37

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

F A C U L T A D D E I N G E N I E R I A

SISTEMA AUTOMATICO DE SINCRONIA DE TIEMPO (SAST)

PARA TELEFONOS DE MEXICO

T E S I S

PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A N:

DOMINGUEZ LOZANO ZORAYA MARIA

MILAN GARCIA DANIEL

ROMERO PEREZ ROBERTO CARLOS

ROSALES VALDEZ LILIANA

DIRECTOR DE TESIS: M.I. LAURO SANTIAGO CRUZ

MEXICO, D.F 2002

PAGINACIÓN DISCONTINUA

SISTEMA AUTOMÁTICO DE SINCRONÍA DE TIEMPO

(SAST)

PARA TELÉFONOS DE MÉXICO

DOMINGUEZ LOZANO ZORAYA.
MILÁN GARCÍA DANIEL.
ROMERO PÉREZ ROBERTO CARLOS.
ROSALES VALDÉZ LILIANA.

DIRECTOR DE TESIS: M.I. LAURO SANTIAGO CRUZ

ÍNDICE

ÍNDICE

INTRODUCCIÓN

INTRODUCCIÓN

En el último siglo, la tecnología ha ido avanzando a pasos agigantados y en los campos más diversos como lo son la medicina, la biología, la genética, la astronomía, la electrónica, la mecánica, la informática, la cibernética y, por supuesto, las telecomunicaciones. Estos adelantos de la tecnología han permitido al hombre contar con un sin fin de comodidades, cambiando su forma de vida de manera muy importante.

En la actualidad las distancias se han reducido considerablemente, ya que uno puede ir de un continente a otro en cuestión de horas. Gracias a los servicios de telecomunicaciones podemos observar programas de TV originados en cualquier parte del mundo vía satélite, podemos tener acceso a una red de área global (Internet) y hacer uso de una infinidad de servicios audio, vídeo y datos a través de la línea telefónica.

El teléfono se ha convertido en el medio de comunicación interpersonal más importante de nuestros tiempos, ya que aparte de cumplir con su propósito original de comunicación de voz, ahora es empleado en la transferencia de datos vía fax o a través de un módem, e incluso, puede ser empleado para servicios de videoconferencia (videoteléfono) aunque con limitaciones de ancho de banda.

Es innegable que la aparición de las computadoras ha impactado favorablemente al desarrollo de la tecnología, y la telefonía no es la excepción. Las grandes centrales telefónicas analógicas basadas en sistemas electromecánicos han sido desplazadas por las modernas centrales digitales completamente computarizadas y controladas por *software* especializado, las cuales cuentan con capacidad de hasta 80,000 líneas de usuario y ofrecen servicios digitales tales como conferencia tripartita (tres a la vez), llamada en espera, transferencia de llamada, identificación de llamada, etc.

En México, Teléfonos de México S.A. de C.V. (TELMEX) se ha consolidado en los últimos años como la empresa líder en servicios de telecomunicaciones (voz, vídeo y datos) en nuestro país y una de las más importantes de Latinoamérica. Cuenta con una amplia red conformada por equipos de diversas tecnologías y para fines diversos tales como conmutación, transmisión y facturación, todos ellos interconectados a través de lo que conocemos como la Red Corporativa de Datos de TELMEX (RCDT).

Debido a la diversidad de los equipos y a la amplitud de la red, ha sido una tarea difícil implementar un sistema que de manera automática sincronice todos los equipos con respecto a una hora de referencia a nivel nacional. Para dar un ejemplo claro, las centrales telefónicas de larga distancia se sincronizan semanalmente y de manera manual por técnicos operadores desde el Centro Nacional de Supervisión (CNS), ubicado en la ciudad de Querétaro, Querétaro. Estos operadores pasan largo tiempo monitoreando y enviando los comandos correspondientes a las centrales para modificar su hora de sistema, tomando +/- 10 segundos como una desviación de tiempo aceptable.

La gerencia de Auditoria de TELMEX, luego de un minucioso análisis de archivos de facturación, detectó que cerca de 3 millones de llamadas telefónicas locales y de larga distancia se dejan de cobrar como consecuencia de la diferencia entre las horas de sistema entre la central origen y la central destino; es decir, por la carencia de un sistema de sincronización de tiempo eficiente. El problema radica en que al momento de establecerse una llamada de la central "A" hacia la central "B", se generan registros de facturación detallada en ambos puntos, cada uno con su hora de sistema correspondiente, entonces, al siguiente corte de facturación, se analizan todos los registros de llamada mediante un algoritmo especial que, cuando por la diferencia de

tiempo entre estos registros no puede precisar si se trata de una misma llamada o de llamadas diferentes, factura únicamente una de ellas, con la consecuente pérdida del cobro correspondiente a la llamada o llamadas restantes.

Este problema se presenta comúnmente cuando se realizan varias llamadas desde una misma línea telefónica, en un corto lapso de tiempo (llamadas de traslape). El algoritmo de facturación, con el objeto de evitar cobros injustificados a los usuarios, toma como una sola llamada todas aquellas que se realicen dentro de un intervalo de tiempo de 30 segundos.

La misma problemática se presenta cuando se cursa tráfico entre centrales telefónicas de TELMEX y centrales de otras empresas de telefonía, como por ejemplo Alestra y Avantel. Cuando estas empresas detectan que se les está cobrando un tiempo de interconexión mayor, TELMEX se hace acreedor a multas y sanciones económicas por parte de la Secretaría de Comunicaciones y Transportes.

En el presente trabajo presentaremos como alternativa de solución a esta problemática, el desarrollo de un sistema distribuido que a través de algoritmos estadísticos y módulos de programa en lenguaje C, con comunicaciones de red TCP/IP, bajo los sistemas operativos UNIX y VOS (*Virtual Operating System*, Sistema Operativo Virtual), sea capaz de monitorear las centrales de conmutación que se encuentren fuera de sincronía, y enviarles los comandos de sincronización correspondientes, a través del sistema de gestión NMA *(Network Monitoring And Analisys, Monitoreo y Análisis de la Red)*, ofreciendo reducir la desviación de tiempo a +/-2 segundos.

Para tal fin, se tomará como referencia la hora del equipo *Stratus** donde corre el NMA, y que a su vez se encuentra en sincronía a través del protocolo NTP con un servidor de tiempo en los Estados Unidos.

Los objetivos que perseguimos con el desarrollo del presente trabajo son los siguientes:

- Contribuir en gran medida a la solución de la problemática de errores en tiempos de facturación, atribuibles a deficiencias en la sincronía entre las centrales de conmutación.

- Contribuir en gran medida a la solución de la problemática de errores en tiempos de interconexión con otras empresas que ofrecen servicios de telefonía, atribuibles a problemas de sincronía.

- Evitar en la medida de lo posible, la adquisición e instalación de módulos de *hardware* y *software* de alto costo en cada una de las centrales telefónicas para su sincronización con un equipo servidor de tiempo.

* El equipo *Stratus* cuenta con una configuración de *Hardware* y *software* tolerante a fallas, lo cual lo hace muy confiable para albergar al NMA (sistema de gestión de centrales telefónicas digitales). En realidad existen dos equipos *Stratus*: uno ubicado en la Ciudad de México, en las oficinas de la colonia Guadalupe Inn y otro en la Ciudad de Guadalajara, Jalisco. El primero atiende la parte centro y sur del país y el segundo se encarga de la parte Norte. Dado que estos equipos operan de manera muy homogénea, el Sistema Automático de Sincronía de tiempo podrá implantarse en ambos subsistemas de gestión.

El presente trabajo está integrado de la siguiente forma:

En el capítulo uno se presenta la historia de la telefonía en México, desde el origen del teléfono, en 1876, hasta nuestros días, en los cuales las centrales telefónicas son completamente digitales y se ha abierto la competencia en servicios de larga distancia.

El capítulo dos contiene un panorama general de conceptos de *hardware* y de *software* que resultan fundamentales para entender la problemática que estamos tratando desde un punto de vista técnico. Estos conceptos son redes de computadoras, dado que la RCDT se conforma a su vez de redes más pequeñas y de distinta naturaleza; programación distribuida, pues es una metodología que permite el desarrollo de sistemas cuyos componentes residen sobre plataformas de *hardware* y *software* diferentes; el sistema operativo UNIX, plataforma que por sus características será la base principal en el desarrollo de este sistema; el protocolo de comunicaciones TCP/IP, el cual es el medio de entendimiento entre los módulos del sistema a través de la red; y por último, el lenguaje de programación C, cuya potencia es indiscutible bajo cualquier plataforma de *software*.

En el capítulo tres se da una introducción a la Programación Cliente-Servidor empleando *sockets*, concepto fundamental en la programación de cada uno de los módulos que conformarán el sistema.

En el capítulo cuatro se presentará un análisis completo de la problemática que pretendemos resolver mediante el desarrollo del Sistema Automático de Sincronía de Tiempo (SAST).

En el capítulo cinco presentará el diseño detallado de cada uno de los módulos que van a integrar el sistema.

En el capítulo seis se mostrará el código fuente de cada uno de los módulos del sistema, así como una descripción detallada de los mismos, a fin de resaltar los puntos importantes de los mismos e incluyendo diagramas de flujo.

En el capítulo siete hará referencia a los protocolos de prueba aplicados al sistema, al proceso de liberación hacia los usuarios finales y a los procedimientos de mantenimiento y administración que habrán de seguirse una vez liberado el sistema.

En el capítulo ocho se realizará un análisis de los resultados obtenidos, haciendo una comparación de los mismos con los objetivos previamente planteados desde el inicio del presente trabajo.

Finalmente presentamos la bibliografía consultada y los apéndices generados.

I. HISTORIA DE LA TELEFONÍA EN MÉXICO

Desde tiempos ancestrales la importancia de la humanidad por comunicarse entre sí ha sido vital, gracias a los avances de la tecnología, hoy en día podemos comunicarnos casi inmediatamente a todos los continentes del mundo. Desde el invento del teléfono, éste ha revolucionado la comunicación entre los seres humanos a pasos agigantados. En todos los países se cuenta con empresas que se dedican a proveer este servicio a todos sus usuarios, en México la empresa de mayor renombre es Teléfonos de México S.A. de C.V. (TELMEX) que pone todos sus esfuerzos por estar a la vanguardia en telecomunicaciones.

1.1 Origen del teléfono

El descubrimiento y aplicación de la electricidad representó un enorme avance social y económico. Pronto las comunicaciones tuvieron que adaptarse a ese nuevo estilo y, entre 1830 y 1844, se inventó el telégrafo eléctrico pero, por desgracia, no ofrecía un contacto personal y directo.

Años más tarde, el sacerdote francés Gauthey propuso un sistema de transmisión de voz mediante tubos acústicos. Posteriormente varios científicos, como Robert Hooke, Joseph Henry, Michael Faraday y muchos otros, realizaron importantes avances pero sin éxito. Hasta que, en 1860, el alemán Philipp Reis inventó un aparato al que llamó teléfono, del griego "hablar a lo lejos", que transmitía sonidos en breves intervalos de tiempo. Sin embargo sería Alexander Graham Bell, un maestro de sordomudos en Quebec, quien realizaría el invento del siglo. En realidad, él estaba entusiasmado por el telégrafo y en 1871 inició sus investigaciones.

Cuatro años más tarde, fabricó un sistema muy elemental que patentó bajo el título "Mejoramiento de transmisores y receptores para telégrafos eléctricos" y, en 1876, patentó otro avance: "Mejoras a la telegrafía". Al mismo tiempo Bell profundizó sus investigaciones para perfeccionar la transmisión de la voz humana. Sus ensayos culminaron aquel 10 de marzo de 1876, al probar su nuevo aparato que funcionaba con una pila eléctrica. Con ese aparato su ayudante sólo recibía señales audibles muy débiles por lo que, para reforzarlas, se le ocurrió aumentar la densidad de la pila eléctrica y le agregó ácido sulfúrico, parte del líquido se derramó y le quemó la pierna: "*Mr. Watson, come here, I want you* [Señor Watson, venga aquí, lo necesito]". Watson escuchó el llamado con insólita claridad... El teléfono había nacido.

Bell solicitó la patente el 8 de abril de 1876, para su invento al que llamó "Receptores telegráficos telefónicos", pero se la concederían hasta el 6 de junio, pues Elisha Grey había presentado un invento similar, y se tuvieron que realizar las investigaciones pertinentes.

A pesar de ello, el 10 de mayo de 1876, en la Academia de Artes y Ciencias de Boston, Bell presentó y demostró su sistema y sus fundamentos científicos, ganándose la admiración de todos. Tiempo después, ya con la patente, el 12 de febrero de 1877, Bell llevó a cabo la primera comunicación de larga distancia y habló por teléfono desde Boston, a través de una línea telegráfica, con un periodista que estaba en Salem, a 25 kilómetros de allí. Un año después se inició la comercialización del teléfono, cuando George W. Coy construyó, en New Haven, la primera central telefónica, con una veintena de clientes. Así surgió la Bell Telephone System Co., que posteriormente se convertiría en la National Bell Telephone Co.

Pronto ingresó a la compañía Francis Blake, quien inventó un nuevo tipo de transmisor que permitía una comunicación bastante más clara. Éste fue el detonador para que el teléfono adquiriera gran popularidad en las grandes ciudades de Estados Unidos y algunas de América Latina. En Europa el impacto fue inmediato. En Gran Bretaña se instaló una central telefónica y luego el servicio pasó a ser monopolio gubernamental, situación similar a la que se produjo en Francia y Alemania. A partir de entonces el avance no cesó, la vida del ser humano había cambiado, al fin lograba rebasar con su voz las distancias.

1.1.1 El inicio de la telefonía en México

Desde que surgió a la vida independiente, México buscó el camino de su desarrollo. Un ejemplo de ello fue el primer enlace telefónico, realizado el 13 de marzo de 1878, entre la ciudad de México y la población de Tlalpan (a una distancia de 16 kilómetros), con la consiguiente admiración popular.

Meses después, la Alfred Westrup & Co. obtuvo permiso para instalar una red telefónica entre la Inspección General de Policía, sus seis Comisarías, la oficina del Gobernador de la Ciudad y el Ministerio de Gobernación; y en 1881, el empresario estadounidense Greenwood obtuvo otra para instalar una red telefónica en la ciudad de México con lo que inició el tendido del cableado público. Pero los habitantes se quejaron de que "los postes y alambres colocados perjudicaban el buen aspecto de la ciudad" y fue necesario explicar la utilidad del Teléfono para lograr que se aceptara.

Varios extranjeros también deseaban prestar este servicio en México, así que se asociaron con la Compañía Telefónica Mexicana (sustentada técnica y financieramente por la Western Electric Telephone Co.) para constituir una nueva empresa en 1882. Sus oficinas se ubicaron en Santa Isabel número 6 1/2, donde hoy se encuentra el Palacio de Bellas Artes.

Al año siguiente esta compañía llevó a cabo la primera conferencia de Larga Distancia internacional entre las Ciudades de Matamoros, Tamaulipas, y el Puerto de Brownsville, Texas, en los Estados Unidos y, muy pronto, obtuvo la concesión para ofrecer el servicio público telefónico.

En menos de seis años ya contaban con 800 suscriptores y editaron el primer Directorio Telefónico. Pero un año después, en 1889, el número de clientes creció con la introducción del teléfono de extensión, los aparatos de escritorio tipo "candelero" y el servicio telefónico de veladores, para dejar mensajes nocturnos que las operadoras entregaban al día siguiente.

En 1891 inició el servicio telefónico local en diversas ciudades del país, donde la Telefónica compraba las pequeñas compañías que lo otorgaban. Con más clientes, en 1894 la Telefónica editó, en su directorio, la primera "sección clasificada" y contrató operadoras que dominaran el idioma inglés, para atender mejor a los suscriptores extranjeros.

Sin embargo, un hecho revolucionaría a la naciente industria telefónica, cuando José Sitzenstätter se relacionó con la L.M. Ericsson de Estocolmo para venderle su concesión en 1905, la cual permitía la explotación del servicio telefónico en la capital y alrededores. Iniciaba la competencia.

En el mismo año la Compañía Telefónica Mexicana aumentó su capital americano y cambió a Compañía Telefónica y Telegráfica Mexicana, S.A. Dos años después la Compañía de Teléfonos Ericsson, S.A., filial de la sueca Mexikanska Telefonaktiebolaget Ericsson, inauguró su servicio el 1 de octubre con 300 suscriptores. A fin de año contaba ya con 650.

Con la Revolución Mexicana los planes de desarrollo de todos los sectores económicos del país cambiaron. La telefonía no fue la excepción, aunque no sufrió las pérdidas de la telegrafía y el ferrocarril. Prueba de ello, es que en 1911 la Ericsson construyó líneas de Tlalnepantla a Cuautitlán, para iniciar el servicio interurbano. Como presidente, Madero estableció que el gobierno tendría preferencia para hacer uso gratuito de la comunicación telefónica a fin de tratar asuntos oficiales, con las restricciones necesarias.

Poco tiempo después las compañías telefónicas frenaron su ritmo de crecimiento, no sólo por la revolución, sino también por el inicio de la Primera Guerra Mundial, ya que en 1914 los materiales telefónicos escaseaban, porque la materia prima de éstos se destinó a la fabricación de armamentos. La Ericsson se había protegido de los acontecimientos nacionales con las gestiones del coronel sueco Thord Hallstrom, miembro del estado mayor del general Villa. Pero en el ámbito internacional la guerra acarreó como consecuencia el aumento de precio de los materiales.

En tanto a la Compañía Telefónica y Telegráfica Mexicana se le intervino el servicio el 6 de enero de 1915, pues los conflictos laborales se agudizaron tanto que el gobierno embargó las redes de la empresa. Esta situación tardaría diez años en resolverse; mientras tanto, de conformidad con la nueva Constitución de México (1917) el gobierno emitió una ley que incrementaba el monto de los impuestos, incluido el que gravaba al servicio telefónico, y que habría de suprimirse en 1920.

1.1.2 Llamando desde México al mundo

Tras el fin de la Primera Guerra Mundial, en el ámbito de las telecomunicaciones, y en especial en el de la telefonía, iniciaron los planes de utilizar las comunicaciones eléctricas con ondas portadoras. La Ericsson, que ya tenía 32 concesiones para ofrecer servicio público y privado, adquirió dos estaciones portátiles inalámbricas con un radio de comunicación de 200 kilómetros. Con lo que introdujo en México el sistema telefónico automático, que sería inaugurado años después y sustituiría gradualmente al sistema de operadoras.

En 1924 la compañía Ericsson inauguró la primera central telefónica automática, conocida como la central Roma, que entraría en funciones dos años después, con capacidad para conectar diez mil líneas. Al año siguiente, para concluir el conflicto en la Compañía Telefónica y Telegráfica Mexicana, S. A., ésta fue adquirida por la *ITT (ITT: International Telephone and Telegraph Co., Compañía Internacional de Teléfonos y Telégrafos)*, con lo que pudo competir en un mismo nivel con la Ericsson.

Las redes telefónicas crecieron de tal manera por el aumento del número de suscriptores que, para distinguir los teléfonos de cada compañía, se decidió que la Ericsson utilizara exclusivamente dígitos, mientras que la otra compañía usaría dígitos y letras. Las mencionadas compañías, funcionaban con diferentes tecnologías por lo que los clientes

no podían comunicarse con suscriptores cuyos contratos no estuvieran suscritos con la misma.

En 1925 el Gobierno Federal celebró un convenio para tender el cableado telefónico entre México y Estados Unidos. La Compañía Telefónica y Telegráfica Mexicana, S.A. obtuvo la concesión para explotar el servicio de larga distancia, un año después la Ericsson obtendría la suya. Obviamente, el despegue del servicio de larga distancia fue inmediato.

El 29 de septiembre de 1927 la Compañía Telefónica y Telegráfica Mexicana, S. A., enlazó la primera conferencia telefónica entre un presidente mexicano, el general Plutarco Elías Calles, y uno norteamericano, Calvin Coolidge. Poco después, el 29 de noviembre, se inauguró la línea telefónica entre México y Canadá.

El 1 de julio de 1928 tuvieron éxito los esfuerzos por lograr una comunicación telefónica con Europa. "Señor Valenzuela: habla Estrada. Tengo mucho gusto en saludarlo", fueron las primeras palabras transmitidas entre México y Europa, mediante un sistema que consistía en una combinación de líneas telefónicas de tierra y circuitos radiotelefónicos a través del Atlántico. La conversación se inició entre el licenciado Genaro Estrada, subsecretario de Relaciones Exteriores, y el licenciado Valenzuela, ministro plenipotenciario en la Gran Bretaña. Otro éxito más de la Compañía Telefónica y Telegráfica Mexicana.

En el nuevo servicio telefónico transoceánico se incluyeron la ciudad de México, Querétaro, San Luis Potosí, Saltillo, Monterrey, Tampico y Nuevo Laredo, localidades que se podrían comunicar, en Europa, con Inglaterra, Escocia, Gales, Alemania, Holanda, Bélgica, Francia, Suecia y Dinamarca. Con España se entró en contacto el 30 de noviembre de 1928.

Inicialmente este servicio se limitaba de las 6:30 a.m. a las 10:00 p.m., hora de México; muy pronto, 30 mil de los 60 mil aparatos telefónicos instalados se conectaron al servicio internacional. Hasta esos momentos los únicos países de América con los que se había logrado establecer comunicación telefónica eran Estados Unidos, Canadá y Cuba. No fue sino hasta 1930 cuando se enlazaron Norte y Sudamérica gracias al circuito, transmisor y receptor entre la Compañía Internacional de Radio (de Argentina) y la American Telephone and Telegraph Co. (de Estados Unidos).

Por otra parte, en la ciudad de México funcionaban 14 centrales automáticas, en su mayoría de la compañía Ericcson: Apartado, Chapultepec, Roma, Valle, Coyoacán, Mixcoac, Madrid, Peralvillo, Portales, San Ángel, Condesa, Santa María, Tacubaya y Victoria. Desde entonces se aceleró la competencia entre la L. M. Ericsson y la ITT, con el incremento de sus conflictos. Por lo que, en junio de 1936, el presidente Cárdenas comunicó a ambas compañías su obligación de enlazar sus líneas y combinar sus servicios, sustentando su orden en razones de interés público.

1.2 Nacimiento de Teléfonos de México (TELMEX)

Desde 1936 se ordenó la fusión de líneas de las dos principales compañías telefónicas de México, sin embargo, el proyecto se suspendió debido a la Segunda Guerra Mundial que afectó a la mayoría de las empresas transnacionales. Además los avances tecnológicos separaban a las empresas, pues mientras Ericsson de capital sueco aumentó de cinco a seis las cifras de sus números telefónicos en 1940, la Compañía Telefónica y Telegráfica

Mexicana, S. A. adquirida por ITT, adoptó este sistema hasta 1946. Con lo que el 2 de agosto de 1946 el gobierno anunció el enlace definitivo de las líneas de ambas telefónicas. Con esta comunicación entre ambas empresas y la situación económica tan favorable por la que atravesaba el país, fue posible que, el 23 de diciembre de 1947, se constituyera una de las empresas más trascendentes en la historia del México contemporáneo: Teléfonos de México, S. A. (TELMEX).

TELMEX surgió de las diversas negociaciones entabladas entre la L. M. Ericsson de Estocolmo y Axel Wenner-Gren, para crear una empresa mexicana que asumiera el servicio que prestaba la Empresa de Teléfonos Ericsson, S. A., filial de la matriz sueca. Las negociaciones culminaron con la formación de la nueva empresa telefónica, que inició sus actividades el 1 de enero de 1948 con cinco accionistas: Corporación Continental, S. A., Empresa Ericsson, Bruno Pagliai, Octavio Fernández R. y José Joaquín César.

Durante el primer año de labores TELMEX emprendió la ardua tarea de enlazar en forma automática los dos sistemas telefónicos existentes: el suyo propio (antes compañía Ericsson) y el de la Compañía Telefónica y Telegráfica Mexicana, S. A.; el enlace fue finalmente inaugurado por el presidente Alemán, el 9 de enero de 1948, y benefició a 149,612 clientes.

El 16 de febrero de 1950 se llegó a un acuerdo para que Teléfonos de México adquiriera la Compañía Telefónica y Telegráfica Mexicana, S. A. y el 29 de abril se firmó el documento formal. El 3 de mayo el gobierno mexicano, Wenner-Gren, la ITT y la L. M. Ericsson asignaron un acuerdo definitivo que consolidó a TELMEX como la principal empresa telefónica del país.

El 1 de abril de 1952 entró en vigor la nueva Ley del Impuesto sobre ingresos por Servicios Telefónicos, entre otros servicios, que gravaría con 10 por ciento el servicio de larga distancia y con un 5 por ciento el servicio local. El producto del impuesto sería destinado a financiar el mejoramiento y la ampliación del servicio telefónico. Como consecuencia, ese mismo año se inauguró la nueva central de Chapultepec, con una capacidad inicial de 18 mil líneas.

Durante el primer año de gobierno de don Adolfo Ruiz Cortines se puso en servicio el sistema de microondas entre el Distrito Federal y Puebla, con 23 canales telefónicos, y se introdujo el servicio medido, que permitió aplicar un principio equitativo y justo de pago debido a que las cuotas no aumentarían, siempre y cuando no se sobrepasara el número de conversaciones establecidas dentro de la cuota básica. A finales de ese año, se equilibraron las participaciones de la L. M. Ericsson y la ITT mediante la firma de diversos acuerdos, entre los cuales destaca el retiro de la participación Axel Wenner-Gren en el capital de la empresa, lo que dejó al público contratante del servicio una pequeña participación accionaria.

1.2.1 La nueva tecnología

TELMEX, en los siguientes cinco años, colocó en el mercado acciones y obligaciones, tanto comunes como nominativas y al portador, lo que le permitió obtener recursos para poder establecer 25 mil nuevos servicios por año. El gobierno tomó las medidas necesarias para que todo suscriptor, al cambiar de domicilio sus aparatos, adquiriera acciones y obligaciones de la compañía.

En 1955 casi todas las centrales de la zona metropolitana aumentaron los equipos de intercomunicación y los cables troncales, lo que repercutió en un alza de las tarifas, tanto en el servicio local como en el de larga distancia. Pero, durante ese año, TELMEX sufrió grandes pérdidas debido a la interrupción del servicio por los huracanes y graves inundaciones ocurridos en el estado de Tamaulipas, por lo que hubo que emprender la reconstrucción de las redes dañadas.

En 1956, la empresa decidió proveerse de equipo telefónico fabricado en el país, por lo que el 5 de diciembre se constituyó la compañía Industria de Telecomunicación, S.A. de C.V. (INDETEL). En ese mismo año, la sociedad capitalina fue testigo de la instalación de los primeros teléfonos de alcancía, sistema de aparatos de servicio público que, hasta nuestros días, sigue operando.

En el mes de julio del año siguiente la ciudad de México fue golpeada por un fuerte sismo que afectó a más de 1,500 suscriptores, de los cuales 400 eran conmutadores privados. No obstante, el 30 de octubre se dio un gran paso al inaugurar el servicio de télex entre el Distrito Federal y Acapulco, Guerrero.

A fin de aumentar la capacidad de las centrales automáticas, tanto cualitativa como cuantitativamente, la empresa inició la construcción del edificio de TELMEX ubicado en Parque Vía, que serviría como sede coordinadora de la administración telefónica en el país, y que fue inaugurado dos años después.

Gracias a la permanente venta de acciones al público, aumentó el capital de la compañía, lo que dio origen a la mexicanización de Teléfonos de México.

El satélite de comunicaciones Telstar fue lanzado al espacio en el verano de 1962, estaba patrocinado por el sistema Bell y la NASA, y fue el primero en funcionar con frecuencias de microondas. Gracias a ello, el sistema de microondas quedó instalado en forma definitiva entre las ciudades de México, Monterrey y Nuevo Laredo. Éste fue inaugurado por el presidente López Mateos y el secretario de Comunicaciones y Transportes, ingeniero Walter C. Buchanan, el 11 de enero de 1963. La instalación contribuyó en forma inmediata al perfeccionamiento del servicio de conmutación automática de larga distancia, ya que las operadoras de nuestro país se vieron en la posibilidad de marcar directamente el número telefónico de cualquier abonado incluido en la ruta, además de los respectivos de Canadá y Estados Unidos.

El 14 de mayo de 1963 se produjo un acontecimiento memorable para las telecomunicaciones: se llevó a cabo la primera transmisión desde Cabo Cañaveral, Estados Unidos, hacia México, para cubrir el lanzamiento al espacio del astronauta estadounidense Gordon Cooper; pero fue postergado y se televisó al día siguiente. TELMEX transmitió el acontecimiento a través del sistema de microondas, que por entonces estaba equipado con dos canales, uno en operación y otro de reserva con 21 estaciones repetidoras.

Esto hizo posible transmitir sucesos de trascendencia mundial como el asesinato del presidente J. F. Kennedy, la visita del primer mandatario francés Charles De Gaulle, la ceremonia de entrega del Chamizal, y algunos eventos deportivos.

Con la puesta en órbita del satélite norteamericano Pájaro Madrugador se dio inicio al programa de televisión Mundo Visión, en el cual se utilizaron la red de microondas de

TELMEX, en el territorio nacional, y la red de la ATT, en Estados Unidos. Las señales del satélite se retransmitieron a la tierra mediante sus 240 canales de telefonía.

Después, Teléfonos de México realizó considerables inversiones en la ampliación del sistema telefónico, como la del télex "Telégrafo a domicilio" para grandes empresas, se instalaron 28 centrales de telex.

1.2.2 Expansión de la empresa

En 1964, como parte de su programa de telecomunicaciones, TELMEX celebró contrato con la empresa Guatemalteca de Telecomunicaciones Internacionales para establecer, por primera vez, un enlace telefónico directo con el país del sur.

En ese mismo año, ante la creciente demanda del servicio de larga distancia, se determinó establecer el sistema de larga distancia automática (LADA) para que los suscriptores hicieran sus llamadas de teléfono a teléfono sin intervención de la operadora. Par ello, fue necesario establecer un sistema de numeración nacional.

En septiembre de 1965 se instaló el primer equipo LADA 91 (nacional) en Toluca, Estado de México, pero no fue sino hasta 1967 cuando varias ciudades del interior de la República se incorporaron a este servicio. En este último año se instalaron los equipos para el control de operación de los aparatos telefónicos públicos de alcancía en las centrales del Distrito Federal y se colocó el teléfono un millón.

En 1968, cuando TELMEX celebraba el décimo aniversario de su mexicanización, se llevó a cabo la incorporación total de la ciudad de México al servicio LADA. Al mismo tiempo entró en operación el servicio de información de emergencia con la clave 07.

Pero el acontecimiento del año, en materia de telecomunicaciones, fue la transmisión de las imágenes de la XIX Olimpiada. Fue necesario instalar una red subterránea con una longitud de 284 kilómetros de ductos, 203,400 kilómetros de conductores, 19,840 teléfonos en cables (por pares) y un cableado coaxial para troncales urbanos, el primero en el mundo; se invirtieron 19,840 horas-hombre. Se inauguró un edificio especial para el evento, conocido hoy como "Urraza".

En 1969 las comunicaciones internacionales entraron en una nueva era cuando se consolidó el servicio oficial internacional de larga distancia vía satélite, con una comunicación directa a Roma, Italia, para después extenderse a otros países de América del Sur y algunos más de Europa. Esto fue posible gracias al uso del satélite IS-III-F2 de la serie Intelsat III, con capacidad para 1,200 canales telefónicos y un promedio de vida activa de cinco años.

El 7 de agosto de 1969 se inició la construcción del centro telefónico San Juan. Éste tendría una torre de 100 metros de altura que se utilizaría para sustentar, en tres plataformas, todas las antenas de los sistemas de microondas de alta, mediana y baja capacidad. Habría también un edificio para operadoras, servicios especiales, departamentos técnicos del Valle de México y un auditorio.

Al año siguiente nuestro país volvió a ser protagonista de un acontecimiento deportivo internacional: el Campeonato Mundial de Futbol. A fin de cumplir eficientemente con la transmisión se instaló la infraestructura adecuada, con la que el comité organizador,

prensa y Telesistema Mexicano quedaron comunicados con más de mil líneas en el Distrito Federal, 334 en Guadalajara, 291 en León, 247 en Toluca y 247 en Puebla, además de 150 líneas especiales, 100 casetas de larga distancia instaladas en los centros de prensa y 129 líneas privadas para el uso de télex y telefoto.

Al mismo tiempo, en el Valle de México, se antepuso un dígito a los números telefónicos: el número 5, para llegar a siete cifras. Esto se debió al incremento de abonados, que ascendieron a un poco menos de millón y medio.

El panorama para la empresa se tornó particularmente halagüeño cuando alcanzó el segundo lugar mundial en desarrollo. En el país se hacía cada vez más común el uso de microondas, por lo que se instalaron 19 nuevos sistemas y se pusieron en operación dos sistemas de cable coaxial; la conmutación automática quedó conectada casi en su totalidad. El 20 de julio de 1970, cuando se llevó a cabo una comunicación telefónica entre Toluca y Washington, D. C., se inauguró un nuevo sistema automático de larga distancia (LADA 95), el primero de su tipo en Latinoamérica, con un alto grado de transmisión y recepción de mensajes directos.

1.2.3 La privatización de TELMEX

1989 representó el inicio de una nueva etapa de desarrollo tecnológico, financiero y de servicios, cuyos objetivos eran el mejoramiento de la calidad del servicio, el crecimiento y la expansión del mismo, la modernización tecnológica y la diversificación de los bienes y servicios. Como una respuesta a este reto de modernización, y para competir en la venta y promoción de los servicios digitales, TELMEX inició las operaciones del Centro de Telecomunicaciones Avanzadas, institución de investigación integrada por un grupo de especialistas altamente calificados en la materia. Este centro contaba con la primera Red Digital de Servicios Integrados (RDSI) y fue constituido para funcionar como un laboratorio cuyo objetivo era desarrollar nuevos servicios.

En ese año Teléfonos de México inició su participación en el mercado de los servicios celulares a través de su filial Radiomóvil Dipsa, S.A. de C.V., para satisfacer eficazmente la demanda de los usuarios que requieren telecomunicaciones personales y tecnología de vanguardia. En tanto, el Centro de Investigación y Desarrollo logró la aplicación de tecnología avanzada para el mejoramiento de la red nacional, diseñó sistemas y desarrolló equipos complementarios que han hecho más eficiente la operación de los ya existentes.

A fin de modernizar su actividad interna TELMEX inició un programa de automatización para los procesos básicos de atención al usuario, tales como la recepción de solicitudes de líneas y servicios, quejas y aclaraciones, reparación, instalación y cobranza. El presidente Salinas en su "Plan Nacional de Desarrollo 1989-1994" en lo referente a la modernización de las telecomunicaciones señaló: "La indispensable modernización y expansión de las telecomunicaciones requerirá de grandes inversiones, que deberán financiarse con participación de los particulares; el propósito es no distraer recursos fiscales necesarios para atender las legítimas demandas de salud, educación, vivienda y adecuación del resto de la infraestructura. El Estado ejercerá la rectoría en las telecomunicaciones e inducirá su desarrollo, mediante un nuevo marco regulador que tenga en cuenta el cambio tecnológico habido en los últimos años. La regulación dará la debida seguridad jurídica a los participantes en el sector".

En septiembre de 1989, el gobierno federal anunció su intención de privatizar Teléfonos de México, vendiendo su participación en el capital de la empresa y facilitando así el proceso de modernización de las telecomunicaciones en nuestro país. Pues el desarrollo tecnológico modificó las condiciones tradicionales de organización existentes en el sector y varios países cuyas telecomunicaciones estaban bajo un régimen de propiedad estatal, procedieron a abrirlas a la competencia y a encomendar la parte medular de ellas al sector privado.

Las condiciones fundamentales que se persiguieron con la privatización de TELMEX, establecidas por el licenciado Carlos Salinas de Gortari fueron: mantener la soberanía del Estado en el sector; conservar la mayoría del capital en manos de empresarios mexicanos; garantizar la expansión continua de la red; permitir la participación de los trabajadores en el capital de la empresa; elevar la calidad del servicio hacia niveles internacionales; fortalecer la Investigación y el desarrollo tecnológico. La condición de que se mantuviera el control mayoritario de la institución en manos de empresarios mexicanos requirió el diseño de una nueva estructura accionarla que, sin modificar los derechos de los accionistas existentes, permitiera simultáneamente una amplia participación de inversionistas extranjeros. Las instalaciones de TELMEX fueron visitadas por 23 empresas nacionales y extranjeras. El 15 de noviembre de 1990 se recibieron ofertas de tres grupos encabezados por empresarios mexicanos. Finalmente, en estricto apego al calendario dado a conocer públicamente, y después de una cuidadosa homologación de las posturas, el gobierno federal anunció el grupo ganador el 9 de diciembre. Consorcio está integrado por el Grupo Carso, Southwestern Bell International Holdings y France Cables et Radio.

Grupo Carso, S.A. de C V., es una controladora diversificada, que cuenta con una capacidad demostrada en la administración de empresas que operan en mercados altamente competitivos, tanto en el ámbito nacional como en el internacional. Sus principales áreas de operación son los productos de consumo, tiendas departamentales y restaurantes, construcción y exportación. France Cables et Radio es una empresa filial de France Telecom. Este grupo opera 28 millones de líneas telefónicas y cuenta con más de cinco millones de abonados a su sistema de videotexto. France Telecom logró triplicar la red francesa en sólo diez años y fue el primer operador en el mundo que comercializó la Red Digital de Servicios Integrados. Southwestern Bell International Holdings es una subsidiaria de Southwestern Bell Corporation, que cuenta con 66,700 empleados y administra 12 millones de líneas telefónicas en los Estados Unidos. La participación en TELMEX de estos socios tecnólogos es garantía para que el país pueda desarrollar una red de telecomunicaciones más moderna que impulse el progreso económico de México. También abre la puerta a una revolución tecnológica que no sólo ha multiplicado las formas posibles de acceso a los últimos avances en materia de telefonía sino que a la vez ha modificado dramáticamente sus costos.

1.2.4 10 Millones de líneas

Desde su privatización y a lo largo de estos últimos ocho años, la Empresa ha continuado con pasos seguros en la consolidación de sus productos y servicios, además de estar acercándose firmemente hacia la internacionalización. TELMEX tiene la fortaleza para enfrentar retos cada vez mayores porque ha consolidado una planta laboral capacitada y en creciente desarrollo, porque cuenta con un equipo directivo capaz, porque sus finanzas son sanas y porque día con día sus trabajadores se esfuerzan por brindar al Cliente todas

las ventajas que les ofrece TELMEX, además de que su evolución ha requerido inversiones de más de 17 mil millones de dólares.

Al finalizar 1998, TELMEX suministraba servicios de telecomunicaciones a 24 mil 711 poblaciones de todo el territorio nacional y, a un ritmo de instalación de 4 mil líneas diarias, se llegaron a sumar 10 millones de líneas en servicio. En 1991, se inició la construcción de la Red Nacional de Fibra Óptica de Larga Distancia, que enlaza a 54 ciudades del país a través de más de 37 mil kilómetros de cable de Fibra Óptica, que pueden transmitir 270 mil conferencias de larga distancia simultáneamente. Lo que constituye la columna vertebral de la red de telecomunicaciones de larga distancia.

Con la modernización de la red de larga distancia, se desarrollaron los servicios multimedia, integrando tecnologías y permitiendo la interacción simultánea de cada una de ellas. Por lo que desde 1992, TELMEX proporciona el servicio de videoconferencia, denominado Video Enlace Digital, a empresas e instituciones que requieren comunicarse con varios puntos en México o en el mundo, ya que cuenta con 100 salas de Videoenlace ubicadas en las principales ciudades del país, que a su vez están enlazadas con salas de 42 países.

Un paso fundamental para la prestación de este servicio fue la introducción de nuevos protocolos de transmisión, que facilitaran el manejo de diversas tecnologías. Dos de estos protocolos son el *SDH (SDH: Syncronous Digital Hierarchy, Sincronía Digital Jerárquica)* y el ATM *(ATM: Asynchronous Transfer Mode, Modo Asíncrono de Transferencia)* que permiten la transmisión a alta velocidad y manejar, al mismo tiempo, señales de voz, datos e imágenes.

Por otra parte, TELMEX emprendió un ambicioso programa de modernización en los Centros de Tráfico por Operadora y puso a funcionar 39 Centros Digitales de Tráfico Avanzado en las principales ciudades del país con 420 posiciones digitales computarizadas, que sustituyeron a los viejos conmutadores con más de 30 años de antigüedad.

En los primeros cinco años casi se duplicó la cobertura total de la Red. Se sustituyeron las centrales analógicas y electromecánicas con equipos digitales, logrando que la planta telefónica de las ciudades de México, Monterrey y Guadalajara fuera 100 por ciento digital. En las principales ciudades del país se pusieron en operación centros de Atención Telefónica a Clientes, con los servicios de reportes por fallas y de información, mediante la introducción de posiciones digitales por operadora; este importante cambio se realizó en 1991 para el 04 (hoy 040), y a partir de 1995 para el 05 (ahora 050).

En 1995 inició sus operaciones el Centro de Administración de la Red (CAR) en la ciudad de México, desde entonces a la fecha, se ha abierto un CAR en siete ciudades importantes del país. Los CAR son centros avanzados de monitoreo y análisis que permiten establecer un control más estricto de todos los elementos que conforman el sistema telefónico, supervisando la operación de las centrales telefónicas, para poder detectar oportunamente fallas en las mismas, así como en equipos de transmisión y de fuerza y clima con el fin de reducir tiempos de atención y solución de problemas.

Asimismo, se renovó la Red Local de Planta Exterior en los distritos telefónicos más afectados por el paso del tiempo y la falta de mantenimiento. Se puso en operación la Señalización por Canal Común No. 7 que es un protocolo de comunicación de datos a alta

velocidad que posibilita la optimización de la red telefónica y la introducción de una plataforma de servicios avanzados de telecomunicaciones en forma masiva. Este sistema ya obtuvo la certificación ISO 9002.

Para diciembre de 1995 entró en operaciones el Sistema Trasatlántico de Cable Submarino Columbus II, capaz de transmitir señales de voz, datos e imágenes con 23 mil canales, pudiendo manejar 320 mil llamadas a una velocidad de 565 megabytes por segundo por cada par de fibras.

Asimismo, se creó el Centro Nacional de Supervisión de la Red de Larga Distancia, en Querétaro, cuyas instalaciones fueron puestas en operación en 1995 y en 1997 recibió la certificación ISO 9002. A través de este centro se controla y vigila, en forma permanente y confiable, el estado técnico del sistema de larga distancia, la operación y mantenimiento, la corrección de averías, la prevención de siniestros, el suministro del servicio y la expansión de la red misma.

En enero de 1996 TELMEX se ubicó, nacional e internacionalmente, como una empresa completa y sólida de telecomunicaciones, ante la apertura a la competencia, en el mercado de larga distancia. La participación de mercado de la Empresa, en términos de líneas, en servicio de Larga Distancia en las 100 ciudades en competencia al finalizar 1998, fue de 79.2 por ciento, en comparación con 74.8 obtenido al cierre de 1997.

TELMEX cuenta ya con la más amplia red de telefonía pública, que para 1998 alcanzó un total de 231 mil 873 aparatos instalados que funcionan con tecnología chip, cifra 46.9 por ciento superior a la registrada en 1997.

Actualmente, 360 Centros de Atención TELMEX ofrecen atención integral: recepción de pagos, contrataciones, área de tienda, videoaclaraciones, acceso a Internet, cajero automático para pagos las 24 horas y sala de espera, en modernas y cómodas instalaciones, más cercanas al Cliente.

Como parte de la visión a largo plazo de construir una infraestructura de telecomunicaciones de clase mundial, en 1998 se puso en marcha el proyecto Cable Submarino Mar de Cortés, para integrar a Baja California a la Red Nacional de Fibra Óptica.

También en 1998, se realizaron las gestiones necesarias para que, mediante la Certificación ISO 9002, se reconociera el nivel de calidad internacional que posee el Centro de Atención al Mercado Empresarial de TELMEX. Esta certificación complementa el esfuerzo de calidad de clase mundial previamente alcanzado por el Centro Nacional de Supervisión de la Red de Larga Distancia, la Red Inteligente y el Sistema de Señalización por Canal Común No. 7.

Como podemos observar la telefonía en México ha avanzado a grandes pasos, donde TELMEX es la empresa de mayor prestigio que cubre las necesidades de comunicación entre México y el mundo. Para poder llevar a cabo esta labor TELMEX cuenta con una serie de redes de comunicaciones a escala mundial las cuales permiten un tiempo de respuesta inmediato entre la persona que realiza la llamada y su receptor, independientemente que sea vía persona – persona, haciendo uso de una operadora o mediante la Internet. En el siguiente capítulo abordaremos las características de las redes incluyendo someramente la evolución de las computadoras a través del tiempo.

II. GENERALIDADES

En el presente capítulo se presenta al lector una visión general de conceptos de *hardware*, los cuales resultan fundamentales para comprender la plataforma sobre la cual residen los sistemas de *software*, como el que estamos proponiendo como alternativa de solución a la problemática ya descrita. Haremos un repaso que abarcará desde los orígenes de la computadora y su evolución hasta llegar a las grandes redes de datos que conocemos hoy en día.

2.1 Conceptos de *Hardware*

En el desarrollo de cualquier sistema de información es de vital importancia el conocimiento del *hardware* como del *software* donde va a residir. Del diseño y optimización de ambas plataformas dependerá la confiabilidad y eficiencia de cualquier sistema.

2.1.1 El Origen de las Computadoras

- Antecedentes

Una de las cuestiones que siempre ha fascinado al hombre es la relacionada con la actividad de contar y el concepto de número. La realización de cálculos aritméticos mediante las cuatro operaciones básicas (suma, resta, multiplicación y división) no supone dificultad para cualquier persona con una mínima formación escolar. Sin embargo, el simple intento de efectuar una multiplicación con cifras romanas representa dificultades muy considerables. Esta consideración permite entrever el largo camino recorrido por el hombre hasta idear los ágiles métodos actuales de cálculo.

Es muy probable que el hombre primitivo utilizara los dedos para realizar cálculos sencillos; con los dedos de las manos sólo podían llegar hasta diez y con la ayuda de las manos de otros individuos podía ir acumulando decenas. Una cuerda con nudos, hechos a intervalos regulares, contribuyó a su vez a la tarea de contar, y en la antigua civilización egipcia se agrupaban guijarros en montones de a diez con el mismo objeto.

De ahí que entre las primeras herramientas que inventó esté un ingenio mecánico capaz de liberarlo de la pesada tarea de calcular a mano. De hecho, la palabra cálculo proviene del latín *calculus*, que nombra las pequeñas piedras que se usaban hace miles de años como auxiliares en las cuentas (en una especie de ábaco formado con ranuras en el suelo y operado manualmente) y que se han encontrado en excavaciones arqueológicas.

El paso siguiente fue la invención de un tablero o ábaco para el cálculo. En su forma más sencilla el ábaco consiste en una tabla con una serie de hendiduras; en la primera se colocan tantas piedras como unidades hay que representar; en la segunda, tantas como decenas, y así sucesivamente. El ábaco supuso un gran avance en todas las culturas que lo adoptaron; hay indicios de su utilización tanto en el mundo mediterráneo como en la China de Confucio o en las civilizaciones precolombinas de América. En la actualidad, si bien en formas más evolucionadas, aún se emplea en muchos países, especialmente en el extremo oriente.

El ábaco representa la primera calculadora mecánica, aunque no se le puede llamar computadora porque carece de un elemento fundamental: el programa, que no se logrará sino hasta mucho tiempo después.

Otro ingenio mecánico, que tampoco es una computadora, fue la máquina de calcular inventada por Blaise Pascal (1623-1662). Se trata de una serie de engranes en una caja que proporcionan resultados de operaciones de suma y resta en forma directa – mostrando un número a través de una ventanilla- y que por este simple hecho tiene la ventaja de que evita tener que contar, como en el caso del ábaco; además, presenta los resultados en forma más accesible y directa.

Podría decirse que la computadora nació alrededor de 1830 con la invención de la "máquina analítica" de Charles Babbage (1792, 1871). Este diseño, nunca llevado por completo a la práctica, contenía todos los elementos que configuran a una computadora moderna y la diferencian de una calculadora.

La máquina analítica estaba dividida funcionalmente en dos grandes partes: una que ordenaba y otra que ejecutaba las órdenes. Esta última era una versión muy ampliada de la máquina de Pascal, mientras que la otra era la parte clave. La innovación consistía en que el usuario podía, cambiando las especificaciones de control, lograr que la misma máquina ejecutara operaciones complejas, diferentes de las que se hicieron antes.

Esta verdadera antecesora de las computadoras contaba también con una sección en donde se recibían los datos para trabajar. La máquina seguía las instrucciones dadas por la unidad de control, las cuales indicaban que hacer con los datos de entrada y se obtenían luego los resultados deseados. La aplicación fundamental para la que el gran inventor inglés desarrolló su máquina era elaborar tablas de funciones matemáticas usuales (logaritmos, tabulaciones trigonométricas, etc.) que requerían mucho esfuerzo manual.

Esta primera computadora "leía" los datos (argumentos) de entrada por medio de las tarjetas perforadas, invento del francés Joseph M. Jacquard (1752-1834), y que dieron inicio al surgimiento de la industria de los telares mecánicos durante la revolución industrial.

No obstante, la máquina analítica nunca se puso en funcionamiento precisamente por la dificultad para lograr los cambios necesarios en las especificaciones de la unidad de control, lo cual requería alterar la disposición de ciertos elementos mecánicos de la máquina. Es perfectamente válido, sin embargo, referirse a esta máquina como la primera computadora digital, porque el concepto "digital" no presupone el concepto "electrónico".

Hoy en día casi todas las computadoras en uso son digitales, ya que el empleo de las analógicas se restringe a aplicaciones muy particulares en la ingeniería o biología.

En la figura 2.1 se describe el esquema elemental de la máquina inventada por Charles Babbage, con el propósito de explicar algunas de las características más importantes de toda computadora digital moderna.

La sección de control se convierte en el concepto fundamental, pues es la parte que dirige el procesamiento, de acuerdo con un programa previamente introducido en el "almacén" (como llamó Babbage a la memoria) de la máquina. Así, una computadora está formada por una **unidad de entrada**, que recibe tanto la información a procesar como las instrucciones (programa); la **unidad de memoria**, que almacena la información; la **unidad de procesamiento** (aritmética y lógica), que ejecuta los cálculos sobre la información; la **unidad de control**, que dirige a todas las demás unidades, determinando cuando se debe

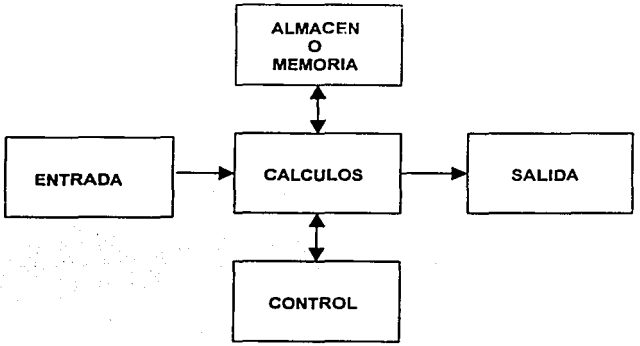


Fig. 2.1 Esquema básico de la máquina analítica de Babbage.

de leer la información, en qué lugares debe almacenarse, cuando debe funcionar la unidad aritmética, etcétera; y una **unidad de salida**, que muestra la información ya procesada en forma de números o gráficas.

A casi cien años de distancia de Babbage, en 1947, se diseño la primera computadora electrónica digital, que tenía gran parecido con la máquina analítica –y esto habla del genio de su antecesor, aunque antes hubo algunos otros esfuerzos.

En 1932, el investigador estadounidense Vannevar Bush construyó en el Instituto Tecnológico de Massachusetts una calculadora electromecánica conocida como el analizador diferencial, pero era de propósito específico y no tenía capacidades de programación.

Asimismo, en 1944 se construyó en la Universidad de Hardvard, en Estados Unidos, la computadora IBM Mark I, diseñada por un equipo encabezado por Howard H. Aiken, No obstante, esta máquina no califica para ser considerada como la primera computadora electrónica, porque no era de propósito general y su funcionamiento estaba basado en dispositivos electromecánicos, llamados relevadores.

- La Primera Computadora

Un equipo dirigido por los doctores en ingeniería, John Mauchly y John Eckert, de la Universidad de Pennsylvania, terminó en 1947 una gran máquina electrónica llamada ENIAC (*ENIAC: Electronic Numerical Integrator And Calculator*, Integrador y Calculador Numérico Electrónico) que puede ser considerada como la primera computadora digital electrónica de la historia.*

* Aunque eso está en duda, porque hubo otros pioneros: Konrad Zuse en Alemania (computadora Z3), Alan Turing y Maxwell Newman en Inglaterra (computadora Colossus), y John Atanasoff y Clifford Berry en Estados Unidos (computadora ABC).

Esta máquina era enorme, ocupaba todo un sótano de la universidad, pesaba toneladas, tenía 18 000 tubos de vacío, consumía 140 kW de energía eléctrica y necesitaba todo un sistema de aire acondicionado industrial. Pero era capaz de efectuar alrededor de cinco mil operaciones aritméticas en un segundo, dejó atrás de manera definitiva las limitaciones humanas de velocidad y precisión e inauguró una nueva etapa en las capacidades de procesamiento de datos.

El proyecto, auspiciado por el Departamento de Defensa de Estados Unidos, culminó dos años después, cuando se integró a ese equipo el ingeniero y matemático húngaro naturalizado estadounidense John Von Neumann (1903-1957). Sus ideas resultaron tan fundamentales para los desarrollos posteriores que bien puede considerarse el padre de las computadoras.

La computadora diseñada por este nuevo equipo se llamó EDVAC (*EDVAC: Electronic Discrete Variable Automatic Computer*, Computadora Electrónica Automática de Variables Discretas); que tenía cerca de cuatro mil bulbos y usaba un tipo de memoria basado en tubos llenos de mercurio por donde circulaban señales eléctricas sujetas a retardos.

La nueva idea fundamental resulta muy sencilla, pero de vital importancia: permitir que en la memoria coexistan datos con instrucciones, para que entonces la computadora pueda ser programada de manera "suave"[†], y no por medio de alambres que interconectaban eléctricamente varias secciones de control, como en la ENIAC.

Esta idea, que incluso obliga a una completa revisión de la arquitectura de las computadoras, recibe entonces el nombre de **modelo de von Neuman**. Alrededor de este concepto gira toda la evolución posterior de la industria y la ciencia de la computación.

De 1947 a la fecha, las cosas han avanzado muy rápido, más que cualquier otro proceso en la historia de la ciencia y la tecnología, a tal grado que en la actualidad hay computadoras mucho más poderosas que la ENIAC y que utilizan un circuito de silicio tan pequeño que cabe en la palma de la mano.

	ENIAC	Intel 8080
Año	1947	1973
Componentes electrónicos	Casi 18,000 bulbos	Un circuito integrado con más de 100, 000 transistores
Tamaño	120 m2	Menos de un cm2
Requerimientos de potencia	140 kilowatts	Pocos miliwatts
Velocidad	5,000 sumas/segundo	150,000 sumas/segundo
Frecuencia de reloj	100 kHz	2 MHz
Costo	Varios millones de dólares	50 dólares en 1974

Tabla 2.1 Comparación entre la ENIAC y el Intel 8080.

[†] Tal vez el uso de la palabra "suave" pueda parecer extraño, pero deja de serlo cuando se contrasta con el hecho de que en las computadoras anteriores a von Neumann (y en las actuales calculadoras), las operaciones que se pueden hacer están "alambradas" y predefinidas, por lo que el usuario no puede cambiarlas. Esta flexibilidad es lo que define a la programación y al término *software* empleado tanto en inglés como en español.

Como contraste inicial, antes de proceder a describir lo que ha sucedido en el transcurso de tiempo desde la invención de la computadora, se hace aquí una comparación entre dos máquinas: la ENIAC y uno de los primeros microprocesadores (ya obsoleto).

Es prácticamente imposible encontrar otro ejemplo de un avance de esta naturaleza en la historia de la tecnología, y esto hace más interesante el estudio de lo que ha pasado desde los años iniciales hasta la fecha.

2.1.2 Evolución de las Computadoras

Las condiciones históricas que Babbage no encontró para la construcción de su máquina analítica habían madurado durante el primer tercio del siglo XX. La composición de las fuerzas sociales había sufrido un cambio notable y empezaban a surgir concentraciones financieras importantes. La nueva situación derivada de la necesidad de controlar la creciente complejidad de las actividades sociales, tanto a nivel de funcionamiento administrativo de las grandes empresas como con respecto a los programas de investigación de nuevas y sofisticadas tecnologías militares (que después revertirían en formas civiles), crearon las condiciones para la aparición del ordenador, instrumento que haría posible el tratamiento de grandes masas de información, hasta el momento objeto de las máquinas clásicas, y la realización de cálculos científicos, ya probablemente auxiliados por las máquinas de calcular de sobremesa.

La evolución de las computadoras solía dividirse en **generaciones**, aunque esto cada vez se emplea menos. El criterio para determinar cuando se dio el cambio de una generación a otra no está claramente definido, pero resulta aparente que deben cumplirse al menos dos requisitos estructurales:

a) Forma en que están construidas: que haya tenido cambios sustanciales.
b) Forma en que el ser humano se comunica con ellas: que haya experimentado progresos importantes.

En lo que respecta al primer requisito, los cambios han sido drásticos en el corto lapso que tienen de existencia las computadoras (desde los tubos de vacío hasta los circuitos microelectrónicos), mientras que el avance del segundo requisito ha sido más cauteloso. Debido a la falta de una definición formal de la frontera entre generaciones se crea confusión cuando se intenta determinar cuál es la generación actual.

Desde el punto de vista estricto, aún estamos en la tercera generación de computadoras – o en lo que podría llamarse la segunda parte de ésta, porque sólo ha habido adelantos significativos en el punto a), pues en lo relativo al punto b), entre las computadoras actuales y las de hace quince años no hay diferencia sustancial alguna; la comunicación entre el usuario y la máquina sólo se ha vuelto más cómoda y conveniente.

Así pues, no está claro si ya estamos en la cuarta generación o si aún no se cumplen los requisitos para el cambio. Sin embargo, la suposición general, avalada fuertemente por los fabricantes de equipo, es que nos encontramos ya en ésta desde el advenimiento de los microprocesadores, y cómo ésta resulta ser la opinión más popular, aunque no la más correcta desde nuestro punto de vista, es la que se supondrá en adelante.

- Primera Generación

Esta primera etapa abarcó la década de 1950 y se conoce como la primera generación de computadoras. Las máquinas de esta generación cumplen los requisitos antes mencionados de la siguiente manera:

a) Por medio de circuitos de tubos de vacío o bulbos.
b) Mediante la programación en lenguaje de máquina (lenguaje binario).

Estos aparatos eran grandes y costosos –del orden de decenas o cientos de miles de dólares.

En 1951 aparece la primera computadora comercial, es decir, fabricada con el objetivo de venderse en el mercado: la UNIVAC I (*UNIVAC: UNIVersAl Computer*, Computadora Universal). Esta máquina disponía de mil palabras de memoria central y podía leer cintas magnéticas, se utilizó para procesar los datos del censo de 1950 en Estados Unidos. Estos eran los años de la posguerra y la nueva invención no presagiaba su gigantesco potencial en la competencia económica internacional, que no llegaría sino hasta una década más tarde.

Durante la primera generación y hasta mediados de la tercera, las unidades de entrada estaban completamente dominadas por las tarjetas perforadas, retomadas a principios de siglo por Herman Hollerith (1860-1929), quien además fundó una compañía que con el paso de los años se conocería como IBM (*IBM: International Bussines Machines*, Máquinas de Negocios Internacionales).

A la UNIVAC I le siguió una máquina desarrollada por IBM, que apenas incursionaba en ese campo; es la IBM 701 (de la que se entregaron 18 unidades entre 1953 y 1956), que inaugura la larga serie por venir.

Posteriormente, la compañía Remington Rand produjo el modelo 1103, que competía con la 701 en el campo científico, y la IBM fabricó la 702, que no duró mucho en el mercado por problemas con la memoria.

La más exitosa de las computadoras de la primera generación fue el modelo 650 de IBM, de la que se produjeron varios cientos. Esta máquina usaba un esquema de memoria secundaria llamada tambor magnético, antecesor de los discos que actualmente se emplean.

La competencia contestó con los modelos UNIVAC 80 y 90 , que pueden situarse ya en los inicios de la segunda generación.

También de esa época son los modelos IBM 704 y 709, Burroughs 220 y UNIVAC 1105.

- Segunda Generación

Se acercaba la década de 1960 y las computadoras seguían en constante evolución: reducían su tamaño y aumentaban sus capacidades de procesamiento. Al mismo tiempo se definía con mayor claridad toda una nueva ciencia: la de comunicarse con las computadoras, que recibiría el nombre de programación de sistemas.

En esta etapa puede hablarse ya de la segunda generación de computadoras, que se caracteriza por los siguientes aspectos primordiales:

a) Están construidas con circuitos de transistores.
b) Se programan en nuevos lenguajes, llamados de "alto nivel".

En general, las computadoras de la segunda generación son de tamaño más reducido y de costo menor que las anteriores. En esta etapa hubo una gran competencia y muchas compañías nuevas, y se contaba con máquinas bastante avanzadas para su época, como la serie 5000 de Burroughs y la máquina ATLAS, de la Universidad de Manchester. Cabe decir que esta última incorporaba –con varios años de anticipación- técnicas de manejo de memoria virtual, que se perfeccionarían tiempo después.

Entre los primeros modelos se puede mencionar la Philco 212 (esta compañía se retiró del mercado de computadoras en 1964) y la UNIVAC M460. Una empresa recién formada, Control Data Corporation, produjo la CDC 1604, seguida por la serie 3000. Estas máquinas comenzaron a imponerse en el mercado de las grandes computadoras.

IBM mejoró la 709 y produjo la 7090 (luego ampliada a la 7094), que ganó el mercado durante la primera parte de la segunda generación. UNIVAC continuó con el modelo 1107, Mientras que NCR (*National Cash Register*) empezó a producir máquinas más pequeñas para proceso de datos de tipo comercial, como la NCR 315.

RCA (*RCA: Radio Corporation of America, Corporación de la Radio de América*) introdujo el modelo 501, que manejaba el lenguaje COBOL, para proceso administrativo y comercial. Más tarde introdujo el modelo RCA 601.

La segunda generación no duró mucho, sólo unos cinco años, y debe considerarse como una transición entre las recién inventadas máquinas electrónicas, que nadie sabía con precisión para que podían ser útiles, y el actual concepto de computadora, sin el cual el funcionamiento de las modernas sociedades industriales sería difícil de concebir.

- Tercera Generación

Con la aparición de nuevas y mejores formas de comunicarse con las computadoras, junto con los progresos en electrónica, surge la que se conoce como tercera generación de computadoras a mediados de la década de 1960.
Como parte de una enorme estrategia comercial y de mercadotecnia esta etapa se inaugura con la presentación, en abril de 1964, de la serie 360 de IBM.

Las características estructurales de la tercera generación consisten en:

a) Su fabricación electrónica está basada en circuitos integrados:[‡] agrupamiento de circuitos de transistores grabados en pequeñísimas placas de silicio.

[‡] Estos microcircuitos reciben el nombre de circuitos integrados, y son conocidos también por su nombre popular en inglés, *chip* (pedacito). Su origen se remonta a 1958-1959, cuando la idea de obtener e interconectar capacitores y transistores a partir de un pequeño bloque de silicio surge en las mentes, en forma independiente, del doctor Robert Noyce, de la recién creada compañía Fairchaild Semiconductors (quien luego fundaría la empresa Intel), y del ingeniero Jack Kilby, de la compañía Texas Instruments.

b) Su manejo es por medio de los lenguajes de control de los sistemas operativos (que se estudiarán más adelante).

Las computadoras de la serie IBM 360 (modelos 20, 22, 30, 40, 50, 65, 67, 75, 85, 90, 195) manejaban técnicas especiales de utilización del procesador, unidades de cinta magnética de nueve canales, paquetes de discos magnéticos y otras características ahora usuales. No todos los modelos empleaban esas técnicas, sino que estaban divididos por aplicaciones.

El sistema operativo de la serie 360, llamado simplemente OS (*Operating System*, sistema Operativo), en varias configuraciones incluía un conjunto de técnicas de manejo de memoria y del procesador que pronto se convirtieron en estándares.

Esta serie alcanzó un éxito enorme, a tal grado que la mayoría de la gente (los ciudadanos comunes y corrientes) pronto llegó a identificar el concepto de computadora con el nombre de IBM. Sin embargo, sus máquinas no fueron las únicas, ni necesariamente las mejores. También en 1964 CDC introdujo la serie 6000, con la máquina 6600, que durante varios años fue considerada como la más rápida.

Esta fue una época de pleno desarrollo acelerado y de competencia por los mercados internacionales, debido a que la industria de la computación había crecido hasta alcanzar proporciones insospechadas.

Al inicio de la década de 1970 IBM produce la serie 370 (modelo 115, 125,135, 145, 158, 168) como mejora, aunque no radical, de la serie 360. UNIVAC compite con los modelos 1108 y 1110, máquinas de gran escala; mientras que CDC inaugura su serie 7000 con el modelo 7600, reformado después para introducir la serie cyber. Estas computadoras son tan veloces y tan potentes que se convierten ya en un asunto de Estado y de seguridad nacional para el país que las produce: en los más altos niveles gubernamentales se cuida su exportación y comercialización internacional.

A finales de esa década IBM introduce las nuevas versiones de la serie 370 con los modelos 3031, 3033 y 4341, en tanto que Burroughs con las computadoras de la serie 6000 (modelos 6500, 6700) de avanzado diseño, luego reemplazadas por la serie 7000. La compañía Honeywell participa con las computadoras de la línea DPS, en varios modelos.

En Japón, la compañía Fujitsu produjo computadoras poderosas, desde máquinas relativamente pequeñas hasta verdaderos gigantes (de la serie FACOM), comparables sólo con los más grandes sistemas de CDC o IBM.

Las grandes computadoras reciben el nombre en inglés de *mainframes*, que significa precisamente gran sistema.

- Minicomputadoras

A mediados de la década de 1970, en plena tercera generación, surge un gran mercado para computadoras de tamaño mediano o **minicomputadoras**, que no son tan costosas como las grandes máquinas y disponen de una gran capacidad de proceso. En un principio, el mercado de estas máquinas estuvo dominado por la serie PDP-8 de DEC (*Digital Equipment Corporation*).

Otras minicomputadoras populares fueron la serie PDP-11 de DEC, reemplazada luego por las máquinas VAX (*Virtual Address eXtended*) de la misma compañía; los modelos Nova y Eclipse de Data General; las series 3000 y 9000 de Hewlett-Packard, en varias configuraciones y el modelo 34 de IBM, que luego fue reemplazado por los modelos 36 y 38.

Dentro de esta categoría estaban también las máquinas Wang y Honeywell-Bull en diversas configuraciones, así como computadoras Prime, ICL (*International Computers Limited*, inglesa), Siemens (alemana), etcétera.

En la ex Unión Soviética fueron de amplio uso las computadoras de la serie SU (Sistema Unificado, Ryad), que también pasaron por varias generaciones. La primera de estas máquinas era, en cuento a la arquitectura, una copia de la serie 360 de IBM, con los modelos ES 1020 a 1060. A fines de la década de 1970 surgió la serie Ryad-2, cuya arquitectura seguía a la serie 370.

Asimismo, los países socialistas desarrollaron una serie de computadoras dedicadas al control industrial además de las máquinas de la serie Minsk y BESM.

En la actualidad el mercado de las minicomputadoras es muy dinámico, sobre todo por su uso como servidores[5] de las cada vez más comunes redes metropolitanas y amplias. Algunas de las empresas mencionadas continúan con nuevos modelos: Hewlett-Packard, DEC (alpha), IBM (series AS/400 y RS/6000), y han surgido con gran éxito nuevas compañías especializadas, como Sun Microsystems o Silicon Graphics, pero otras más han desaparecido recientemente.

- Cuarta Generación

El adelanto de la microelectrónica avanza a una velocidad impresionante, y ya por el año de 1972 surge en el mercado una nueva familia de circuitos integrados de alta densidad que reciben el nombre de **microprocesadores**. Las **microcomputadoras** diseñadas con case en estos circuitos son en extremo pequeñas y baratas, por lo que su uso se extiende al mercado de consumo. Hoy en día hay microprocesadores en muchos aparatos de uso común, como relojes, televisores, hornos, juguetes, etcétera, y naturalmente en toda una nueva generación de máquinas, aunque sólo en lo que respecta al equipo físico (requisito *a* antes mencionado), puesto que en el otro aspecto, (requisito *b* para determinar el cambio de una generación a otra) no existen progresos de esta magnitud, aunque los cambios que han sucedido tampoco son despreciables.

El nacimiento de las microcomputadoras tuvo lugar en los Estados Unidos a partir de la comercialización de los primeros microprocesadores de 8 bits (Intel 8008, 8080) a comienzos de la década de 1970. Las primeras de éstas máquinas se conocían sencillamente como microcomputadoras, y comenzaron a tener aceptación primero sólo en el mercado de los técnicos e ingenieros que deseaban (o podían) experimentar con esta nueva tecnología, pues ponía al alcance de su mano un equipo antes completamente inaccesible. La empresa IMSAI vendía una microcomputadora en forma de partes que el entusiasta experimentador armaba y probaba. Pronto otras compañías entraron a la

[5] Se da ese nombre a la computadora que sirve como depositaria de los archivos manejados por los múltiples usuarios de una red. En el servidor –puede haber más de uno- también residen los programas de control de la red.

competencia y algunas empezaron a ofrecer equipos ya armados, más fáciles de utilizar, aunque seguía siendo absolutamente necesario un conocimiento especializado sobre la materia o al menos la disposición de dedicar muchas horas del día a obtenerlo.

Durante la década de 1970 aparecieron microcomputadoras de múltiples marcas y pronto se impusieron dos tendencias: la de los sistemas Apple, que empleaban su propia tecnología (basada en el microprocesador Mostek 6502) y un sistema operativo particular, y la de casi todas las demás microcomputadoras, que empleaban el sistema operativo CP/M *(Control Program/Monitor*, Programa de Control/Supervisión), basadas en los procesadores Intel 8080 y 8085 y luego en el Zilog Z80. Durante varios años la situación se mantuvo con un ritmo de crecimiento rápido, pero frenado aún por la dificultad de empleo de estos dispositivos.

Cuando el mercado adquirió proporciones importantes, y una vez que la nueva tecnología estaba ya aceptada, comenzó la verdadera explosión comercial masiva con la introducción, en 1981, de la *Personal Computer* (computadora personal) de IBM. Esta máquina (basada en el microprocesador Intel 8088) tenía características importantes que hacían mucho más amplio su campo de aplicaciones, sobre todo por un nuevo sistema operativo estandarizado (MS-DOS: *Microsoft Disk Operating System*, Sistema Operativo en Disco de Microsoft) y una capacidad mejorada de graficación, lo cual la hacía más atractiva y relativamente más fácil de usar.

En la actualidad existen centenares de fabricantes de computadoras personales (o de partes), mayoritariamente distribuidos en Estados Unidos, Japón Taiwan, Corea del Sur y Europa, y juntos configuran una enorme industria que ha colocado centenares de millones de microcomputadoras en el mercado durante la última década.

A la cabeza tecnológica de esta industria llegaron a estar prácticamente solo dos compañías: IBM y Apple; juntas llegaron a manejar el 50% del mercado mundial de las computadoras personales aunque sus modelos eran mutuamente incompatibles. Como ya mencionamos, en agosto de 1981 IBM lanzó la computadora personal PC y determinó la dirección a seguir durante media década, hasta que en 1987 descontinuó la línea (pero sólo durante pocos años porque no resultó un buen negocio) e inauguró una nueva, llamada PS/2 (*Personal Systems*, Sistemas Personales), con varios modelos basados en los procesadores Intel 8086, 80286 y 80386.

Apple por su parte mantuvo vigente su familia Macintosh (inicialmente basada en el procesador Motorola 68000), en varias versiones, y además introdujo nuevos modelos con microprocesadores especiales. Asimismo, muchas compañías aprovecharon que desde el inicio IBM hizo pública la arquitectura(o sea, el diseño global) de su computadora PC para producir una gran cantidad de computadoras parecidas, tanto en funcionamiento como en apariencia, por lo cual es muy común que existan modelos similares (a veces más poderosos y económicos), conocidos en inglés como *clones*: copias. En vista de la gran competencia que estas máquinas "apócrifas" hicieron a las de IBM, esta compañía decidió mantener privado el diseño y especificaciones de la serie PS/2, por lo que no surgieron competidores directos.

Sin embargo, a partir de que IBM abandonó la línea PC, este segmento del mercado a seguido en aumento, gracias a la introducción de modelos cada vez más poderosos basados en microprocesadores de 32 bits y debido también al desarrollo de los sistemas operativos y programas de *software* para múltiples aplicaciones.

2.1.3 Organización de la Computadora

Un sistema de cómputo es la configuración completa de una UCP (Unidad Central de Procesamiento) con unidades periféricas y con los programas de base (sistema operativo y programas de aplicación) que la hacen comportarse como un todo coherente.

Un microprocesador combinado con memoria y capacidades de entrada-salida se vuelve un microcomputador. La palabra *micro* se utiliza para indicar el tamaño físico de los componentes que están involucrados. La segunda parte de la palabra en microprocesador y microcomputador es la que realmente los coloca aparte. La palabra procesador es una abreviación para la unidad procesadora central. El término microcomputador es utilizado para indicar un sistema de computador completo, que consta de tres unidades básicas: CPU, memoria y dispositivos de entrada-salida. Para entender como funciona un microcomputador, es necesario familiarizarse con la interacción entre los componentes que lo conforman.

La Figura 2.2 muestra un diagrama de bloques de un sistema microcomputador. Típicamente, el microcomputador tiene un solo microprocesador. Si muchos procesadores se incluyen, entonces tenemos un sistema multiprocesador.

- El microprocesador

El conjunto formado por la unidad de control y la unidad de aritmética y lógica se llama procesador central o unidad central de procesamiento, UCP. Sus funciones consisten en decodificar y ejecutar las instrucciones de un programa, leer y escribir contenidos de las celdas de memoria y llevar y traer datos entre celdas de memoria y registros especiales.

El procesador es, pues, el "corazón" de la computadora. De éste dependen las demás funciones del sistema integrado, y allí se controlan todas las operaciones realizadas por la máquina.

Una UCP puede diseñarse y construirse combinando varias funciones digitales, tales como registros, una unidad aritmético-lógica, multiplexores y decodificadores. Un microprocesador es una UCP encerrada en un paquete de circuito integrado (CI), el cual normalmente puede llegar a consistir de varios millones de transistores y medir unos cuantos centímetros.

La unidad de aritmética y lógica de la UCP, como su nombre indica, se encarga de efectuar las operaciones relacionadas con los cálculos numéricos y simbólicos. Una unidad típica sólo es capaz de realizar un número reducido de operaciones muy elementales, pero a una gran velocidad. Las operaciones que esta subunidad puede efectuar son: suma y resta de dos números de punto fijo, multiplicación y división de punto fijo, manipulación de los bits de los registros y del acumulador (operaciones lógicas AND, OR, NOT), y comparación del contenido de dos registros (para averiguar si los números que contienen son iguales o uno es mayor).

Pocos procesadores tienen la capacidad de efectuar operaciones más complejas que éstas, lo cual significa que, por ejemplo, para elevar un número a una potencia hay que usar un programa especial escrito en lenguaje de máquina. Todos los sistemas de cómputo proporcionan a los usuarios bibliotecas de programas y funciones matemáticas

para efectuar estos cálculos, y lo hacen "armando" las funciones complejas con base en las operaciones elementales que la unidad aritmética y lógica si es capaz de efectuar.

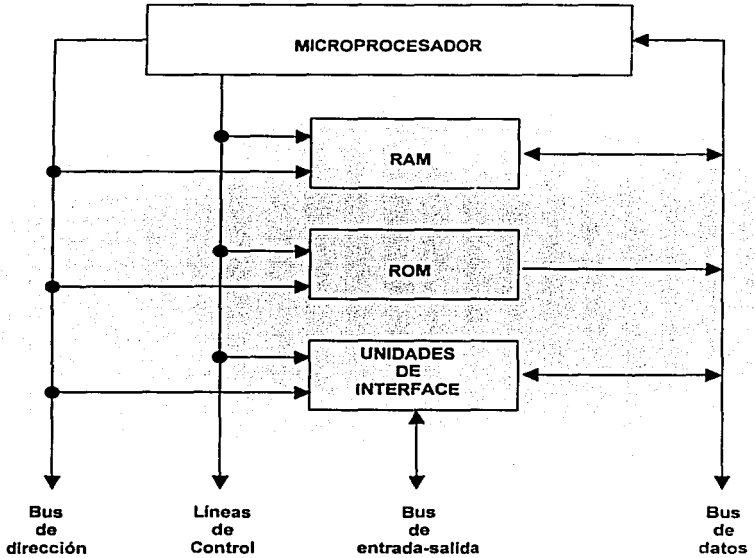


Fig. 2.2 Diagrama de bloques del sistema microcomputador.

La unidad de control es la encargada de marcar los cuatro pasos discretos en los cuales opera el microprocesador. Las micro-operaciones son realizadas durante cada uno de estos pasos. Cuando una instrucción se lee de la memoria, se dice que el microprocesador se encuentra en el ciclo de instrucción *fetch*. Cuando la palabra leída de la memoria se encuentra en la dirección de un operando el microprocesador se encuentra en ciclo *indirecto*. Cuando la palabra leída de la memoria está en un operando, el microprocesador se encuentra en un ciclo de ejecución de datos. El microprocesador tiene un cuarto ciclo denominado ciclo de *interrupción*, el cual se presenta eventualmente cuando el microprocesador deje pendiente la ejecución de una rutina para ejecutar otra o para atender la petición de algún otro dispositivo.

- La Memoria Central

En este conjunto, -generalmente grande- de celdas direccionables es donde la computadora almacena toda la información (datos y programas) que utilizará mientras esté encendida. Cualquier operación que el procesador efectúe debe necesariamente residir en la memoria central, ya que es allí donde la UCP buscará la siguiente instrucción.

La memoria central generalmente consiste de módulos de memoria RAM (*Random Acces Memory*, Memoria de Acceso Aleatorio) que constan a su vez de un número de CI conectados juntos. Esta memoria es necesaria para almacenar datos, parámetros y resultados intermedios que necesitan actualización y están sujetos a cambio.

Existe otro tipo de memoria denominada ROM (Read Only Memory, Memoria de Solo Lectura), la cual también reside en un paquete de CI y es utilizada para almacenar programas y tablas de constantes que no están sujetos a cambio una vez que la construcción de la computadora se ha completado.

- Unidades de Interface

Las unidades de interface proporcionan las rutas necesarias para transferir información entre el microprocesador y los dispositivos externos de entrada-salida conectados al bus de entrada-salida. El microprocesador recibe el estado e información de datos de los dispositivos externos a través de la interface y responde enviando información de control y de datos a través de la misma. Esta comunicación es especificada por instrucciones programadas que dirigen los datos a través de los diferentes tipos de bus de la microcomputadora.

- El bus

La comunicación entre los componentes del microprocesador a través del bus de dirección y de datos. El bus de datos es unidireccional desde el microprocesador hasta las otras unidades. La información binaria que el procesador coloca en el bus de dirección especifica una palabra de memoria en particular en RAM o ROM. El bus de dirección también se utiliza para seleccionar una de las muchas unidades de interface conectada al sistema o a un registro particular dentro de una unidad de interface. El número de líneas disponibles en el bus de dirección determina el máximo tamaño de memoria que se puede acomodar en el sistema. Para n líneas, el bus puede especificar hasta 2 n palabras de memoria. La longitud típica de un bus de dirección de un microprocesador es de 16, proporcionando una capacidad de memoria máxima de 2 16 = 65,536 palabras. La cantidad de memoria empleada en un sistema microcomputador depende de la aplicación particular y bastante a menudo es menor que el máximo disponible en el bus de dirección.

El bus de datos transfiere datos hacia y desde el microprocesador y la memoria o interface que es seleccionada por el bus de dirección. El bus de datos es bidireccional, lo que significa que la información binaria puede fluir en cada dirección. El Objeto de emplear un bus de datos bidireccional es la de optimizar el número de terminales dentro del paquete de CI. Si un microprocesador no utilizara un bus de datos bidireccional, sería necesario proporcionar terminales de entrada y salida por separado, lo cual implicaría un mayor número de patillas en el paquete de CI. El número de líneas para el bus de datos dentro del microprocesador comúnmente es de 8, 16 o 32, dependiendo de la arquitectura y aplicación especial del mismo.

- Unidades de Entrada y Salida

Las unidades de entrada más comunes fueron las lectoras de tarjetas perforadas (que ya no se emplean) y actualmente las terminales de video (pantallas o monitores). Las unidades de salida más usuales actualmente son las impresoras y las terminales de video. Existe gran diversidad de modelos de terminales de entrada/salida, pero la mayoría

utiliza los dos mismos elementos para lograr la comunicación entre el usuario y la máquina: un teclado para escribir a la máquina y una pantalla de video donde la computadora despliega los mensajes.

Desde la segunda parte de la tercera generación, las computadoras se comunican de manera interactiva con el usuario (es decir, mediante un diálogo), para lo cual se requiere una unidad que permita la comunicación bidireccional (entrada/salida), no una unidad de entrada exclusivamente. La opción más adecuada para este fin es la terminal de video ya mencionada, cuyo teclado muchas veces suele acompañarse de un dispositivo apuntador conocido en inglés como *mouse*[**], porque es una pequeña unidad móvil con dos grandes botones en la parte superior y una larga cola (como de ratón) que en realidad es el cable que se conecta a la computadora.

Cuando se oprime una tecla, los circuitos especiales del teclado envían a la computadora una serie de bits que identifican el carácter seleccionado, y es así como el procesador recibe los datos por parte del usuario.

El programa especial para manejo básico de entradas y salidas, llamado BIOS (*Basic Input-Output System*) es de uso general, y suele residir en forma permanente en una memoria PROM, a partir de donde se copia a RAM cuando se enciende inicialmente la computadora.

Existen otros dispositivos de entrada y salida de aplicación más especial como los graficadores, las impresoras láser, los scanner o digitalizadores e incluso e incluso equipo multimedia para la reproducción de archivos de audio.

Como la memoria central de una computadora es costosa y escasa, se vuelve necesario tener áreas adicionales de almacenamiento para guardar grandes cantidades de información de manera más económica. Además, la memoria central es volátil: pierde los datos almacenados al interrumpirse el suministro de corriente eléctrica, por lo que no resulta práctico utilizarla para almacenamiento permanente de datos.

Esta y otras razones dan lugar a la creación de unidades periféricas de memoria que en conjunto reciben el nombre **de memoria auxiliar (periférica o secundaria)**. Los tipos más comunes de dispositivos son:

- Unidades de Memoria Auxiliar

Como la memoria central de una computadora es costosa y escasa, se vuelve necesario tener áreas adicionales de almacenamiento para guardar grandes cantidades de información de manera más económica. Además, la memoria central es volátil: pierde los datos almacenados al interrumpirse el suministro de corriente eléctrica, por lo que no resulta práctico utilizarla para almacenamiento permanente de datos.

Las cintas magnéticas suelen manejarse en dos presentaciones: carrete y cartucho. La información se almacena grabando cada byte a lo ancho de ésta: los bits del 0 al 7 irán

[**] El culpable de la creación del "dispositivo apuntador" universalmente conocido como mouse es Doug Engelbart, quien lo pensó hacia 1965. Posteriormente fue rediseñado y adoptado por las empresas Xerox y Apple. Engelbart es también el diseñador original del concepto de "ventanas" y del correo electrónico.

ocupando posiciones sobre una vertical hasta llenar todo el ancho de la cinta. De esta manera, los bytes quedan acomodados uno por uno a lo largo de la cinta.

Una forma más conveniente de manejar información consiste en depositarla en (forma magnética) de manera directa sobre la superficie de un disco que gira en forma radial y en cuyo extremo está la cabeza lectora/grabadora. Los discos magnéticos existen en diversas presentaciones: discos rígidos fijos (es decir, que no se pueden sacar del dispositivo que los contiene) y discos flexibles removibles: los conocidos *diskettes.*

En los discos ópticos se aprovechan las características únicas de la luz láser para grabar micrométricas marcas en una superficie. Como el láser es luz monocromática de altísimo grado de coherencia, puede incidir sin dispersarse sobre áreas de millonésimas de metro de radio, con lo cual se obtienen densidades de grabación enormes. Estas microscópicas marcas representan los 0's o 1's sobre la superficie del disco.

2.1.4 Nacimiento de las redes de computadoras

A principios de los años sesenta, investigadores de instituciones de reconocido prestigio como el *MIT (MIT: Institute Technology of Massachusetts; Instituto Tecnológico de Massachusetts)* sentaron las bases tecnológicas que facilitaron en años posteriores la creación de las redes. Leonard Kleinrock fue el primero que habló sobre la teoría de conmutación por paquetes en su artículo "Flujo de Información en Redes Amplias de Comunicación". J.C.R. Licklider y W. Clark, también del MIT escribieron "Comunicación hombre - computadora en línea" y Paul Baran publicó "Redes de Comunicación Distribuida", en el que hablaba de redes conmutadas por paquetes, sin punto único de interrupción.

En 1965 la *DARPA (DARPA: Defense Advanced Research Projects Agency; Agencia de Proyectos de Investigación para la Defensa Avanzada)* promueve un estudio sobre "Redes cooperativas de computadoras de tiempo compartido", y al año siguiente, Larry Roberts, del MIT, publica "Hacia una red cooperativa de computadoras de tiempo compartido". En los años sucesivos se van presentando proyectos sobre redes conmutadas por paquetes, como en el simposio sobre principios operativos de 1967.

Con todo esto, a finales de los años sesenta, una de las preocupaciones de las Fuerzas Armadas de los Estados Unidos era conseguir una manera de que las comunicaciones estuvieran descentralizadas, es decir, evitar un centro neurálgico de comunicaciones que pudiera ser destruido en un eventual ataque militar con armas nucleares y que así, aún sufriendo el ataque, las comunicaciones no se bloquearan, sino que solamente se perdiera un nodo.

En 1969 la DARPA, junto con la compañía *Rand Corporation* desarrolló una red sin nodos centrales basada en conmutación de paquetes. La información se dividía en paquetes y cada paquete contenía la dirección de origen, la de destino, en número de secuencia y una cierta información. Los paquetes al llegar al destino se ordenaban según el número de secuencia y se juntaban para dar lugar a la información. Al viajar por la red paquetes, era más difícil perder datos ya que, si un paquete concreto no llegaba al destino o llegaba defectuoso, el ordenador que debía recibir la información sólo tenía que solicitar al ordenador emisor el paquete que le faltaba. El protocolo de comunicaciones se *llamó NCP (NCP: Network Control Protocol; Protocolo de Control de Redes).* Esta red en principio solo unía a un pequeño número de computadoras y se denominó DARPANET

(DARPANET: Defense Advanced Research Projects Agency Networking; Agencia de Redes de Proyectos de Investigación para la Defensa Avanzada), pero en 1972 se cambió el nombre por ARPANET *(ARPANET: Advanced Research Projects Agency Networking; Agencia de Redes de Proyectos de Investigación Avanzada)*, cuando ya conectaba a unos cuarenta nodos. En 1971 se creó el primer programa para enviar correo electrónico. Fue Ray Tomlinson y combinaba un programa interno de correo electrónico y un programa de transferencia de ficheros.

Los años setenta transcurren con instituciones conectándose directamente o conectando otras redes a ARPANET y con los responsables desarrollando estándares y protocolos, como Telnet, la especificación de transferencia de archivos o el protocolo NVP (NVP: Network Voice Protocol; Protocolo de voz en redes). Vinton Cerf y Bob Kahn publican "Protocolo para Intercomunicación de Redes por paquetes" que especifica en detalle el diseño del *TCP (TCP: Transmission Control Protocol; Protocolo de Control de Transmisión).*

2.1.5 Topologías de redes de computadoras

La topología es la forma física en que es posible conectar las estaciones de trabajo dentro de una red. El término topología es un concepto geométrico con el que se alude al aspecto de una cosa. En la actualidad, las topologías son consecuencia del tipo de tarjeta de red que se utilice, por ejemplo : Token-Ring (Anillo), Arcnet (Arbol), Ethernet (Bus Lineal), etc.

Las principales topologías de red son :

a) Topología en Estrella
b) Topología de Arbol (Jerárquica)
c) Topología de Bus (Horizontal)
d) Topología en Anillo
e) Topología en Malla

- Topología de Estrella

En la figura 2.3 se observa este tipo de conexión, en donde el elemento central es el servidor con sus periféricos. El servidor se mantiene preguntando constantemente a cada estación de trabajo, mediante comunicación exclusiva y por turno, si se desea transmitir información: de ser afirmativo atiende la petición y, al terminar, prosigue su interrogatorio permanente con otra estación de trabajo.

- Topología de Árbol

La topología de Árbol, también llamada de Estrella Distribuida (figura 2.4) ya que es una extensión de la arquitectura en estrella por interconexión de varias de estas. Permite establecer una jerarquía clasificando a las estaciones en grupos y niveles según el nodo a que están conectadas y su distancia jerárquica al nodo central. De características similares a la red en estrella, reduce la longitud de los medios de comunicación incrementando el número de nodos. Se adapta a redes con grandes distancias geográficas y predominio de tráfico local, características más propias de una red pública de datos que de una red privada local.

- Topología de Bus

Esta tipo de conexión se considera la más sencilla de todas (figura 2.5), donde las microcomputadoras incluyendo al servidor, están enlazadas por un sólo cable, y la información viaja en ambos sentidos, por lo que es necesario prevenir las colisiones entre ellas.

- Topología de Anillo

En la figura 2.6 se puede ver este tipo de conexión donde la información viaja ordenadamente en un sólo sentido, a través de un sólo cable, describiendo un ángulo de 360° en cuyo anillo imaginario están conectadas en serie las estaciones de trabajo y el servidor.

- Topología de Malla

Este tipo de topología (figura 2.7) se ha venido empleando en los últimos años. Lo que lo hace atractiva es una relativa inmunidad a los problemas de embotellamiento y averías. Gracias a la multiplicidad de caminos que ofrece a través de los distintos *DTE (DTE: Data Terminal Equipment, Equipo terminal de datos)* y *DCE (DCE: Data Circuit Equipment; Equipo de circuito de datos)*, es posible orientar el tráfico por trayectorias alternativas en caso de que algún nodo esté averiado u ocupado.

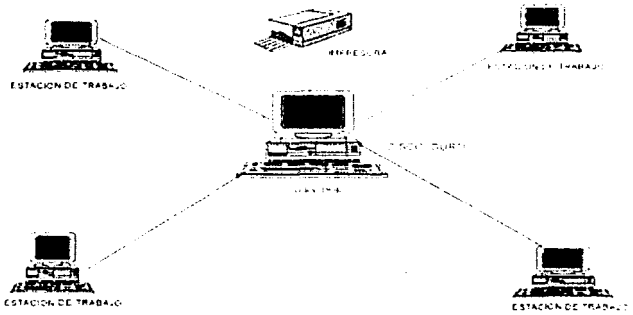


Fig. 2.3 Topología de Estrella.

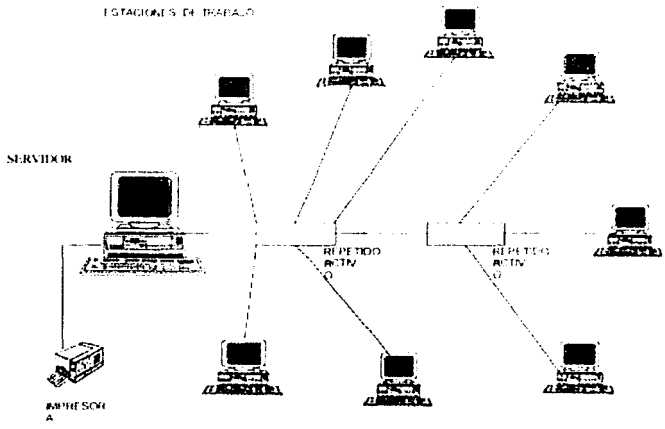


Fig. 2.4 Topología de Arbol.

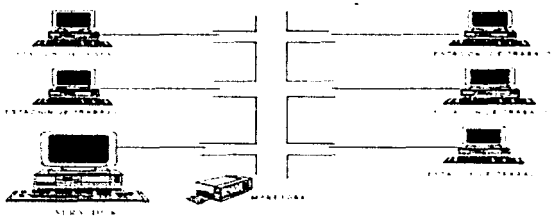


Fig. 2.5 Topología de Bus.

Fig. 2.6 Topologia tipo Anillo.

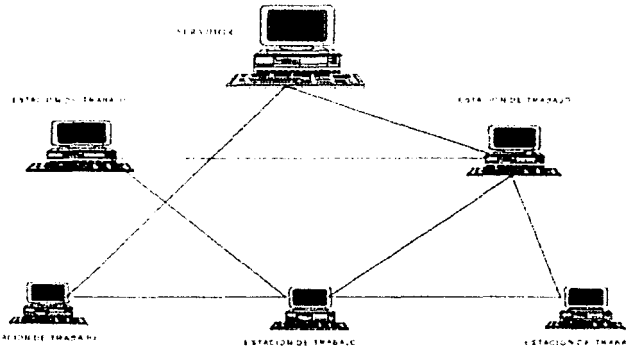


Fig. 2.7 Topologia tipo Malla.

2.1.6 Protocolos de comunicación

Cuando se requiere transferir información a lo largo de la red a través de un canal de comunicación se debe de contar con una serie de reglas que permitan esta comunicación; a estas reglas se les denominan protocolos de comunicación. Existen diversos tipos de protocolos pero para nuestro estudio mencionaremos algunos como referencia, enfocándonos principalmente en el protocolo TCP/IP, protocolo que consta de cuatro niveles que pueden rotularse como físico, de envío, de servicio y de aplicación. La base de muchos protocolos es el modelo OSI, para lo cual abordaremos brevemente una explicación del mismo.

- Modelo OSI

La organización ISO y el Comité Consultivo Internacional Telegráfico y Telefónico (CCITT) han desarrollado el modelo de referencia OSI para definir redes estratificadas y protocolos con varios niveles, como se muestra en la figura siguiente (figura 2.8):

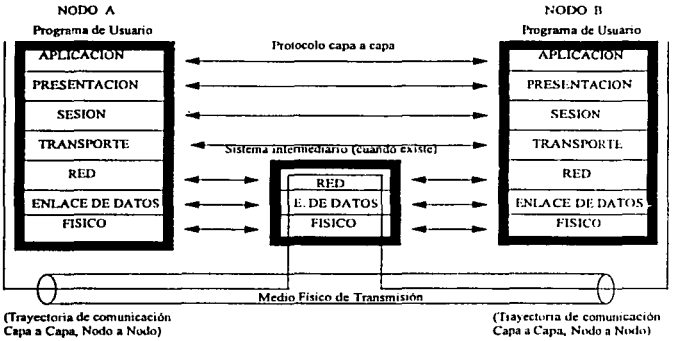


Fig. 2.8 Modelo O.S.I.

Este modelo ha recibido una gran aceptación en todo el mundo y está siendo instalado ya por muchos fabricantes. Los principales objetivos que persigue el modelo OSI son:

a) Proporcionar una serie de normas para la comunicación entre sistemas.
b) Eliminar todos los elementos técnicos que pudieran existir para la comunicación entre sistemas.
c) Abstraer el funcionamiento interno de los sistemas individuales.
d) Definir los puntos de interconexión para el intercambio de información entre los sistemas.
e) Limitar el número de opciones para incrementar las posibilidades de comunicación sin necesidad de onerosas conversiones y traducciones entre diferentes productos.
f) Ofrecer un punto de partida válido desde el comienzo en caso de que las normas del estándar no satisfagan todas las necesidades.

Explicando de manera muy sencilla el funcionamiento de cada una de las capas del modelo OSI tenemos:

La primer capa llamada nivel físico, que es la responsable de la transmisión de la información entre dos puntos en un circuito, tales como un servidor (despachador de archivos) y una terminal (estación de Trabajo).

Después sigue el nivel de enlace que es el responsable de la transferencia de datos por el canal, proporciona los datos de sincronización necesaria para delimitar el flujo de bits del nivel físico, así mismo garantiza la identidad de los bits, encargándose de que los datos lleguen sin errores al DTE.

Enseguida aparece el nivel de red que define la interfaz entre el DTE de usuario y la red de conmutación de paquetes, además de la interfaz de un DTE con otro a través de esta red.

El nivel de transporte proporciona la interfaz entre la red de comunicación de datos y los tres niveles superiores, es el nivel que permite al usuario elegir entre diversas opciones de calidad (y de precio) dentro de una misma red (es decir dentro del nivel de red).

Después de esta capa aparece el nivel de sesión, que funciona como interfaz del usuario con el nivel de transporte, ofrece un mecanismo organizado de intercambio de datos entre usuarios, cada usuario puede seleccionar el tipo de control y de sincronización que desea de la red.

El nivel de presentación asigna una sintaxis a los datos, es decir, determina la forma de presentación de los datos sin preocuparse de su significado o semántica, su principal misión es, aceptar tipos de datos (caracteres enteros, etc.) procedentes del nivel de aplicación y negociar con el nivel homólogo del otro extremo, la sintaxis escogida (por ejemplo ASCII).

La capa de nivel de aplicación se encarga de atender al proceso de aplicación del usuario final. A diferencia del nivel de presentación, este nivel toma en cuenta la semántica de los datos.

- Protocolo X.25

Este protocolo fue la primera norma mundialmente aceptada entre una terminal de datos (DTE en modo de paquetes) y una red de paquetes (DCE) para el acceso a una red de paquetes pública o privadas. Este protocolo admite corrección y detección de errores, lo cual, lo torna ideal para entornos de baja calidad con líneas ruidosas cuando las aplicaciones en cuestión exigen una transmisión sumamente confiable. La recomendación X.25 maneja generalmente velocidades desde 1200 bps hasta 64 kbps, por lo que es muy utilizada en ampliaciones transaccionales y solicitudes de información de volumen moderado. Los protocolos definidos en X.25 corresponden a los tres niveles más bajos del modelo *OSI (OSI: Open System Interconection; Sistema de Interconexión abierta)*: físico, de enlace de datos o tramas y de red o paquetes; donde cada capa es funcionalmente independiente de las otras, con la excepción de que una falta de las capas puede afectar la operación de las capas más altas.

- Protocolo Frame Relay

Las técnicas actuales de conmutación de paquetes llamadas Conmutación rápida de paquetes toman ventaja de los medios de transmisión confiables como la fibra óptica y de alta velocidad de procesamiento de los conmutadores para eliminar algunos procesos de control de flujo y corrección de errores de X.25, incrementando de esta forma la velocidad de transmisión de datos de las redes. Un técnica de conmutación rápida es la Conmutación de tramas *(Frame Relay)*, diseñado para maximizar la capacidad caudal y minimizar los costos simplificando el procesamiento en la red. El sistema *Frame Relay* se presta especialmente a las aplicaciones en las cuales los puntos extremos son dispositivos inteligentes (p. ej., workstations) y las líneas de transmisión son de alta calidad, éste concepto permite reducir el procesamiento de protocolos en cada nodo de la red disminuyendo el retardo global de extremo a extremo. El *Frame Relay* utiliza solo las

dos primeras capas del modelo OSI, proporcionando múltiples conexiones lógicas sobre un solo circuito físico y permite a la red enviar datos sobre esas conexiones hacia sus destinos finales.

- Protocolos HDLC Y LAPB

IBM después de desarrollar este protocolo, lo envío al ANSI para que fuera aceptado como una norma en los Estados Unidos de Norteamérica y a la *ISO (ISO: International Standaritation Organization, Organización Internacional de Estándares)* para que lo fuera internacionalmente. La OSI lo modificó para llegar a tener el protocolo *HDLC (HDLC: High Level Data Link Control, Control de Alto Nivel de Enlace de Datos)*. Posteriormente, la *UIT (UIT: Union International Telecomunications, Unión Internacional de Telecomunicaciones)* adoptó y modificó el HDLC para dar lugar a su protocolo *LAP (LAP: Link Access Protocol, Procedimiento de Acceso al Enlace)*, como parte de la norma de interfase de la red X.25, pero más tarde lo modificó nuevamente para crear el protocolo *LAPB (LAPB: Link Access Protocol on channel B, Procedimiento de Acceso al Enlace en el canal B)*, con el objeto de hacerlo más compatible con la última versión del HDLC. Estos protocolos están orientados a bit donde una de las ventajas es la reducción del número de caracteres necesarios para control, ya que cada bit dentro de un Byte puede tener un significado diferente.

- Protocolo PLP

La capa de red o de paquetes es la más compleja de la recomendación X.25, gran parte de esta complejidad se debe a la flexibilidad y confiabilidad que por naturaleza debe tener un protocolo de este nivel. Al protocolo de esta capa se le conoce normalmente como protocolo PLP (*PLP: Package Level Protocol, Protocolo de la Capa de Paquetes*). Una de las funciones principales de la capa de paquetes es el establecimiento de conexiones por medio de circuitos virtuales.

- Protocolo UDP

Este protocolo es implementado en hosts finales, y sólo se limita a intercambiar información que confirme que los datos que se enviaron llegaron de una manera segura. Una aplicación que se desee enviar vía UDP *(UDP: User Dataram Protocol: Protocolo de Datagramas de Usuarios*), tiene que pasar un bloqueo de datos al UDP. Posteriormente el datagrama del usuario pasa al IP y se compacta en un datagrama IP. El UDP tiene las mismas funciones que TCP, la diferencia entre ellos es que TCP es bidireccional y UDP no lo es, por lo tanto no tiene un control sobre los errores que pudieran existir para una mayor confiabilidad de datos.

2.1.7 Protocolo TCP/IP

El IP (*IP: Internet Protocol; Protocolo de Internet*) y el TCP (*TCP: Transmission Control Protocol; Protocolo de Control de Transmisión*), fueron desarrollados inicialmente en 1973 por el informático estadounidense Vinton Cerf como parte de un proyecto dirigido por el ingeniero norteamericano Robert Kahn y patrocinado por la ARPA *(ARPA: Agency Research of Programs Advanced*) del Departamento Estadounidense de Defensa. Internet comenzó siendo una red informática de ARPA (llamada ARPAnet) que conectaba redes de ordenadores de varias universidades y laboratorios en investigación en Estados Unidos.

- Arquitectura de TCP/IP

TCP/IP es el protocolo común utilizado por todos los ordenadores conectados a Internet, de manera que éstos puedan comunicarse entre sí. Hay que tener en cuenta que en Internet se encuentran conectados ordenadores de clases muy diferentes y con *hardware* y *software* incompatibles en muchos casos, además de todos los medios y formas posibles de conexión. Aquí se encuentra una de las grandes ventajas del TCP/IP, pues este protocolo se encargará de que la comunicación entre todos sea posible. TCP/IP es compatible con cualquier sistema operativo y con cualquier tipo de *hardware*.

TCP/IP no es un único protocolo, sino que es un conjunto de protocolos que cubren los distintos niveles del modelo, siendo los dos protocolos más importantes el TCP y el IP. La arquitectura del TCP/IP consta de cinco niveles o capas en las que se agrupan los protocolos, y que se relacionan con los niveles OSI de la siguiente manera:

- Aplicación: Corresponde a los niveles OSI de aplicación, presentación y sesión. Aquí se incluyen protocolos destinados a proporcionar servicios, tales como correo electrónico (SMTP), transferencia de ficheros (FTP), conexión remota (TELNET) y otros más recientes como el protocolo (HTTP).
- Transporte: Coincide con el nivel de transporte del modelo OSI. Los protocolos de este nivel, tales como TCP y UDP, se encargan de manejar los datos y proporcionar la fiabilidad necesaria en el transporte de los mismos.
- Internet: Es el nivel de red del modelo OSI. Incluye al protocolo IP, que se encarga de enviar los paquetes de información a sus destinos correspondientes. Es utilizado con esta finalidad por los protocolos del nivel de transporte.
- Red: Es la interfaz de la red real de TCP/IP sin especificar ningún protocolo concreto, así es que corre por las interfaces conocidas, como por ejemplo: X.25.

El TCP/IP necesita funcionar sobre algún tipo de red o de medio físico que proporcione sus propios protocolos para el nivel de enlace de Internet. Por este motivo hay que tener en cuenta que los protocolos utilizados en este nivel pueden ser muy diversos y no forman parte del conjunto TCP/IP. Sin embargo, esto no debe ser problemático puesto que una de las funciones y ventajas principales del TCP/IP es proporcionar una abstracción del medio de forma que sea posible el intercambio de información entre medios diferentes y tecnologías que inicialmente son incompatibles.

Para transmitir información a través de TCP/IP, ésta debe ser dividida en unidades de menor tamaño. Esto proporciona grandes ventajas en el manejo de los datos que se transfieren y, por otro lado, esto es algo común en cualquier protocolo de comunicaciones. En TCP/IP cada una de estas unidades de información recibe el nombre de datagrama, y son conjuntos de datos que se envían como mensajes independientes.

- Características de TCP/IP

Ya que dentro de un sistema TCP/IP los datos transmitidos se dividen en pequeños paquetes, éstos resaltan una serie de características, que se mencionan a continuación:

La tarea de IP es llevar los datos a granel (los paquetes) de un sitio a otro. Las computadoras que encuentran las vías para llevar los datos de una red a otra (denominadas enrutadores) utilizan IP para trasladar los datos. En resumen, IP mueve los paquetes de datos a granel, mientras TCP se encarga del flujo y asegura que los datos estén correctos.

Las líneas de comunicación se pueden compartir entre varios usuarios. Cualquier tipo de paquete puede transmitirse al mismo tiempo, y se ordenará y combinará cuando llegue a su destino.

Los datos no tienen que enviarse directamente entre dos computadoras. Cada paquete pasa de computadora en computadora hasta llegar a su destino. Éste, claro está, es el secreto de cómo se pueden enviar datos y mensajes entre dos computadoras aunque no estén conectadas directamente entre sí. Lo que realmente sorprende es que sólo se necesitan algunos segundos para enviar un archivo de buen tamaño de una máquina a otra, aunque estén separadas por miles de kilómetros y pese a que los datos tienen que pasar por múltiples computadoras. Una de las razones de la rapidez es que, cuando algo anda mal, sólo es necesario volver a transmitir un paquete y no todo el mensaje.

Los paquetes no necesitan seguir la misma trayectoria. La red puede llevar cada paquete de un lugar a otro y usar la conexión más idónea que esté disponible en ese instante. No todos los paquetes de los mensajes tienen que viajar, necesariamente, por la misma ruta, ni necesariamente tienen que llegar todos al mismo tiempo.

La flexibilidad del sistema lo hace muy confiable. Si un enlace se pierde, el sistema usa otro. Cuando se envía un mensaje, el TCP divide los datos en paquetes, ordena éstos en secuencia, agrega cierta información para control de errores y después los lanza hacia fuera, y los distribuye.

En el otro extremo, el TCP recibe los paquetes, verifica si hay errores y los vuelve a combinar para convertirlos en los datos originales. De haber error en algún punto, el programa TCP destino envía un mensaje solicitando que se vuelvan a enviar determinados paquetes.

2.1.8 Medios de transmisión

Lo descrito anteriormente forma parte del equipo de la red, pero estos necesitan medios de comunicación, interna y externa, los cuales presentamos a continuación:

- Líneas de Transmisión

El cable seleccionado dependerá en parte de sus prestaciones, costo y facilidad de instalación, facilidad de ampliación y algunos otros factores. El punto número 1 (prestaciones) constituye el más importante en la mayoría de las instalaciones, ya que éstas determinan la velocidad y distancia a las que puede transmitir las señales sin que éstas se distorsionen. Otros factores son la resistencia física y la duración. A continuación daremos una descripción de los tipos de cable y sus características.

a) Líneas Abiertas: estos son cables cubiertos por cristales aislantes o cerámicos. En la actualidad no se utilizan mucho, ya que presentan la desventaja de ser lentos en la transmisión ante una gran cantidad de información.

b) Cable Par Telefónico: también conocido como par de hilos trenzados, y como su nombre lo indica consta de un par de alambres de cobre enrollados y aislados por plásticos o cualquier otro aislante .Es uno de los tipos de cable más utilizados en los edificios y en sistemas telefónicos, ya que elimina una gran cantidad de ruido e interferencias. Gran parte de cableado instalado para sistemas telefónicos consta de 25 pares de los cuales utilizamos generalmente unos pocos y en la mayoría de los casos los pares no utilizados los podemos emplear para establecer un sistema de cableado de red. También es importante verificar las especificaciones de la placa de conexión de red, para asegurarse que el cable ofrece la calidad necesaria para la transmisión. Las desventajas del par de hilos reside en que el máximo ancho de banda es muy estrecho, estos no permiten rangos de transmisión de datos altos ni corridas de cables a grandes distancias. En situaciones donde los rangos de transmisión de datos no excedan 2 Mbps y donde los cableados no son mayores de 500 metros, el par de hilos trenzados es un excelente medio de comunicación.

c) Cable Telefónico sin Apantallar: este tipo de cable puede representar una alternativa económica al problema de cableado de una red, aunque la velocidad de transmisión y la distancia cubierta por el cable son menores. Además de resultar de fácil instalación, el cable puede transmitir datos a una velocidad máxima de 1 Mbps.

d) Cable Telefónico Apantallado: este cable ofrece varias ventajas sobre el anterior a nivel velocidad y distancia, pero éste es más costoso debido a que el cable está fabricado siguiendo un control más estricto. Puede transmitir a mayores distancias y velocidades.

e) Cable Coaxial: este tipo de cable consta de un conductor central rodeado por una capa aislante y por encima de ésta una malla de hilos trenzados y finalmente una cobertura aislante que protege todo el conjunto. Estos cables coaxiales son capaces de transmitir mayores frecuencias, en comparación con un cable par telefónico; y es un medio de comunicación utilizado para la implementación de redes locales de trabajo. Existe cable coaxial para transmisión de banda base o banda ancha.

Transmisión en Banda Base: la transmisión en banda base es utilizada para señales digitales de una estación de trabajo a otra a través de cable coaxial a velocidades de 10 Mbps y de distancias de hasta 4300 mts. Se utiliza generalmente en la Red Ethernet.

Transmisión en Banda Ancha: este tipo de cable nos permite transmitir señales de voz, vídeo y datos y se utiliza generalmente para transmisión de T.V. El cable soporta varios canales de señales de radio frecuencia de naturaleza analógica no digital. Debido a esto, las señales de la red tienen que ser convertidas de digitales a analógicas para ser transmitidas por el cable, las líneas de banda ancha pueden transmitir a una velocidad cercana a 5 Mbps.

f) Fibra Optica: este tipo de cable es excelente para tendidos a largas distancias que generalmente serían afectados por grandes interferencias eléctricas causadas por máquinas eléctricas, equipos de iluminación y otros dispositivos. Como el cable no emite señal también es excelente para usarlo en sistemas de alta seguridad. La capacidad de fiabilidad y de transmisión de la fibra óptica permite incrementar las distancias cubiertas. El vidrio de un cable de fibra óptica es tan puro que si fuera posible mirar en un extremo a otro en un cable de 3 a 4 kilómetros veríamos que es completamente transparente. Un tramo de cable de fibra óptica puede cubrir hasta 3 kilómetros sin repetidoras. Los estándares para la transmisión por fibra óptica especifican una velocidad de transmisión

de 100 Mbps pero es posible transmitir a velocidades de 1 Gbps, aunque actualmente no se dispone de los dispositivos eléctricos adecuados. Existen tres tipos de cable de fibra óptica:

- Cable de una sola fibra unimodal: este cable tiene un gran ancho de banda pero es difícil realizar conexiones en este tipo de cable.
- Cable de varias fibras multimodal: este cable es más fácil de conectar y se suministra de 2 a 24 fibras.
- Cable por índice de refracción multimodal: este cable es el que ofrece la mayor velocidad de transmisión a larga distancia.

g) Microondas: se puede mencionar que la mayoría de los circuitos utilizados en los sistemas telefónicos dependen de esta tecnología, la cual usa ondas de radio de alta frecuencia, desde un rango de 4.6 hasta 12 GHz (1 Ghz equivale a mil millones de ciclos por segundo). Esta facilidad de ancho de banda, proporciona la capacidad de transmitir a grandes distancias, pero es necesario tener instaladas unidades repetidoras con intervalos aproximados a 30 millas. Las unidades repetidoras transmiten la señal hacia la siguiente repetidora de microondas.

h) Comunicación Vía Satélite: para la utilización de rangos de transmisión más extensos que los manejados por las fibras ópticas, se pueden utilizar los satélites actuando como estaciones de transmisión, siendo una nueva era de las comunicaciones para el mundo entero. Los satélites reciben las señales en una frecuencia y las transmiten en otra, con el propósito de no interferir con las señales que están recibiendo. En la comunicación vía satélite el tiempo de propagación (250 a 300 milisegundos), implica retraso en la transmisión de datos mucho mayores que en los enlaces terrestres (6 microsegundos).

Como podemos apreciar, existen diferentes tipos de medios de comunicación para la transmisión de datos, pero es conveniente analizar cada una de las características de ellos, para tener la capacidad de elegir el medio óptimo. Entre las características más importantes a considerar pueden citarse: el costo, rango de error, seguridad que permiten y la rapidez de transmisión.

2.1.9 Sistemas operativos

El sistema operativo de una red, es el conjunto de programas que regulan y distribuyen el funcionamiento de la misma, proporciona elementos para establecer la interfaz con el usuario, controla y define los grupos y niveles de seguridad, es el encargado de la integridad y seguridad de la información contenida en ella; además controla la compartición de los recursos. En general, optimiza los recursos del sistema para un mejor rendimiento y aprovechamiento de los mismos.

Novell ha sido el principal protagonista en el campo de las redes desde la década de los 80's con sus diferentes versiones de NetWare, y aún domina más de la mitad del mercado, donde mantiene gran presencia. Microsoft tuvo un inicio lento con su sistema Windows NT, pero recientemente ha tenido un gran despunte que lo ha llevado a estar en segundo sitio detrás del gigante Novell. Con menos fuerza se reparten lo que resta del mercado el sistema OS/2 de IBM, la solución para redes cliente/servidor AppleShare de Apple y las diferentes variantes de UNIX como Solaris, UnixWare y el clon Linux. La supervivencia de estos sistemas operativos de red, dependerá de la renovación de estrategias tanto productivas como en materia de mercadotecnia en los años por venir, ya

que la batalla entre Microsoft y Novell por la supremacía del mercado, parece no querer dejar una tercera opción.

2.1.10 Tipos de redes de computadoras

Las redes de computadoras por su cobertura geográfica se pueden clasificar en:

Redes de Area Local (LAN).
Redes de Area Metropolitana (MAN).
Redes de Area Amplia (WAN).

- Redes de Area Local (Local Area Network)

Son redes diseñadas para operar sistemas de datos a bajo costo y velocidades de 116 millones de bits por segundo (Mbps), en distancias de hasta 2.5 Km., usualmente enlazando terminales, computadoras personales, etc., dentro de un edificio.

- Redes de Area Metropolitana (Metropolitan Area Network)

Son redes estandarizadas de alta velocidad (hasta 100 Mbps), que proporcionan conexiones LAN-LAN y LAN-WAN para sistemas privados y públicos de comunicación de datos que cubren una determinada área metropolitana (hasta 100 kms). Las redes de área metropolitana pueden proporcionar también servicios de voz y vídeo.

- Redes de Area Amplia (Wide Area Network)

Son redes de comunicación de datos que abarcan varios cientos o miles de kilómetros y pueden utilizar enlaces de alta velocidad. Actualmente una de las redes mas utilizadas a nivel mundial es la red Internet también conocida como Red de Redes.

2.1.11 La red de redes: Internet

En 1979 ARPA crea la primera comisión de control de la configuración de Internet y tras varios años de trabajo, por fin en 1981 se termina de definir el protocolo TCP/IP y ARPANET lo adopta como estándar en 1982, sustituyendo a NCP. Son las primeras referencias a Internet, como "una serie de redes conectadas entre sí, específicamente aquellas que utilizan el protocolo TCP/IP". Internet es la abreviatura de *Interconnected Networks*, es decir, Redes interconectadas, o red de redes. Además en estos años se fundan Microsoft (1975) y Apple (1976).

En 1985, quince años después de la primera propuesta, se termina el desarrollo del aún vigente protocolo para la transmisión de ficheros en Internet (FTP), basado en la filosofía de cliente-servidor.

Un punto fundamental en el éxito fue el hecho de que ARPA distribuyera a bajo coste los protocolos, que fueron adoptados por el UNIX de BSD (*Berkeley Software Distribution*), muy difundido entre las universidades. De esta forma se crearon una gran cantidad de servicios y se provocó un importante avance en el desarrollo de la red. Por esta época se crea el sistema de denominación de dominios *DNS (DNS: Domain Name System, Nombre del Sistema Dominio)*

1995 es el año del gran *"boom"* de Internet. Puede ser considerado como el nacimiento de la Internet comercial. Desde ese momento el crecimiento de la red ha superado todas las expectativas. Este hecho se produce porque es en este año cuando la WWW supera a ftp-data transformándose en el servicio más popular de la red, después de que el año anterior superase a telnet. A partir de aquí empiezan a incrementarse de una manera casi exponencial el número de servicios que operan en la red, ya que para esta época ya operan bancos en la red (First Virtual), una radio comercial de difusión exclusiva en Internet (Radio HK). Gobiernos de todo el mundo se conectan a la red, y el registro de los dominios deja de ser gratuito para pagarse una cuota anual de $50.

Internet es un conglomerado de ordenadores de diferente tipo, marca y sistema operativo, distribuidos por todo el mundo y unidos a través de enlaces de comunicaciones muy diversos. La gran variedad de ordenadores y sistemas de comunicaciones plantea numerosos problemas de entendimiento, que se resuelven con el empleo de sofisticados protocolos de comunicaciones.

2.2 Conceptos de *Software*

Para poder realizar cualquier sistema a la medida, primero es necesario analizar al *software* que se deberá utilizar para su programación. Esta parte del capítulo pretende analizar y recordar algunos conceptos de *software* de los cuales se han hecho uso para la elaboración del SAST.

2.2.1 Conceptos de Programación Distribuida

- Definición de Programación Distribuida

Se puede entender como programación distribuida a la distribución de una tarea computacional por varios programas, procesos o procesadores. Todo esto puede hacerse simultáneamente sin que se presente inconsistencia en la información que se está manejando. Esta definición puede ser muy amplia, por lo que se tendrán que definir algunos de sus beneficios, y de esta manera se podrá adquirir una idea más clara del significado de la programación distribuida.

- Beneficios de la Programación Distribuida

Una manera de describir la programación distribuida y clasificarla en varios tipos es definir sus beneficios. A continuación se mencionan cuatro de los principales:

1. La solución a un problema específico en varios pasos, se pueden usar los programas de uso general para manejar algunos de estos pasos, y así que reduce la cantidad de nuevo código que nosotros tenemos que escribir. A menudo, nosotros podemos poder evitar la escritura total de cualquier nuevo código. Nosotros nos referiremos a este beneficio como el *tool building.*
2. Usando varios procesadores concurrentes, se pueden resolver problemas rápidamente, a diferencia que si se usara un solo procesador. A este beneficio se le conoce como el *concurrency.*
3. Un problema que a menudo se presenta es de la forma " Máquina A , B y C en paralelo." En este caso la solución puede ser usar los procesos paralelos por separado. Forzando la solución en una forma estrictamente secuencial para la

ejecución por un solo proceso. Nosotros nos referiremos a este beneficio como el de *paralelismo.*

4. En ocasiones, los recursos necesarios para resolver algún problema que se presenta termina siendo la comunicación que se pueda presentar entra varias computadoras con la red. Los recursos, en este contexto, pueden ser: archivos, modems, una CPU específica, etc. A este beneficio lo conocemos como *resource sharing.*

A veces los recursos que se necesitan para resolver un problema de distribución contemplan la necesidad de utilizar varias computadoras conectadas en una red.

- Métodos de Descomposición

A continuación se tratarán algunos métodos de descomposición de un problema para que la solución pueda distribuirse.

- Distribución de los datos

Una manera muy común de distribuir es dividir el conjunto de datos de entrada en partes, y de cada parte realizar un proceso por separado. Éste es llamado como distribución de los datos o descomposición del dominio.

- Distribución Algorítmica

Una segunda técnica importante es la distribución algorítmica. La característica importante de distribución algorítmica es que cada conjunto de datos ve los mismos comandos, pero realizan un funcionamiento diferente en ellos. En el caso de una aplicación del multiprocesador, cada procesador está ejecutando el código diferente. La tubería o filtro donde pasa el conjunto de datos establece alguna clase de sincronización contenida en sus componentes individuales.

- Granular

Otra manera de caracterizar las aplicaciones distribuidas es la granular. Éste no es un término preciso, pero tiene que ver principalmente con el tamaño del conjunto de datos operando cada proceso.

La información se va procesando paso a paso para evitar la saturación al momento de su compilación.

- Equilibrio de carga

Cuando una tarea de la informática será extendida por los procesadores múltiples para la ejecución actual, es generalmente deseable equilibrar la carga, es decir, se asegura que cada procesador tenga una cantidad igual de trabajo por hacer.

2.2.2 El sistema operativo UNIX

Al igual que otros sistemas operativos, el sistema operativo UNIX es un conjunto de programas de utilidad y un conjunto de instrumentos que permiten al usuario conectar y utilizar esas utilidades para construir sistemas y aplicaciones.

- Orígenes del Sistema Operativo UNIX

UNIX nació en 1969 en una *main*frame 635 de General Electric. A la vez, los Laboratorios Bell de AT&T (*AT&T: American Telephon and Telegraph, Teléfonos y Telégrafos Americanos*) había completado el desarrollo de Multics, un sistema multiusuario que falló por su gran demanda de disco y memoria. En respuesta a Multics, los ingenieros de sistemas Kenneth Thompson y Dennis Ritchie inventaron el UNIX. Inicialmente, Thompson y Ritchie diseñaron un sistema de archivos para su uso exclusivo, pero pronto lo cargaron en una Digital Equipment Corp. (DEC) PDP-7, una computadora con solo 18 kilobytes de memoria. En 1971, UNIX recibió reconocimiento oficial de AT&T cuando la firma lo usó para escribir manuales, en el mismo año la segunda edición de UNIX fue realizada y dio forma al UNIX moderno con la introducción del lenguaje de programación C y el concepto de los *pipes*. Los *pipes* fueron debido a que son un mecanismo orientado a objetos, porque entregan datos desde un objeto, o programa, a otro objeto.
El lenguaje C es otro producto de los Laboratorios Bell y a finales de 1973, después de que Ritchie añadió soporte para variables globales y estructuras, C se convirtió en el lenguaje preferente de programación de UNIX.

El lenguaje C fue responsable del concepto de portabilidad. Escrito en C, el entorno UNIX pudo ser relativamente fácil de trasladar a diferentes plataformas de *hardware*. Las aplicaciones escritas en C pudieron ser fáciles de transportar entre diferentes variantes de UNIX. En esta situación nació el primer criterio de sistema abierto: portabilidad OS *(OS: Open System, Sistemas Abiertos)*, la posibilidad de mover *software* desde una plataforma *hardware* a otra de una forma estándar. En 1974, la quinta edición de UNIX fue realizada para que estuviera disponible para las universidades. En 1975, la sexta edición de UNIX fue desarrollada e iniciada para ser ampliamente usada. Durante este tiempo, los usuarios se hicieron activos, los grupos de usuarios fueron formados, y en 1976, se estableció el grupo de usuarios USENIX. En 1977, Interactive System Corp. inició la venta de UNIX en el mercado comercial. Durante este tiempo, UNIX también adquirió más poder, incluyendo soporte para procesadores punto flotante, microcódigo y administración de memoria.

- Características del Sistema Operativo UNIX

A comienzos de 1984, había sobre 100,000 instalaciones del sistema UNIX en el mundo, funcionando en máquinas con un amplio rango de computadoras, desde microprocesadores hasta *mainframes*. Ningún otro sistema operativo puede hacer esta declaración. Muchas han sido las razones que han hecho posible la popularidad y el éxito del sistema UNIX:

1. El sistema está escrito en un lenguaje de alto nivel, haciéndolo fácil de leer, comprender, cambiar, y mover a otras máquinas.
2. Posee una simple interface de usuario con el poder de dar los servicios que los usuarios quieren.
3. Provee de primitivas que permiten construir programas complejos a través de programas simples.
4. Usa un sistema de archivos jerárquico que permite un mantenimiento fácil y una implementación eficiente.
5. Usa un formato consistente para los archivos, el flujo de bytes, haciendo a los programas de aplicación más fáciles de escribir.
6. Provee una simple y consistente interface a los dispositivos periféricos.

7. Es un sistema multiusuario y multitarea, cada usuario puede ejecutar varios procesos simultáneamente.
8. Oculta la arquitectura de la máquina al usuario, haciendo fácil de escribir programas que se ejecutan en diferentes implementaciones *hardware.*

Sin embargo tiene algunos inconvenientes:

1. Comandos poco claros y con demasiadas opciones.
2. Escasa protección entre usuarios.
3. Sistema de archivo lento.

A pesar de que el sistema operativo y muchos de los comandos están escritos en C, UNIX soporta otros lenguajes, incluyendo Fortran, Basic, Pascal, Ada, Cobol, Lisp y Prolog.
El sistema UNIX puede soportar cualquier lenguaje que tenga un compilador o intérprete y una interface de sistema que defina las peticiones del usuario de los servicios del sistema operativo de la forma estándar de las peticiones usadas en los sistemas UNIX.

El sistema operativo UNIX ha evolucionado durante desde su invención como experimento informático hasta llegar a convertirse en uno de los sistemas operativos más populares e influyentes del mundo. UNIX es el sistema más usado en investigación científica, pero su aplicación en otros entornos es bastante considerable. UNIX tiene una larga historia y muchas de sus ideas y metodología se encuentran en sistemas como DOS *(DOS: Disk Operative System; Sistema Operativo de Disco)* y *Windows.*

Las características fundamentales del UNIX moderno son: memoria virtual, multitarea y multiusuario. La filosofía original de diseño de UNIX fue la de distribuir la funcionalidad en pequeñas partes: los programas. De esta forma, el usuario puede obtener nueva funcionalidad y nuevas características de una manera relativamente sencilla, mediante las diferentes combinaciones de pequeñas partes (programas). Además, en el caso de que aparezcan nuevas utilidades (y de hecho aparecen), pueden ser integradas al espacio de trabajo. Las versiones modernas del sistema UNIX están organizadas para un uso de red fácil y funcional, por lo que es muy frecuente encontrar versiones del sistema UNIX sobre grandes unidades centrales sosteniendo varios cientos de usuarios al mismo tiempo.

Las herramientas de comunicación internas del sistema, la fácil aceptación de rutinas de dispositivo adicionales de bajo nivel y la organización flexible del sistema de archivos son naturales para el entorno de red de hoy en día. El sistema UNIX, con su capacidad de multitarea y su enorme base de *software* de comunicaciones, hace que la computación por red sea simple, permitiendo también compartir eficientemente dispositivos como impresoras y disco duro.

La versión SVR4 (Sistema V versión 4), es la versión más actualizada del sistema UNIX de AT&T. Ha sido portada a la mayoría de las máquinas computadoras centrales y es el estándar actual para la línea AT&T. SVR4 ha sido significativamente mejorado con respecto a versiones anteriores. Una de estas mejoras es la interfaz gráfica de usuario (GUI), que permite la utilización de X Windows. Los sistemas comerciales UnixWare de SCO y Solaris de Sun Microsystems están basados en el SVR4. La mejora más importante de SVR4 es la adición de soporte completo para redes de área local. La administración de máquinas conectadas en red se ha mejorado en gran medida y la administración remota es ahora posible a través de la red.

El soporte para redes de área local está muy mejorado en SVR4, en comparación con versiones más antiguas del sistema operativo UNIX. Además del soporte de rutinas de bajo nivel en el núcleo, se dispone de un *software* simple y amistoso para conectar la dos LAN's principales disponibles en el mundo UNIX, Ethernet y Starlan.

UNIX es un sistema operativo multiusuario; no sólo puede utilizarlo más de una persona a la vez, sino que los diferentes usuarios recibirán distinto trato. Para poder identificar a las personas, UNIX realiza un proceso denominado ingreso (*login*). Cada archivo en UNIX tiene asociados un grupo de permisos. Estos permisos le indican al sistema operativo quien puede leer, escribir o ejecutar como programa determinado archivo.

UNIX reconoce tres tipos diferentes de individuos: primero, el propietario del archivo; segundo, el "grupo"; por último, está el "resto" que no es ni propietarios ni pertenecen al grupo, denominados "otros". En general, las máquinas UNIX están conectadas en red, es decir, que los comandos no se ejecutarán físicamente en la computadora en la cual se está tecleando, sino en la computadora a la que uno se ha conectado. A veces hay que conectarse explícitamente, dando un nombre de máquina desde un programa emulador de terminal, es decir, un programa que permite que una computadora actúe como teclado y pantalla de otra computadora remota. Hay muchos modos de comunicarse con otros usuarios que estén conectados al mismo sistema, o incluso que sean usuarios de él. Para hacer lo primero se usa la orden **talk**, que conecta con un usuario siempre que esté conectado al sistema. No sólo puede conectarse uno con un usuario del mismo sistema, sino de cualquiera conectado al mismo, por ejemplo, en Internet.

La forma más habitual de enviar mensajes es el correo electrónico. Este método permite enviar mensajes de texto ASCII *(ASCII: American Standart Code for Information Interchange, Código Estándar Americano para el Intercambio de Información)*, a veces con archivos pegados (attachments); estos archivos tienen que ser previamente convertidos a ASCII, para poder ser enviados por este medio. UNIX tiene una orden, mail, para mandar correo electrónico, pero no es demasiado amistosa para el usuario y por ello se usan otros programas, como el Pine, para enviar o recibir correo.

Dado que UNIX es un sistema operativo de red, muchas de las computadoras con UNIX están conectadas unas a otras y a Internet. Una computadora UNIX ofrece generalmente una serie de servicios a la red, mediante programas que se ejecutan continuamente llamados *daemon*. Estos *daemon* escuchan un puerto, o dirección numérica que identifica un servicio y actúan como servidores. Para usar tales servicios se usan programas clientes, que ya saben de qué puerto se trata y cual es el protocolo adecuado para hablar con ese daemon. Por supuesto, para usar estos programas hay que tener primero permiso para usar tal puerto o protocolo, y luego acceso a la máquina remota, es decir, hay que "autentificarse", o identificarse como un usuario autorizado de la máquina. Algunos de estos programas son **telnet**, **rlogin**, **rsh**, **ftp**, etc.

La figura 2.9 describe la arquitectura de alto nivel de UNIX. El sistema operativo interactúa directamente con el *hardware*, suministrando servicios comunes a los programas y aislándolos de la particularización del *hardware*. Viendo el sistema como un conjunto de capas, el sistema operativo es comúnmente llamado como núcleo del sistema o kernel. Como los programas son independientes del *hardware* que hay por debajo, es fácil moverlos desde sistemas UNIX que corren en diferentes máquinas si los programas no hacen referencia al *hardware* subyacente. Por ejemplo, programas que asumen el tamaño

de una palabra de memoria será más difícil de mover a otras máquinas que los programas que no lo asumen.

Los programas como el *shell* y los editores (**ed** y **vi**) mostrados en la capa siguiente interactúa con el kernel invocando un conjunto bien definido de llamadas al sistema. Las llamadas al sistema ordenan al kernel realizar varias operaciones para el programa que llama e intercambiar datos entre el kernel y el programa. Varios programas mostrados en la figura 2.9 están en configuraciones del sistema estándares y son conocidos como comandos, pero los programas de usuario deben estar también en esta capa, indicándose con el nombre a.out, el nombre estándar para los archivos ejecutables producidos por el compilador de C.

Otros programas de aplicaciones pueden construirse por encima del nivel bajo de programas, por eso la existencia de la capa más exterior en la figura 2.9. Por ejemplo, el compilador de C estándar, está en el nivel más exterior de la figura: invoca al preprocesador de C, compilador, ensamblador y cargador, siendo todos ellos programas del nivel inferior. Aunque la figura muestra una jerarquía a dos niveles de programas de aplicación, los usuarios pueden extender la jerarquía a tantos niveles como sea apropiado. En realidad, el estilo de programación favorecida por UNIX estimula la combinación de programas existentes para realizar una tarea.

Muchos programas y subsistemas de aplicación que proporcionan una visión de alto nivel del sistema tales como el shell, editores, SCCS (*SCCS: Source Code Control System, Sistema de Control del Código Fuente)* y los paquetes de documentación, están convirtiéndose gradualmente en sinónimos con el nombre de "Sistema UNIX". Sin embargo, todos ellos usan servicios de menor nivel suministrados finalmente por el kernel, y se aprovechan de estos servicios a través del conjunto de llamadas al sistema. Hay alrededor de 64 llamadas, de las cuales unas 32 son usadas frecuentemente. Tienen opciones simples que las hacen fáciles de usar pero proveen al usuario de gran poder. El conjunto de llamadas al sistema y los algoritmos internos en los que se implementan forman el cuerpo del kernel.

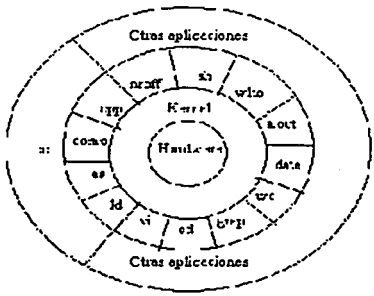


Fig. 2.9 Arquitectura del sistema operativo UNIX.

- Componentes del Sistema Operativo UNIX

Existen tres componentes importantes dentro del Sistema Operativo, los cuales se explicarán a continuación:

Kernel

Al conjunto de programas que componen UNIX y que se encargan de proporcionar los recursos del sistema, asignar tareas, manejar al almacenamiento de datos y de coordinar todos los detalles internos de la computadora se les llama *KERNEL*. Este es el componente principal del sistema operativo. El usuario rara vez opera directamente con el *kernel*, que es la parte residente en memoria del sistema operativo.

UNIX se caracteriza por ser un sistema "MULTIUSUARIO" porque permite que dos o más personas utilicen la computadora al mismo tiempo.

Shell

Los usuarios se comunican con el *kernel* a través de otro programa conocido como el **shell**. El *shell* es un "Intérprete de Línea de Comandos" que traduce los comandos tecleados por el usuario y los convierte en instrucciones que puede entender el *kernel*. Cuando alguien teclea un comando en la terminal, el *shell* interpreta el comando y llama al programa deseado. También es un lenguaje de programación de alto nivel que puede utilizarse en la combinación de programas de utilidad para crear aplicaciones completas.

El *shell* puede soportar múltiples usuarios, múltiples tareas, y múltiples interfaces para sí mismo. Los dos *shells* más populares son el **BourneShell** que es el estándar aceptado para el sistema UNIX y el **Cshell** (BSD Unix) nombrado por "C" el cual es un lenguaje de alto nivel, debido a que usuarios diferentes pueden usar diferentes *shells* al mismo tiempo, entonces el sistema puede aparecer diferente para usuarios diferentes. Existe otro *shell* conocido como **Kornshell** (así llamado en honor a la persona que lo desarrolló, David Korn), que es muy popular entre los programadores.

Sistema de Archivos (File System)

El sistema operativo UNIX está diseñado para manejar información contenida normalmente en discos. Para que esta manipulación sea realmente efectiva, es necesario que la información esté organizada de alguna forma. La manera estándar de organizar la información es en archivos. Los archivos son localizados dentro del disco porque son apuntados desde un lugar determinado, a este lugar se le denomina directorio.

Sin embargo en UNIX no se utiliza un único directorio para apuntar a todos los archivos del sistema, sino que se crea una estructura jerárquica de directorios conocida como estructura en árbol.

2.2.3. El lenguaje de programación C

C fue desarrollado originalmente como un lenguaje de programación de sistemas, los atributos que lo hacen deseable para dichos tipos de programas están originando que muchos programadores se den cuenta de su potencial en muchas aplicaciones de propósito general. Estas características incluyen transportabilidad, modificabilidad y

acceso a operaciones que normalmente están definidas a la programación en lenguaje ensamblador.

- Origen del Lenguaje C

C fue desarrollado originalmente en los años setenta por Dennis Ritchie en los Laboratorios Bell, (ahora laboratorios Bell AT&T), utilizando un DEC PDP-11. C es el resultado de un proceso de desarrollo comenzado con un lenguaje anterior denominado BCPL (BCPL: *Binary Code Program Language*, Lenguaje de Programación en Código Binario), que todavía se sigue usando, principalmente en Europa. BCPL fue desarrollado por Martin Richards e influenció otro lenguaje denominado B, que fue inventado por Ken Thompson. En los años 70 el lenguaje B llevó al desarrollo del C.

Durante muchos años el estándar de C fue de hecho la versión proporcionada con el sistema operativo UNIX versión 5. Con la popularidad de las microcomputadoras se crearon muchas implementaciones de C. En lo que se podía decir que era un milagro, los códigos fuente aceptados por la mayoría de esas implementaciones eran altamente compatibles. Sin embargo, como no existía ningún estándar, aparecieron discrepancias.

Para cambiar la situación, el ANSI (ANSI: *American National Standard Institute*, Instituto Nacional Americano de Estándares) estableció un comité(comité ANSI X3J11), a principios del verano de 1983, para crear un estándar que definiera de una vez por todas el lenguaje C. Esta definición normalizada fue terminada y adoptada por el ANSI en el año de 1987.

- Características y Ventajas del Lenguaje C

En el pasado, cuando necesitábamos tomar un programa que estaba escrito y ejecutándose en una clase de computadora y llevarlo a otra de tipo diferente, una gran parte del mismo tenía que volverse a rescribir. Un gran objetivo de diseño en la realización del C fue abrirse camino en está dependencia de la máquina. A causa de esto, C se convirtió en uno de los lenguajes más transportables que existen hoy día, permitiendo dividir programas que al principio parecen ser tareas grandes e irresolubles en algunas subtareas más pequeñas que son más sencillas de diseñar y codificar o la utilización de estas subtareas en una aplicación no relacionada en principio con aquélla con muy poca o ninguna modificación. Si realizamos la mayoría de nuestros programas con funciones aumentamos la legibilidad del código y disminuimos el desarrollo y tiempo de verificación del programa. Además creamos una serie de rutinas que pueden colocarse en una biblioteca y compartirse por otros programas.

C es un lenguaje de programación estructurado de propósito general. Sus instrucciones constan de términos que se parecen a expresiones algebraicas, además de ciertas palabras clave inglesas como **if**, **else**, **for**, **do** y ***while***. En este sentido, C recuerda a otros lenguajes de programación estructurado de alto nivel como Pascal y FORTRAN-77. C tiene también algunas características adicionales que permiten su uso a nivel más bajo, cubriendo así el vacío entre el lenguaje de máquina y los lenguajes de alto nivel más convencionales.

C se caracteriza por hacer la redacción de programas fuente muy concisos, debido en gran parte al gran número de operadores que incluye el lenguaje. Tiene un con junto de instrucciones relativamente pequeño, aunque las implementaciones actuales incluyen un

numeroso conjunto de funciones de biblioteca que mejoran las instrucciones básicas. Es más, el lenguaje permite a los usuarios escribir funciones de biblioteca adicionales para su propio uso. De esta forma, las características y capacidades se pueden ampliar fácilmente por el usuario.

Hay compiladores de C disponibles para las computadoras de todos los tamaños y los intérpretes de C se están haciendo cada vez más comunes. Los compiladores son frecuentemente compactos y generan programas objeto que son pequeños y muy eficientes en comparación con los programas generados a partir de otros lenguajes de alto nivel. Los intérpretes son menos eficientes, aunque son de uso más cómodo en el desarrollo de nuevos programas. Muchos programadores comienzan utilizando un intérprete, y una vez que han depurado el programa (eliminado los errores del programa) utilizan un compilador.

Otra característica de C es que los programas son muy portables, más que los escritos en otros lenguajes de alto nivel. La razón de esto es que C deja en manos de las funciones de biblioteca la mayoría de las características dependientes de la computadora. Toda versión de C se acompaña de su conjunto de funciones de biblioteca, que están escritas para las características particulares de la computadora en la que se instale. Estas funciones de biblioteca están relativamente normalizadas y se accede a cada función de biblioteca de igual forma en todas las versiones de C. De esta manera, la mayoría de los programas en C, que se diseñaron en una computadora, se pueden llevar a otra diferente y compilarlos y ejecutarlos con muy pocas o ninguna modificación.

- Compiladores e Intérpretes

Los términos compilador e intérprete se refieren a la forma en que se ejecuta un programa. En teoría, cualquier lenguaje de programación puede ser compilado o interpretado, pero algunos pueden ejecutarse en una forma u otra. Por ejemplo; BASIC normalmente es interpretado y C normalmente es compilado. La forma en que se ejecuta un programa no viene definida por el lenguaje en que haya sido escrito. Los intérpretes y los compiladores son simplemente programas sofisticados que trabajan sobre el código fuente del programa.

Un intérprete lee el código fuente de un programa línea a línea, realizando las instrucciones específicas contenidas en esa línea. Un compilador lee el programa entero y lo convierte a código objeto, que es una traducción del código fuente del programa a una forma que puede ser ejecutada directamente por la computadora. El código objeto también se suele denominar código binario o código máquina. Una vez que el programa está compilado las líneas de código fuente dejan de tener sentido en la ejecución del programa.

Cuando se usa un intérprete, el código fuente debe estar presente cada vez que se quiere ejecutar el programa. Este lento proceso se da cada vez que se ejecuta el programa. Por contra, un compilador convierte el programa en código objeto que puede ser directamente ejecutado por la computadora. Como el compilador traduce el programa de una vez, todo lo que hay que hacer es ejecutarlo directamente, normalmente mediante el simple proceso de escribir su nombre. Por tanto, la compilación sólo cuesta una vez. El código interpretado lleva esa carga cada vez que se ejecuta un programa.

El proceso de compilación en sí mismo lleva tiempo, pero queda fácilmente desplazado por el tiempo que se ahorra al usar el programa. Los programas compilados ejecutan mucho más rápido que los programas interpretados. La única situación en la que esto no es cierto es cuando el programa es muy corto. -- menos de 50 líneas- y no usan ningún bucle o ciclo iterativo.

Dos términos comúnmente usados en C es tiempo de compilación y tiempo de ejecución. El tiempo de compilación se refiere a todo lo que sucede durante el proceso de compilación. El tiempo de ejecución se refiere a todo lo que sucede cuando el programa se ejecuta realmente.

- Esquema de un programa

Todo programa en C consta de una o más funciones, una de las cuales se llama ***main***. El programa siempre comenzará por la ejecución de la función ***main***. Las definiciones de las funciones adicionales pueden preceder o seguir a ***main***.

Cada función debe contener:

1. Una cabecera de la función, que consta del nombre de la función, seguido de una lista opcional de argumentos encerrados con paréntesis.

2. Una lista de declaración de argumentos, si se incluyen éstos en la cabecera.

3. Una sentencia compuesta, que contiene el resto de la función.

Los argumentos son símbolos que representan información que se le pasa a la función desde otra parte del programa. (También se llaman parámetros o argumentos).

Cada sentencia compuesta se encierra con un par de llaves, {y}. Las llaves pueden contener combinaciones de sentencias elementales (denominadas sentencias de expresión) y otras sentencias compuestas. Así las sentencias compuestas pueden estar anidadas, una dentro de otra. Cada sentencia de expresión debe acabar en punto y coma(;)

Los comentarios pueden aparecer en cualquier parte del programa, mientras estén situados entre los delimitadores /* y */. Los comentarios son útiles para identificar los elementos principales de un programa o para la explicación de la lógica subyacente de éstos.

La forma general de un programa en C se ilustra en la Figura 2.10, donde f1() hasta fN() representan funciones definidas por el usuario.

- Creación de un programa

La compilación de un programa en C consiste en los siguientes 4 pasos:

1. Redacción del programa
2. Compilación del programa
3. Enlazado del programa con todas aquellas funciones de la biblioteca que se necesiten.

4. Ejecución del Programa.

La mayoría de los compiladores de C deben disponer de un editor aparte para crear los programas. Si se usa CP/M, se puede usar ED, el editor estándar del CP/M. Si se usa MS-DOS (MS-DOS: *Microsoft Disk Operating System*, Sistema Operativo en Disco de Microsoft) o PC-DOS (PC-DOS, *Personal Computer Disk Operating System*, Sistema operativo para Computadora Personal) se puede usar EDLIN, y por supuesto, en las implementaciones de UNIX se utiliza el editor de texto vi. Los compiladores sólo aceptan archivos de texto estándar como entrada. Por ejemplo, su compilador no aceptará archivos creados con ciertos procesadores de texto debido a que incluyen códigos de control caracteres no imprimibles.

```
archivos de cabecera
declaraciones globales
main()
{
        variables locales
        flujo de sentencias
}
f1()
{
        variables locales
        flujo de sentencias
}
f2()
{
        variables locales
        flujo de sentencias
}
.
.
.
fN()
{
        variables locales
        flujo de sentencias

}
```

Fig. 2.10 Esquema de un programa en C.

- Mapa de memoria

Un programa compilado en C crea y usa cuatro regiones de memoria lógicas diferentes que sirven para funciones específicas. La primera región es la memoria que contiene realmente el código del programa. La siguiente región es la memoria donde se guardan las variables globales. Las dos regiones restantes son la pila y un espacio libre. La pila se usa para muchas cosas mientras se ejecuta el programa. Mantiene las direcciones de retorno para las llamadas a funciones, así como las variables locales. También sirve para salvar el estado de la CPU. El espacio libre es la región de memoria libre, que puede usar

el programa mediante las funciones de asignación dinámica para cosas como el uso de listas enlazadas y árboles.

Aunque la disposición física exacta para cada una de las cuatro regiones de memoria difiere entre los tipos de CPU y las implementaciones de C, el diagrama de la figura 2.11 muestra de forma conceptual cómo aparece el programa de C en memoria.

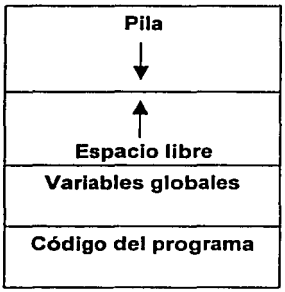


Fig. 2.11 Un mapa de memoria de un programa en C.

- Variables, constantes, operadores y expresiones

Las variables y las constantes son manipuladas por los operadores para formar expresiones. Se trata de los elementos más básicos del lenguaje C.

- Tipos de datos

Existen cinco tipos de datos básicos en C: carácter, entero, coma flotante, coma flotante de doble precisión y sin valor. Aunque el tamaño y rango de estos tipos de datos varía con cada tipo de procesador y con la implementación del compilador de C, se asume que un entero tiene la misma longitud en bits que la palabra de la CPU. Normalmente un carácter es un *byte*. El formato exacto de los valores en coma flotante depende de cómo se implementen. Para la mayoría de las microcomputadoras, incluyendo las basadas en el microprocesador 8088 y el 8086, la información sobre tamaños y rangos se muestra en la tabla de la tabla 2.2.

El rango de los tipos ***float*** y ***double*** normalmente se da en dígitos de precisión. Las magnitudes de los tipos *float* y *double* dependen del método utilizado para representar los números en coma flotante. Cualquiera que sea el método, los números son bastante grandes.

El tipo *void* (añadido por el estándar ANSI propuesto) tiene dos usos. El primero es para declarar explícitamente una función como que no devuelve valor alguno; el segundo es para crear punteros genéricos.

C soporta varios tipos agregados, incluyendo estructuras, uniones campos de bits, enumeraciones y tipos definidos por el usuario.

- Declaración de variables

Todas las variables en C han de ser declaradas antes de poder ser usadas. Existen tres lugares básicos donde se pueden declara variables, dentro de funciones, en la definición de parámetros de funciones y fuera de las funciones. Estas variables son, respectivamente, variables locales, parámetros formales y variables globales.

Tipo	Longitud en bits (las longitudes y rangos asumen palabras de 16 bits)	Rango(Este tipo aparece
Char	8	0 a 255
Int	16	-32768 a 32767
Float	32	Aproximadamente 6 dígitos de precisión.
Double	64	Aproximadamente 12 dígitos de precisión
Void	0	Sin valor

Tabla 2.2 Tamaños y rangos de los tipos básicos.

- Sentencias de control

Muchos programas requieren que un grupo de instrucciones se ejecute repetidamente, hasta que alguna condición lógica se satisface, se denomina a estas estructuras bucles (la sentencia ***while***). A veces no se conoce por adelantado el número de repeticiones; se repite la ejecución hasta que alguna condición lógica se hace cierta (por la sentencia ***do-while***). Otras veces, un grupo de instrucciones consecutivas se repite un cierto número especificado de veces (la sentencia ***for***). Esto es otra forma de bucle.

Muchos programas requieren que se realice una prueba o comprobación lógica (la sentencia ***if-else***) en algún punto concreto dentro de ellos. Para ello se realizará a continuación alguna acción que dependerá del resultado de la prueba lógica. Esto se conoce como ejecución condicional. Y finalmente existe una clase especial de ejecución condicional en la que un grupo de sentencias es seleccionado entre varios grupos disponibles. Esto se denomina a veces selección (la sentencia ***switch***). La sentencia ***break*** se utilizará para terminar la ejecución de bucles o salir de una sentencia *switch*. Se puede utilizar dentro de una sentencia ***while***, ***do-while***, ***for*** o ***switch***. Para saltarse el resto de bucle sin terminarlo se utiliza la sentencia continúe, la cual, sino, simplemente no ejecuta las sentencias que se encuentran a continuación y se salta directamente a las siguientes pasada a través del bucle.

La sentencia ***continue*** se puede incluir dentro de una sentencia ***while***, ***do-while*** o ***for***. Para alterar la secuencia de ejecución normal del programa se utiliza la sentencia ***goto***, la cual transfiere el control a otra parte de él.

Todas estas operaciones se pueden realizar utilizando diversas sentencias de control incluidas en C.

- Funciones

El uso de funciones definidas por el programador permite dividir un programa grande en un cierto número de componentes más pequeños, cada una de las cuales con un propósito único e identificable. Por tanto, un programa en C se puede modularizar mediante el uso inteligentes de las funciones, las cuales permiten acceder repetidamente a un grupo determinado de instrucciones de varias partes distintas de programa. Las instrucciones repetidas se pueden incluir dentro de una sola función, a la que se puede acceder cuando sea necesario. Además, se puede transferir un diferente conjunto de datos a la función cada vez que se accede a ella. Por tanto el uso de funciones evita la necesidad de repetir las mismas instrucciones de forma redundante.

Igualmente, es importante la claridad lógica resultante de la descomposición de un programa en varias funciones concisas, representando cada función alguna parte bien definida del problema global. Estos programas son más fáciles de escribir y de depurar, y su estructura lógica es más fácil de entender que la de aquellos que adolecen de este tipo de estructuras. Esto es especialmente cierto en programas largos y complicados. La mayoría de los programas en C se modularizan de esta manera, aun cuando no impliquen la ejecución repetida de las mismas tareas. De hecho, la descomposición de un programa en módulos individuales se considera generalmente la parte importante de la buena práctica de la programación.

La utilización de funciones permite también al programador construir una biblioteca a medida de rutinas de uso frecuente o de rutinas que se ocupen del manejo de elementos dependientes del sistema. Cada rutina se puede programar como una función por separado y almacenar un determinado archivo de biblioteca. Si un programa requiere una determinada rutina, se puede añadir la correspondiente función de biblioteca al programa durante el proceso de compilación. Por tanto muchos programas distintos pueden utilizar una misma función. Esto evita la reescritura del código de las funciones. Además favorece la portabilidad, ya que se pueden escribir programas sin prestar atención a las características dependientes del sistema.

Una función es un segmento de programa que realiza determinadas tareas bien definidas. Todo programa en C consta de una o más funciones. Una de estas funciones se debe llamar ***main***. La ejecución del programa siempre comenzará por la de las instrucciones contenidas en *main*. Se pueden subordinar funciones adicionales a ***main***, y posiblemente unas a otras.

Si un programa contiene varias funciones, sus definiciones deben aparecer en cualquier orden, pero deben ser independientes unas de otras. Esto es, una definición de una función no puede estar incluida en otra.

Generalmente, una función procesará la información que le es pasada desde el punto del programa en donde se accede a ella y devolverá un solo valor. La información se le pasa a la función mediante unos identificadores especiales llamados argumentos (también denominados parámetros) y es devuelta mediante la sentencia ***return***. Sin embargo, algunas funciones aceptan información pero no devuelven nada, mientras que otras funciones devuelven varios valores.

Los argumentos formales permiten que se transfiera información desde el punto del programa en donde se llama a la función a ésta. También se llaman parámetros o

parámetros formales. Los identificadores utilizados como argumentos formales son "locales" en el sentido de que no son reconocidos fuera de la función.

Se puede acceder (llamar) a una función especificando su nombre, seguido de una lista de argumentos cerrados entre paréntesis y separados por comas.

Los argumentos que aparecen en la llamada a la función se denominan argumentos actuales, mientras que los argumentos formales son los que aparecen en la primera línea de definición de la función. En una llamada normal a una función, habrá un argumento actual por cada argumento formal. Los argumentos actuales pueden ser constantes, variables simples, o expresiones más complejas. No obstante, cada argumento actual debe ser del mismo tipo de datos que el argumento formal correspondiente.

Cuando se le pasa un valor simple a una función mediante un argumento actual, se copia el valor del argumento actual al argumento formal de la función. Por tanto, se puede modificar el valor del argumento formal dentro de la función, pero el valor del argumento actual en el punto de llamada no cambiará. Este procedimiento para pasar el valor de un argumento se denomina paso por valor. El compilador convertirá el valor de cada argumento actual al tipo de datos declarados y comparará si coincide el tipo de datos de cada argumento actual con el de argumento formal correspondiente.

Se llama recursividad a un proceso mediante el que una función se llama a sí misma de forma repetida, hasta que se satisface alguna determinada condición. Se deben satisfacer dos condiciones para que se pueda resolver un problema recursivamente. Primero, el problema se debe escribir en forma recursiva, y segundo, la sentencia del problema debe incluir una condición de fin.

- *Arrays*

Los ***Arrays*** *(ó arreglos)* se definen en gran parte como las variables ordinarias, excepto en que cada ***Array*** debe acompañarse de una especificación de tamaño. Para un ***Array*** unidimensional, el tamaño se especifica con una expresión entera positiva encerrada entre paréntesis cuadrados. La expresión es generalmente una constata entera positiva.

El nombre de un ***Array*** se puede usar como argumento de una función, permitiendo así que el ***Array*** completo sea pasado a la función. Sin embargo, la manera de la que se pasa difiere mucho de la de una variable ordinaria.

Para pasar un ***Array*** a una función, el nombre del ***Array*** debe aparecer solo, sin corchetes o índices, como un argumento actual dentro de la llamada a la función. El correspondiente argumento formal se escribe de la misma manera, pero debe ser declarado como un ***Array*** dentro de la declaración de argumentos formales. Cuando se declara una ***Array*** unidimensional como un argumento formal, el ***Array*** se escribe con una par de corchetes cuadrados vacíos.

Los ***Arrays*** multidimensionales son definidos prácticamente de la misma manera que los ***Arrays*** unidimensionales, excepto que se requiere un par separado de corchetes para cada índice. Así un **arrray** bidimensional requerirá dos pares de corchetes, un ***Array*** tridimensional requerirá tres pares de corchetes, y así sucesivamente.

- Punteros

El objetivo básico de los punteros es identificar posiciones de memoria. Como la memoria está dividida en muchas posiciones, cada una de las cuales es capaz de almacenar información, necesita un método de colocar un valor en una posición especifica de la memoria y más tarde recuperar ese valor. La manera en que esto se hace es asignando a cada posición de memoria una dirección única. En C utiliza los símbolos & y *. El símbolo (&) se utiliza para especificar la dirección de una variable, no el valor que contiene. El asterisco (*) se utiliza para recuperar el valor contenido en la posición referenciada por un puntero.

- Estructuras, uniones y variables definidas por el usuario

La declaración de estructuras es algo más complicada que la declaración de ***Arrays***, ya que una estructura debe ser definida en términos de sus miembros individuales. Los miembros individuales pueden ser variables ordinarias, punteros, ***Arrays***, u otras estructuras. Los nombres de los miembros dentro de una estructura particular deben ser todos diferentes, pero el nombre de un miembro puede ser el mismo que de una variable definida fuera de la estructura.

No se puede asignar un tipo de almacenamiento a un miembro individual, y tampoco puede inicializarse dentro de la declaración del tipo de la estructura.

La característica **typedef** permite a los usuarios definir nuevos tipos de datos que sean equivalentes a los tipos de datos existentes. Una vez que el tipo de dato definido por el usuario ha sido establecido, entonces las nuevas variables, ***Arrays***, estructuras pueden ser declaradas en términos de este nuevo tipo de dato.

La característica **typedef** es particularmente útil cuando se definen estructuras, ya que elimina la necesidad de escribir repetidamente el modificador **estruct**, en cualquier lugar que una estructura sea referenciada. Como resultado, la estructura puede ser referenciada más concisamente. Además, el nombre dado al tipo de estructura definido por el usuario sugiere a menudo el propósito de la estructura dentro de un programa.

La dirección de comienzo de una estructura puede accederse de la misma manera que cualquier otra dirección, mediante el uso del operador dirección (&).

Los miembros individuales de estructura se pueden pasar a una función como argumentos en la llamada a la función, y un miembro de una estructura puede ser devuelto mediante la sentencia ***return***. Para hacer esto, cada miembro de la estructura se trata como una variable ordinaria unievaluada.

La mayoría de las nuevas versiones de C permiten que una estructura completa sea transferida directamente a una función como un argumento y devuelta directamente mediante la sentencia ***return***. Cuando una estructura se pasa directamente a una función, la transferencia es por valor y no por referencia. Esto es consistente con otra transferencia directa (no puntero) en C. Por lo tanto, si cualquiera de los miembros de la estructura es alterado dentro de la función, las alteraciones no serán reconocidas fuera de la función. Sin embargo, si la estructura modificada es retornada a la parte llamante del programa, entonces los cambios serán reconocidos dentro de est ámbito mayor.

Las estructuras autorreferenciadoras son muy útiles en aplicaciones que involucren estructuras de datos enlazadas, tales como listas y árboles.

La idea básica de la estructura enlazada de datos es que cada estructura incluye un puntero indicando dónde está el siguiente componente. Por tanto, el orden relativo de las componentes puede ser fácilmente cambiado, simplemente alterando los puntos. Además las componentes individuales pueden ser fácilmente añadidas o borradas, alterando los punteros nuevamente. Como resultado, una estructura de datos enlazada no se confina a un número máximo de componentes.

Las uniones, como las estructuras, contienen miembros cuyos tipos de datos pueden ser diferentes. Sin embargo, los miembros que componen una unión comparten el mismo área de almacenamiento dentro de la memoria de la computadora, mientras que cada miembro dentro de la estructura tiene asignada su propia área de almacenamiento. Así, las uniones se usan para ahorrar memoria. Son útiles para aplicaciones que impliquen múltiples miembros donde no se necesita asignar valores a todos los miembros a la vez.

Dentro de una unión, la reserva de memoria requerida para almacenar miembros cuyos tipos de datos en diferentes (tienen diferentes requerimientos de memoria) son manejados automáticamente por el compilador. Sin embargo, el usuario debe conservar una pista del tipo de información que está almacenado en cada momento. Una tentativa de acceso al tipo de información equivocado producirá resultados sin sentido.

Una unión puede ser miembro de una estructura y una estructura puede ser miembro de una unión. Además, las estructuras y las uniones pueden ser mezcladas libremente con ***Arrays***.

Las variables de enumeración son útiles en programas que requieren indicadores para identificar condiciones lógicas internas.

Una enumeración es un tipo de datos, similar a la estructura o a la unión. Sus miembros son constantes que están escritas como identificadores y sin embargo tienen asignados valores numéricos. Estas constantes representan valores que pueden ser asignados a variables de enumeración.

- Archivos de datos

Los archivos de datos permiten almacenar información de modo permanente, para ser accedida o alterada cuando sea necesario.

A diferencia de otros lenguajes de programación, en C no se distingue entre archivos secuenciales y de acceso directo (acceso aleatorio). Pero existen dos tipos distintos de archivos de datos, llamadas archivos secuenciales de datos (o estándar) y archivos orientados a sistema (o de bajo nivel).

Los archivos de datos secuenciales se pueden dividir en dos categorías. En la primera categoría están los archivos que contienen caracteres consecutivos. Estos caracteres pueden interpretarse como datos individuales, como componentes de una cadena o como números. La manera de interpretarlos es determinada dentro de las funciones de biblioteca usadas para transferir la información por las especificaciones de formato dentro de las funciones de biblioteca ***printf*** y ***scanf***.

La segunda categoría de archivos de datos secuenciales, a menudo llamados archivos sin formato, organiza los datos en bloques contiguos de información. Estos bloques representan estructuras de datos más complejos como ***Arrays*** y estructuras. Hay un conjunto diferenciado de funciones de biblioteca para tratar este tipo de archivos. Estas funciones proveen instrucciones simples que pueden transferir ***Arrays*** completos o estructuras a o desde un archivo de datos.

Los archivos orientados al sistema están más relacionados con el sistema operativo de la computadora que los archivos secuenciales. Es algo más complicado trabajar con ellos, pero su uso puede ser más eficiente para cierto tipo de aplicaciones. Para procesar archivos orientados a sistemas se requiere un conjunto separado de procedimientos con sus funciones de biblioteca correspondientes.

Cuando se trabaja con archivos secuenciales de datos, el primer paso es establecer un área de memoria denominada ***buffer***, donde la información se almacena temporalmente mientras se está transfiriendo entre la memoria de la computadora y el archivo de datos. Esta área de ***buffer*** permite leer y escribir información del archivo más rápidamente de los que sería posible de otra manera.

Un archivo de datos deber abierto antes de ser creado o procesado. Esto asocia el nombre de un archivo con el área de ***buffer***. También se especifica cómo se va a usar el archivo, sólo para lectura, sólo para escritura, para lectura/escritura, en el que se permiten las dos operaciones. En la tabla 2.3 se muestra la formas en que se puede abrir un archivo de datos.

Tipo de archivo	Significado
"r"	Abrir un archivo existente sólo para lectura
"w"	Abrir un nuevo archivo sólo para escritura. Si existe un archivo con el nombre-archivo especificado, será destruido y creado uno nuevo en su lugar.
"a"	Abrir un archivo existente para añadir (añadir información nueva al final del archivo). Se creará un archivo nuevo si no existe un archivo con el nombre nombre-archivo especificado.
"r+"	Abrir un archivo existente para lectura y escritura.
"w+"	Abrir un archivo nuevo para escritura y lectura. Si existe un archivo con el nombre-archivo especificado, será destruido y creado uno nuevo en su lugar.
"a+"	Abrir un archivo existente para leer y añadir. Se crea un archivo nuevo si no existe un archivo con el nombre nombre-archivo especificado.

Tabla 2.3-Especificación de tipo de archivo.

Finalmente, un archivo de datos debe cerrarse al final del programa. Esto puede realizase con la función de biblioteca ***fclose***.

Un archivo secuencial de datos puede crearse de dos formas distintas. Una es crear el archivo directamente, usando un procesador de texto o un editor. La otra es escribir un programa que introduzca información en la computadora y la escriba en un archivo. Los archivos sin formato sólo pueden crearse con programas especialmente escritos para tal fin. Si el archivo consta de caracteres individuales, se pueden usar las funciones de biblioteca ***getchar*** y ***putc*** para introducir los datos desde el teclado y escribirlos en el archivo.

Los archivos de datos que contiene sólo cadenas de caracteres pueden crearse y leerse más fácilmente con programas que utilizan funciones de biblioteca especialmente orientadas para cadenas. Algunas funciones de este tipo comúnmente usadas son ***gets***, ***puts***, ***fgets*** y ***fputs***. Las funciones ***puts*** y ***gets*** leen o escriben cadenas a o desde los periféricos estándar de salida, mientras que ***fgets*** y ***fputs*** intercambian cadenas con archivos.

Muchos archivos de datos contiene estructuras de datos más complicadas, como registros que incluyen combinaciones de información, caracteres y números. Tales archivos se pueden procesar usando las funciones de biblioteca ***fscanf*** y ***fprintf***, que son análogas a las funciones ***scanf*** y ***printf***.

La mayoría de las aplicaciones con archivos requieren que se altere el archivo cuando se procesa. Por ejemplo, en aplicaciones que impliquen el procesado de registros de clientes, se puede desear añadir nuevos registros al archivo (ya sea al final del archivo o entre los registros existentes) para borrar registros existentes, para modificar los contenidos de registros, o para reordenar los registros. Estos requerimientos sugieren varias estrategias computacionales distintas.

Trabajar con dos archivos diferentes: un archivo antiguo (la fuente) y otro nuevo. Se lee cada registro del archivo antiguo, se actualiza y se escribe el nuevo archivo. Cuando se han actualizado todos los registros. Se borra o se almacena el archivo antiguo y se renombra el archivo nuevo. Así el nuevo archivo se convierte en la fuente para el siguiente turno de modificaciones.

Históricamente el origen de este método proviene de los primeros días de la computación, cuando los archivos de datos se almacenaban en cintas magnéticas. Sin embargo, este método se usa todavía ya que provee de una serie de antiguas fuentes que pueden usarse para generar el pasado de los clientes. La fuente más reciente puede usarse también para recrear el archivo actual si éste se daña o se destruye.

Algunas aplicaciones implican el uso de archivos para almacenar bloques de datos, donde cada bloque consiste en un número fijo de bytes continuos. Cada bloque representará generalmente una estructura de datos compleja, como una estructura o un ***Array***. Por ejemplo, un archivo de datos puede constar de varias estructuras que tengan la misma composición, o puede contener múltiples ***Arrays*** de mismo tipo y tamaño. Para estas aplicaciones sería deseable leer o escribir el bloque entero del archivo de datos en vez de leer o escribir separadamente las componentes individuales (miembros de la estructura o elementos del ***Array***) de cada bloque.

Las funciones de biblioteca ***fread*** y ***fwrite*** deben usarse en estas situaciones. A estas funciones se les conoce frecuentemente como funciones de lectura y escritura sin formato. Igualmente se hace referencia a estos archivos de datos como archivos de datos sin formato.

Hasta este momento hemos tratado los conceptos de *hardware* y *software* que necesitamos como base para la elaboración de lo que será nuestro sistema. A continuación se mencionará con más detalle lo que se utilizó para la creación del SAST:

III. PROGRAMACIÓN CLIENTE-SERVIDOR EMPLEANDO SOCKETS

En este capítulo se presentará al lector una visión rápida de lo que es la programación cliente-servidor empleando *sockets*. Este concepto ofrece un mecanismo de comunicación general que permite el desarrollo de sistemas de *software* cuyos módulos que lo conforman se ejecutan en un mismo equipo o en equipos diferentes, e incluso, sobre distintas plataformas de sistema operativo.

3.1. Concepto de *Socket*

La manera más simple de aprender la noción de *socket* es haciendo una analogía con la comunicación entre individuos por correo o por teléfono. Cada una de las entidades implicadas en una comunicación debe disponer de un punto de contacto: se puede tratar de un buzón o de un receptor telefónico. Un *socket* no es otra cosa que el equivalente en el sistema a uno de estos objetos: se trata de un punto de comunicación por el cual un proceso podrá emitir o recibir información. La naturaleza de esta última dependerá del tipo de comunicación: de un buzón se extraen mensajes completos, mientras que el teléfono permite el envío de flujos de información que no tienen una estructura claramente definida (su definición se deja a la interpretación de los interlocutores). Esta dualidad se encuentra en los *sockets* a nivel de sus tipos.

Otra característica importante de los puntos de comunicación concierne al conjunto de otros puntos que permite acceder. Un receptor de teléfono interior de una empresa no permite comunicar más que con los otros interiores de la empresa, mientras que un teléfono exterior permite salir del marco local. La noción de **dominio** de un *socket* permite definir el conjunto de *sockets* con los cuales se podrá establecer una comunicación por medio de él.

Un *socket* es un "punto final para comunicación", por el cual un proceso puede emitir o recibir información. En el interior de un proceso, un *socket* se identificará por un descriptor de la misma naturaleza que los que definen los archivos. Esta propiedad es esencial, puesto que permite, por ejemplo, la redirección de los archivos de entrada-salida estándar (descriptores 0, 1 y 2) a los *sockets* y, por tanto, la utilización de aplicaciones estándar sobre la red. Esto significa también que un nuevo proceso (creado por *fork*) hereda los descriptores de *socket* de su proceso padre.

La creación de un *socket* se realizará por medio de la primitiva *socket* cuyo valor de vuelta es un descriptor sobre el cual es posible realizar operaciones de lectura y escritura. Un *socket* permite la comunicación en los dos sentidos, es decir, es un mecanismo de comunicación bidireccional.

3.2. El Dominio de un *Socket*

El dominio de un *socket* especifica el formato de las direcciones que se podrán dar al *socket* y los diferentes protocolos soportados por las comunicaciones vía los *sockets* de este dominio.

La estructura genérica que se utilizará para describir las diferentes primitivas es:

```
struct sockaddr {
        u_short         sa_family;      /* familia de direcciones */
        char            sa_data[14];    /* 14 octetos de dirección (máximo) */
};
```

Para una aplicación particular, esta estructura se deberá reemplazar por la estructura correspondiente del dominio de comunicación utilizado.

a) Para el dominio UNIX (AF_UNIX), los *sockets* son locales al sistema donde han sido definidos. Permiten la comunicación interna de procesos y su designación se realiza por medio de una referencia UNIX. La estructura de una dirección en este dominio está predefinida en <sys/un.h>.

Las estructuras son definidas por dos tipos de direcciones de *sockets*. En un dominio como el siguiente:

```
struct  sockaddr un {
        short  sun_family;          /* dominio UNIX: AF_UNIX*/
        char   sun_path[108];       /* referencia de dirección  */
};
```

b) Para el dominio internet (AF_INET), las direcciones de los *sockets* tienen la estructura sockaddr_in, de la siguiente forma:

```
Struct in_addr {
        U_long          s_addr;
};

struct  sockaddr_in {
        short  sin_family;          /* familia de direcciones: AF_INET      */
        u_short sin_port;           /* Número de Puerto     */
        struct    in_addr  sin_addr;  /* Dirección de internet  */
        char      sin_zero[8];      /* Un campo de 8 ceros */
};
```

c) Pueden existir otro cierto número de dominios. Citemos:

```
AF_NS           /* protocolos XEROX NS */
AF_CCITT        /* protocolos CCITT, protocolos X25, etc. */
AF_SNA          /* IBM SNA */
AF_DECnet       /* DECnet */
AF_APPLETALK    /* Apple Talk */
```

3.3. Tipos de *Sockets*

3.3.1 Propiedades de la comunicación por sockets

El tipo de un *socket* define un tipo de propiedades de las comunicaciones en las cuales está implicado.

a) La fiabilidad de la transmisión. Ningún dato se pierde entre ambos puntos de la comunicación.

b) La conservación del orden de los datos. Los datos llegan en el orden en que han sido emitidos.

c) La no duplicación de datos. Sólo llega al destino un ejemplar de cada dato emitido.

d) La comunicación en modo conectado. Se establece una conexión entre dos puntos antes del principio de la comunicación. A partir de entonces, una emisión desde un extremo está implícitamente destinada al otro extremo conectado.

e) La conservación de los límites de los mensajes. Los límites de los mensajes emitidos se pueden encontrar en el destino.

f) EL envío de mensajes "urgentes". Corresponde a la posibilidad de enviar datos fuera de flujo normal y por consecuencia accesibles inmediatamente.

3.3.2 Tipos de sockets disponibles

Otra propiedad de los *sockets*, tan separada de su familia de direcciones, es su tipo. El tipo de un *socket* se refiere al tipo de capa de transporte que emplean, los principales son:

a) El tipo SOCK_DGRAM

Corresponde a los *sockets* destinados a la comunicación en modo no conectado para el envío de datagramas de tamaño limitado. Las comunicaciones correspondientes tienen la propiedad mencionada en el inciso e, del apartado anterior. En el dominio internet, el protocolo subyacente es el protocolo UDP.

b) El tipo SOCK_STREAM

Los *sockets* de este tipo permiten comunicaciones fiables en modo conectado (propiedades a, b ,c y d) y eventualmente autorizan (según el protocolo apropiado) los mensajes fuera de banda (propiedad f). El protocolo subyacente en el dominio internet es TCP.

c) El tipo SOCK_RAW

Permite el acceso a los protocolos de más bajo nivel (por ejemplo, el protocolo IP en el dominio Internet). Su uso está reservado al superusuario. Permite implantar nuevos protocolos.

d) El tipo SOCK_SEQPACKET

Corresponde a las comunicaciones que poseen las propiedades a, b, c, d y e. Estas comunicaciones se encuentran en el dominio XEROX NS.

Los dos tipos de *sockets* más utilizados y de los cuales estudiaremos su manipulación, son los dos primeros.

3.4. La Creación de un *Socket*

La función ***socket*** permite la creación de un *socket*, es decir, la creación e inicialización de entradas en las diferentes tablas del sistema de gestión de archivos: tabla de descriptores de procesos, tabla de archivos y estructuras de datos conteniendo las

características del *socket* (teniendo un papel análogo al de los i-nodos para los archivos). Entre estas características se encuentran:

- El tipo, el dominio y el protocolo;
- El estado del *socket* (conectado o no, enlazado, en estado de recibir o de emitir);
- La dirección del *socket* conectado (si hay alguno): en efecto, al *socket* se le asocia un extremo de emisión y otro de recepción;
- Punteros a los datos (en emisión y recepción);
- Un grupo de procesos para la gestión de mecanismos asíncronos (en relación con las señales SIGURG y SIGIO).

La forma general de la función que permite crear un *socket* y obtener un descriptor para manipularlo es:

Int *socket*(dominio, tipo, protocolo)

```
Int dominio;        /* AF_INET, AF_UNIX, ... */
Int tipo;           /* SOCK_DGRAM, SOCK_STREAM, ... */
Int protocolo;          /* 0: protocolo por defecto */
```

En caso de fallo de la función se devuelve un valor de −1. El tercer parámetro es generalmente 0: el sistema elige entonces un protocolo por defecto asociado al tipo y dominio del *socket* creado.

3.5. La Supresión de un *Socket*

Un *socket* es suprimido cuando ya no hay ningún proceso que posea un descriptor para accederlo. Esta supresión implica la liberación de entradas en las diferentes tablas y los extremos reservados por el sistema relacionados con el *socket*. Por tanto, esta supresión es el resultado de al menos una llamada a la función ***close***.

3.6 La función *bind*

Después de su creación un *socket* no es accesible mas que por los procesos que conocen su descriptor. Por consecuencia, sin un mecanismo suplementario de designación, sólo los procesos que hayan heredado este descriptor en su creación (por la función ***fork***) podrían utilizar un *socket* (como es el caso de los cauces creados con la función ***pipe***). La función *bind* permite con una sola operación dar un nombre a un *socket*:

Int *bind*(sock, p_dirección, lg)

```
Int sock;                       /* descriptor de socket */
Struct sockaddr *p_dirección;   /* puntero de memoria a la dirección */
Int lg;                         /* longitud de la dirección */
```

Para una utilización de un dominio en particular, el puntero p_dirección apunta a una zona cuya estructura es la de una dirección en ese dominio (sockaddr_un para el dominio AF_UNIX y sockaddr_in para el dominio AF_INET).

El *socket* nombrado después de realizarse con éxito la función (valor de retorno 0), puede ser identificado por cualquier proceso por medio de la dirección que se le ha dado sin

necesidad de poseer un descriptor (lo que es a priori imposible si el proceso no pertenece al sistema del *socket*). Un cierto número de funciones específicas permiten explotar estas direcciones.

Las causas de error (valor devuelto –1) de petición de conexión son múltiples: descriptor incorrecto (EBADF, ENOTSOCK), dirección incorrecta, inaccesible o ya utilizada (EADDRNOTAVAIL, EADD

3.7. *Sockets* No Orientados a Conexión

Este punto está dedicado al estudio de la comunicación por datagramas por medio de los *sockets* del tipo SOCK_DGRAM (cualquiera que sea su dominio de definición).

Recordemos las principales características de estas comunicaciones:

- Cuando se envía un mensaje desde un *socket* con destino a otro *socket*, el que lo expide no obtiene ninguna información sobre la llegada de su mensaje al destino.
- Los límites de los mensajes son preservados.

La falta de seguridad debido a la primera característica no es más que relativa. Aplicaciones esenciales tales como el sistema de archivos distribuidos NFS (Network File System, Sistema de Archivos a Través de la Red), utilizan este modo de comunicación (apoyándose en el protocolo UDP dentro del dominio Internet).

3.7.1 Principio General

El intercambio de datagramas debe ser realizado por medio de *sockets* del tipo SOCK_DGRAM. Un proceso que desee emitir un mensaje con destino a otros debe, por una parte, disponer de un punto de comunicación local (descriptor del *socket* sobre el sistema local) y por otra parte, conocer una dirección del sistema al cual pertenezca su interlocutor (esperando que este interlocutor disponga de un *socket* conectado o enlazado a esta dirección...este tipo de comunicación no permite estar seguro de ello). Los *sockets* de este tipo se utilizan en modo no conectado: en principio toda petición de envío de un mensaje debe incluir la dirección del *socket* de destino.

3.7.2 Operaciones de Envío y recepción

Para realizar las operaciones de envío y recepción en *sockets*, se hace uso de dos primitivas, las cuales nos indican si las operaciones realizadas tuvieron éxito o no, a continuación se describirá como funcionan estas.

- La Primitiva ***sendto***

Int *sendto*(sock, msg, lg, opcion, p_dest, lgdest)

```
Int sock;                 /* descriptor del socket de emisión */
Char msg;                 /* dirección del mensaje a enviar */
Int lg;                   /* longitud del mensaje */
Int opcion;               /* = 0 para el tipo SOCK_DGRAM */
Struct sockaddr pdest;    /* puntero a la dirección del socket destino */
Int lgdest;               /* longitud de la dirección del socket destino */
```

El valor de retorno de la primitiva es, en caso de éxito, el número de caracteres enviados y −1 en caso de fallo. Es necesario subrayar el hecho de que los únicos errores detectados son los locales. Así, no hay ninguna detección de que un *socket* esté enlazado a la dirección dada. Esto significa en particular que si un mensaje se emite con destino a una máquina existente sobre un puerto no asociado a un *socket*, el mensaje se perderá y el emisor no será avisado.

- La Primitiva ***recvfrom***

Int *recvfrom*(sock, msg, lg, opcion, p_expt, lgexp)

```
Int sock;                    /* descriptor del socket de recepción */
Char msg;                    /* dirección de recuperación del mensaje recibido
*/
Int lg;                      /* longitud del mensaje */
Int opcion;                  /* 0 o MSG_PEEK*/
Struct sockaddr pexp;        /* para recuperar la dirección de expedición */
Int lgexp;                   /* tamaño del espacio reservado a p_exp */
```

Esta primitiva permite a un proceso extraer un mensaje de un *socket* del cual posea un descriptor. Para que haya un mensaje, es preciso que este *socket* haya sido enlazado o conectado a una dirección y que un proceso haya enviado un mensaje después de que haya tenido lugar este enlace.

El papel del parámetro ***lg*** es el de dar el tamaño del espacio reservado para memorizar el mensaje que será extraído. Es preciso no perder de vista que se extraerá del *socket* un mensaje completo, que corresponde exactamente a la información enviada en una sola operación ***sendto***. La información recibida provendrá de un solo mensaje y los caracteres del mensaje se perderán si el parámetro ***lg*** es menor que la longitud del mensaje recibido. La primitiva devuelve el número de caracteres recibidos (-1 en caso de error)

En el caso de que el puntero p_exp es distinto de ***NULL***, a la vuelta, la zona de dirección p_exp contiene la dirección del *socket* emisor. Es esencial en el caso en que la llamada *p_lgexp contiene el tamaño de la zona reservada a la dirección p_exp; a la vuelta *p_lgexp tiene el valor de la longitud efectiva de la dirección. El proceso receptor tiene entonces la posibilidad de responder, si hay necesidad de ello, al mensaje recibido por medio de la función *sendto* (recordemos que los *sockets* permiten la comunicación en los dos sentidos).

3.7.3 Ejemplo Completo

Como primer ejemplo presentaremos un proceso cliente con un proceso servidor (un proceso "demonio"). Los dos procesos intercambian datagramas en el dominio de UDP. El proceso servidor está en espera de recepción de mensajes sobre un *socket* enlazada hacia un puerto UDP conocido por los clientes que quieren dirigirse a él (para este ejemplo emplearemos el puerto 2222). Una petición formulada por un cliente consiste simplemente en una cadena de caracteres que el cliente quiere saber si corresponde a la identificación de un usuario sobre el sistema al que pertenece el servidor, y si éste es el caso, obtiene la información (contenida en el archivo etc/passwd) relativas a este usuario.

- El Programa Servidor

El código del servidor comprende las siguientes etapas:

a) Creación y enlace al puerto de servicio de un *socket*;
b) Desconexión del servidor de su terminal de lanzamiento o ejecución;
c) Bucle infinito en el cual el servidor:

- Espera una petición;
- La trata (llamada a la función estándar ***getpwnam*** que permite consultar el archivo /etc/passwd);
- Pone la forma de respuesta (el campo tipo permite distinguir el caso que el usuario es desconocido ['1'] y el que no lo es ['2']);
- Envía la respuesta.

$ cat serv_user.c

/* Servidor Internet en el puerto UDP 2222 que devuelve las informaciones relativas a un usuario cuyo nombre se le ha dado */

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <netinet/in.h>
#include <pwd.h>

#define LGUSER 20
#define LGREP 256
#define PORT 2222

main(n, v)
int n;
char *v[];
{
        int lg, sock, port;
        int d;
        char user[LGUSER];                  /* el mensaje recibido */
        struct sockaddr_in adr;     /* dirección del socket remoto */
        struct passwd *getpwnam(), *p;

        /* la estructura enviada */
        struct respuesta {
                char type;              /* tipo '1': error, tipo '2': OK */
                char info[LGREP];   /* la información pedida si tipo = '2' */
        } rep;

        port = PORT;

        if ( (sock = crearsock(&port, SOCK_DGRAM)) == -1 ) {
                fprintf(stderr, "Fallo en la creación/conexión del socket\n");
```

```
            exit(2);
        }

        /* desconexión del servidor de la terminal */
        close(0); close(1); close(2);

        if ( (d = open("/dev/tty")) > -1) {
                ioctl(d, TIOCNOTTY, 0);
                close(d);
        }

        setpgrp();

        while ( 1 ) {   /* espera de pregunta sobre el socket */
                lg = sizeof(adr);
                bzero(user, LGUSER);
                bzero((char *) &rep, sizeof(rep));

                n = recvfrom(sock, user, LGUSER, 0, &addr, &lg);

                if ( (p = getpwnam(user)) == NULL )
                        rep.type = '1';

                else {
                        rep.type = '2';
                        sprintf(rep.info, "%d %d %s %s %s\n", p->pw_uid,
                                                              p->pw_gid,
                                                              p->pw_gecos,
                                                              p->pw_dir,
                                                              p->pw_shell);
                }

                /* envío de la respuesta */

                n = sendto(sock, (char *) &rep, sizeof(struct respuesta), 0, &adr, lg);

    }
}
```

La forma correcta de ejecutar este programa con profundidad es la siguiente:

$ serv_user &

- El Programa Cliente

Su código se compone de las siguientes etapas:

a) Creación del *socket* local (su enlace o conexión no es necesario, ya que el cliente comenzará emitiendo y el enlace se realizará automáticamente.;
b) Preparación de la dirección del servidor;
c) Envío del mensaje;

d) Espera del resultado;
e) Explotación del resultado.

A continuación se lista el código completo y ejemplos de utilización:

$ cat user.c

```
/* Programa que permite obtener la información contenida en el archivo /etc/passwd de
una máquina de nombre dado como primer parámetro, concerniente a un usuario que se
da como segundo parámetro */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>

#define PORT 2222
#define LGREP 256

main(n, v)
int n;
char *v[];
{
        int lg, sock, uid, gid;
        char p[LGREP];
        struct {
                char type;
                char info[LGREP];
        } rep;

        struct sockaddr_in adr;
        struct hostent *hp;

        if ( n < 3 ) {
                fprintf(stderr, "número de parámetros incorrecto\n");
                exit(2);
        }

        /* preparación de la dirección del socket destino */
        if ( (hp = gethostbyname(v[1]) == NULL ) {
                perror("nombre de la máquina");
                exit(2);
        }

        bzero((char *) &adr, sizeof(adr));
        sock = socket(AF_INET, SOCK_DGRAM, 0);
        adr.sin_family = AF_INET;
        adr.sin_port = htons(PORT);
        bcopy(hp->h_addr, &adr.sin_addr, hp->h_length);
```

```
        /* Envío del mensaje constituido por el nombre de usuario */
        if ( sendto(sock, v[2], strlen(v[2]), 0, &adr, sizeof(adr)) == -1 ) {
                perror("sendto");
                exit(2);
        }

        /* Espera de la respuesta por parte del servidor */
        lg = sizeof(adr);
        n = recvfrom(sock, (char *) &rep, sizeof(rep), 0, &addr, &lg);

        /* Explotación de la respuesta */
        if ( rep.type == '1' )
                fprintf(stderr, "%s: usuario desconocido en %s\n", v[2], v[1]);
        else {
                printf("usuario %s en %s\n\n", v[2], v[1]);
                sscanf(rep.info, "%d %d %[ \n]", &uid, &gid, p);
                printf("                uid = %d      gid = %d\n, uid, gid);
                printf("                %s\n", p);
        }
}
```

Ejemplos de cómo se puede ejecutar este programa cliente se presentan a continuación:

$ user saturno mike
mike: usuario desconocido en saturno

$ user saturno jim
usuario jim en saturno

uid = 102 gid = 20
Jim /ens/jim/bin/csh

3.7.4 Las "Pseudoconexiones"

Este mecanismo permite simplemente alegar la escritura de aplicaciones en el caso de que un *socket* del tipo SOCK_DGRAM no se utilice más que para comunicar sólo con otro. Sin embargo, es necesario tener en cuenta que una pseudocomunicación de este tipo no modifica en nada las características generales del modo de comunicación (en particular en lo que concierne a su fiabilidad).

- La Función ***connect***

Utilizada con *sockets* del tipo SOCK_DGRAM, permite asociar un *socket* de dirección dada a un *socket* local (dado por su descriptor). Se verá que tiene un papel diferente con los *sockets* SOCK_STREAM.

La función tiene como forma general:

```
Int connect(sock, p_adr, lgard)

        Int sock;                       /* descriptor del socket local */
```

```
struct sockaddr *p_adr;    /* puntero a la dirección del socket asociado */
Int lgadr;                 /* longitud de la dirección */
```

Después de la "pseudoconexión" de un *socket* a otro, todas las emisiones de mensajes, vía el primero, tendrán como destino el segundo, que no podrá recibir más que los mensajes que provengan del otro. Es importante notar que esta pseudoconexión es local: no tiene ninguna incidencia sobre el *socket* del cual de ha dado la dirección (la dirección del *socket* remoto está memorizada en la estructura asociada al *socket* local). Un segundo parámetro a la primitiva *connect* suprime una asociación anterior.

3.7.5 Operaciones de Lectura y Escritura

Como la dirección de destino o de origen es implícita, es posible utilizar las funciones de entrada/salida usuales read y write. Además existen dos funciones específicas del *socket* como punto de comunicación: ***recv*** y ***send***.

Int *send*(sock, msg, lg, opción)

```
Int sock;          /* descriptor del socket local */
char *msg;         /* dirección de memoria del mensaje */
Int lg;            /* longitud del mensaje */
Int opción;        /* = 0 */
```

Y también

Int *recv*(sock, msg, lg, opción)

```
Int sock;          /* descriptor del socket local */
char *msg;         /* dirección de memoria para guardar el mensaje */
Int lg;            /* longitud de la zona reservada a la dirección msg */
Int opción;        /* = 0 */
```

Los parámetros son idénticos a los cuatro primeros de las funciones ***sendto*** y ***recvfrom***.

3.8 *Sockets* Orientados a Conexión

Es el modo por las aplicaciones estándar de la familia Internet tales como telnet, ftp o aplicaciones UNIX como rlogin. Aporta la fiabilidad de los intercambios de información con el precio de un incremento de su volumen. (Se pueden dirigir al epígrafe dedicado al protocolo TCP sobre el que se apoyan los intercambios en modo conectado en el dominio Internet).

3.8.1 Principio General

Como indica el nombre de este modo de comunicación, es necesario para utilizarlo realizar una conexión (establecer un <<circuito virtual>>) entre dos puntos. Uno de los dos procesos (en posición de cliente) pregunta al otro (en posición de servidor) si acepta esta comunicación. Se encuentra aquí la analogía que hemos hecho al principio, con la comunicación por correo y por teléfono. Mientras que el modo de comunicación sin conexión presenta un aspecto simétrico en la medida que el <<iniciador>> del diálogo

puede ser cualquiera de los dos que intervienen, haciendo nuevamente la analogía, cuando se ha franqueado una carta, el que la envía no tiene ningún control sobre lo que pasa, a lo más puede esperar que le llegue una respuesta después de un cierto lapso de tiempo.

Las características importantes de la comunicación entre dos puntos son:

- La fiabilidad;
- El aspecto continuo de la información (<<flujo>>): el receptor no tiene la posibilidad de encontrar la estructura de la información enviada (salvo la información y el protocolo particular establecido entre los dos procesos).

Para traducir el aspecto asimétrico inicial que se ha mencionado, se presentará sucesivamente el punto de vista del servidor y el del cliente en la fase inicial, además de los mecanismos generales utilizables por los dos procesos en la fase de comunicación propiamente dicha, donde reaparece la simetría.

3.8.2. El punto de vista del servidor

Su papel es pasivo en el establecimiento de la comunicación: después de haber avanzado en el sistema al que pertenece el cual está preparado para responder a las peticiones de servicio, el servicio se pone a la espera de peticiones de conexión que provienen de clientes. Para esto dispone de un ***socket* de escucha**, enlazado al puerto TCP correspondiente al servicio, sobre el que espera las peticiones de conexión. Cuando llega al sistema una petición de este tipo, se despierta al proceso servidor y se crea un nuevo *socket*: es este último, que llamaremos ***socket* de servicio**, el que se conecta al del cliente. Entonces el servidor podrá, por una parte delegar el trabajo necesario para la realización del servicio a un nuevo proceso (creado por *fork*) que utilizará entonces efectivamente la conexión y, por otra parte, retomará su <<vigilia>> sobre *socket* de escucha.

El esquema general de funcionamiento de un servicio se presenta en la Figura 3.1.

No se volverá sobre la primera etapa cuyo objetivo es simplemente dotar al servidor de un punto de comunicación direccionable desde el exterior.

- La función ***listen***

Esta primitiva permite a un servidor señalar a un sistema que acepta las peticiones de conexión.

Para llamar con éxito a la primitiva ***listen*** (por tanto, con valor de retorno 0), el proceso debe disponer de un descriptor de *socket* del tipo SOCK_STREAM. Si el *socket* no está previamente enlazado a un puerto, el sistema procede a este enlace. En este caso, será necesario hacer una llamada a la primitiva ***getsockname*** para conocer el número del puerto atribuido. La primitiva tiene la forma siguiente:

int *listen* (sock, nb)

int sock; /* descriptor del *socket* de escucha/
int nb; /* número máximo de peticiones de conexión pendientes/

El segundo parámetro define el tamaño de un archivo en el cual se memorizan las peticiones de conexión formuladas, preparadas (desde el punto de vista de los clientes están establecidas) pero todavía no notificadas al servidor. El tamaño del archivo está limitado a un cierto valor por el sistema (en general cinco).

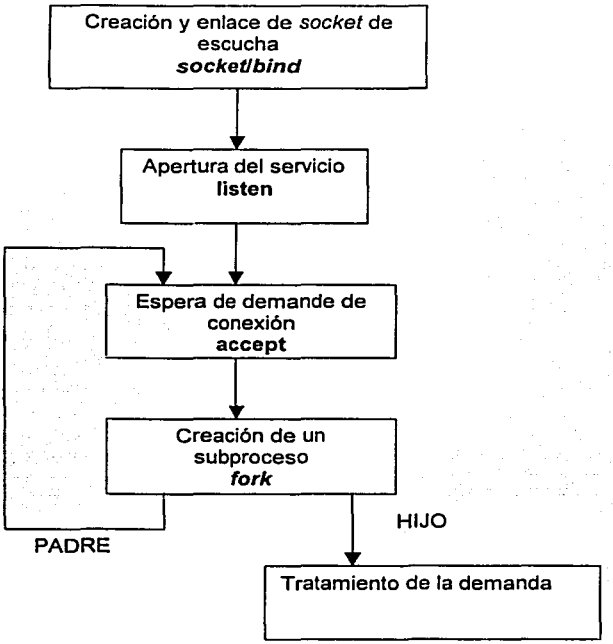


Fig. 3.1 Diagrama de flujo del funcionamiento del servidor.

- La primitiva ***accept***

Esta primitiva permite extraer una conexión pendiente en el archivo asociado a un *socket* para la cual se ha realizado una llamada a *listen*. De esta manera, el sistema toma conocimiento de un enlace realizado. Como se ha dicho, el enlace con el *socket* del cliente se realiza con un nuevo *socket* cuyo descriptor se envía como resultado de la función:

```
int accept (sock, p_adr, p_lgadr)

        int sock;                       /*descriptor del socket*/
        struct sockaddr p_adr;          /dirección del socket conectado*/
        int p_lgadr;            /* puntero al tamaño de la zona reservada a p_adr */
```

El *socket* de servicio creado se enlaza a un nuevo puerto (tomado del conjunto de puertos no reservados).

A la vuelta, también se recupera en memoria, en la zona apuntada por **p_adr**, la dirección del *socket* del cliente con el cual se ha establecido la conexión. El valor de ***p_lgadr** se modifica: si en la llamada era el tamaño de la zona reservada para guardar la dirección, en el retorno da el tamaño efectivo de la dirección.

En el caso en el que no existe ninguna conexión pendiente en el archivo, el proceso se bloquea hasta que exista una (¡0 hasta que llegue una señal!), a menos que el *socket* esté en modo no bloqueante. En este último caso, la primitiva devuelve el valor –1 y la variable errno tiene como valor EWOULDBLOCK.

- El subtratamiento del servicio

Después de la aceptación de la comunicación, el servidor tiene dos posibilidades:

- Hacerse cargo de ella: lo que significa eventualmente que otras conexiones pendientes no serán efectivamente aceptadas hasta que el servicio demandado haya sido satisfecho (a menos que se utilicen los mecanismos de aviso asíncrono);
- Subtratar le gestión de la conexión y la realización del servicio mediante un proceso hijo. El código de un servicio de este tipo corresponde en una primera aproximación al esquema siguiente:

```
    .
    .
exterrn void service( );
int sock_escucha, sock_service;
struct sockaddr_in adr;
int lgadr;
    .
    .
sock_escucha = socket(AF_INET, SOCK_STREAM, 0);
    .
    .
listen (sock_escucha, 5);
while (1) {
        lgadr = sizeof(adr);
        sock_service = accept(sock_escucha, &adr, &lgadr);
        if(fork( ) = = 0) {
            /* El proceso de servicio no usa el socket de escucha*/
            close(sock_escucha);
            /* Realización del servicio */
            service( );
            exit(0); }
        /* El proceso padre no utiliza el socket de servicio */
        close(sock_service);
        }
  }
```

3.8.3. El punto de vista del cliente

Un cliente es la entidad activa en el establecimiento de la conexión: es el que toma la iniciativa de la demanda de conexión de un servidor.

Esta demanda se realiza por medio de la función *connect* de la que ya se ha hablado para el establecimiento de pseudoconexiones de *sockets* del tipo SOCK_DGRAM. Sin embargo, aquí la semántica de la función es completamente diferente: solicita el establecimiento de una conexión que será conocida por los dos extremos. Además el cliente está informado del éxito o del fracaso del establecimiento de la conexión.

El funcionamiento general del cliente se muestra a en la figura 3.2.

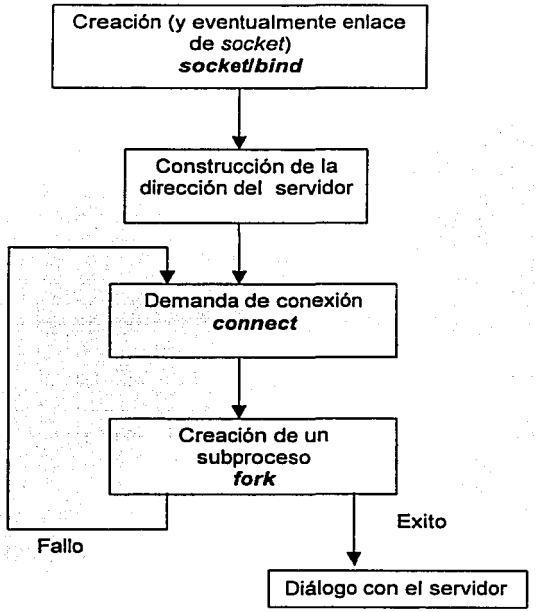


Fig. 3.2 Diagrama de flujo del funcionamiento del cliente.

3.8.4 La función ***Connect***

Toda conexión de dos *sockets* del tipo SOCK_STREAM, en un dominio distinto del UNIX, es el resultado de una llamado con éxito a esta función (en el dominio UNIX, una llamada a la primitiva ***socketpair*** crea un par de *sockets* conectados).

Así, se crea un circuito virtual entre los dos procesos cuyos extremos son los *sockets*. Este circuito permite intercambios bidireccionales.

La forma de la función es idéntica para la que anteriormente presentamos para la pseudoconexión de *sockets* dedicados al intercambio de datagramas.

Int *connect*(sock, p_adr, lgadr)

```
Int sock;                  /* descriptor del socket local */
struct sockaddr *p_adr;    /* dirección del socket remoto */
int lgadr;                 /* longitud de la dirtección remota */
```

El *socket* correspondiente al descriptor sock será conectado o enlazado a una dirección local, en el caso de que no lo estuviera ya previamente. La conexión puede establecerse (y la función *connect* devuelve 0) si se cumplen las siguientes condiciones:

a) Los parámetros son "localmente" correctos;
b) La dirección *p_adr se asocia a un *socket* del tipo SOCK_STREAM en el mismo dominio que el *socket* local de descriptor sock y un proceso (servidor) tiene solicitado escuchar sobre este *socket* (por una llamada *listen*);
c) La dirección *p_adr no está utilizada por otra conexión;
d) El archivo de conexiones del *socket* distante o remoto no está lleno.

En caso de éxito (valor de retorno 0 de la función *connect*), el *socket* local sock está conectado con un nuevo *socket* y la conexión está pendiente hasta que el servidor tenga conocimiento de ella a través de la función ***accept***. Sin embargo, el cliente puede comonzar a escribir o a leer de el *socket*.

En el caso de que no se cumpla una de las condiciones a), b) o c), el valor devuelto por la función es –1 y la variable errno permite conocer la razón del fallo. El comportamiento de la función es particular si no se cuimple la condición d).

- Si el *socket* es de modo bloqueante, el proceso se bloquea. La petición de conexión se repite durante un cierto tiempo; si al cabo de este lapso de tiempo, la conexión no se ha podido establecer, el proceso es despertado (valor devuelto –1 y errno = ETIMEDOUT).
- Si el *socket* es de modo no bloqueante, la vuelta es inmediata (errno = EINPROGRESS). Sin embargo, la petición de conexión no se abandona enseguida (se repite durante el mismo lapso de tiempo). El proceso puede saber si la conexión se ha establecido por medio de la primitiva **select**, el descriptor del *socket* forma parte de los descriptores comprobados en escritura. Finalmente, en el caso de que se realice una nueva solicitud de conexión del mismo *socket* (por una llamada a *connect*) antes de que haya abandonado la petición anterior, la vuelta de la primitiva es igualmente inmediata con errno = EALREADY.

3.8.5 El diálogo Cliente-Servidor

Una vez establecida la conexión entre un servidor y un cliente a través de dos *sockets*, los dos procesos pueden intercambiar flujos de información. A diferencia de lo que pasa en la comunicación por datagramas, el corte en diferentes mensajes no está preservado en el *socket* destino. Esto significa que el resultado de una operación de lectura puede provenir

de la información resultado de varias operaciones de escitura. Además en el caso de los *sockets* del dominio Internet, una petición de escritura de una cadena de caracteres larga, puede provocar el particionamiento de esta cadena, siendo accesibles los diferentes fragmentos por el *socket* destino. En este caso la única garantía que proporciona el protocolo TCP es que los fragmentos son accesibles en el orden correcto. Esto implica que la sincronización de una recepción y una emisión en una conexión de un mismo número de elementos no está asegurada por este mecanismo.

3.8.6 El Corte de la Conexión

Las cosas son un poco más complicadas por el hecho de que el protocolo subyacente (TCP en el dominio Internet) se supone que garantiza la fiabilidad de la transferencia. En estas condiciones una llamada a la función ***close*** con un descriptor de *socket* no puede suprimir de modo brutal el *socket* correspondiente y liberar el espacio reservado para los datos cuya emisión ha sido solicitada, pero que todavía no se ha realizado (es decir, todavía están almaxcenados en los *buffers* locales). Esta supresión tiene lugar posteriormente.

Además, si la conexión se ha cortado en la lectura en el otro extremo, una tentativa de escritura provocará la recepción de la señal SIGPIPE.

La función:

Int shutdown(desc, sens)

```
Int desc;    /* descriptor de socket */
Int sens;    /* 0, 1 ó 2 */
```

Permite a un proceso especificar que ya no desea recibir (sens=0), emitir (sens=1) o ni recibir ni emitir (sens=2) sobre un *socket* conectado de descriptor ***desc***.

Hasta este momento hemos analizado los diversos antecedentes en los que nos basaremos para la realización del proyecto, en los capítulos siguientes entraremos a la parte medular de este compendio, debido a que con las bases tomadas anteriormente podemos dar paso al análisis del SART.

IV. ANÁLISIS DEL SISTEMA AUTOMÁTICO DE SINCRONÍA DE TIEMPO (SAST)

Para iniciar propiamente con el tema central del presente trabajo, en este capítulo se hará un análisis completo de la problemática que se presenta en la planta telefónica de TELMEX, encaminado al desarrollo de una sistema automatizado que contribuya a la solución de la misma, mediante el cumplimiento de objetivos previamente definidos.

4.1 Antecedentes

A mediados de la década de los ochentas, Teléfonos de México S. A. de C.V. dio inicio a un ambicioso proyecto: la digitalización de su planta telefónica. En aquel entonces, TELMEX operaba con centrales telefónicas analógicas basadas en sistemas electromecánicos (consistentes en miles de relevadores y sistemas de rotación y desplazamiento), las cuales ocupaban grandes espacios en los edificios, requerían de un constante mantenimiento por parte de un grupo de técnicos, ofrecían dificultades a ciertas funciones, tales como la detección de fallas y la tarificación de llamadas; y sobre todo, contaban con capacidades relativamente bajas para atender usuarios (10,000 usuarios máximo por central). Algunos ejemplos de estas centrales telefónicas son los tipo AGF, ARF, Rotary y Pentaconta, las cuales funcionaron bien por mucho tiempo, pero al transcurso de los años fueron quedando obsoletas.

Las nuevas centrales telefónicas digitales, que vinieron a sustituir a las anteriores, cuentan ahora con módulos completamente computarizados (conformados por paneles de tarjetas de circuito impreso basadas en microprocesadores) y son controladas y administradas por *software* especializado. Estos equipos son de tamaño más compacto, cuentan con capacidad para atender hasta 80,000 líneas de usuario, no requieren de un mantenimiento exhaustivo, y ofrecen facilidades en la detección de fallas y en el proceso de facturación de llamadas, así como una amplia variedad de nuevos servicios a los clientes. Ejemplos de estas centrales telefónicas digitales son los equipos AXE (Ericsson), Sistema 12 (Alcatel), 5ESS (Lucent Technologies), DMS (Northern Telecom) y PTS (DSC Communications). Los cuatro primeros tipos son centrales de conmutación de circuitos, y pueden emplearse en distintas configuraciones según las necesidades de la planta (central de tráfico local, central de larga distancia, central tandem o de tráfico en tránsito y central de punto de transferencia de señalización). El último de los tipos, las centrales PTS, no cursan llamadas telefónicas propiamente dicho, sino que se encargan de la conmutación de paquetes de señalización entre centrales telefónicas; es decir, su función principal es comportarse como un punto de convergencia entre las mismas para el establecimiento de llamadas.

Luego de este proceso de digitalización de su planta telefónica y la modernización de todos los elementos de red en general, TELMEX quedó a la vanguardia en servicios de telecomunicaciones, al poder brindar a sus subscriptores una amplia gama de nuevos servicios adicionales, entre los cuales podemos mencionar los denominados servicios digitales (transferencia de llamada, llamada en espera, identificación de llamada, conferencia tripartita, correo de voz, despertador, etc). Asimismo, TELMEX cuenta ahora con la infraestructura para brindar servicios de telefonía celular, localizador, servicios ISDN (*Integrated Services Digital Network*, Red Digital de Servicios Integrados), servicios de audioconferencia, videoconferencia, acceso a la red Internet y enlaces de alta velocidad (satelitales o por fibra óptica) para grandes y medianos clientes.

Debido a la diversidad de tecnologías en los equipos de conmutación y a la amplitud de la red (alrededor de 600 centrales en toda la república), TELMEX se vio en la necesidad de adquirir un sofisticado sistema de gestión centralizado para sus centrales, denominado

NMA (*Network Monitoring And Analisys*, Monitoreo y Análisis de la Red), a través del cual es posible monitorear las alarmas especificas de cada elemento de red, tener acceso para la ejecución de cualquier rutina o comando de sistema, e incluso la creación de programas especiales (denominados *scripts*) cuya función es la explotación de la información.

EL sistema NMA (desarrollado por Bellcore) opera sobre una plataforma redundante y tolerante a fallas, consistente en un *mainframe* Stratus de alta capacidad corriendo con sistema operativo VOS (VOS. *Virtual Operating System*, Sistema Operativo Virtual).

Para poder cubrir toda la república, TELMEX cuenta con dos de estos equipos, uno para cubrir la parte sur del país (divisiones Metro, Pacifico, Centro, Sureste y larga distancia) y otro para atender la parte norte (divisiones Occidente, Norte, Noreste y Noroeste). Ambos equipos operan en las mismas condiciones de *hardware* y *software*, pero de manera independiente.

La figura 4.1 muestra la forma en que se encuentran interconectadas las centrales telefónicas a través de la RCDT y son gestionadas a través del NMA.

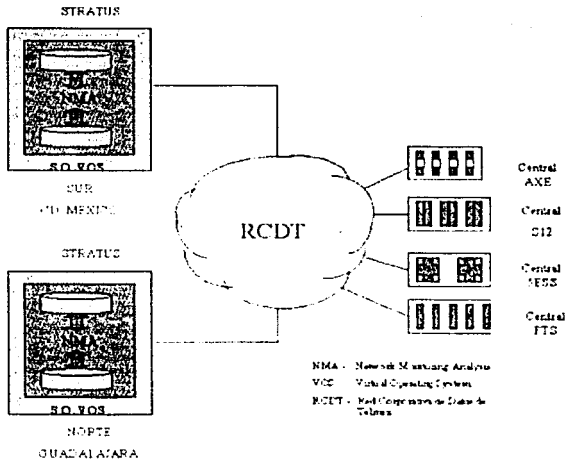


Fig. 4.1 Interconexión y gestión de las centrales telefónicas.

4.2 Definición del Problema

Para entender esta problemática, es necesario decir que cada central telefónica cuenta con una hora de sistema, la cual se toma en cuenta para registrar cada evento que ocurre en la misma; por ejemplo, cuando se realiza una llamada telefónica de un usuario de la central "A" a un usuario de la central "B", se genera en ambos puntos, un registro denominado "registro de facturación detallada", el cual contiene, entre otros datos, la hora de inicio y de terminación de la llamada (según la hora de sistema de cada central).

Lo anterior constituye un problema de magnitud importante para la empresa, ya que hasta el momento de escribir este trabajo, TELMEX no cuenta con un sistema completamente automatizado para sincronizar la hora de sistema de sus centrales, con respecto a una hora de referencia confiable. Actualmente, todas las centrales son monitoreadas por técnicos especialistas en el CNS (Centro Nacional de Supervisión) y en los CAR (Centro de Administración de la Red a través del NMA, y cuando alguno de ellos detecta que una central está muy desviada de la hora de referencia, envía manualmente el comando correspondiente (según el tipo de tecnología de la central) para ajustar la hora. Este procedimiento se lleva a cabo semanalmente y se toma +/- 10 segundos como una desviación aceptable con respecto de la hora de referencia. Cabe mencionar que esta forma de sincronizar las centrales no es del todo eficiente, ya que requiere de técnicos especialistas capacitados en el manejo del NMA y con los conocimientos suficientes relativos a comandos de la central.

Dado lo anterior, y después de un minucioso análisis de archivos de facturación, la gerencia de Auditoria de TELMEX detectó que cerca de tres millones de llamadas telefónicas, locales y de larga distancia, se dejan de cobrar como consecuencia de la diferencia entre las horas de sistema entre la central origen y la central destino; esto se da debido a la ausencia de un sistema de sincronización de tiempo eficiente. Concretamente, este problema se presenta cuando se realizan varias llamadas desde una misma línea telefónica, en un corto lapso de tiempo (llamadas de traslape), ya que cuando se hace el corte periódico de facturación, se analizan todos los registros de llamada mediante un algoritmo especial que, cuando por la diferencia de tiempo entre estos registros no puede precisar si se trata de una misma llamada o de llamadas diferentes, factura únicamente una de ellas, con la consecuente pérdida del cobro correspondiente a la llamada o llamadas restantes.

Este algoritmo de facturación, con el objeto de evitar cobros injustificados a los usuarios, toma como una sola llamada todas aquellas que se realicen dentro de un intervalo de tiempo de 30 segundos.

Este problema se presenta también cuando se cursa tráfico entre centrales telefónicas de otras empresas de telefonía (por ejemplo Alestra, Avantel, etc.) y centrales telefónicas de TELMEX. Consiste en que cuando se establece una llamada de larga distancia originada por un subscriptor de una de estas compañías que requiere el uso de la infraestructura de TELMEX, ésta compañía debe pagarle una cuota por interconexión, la cual dependerá del tiempo que dure la llamada. Cuando la hora de sistema de las centrales de TELMEX se encuentra fuera de sincronía, pueden darse dos casos: primero, que TELMEX cobre un tiempo de interconexión menor al real, y segundo, que al cobrar un tiempo de interconexión mayor se haga acreedor a multas y sanciones económicas por parte de la Secretaría de Comunicaciones y Transportes.

Una vez que se ha presentado de manera general la definición de la problemática, podemos establecer como enunciado definitivo del problema lo siguiente:

"Se requiere el desarrollo de un sistema automatizado montado sobre el NMA, que sea capaz de monitorear la hora de sistema de las centrales telefónicas de TELMEX, y de sincronizarlas dentro de un rango de +/- 2 segundos, tomando como referencia la hora del NMA".

4.3 Propuesta de Solución

Como alternativa de solución a esta problemática, proponemos el desarrollo de un sistema distribuido, con módulos de programa en los *mainframe* Stratus y en un servidor UNIX externo, los cuales se van a comunicar mediante *sockets* (puertos lógicos TCP/IP) en un esquema programación cliente-servidor, de tal manera que los programas cliente pueda pedir servicios a los programas servidor tales como el acceso a una central telefónica para tomar muestras de la hora de sistema y el envío de comandos de sincronización a través del NMA.

De manera más explícita, el servidor UNIX contará con un conjunto de programas con rutinas de comunicación TCP/IP, que le permitirán conectarse a los dos equipos Stratus a través de su correspondiente *socket* número 23 (*socket* por el cual responde el sistema operativo por *default*). Esto permitirá la posibilidad de poder enviar comandos específicos al NMA para activar la ejecución de los *scripts* que van a muestrear y finalmente ajustar la hora de sistema de la central telefónica. Todas las muestras de tiempo se van a registrar en un archivo de datos (uno por cada división geográfica) y luego de un análisis algorítmico de dichas muestras de tiempo, se determinará cuales centrales se encuentran fuera de sincronía(fuera del rango de +/- 2 segundos con respecto a la hora del NMA) y se enviarán a una pantalla de monitoreo en línea, donde el usuario podrá observar el comportamiento de las centrales pertenecientes a su división geográfica y eventualmente ejecutar el proceso de sincronización correspondiente, de acuerdo al tipo de central de que se trate.

La figura 4.2 nos ofrece un mejor panorama de la solución que estamos proponiendo, ya que presenta los módulos que van a integrar al sistema, resaltando la comunicación y flujo de información entre los mismos. En cada uno de los mainframe Stratus (México y Guadalajara) van a correr dos módulos: el módulo de muestreo y el módulo de construcción de mensajes. El primero, como su nombre lo dice, será el encargado de tomar muestras de tiempo de cada una de las centrales telefónicas de manera permanente y periódica (cada hora). El segundo, se encargará de dar forma a los mensajes que se harán llegar a las centrales telefónicas a través de un canal del NMA, con el fin de obtener muestras instantáneas de tiempo o de ajustar su hora de sistema.

El resto de los módulos del sistema van a residir en un servidor externo y cada uno de ellos representará un proceso distinto con una función específica. El primero de ellos es el módulo de recolección, cuya función será recibir por un socket todas las muestras de tiempo generadas por el módulo de muestreo y depositarlas en un archivo de datos a manera de registros. Este archivo de datos va a ser consultado por el módulo de reporteo cuando el usuario requiera un reporte específico y por el módulo de monitoreo en línea, el cual identificará las centrales con desviación de tiempo y las presentará al usuario en una pantalla. El módulo de comunicaciones será uno de los más importantes del sistema, ya que éste va a atender las peticiones de ejecución de comandos generadas por el módulo

de recolección (desbloqueo de puertos del NMA), y por los módulos de monitoreo y muestreo y sincronización manual. Finalmente, al módulo de administración permitirá el manejo de cuentas de usuario de manera que sean configurables por especialidad y con restricciones en cuanto a permisos de usuario.

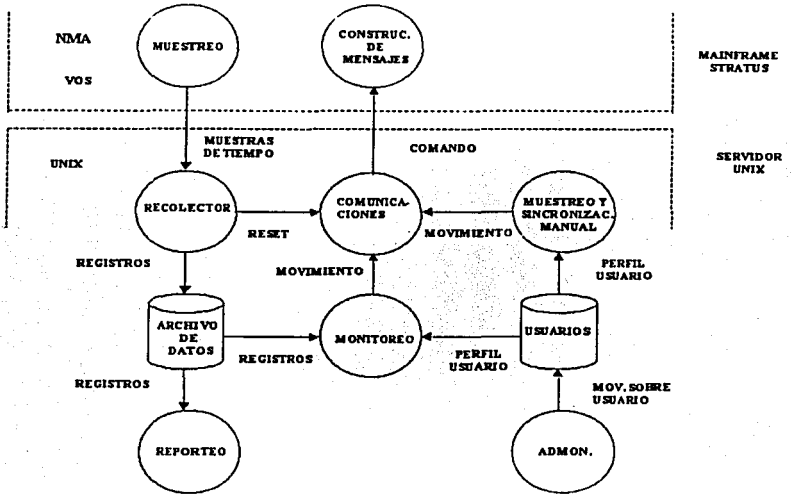


Fig. 4.2 Esquema de la solución propuesta.

De manera inicial, las pruebas preliminares del Sistema Automático de Sincronía de Tiempo se van a realizar sobre las centrales telefónicas de prueba (sin tráfico real), ubicadas en los laboratorios de TELMEX en la Ciudad de México, los cuales ofrecen la posibilidad de ejecutar cualquier prueba en condiciones que se acercan mucho a lo que es en realidad la planta telefónica.

4.3.1 Justificación de Plataformas de Hardware y Software

De acuerdo a las características de la solución que estamos proponiendo (un sistema distribuido con módulos ejecutándose en paralelo y comunicándose en un esquema cliente-servidor), la plataforma de *software* que más se adapta a nuestras necesidades es definitivamente el sistema operativo UNIX, ya que ofrece un ambiente multiusuario-multiproceso y una amplia variedad de herramientas útiles en el desarrollo de sistemas de *software* de este tipo.

La plataforma de *hardware* sobre la que vamos a trabajar consistirá de un servidor arquitectura RISC (*Reduced Instruction Set Computer*, Computadora de Conjunto de Instrucciones Reducido), ya que los equipos basados en esta arquitectura ofrecen una operación muy confiable y de gran desempeño. Para el caso específico de la aplicación que vamos a desarrollar, emplearemos un servidor HP-9000 modelo 827s con la versión 10.20 de sistema operativo HP-UX (nombre de la versión de UNIX propia de Hewlett Packard).

Al principio, las opciones de lenguajes de programación que teníamos para el desarrollo de este sistema eran BASIC, Pascal, Fortran y C. Requeríamos de un de un lenguaje de programación que reuniera ciertas características tales como la potencia, la portabilidad, la modularidad y la reutilización de código; y además, que contara con funciones de biblioteca para la programación de sockets. Definitivamente nuestra elección fue, el lenguaje C.

Las principales razones que nos llevaron a elegir lenguaje C fueron las siguientes:

- Fue el único lenguaje que cubría todas las características que buscábamos.
- Es el único lenguaje de programación que tiene cargado un compilador en los mainframe Stratus (aparte del compilador de PL/I, que es un lenguaje de programación poco conocido).
- La estrecha relación que tiene con el sistema operativo UNIX, sobre el cual residirán la mayor parte de los módulos del sistema.
- Es el lenguaje de programación sobre el cual el grupo de desarrollo participante tiene más conocimiento y experiencia.

En resumen, todos los módulos de programa van a ser escritos en lenguaje C (salvo los *scripts* que conformarán el módulo de muestreo), uno de ellos (módulo de construcción de mensajes) bajo sistema operativo VOS, ya que va a residir en los mainframe Stratus; y los otros seis restantes, bajo sistema operativo UNIX, ejecutándose en el servidor externo HP-9000.

4.3.2 Detalles de la Fuente de Sincronía

Como se puede observar en la figura 4.3, para efectos de sincronización, se tomará como hora de referencia, la hora de los equipos Stratus, ya que estos se encuentran sincronizados por *software*, a través del protocolo estándar de sincronización empleado a nivel mundial NTP (NTP: *Network Time Protocol*, Protocolo de Sincronización de Tiempo), con un servidor de tiempo ubicado en la ciudad de Tijuana, B.C. Este servidor de tiempo está catalogado en la categoría *stratum* 1, ya que a su vez se encuentra sincronizado con un servidor *stratum* 0 de la armada de los Estados Unidos a través de la red Internet. Este último servidor está catalogado como *stratum* 0 ya que se encuentra directamente en sincronía con el CUT (*Coordinated Universal Time*, Tiempo Coordinado Universal) a través de un receptor GPS (*General Positioning Service*, Servicio de Posicionamiento General), el cual permite tener una conexión directa por satélite. De esta manera se garantiza que la hora de los *mainframe* Stratus será confiable para tomarla como hora de referencia central para sincronizar los relojes de sistema de las centrales telefónicas.

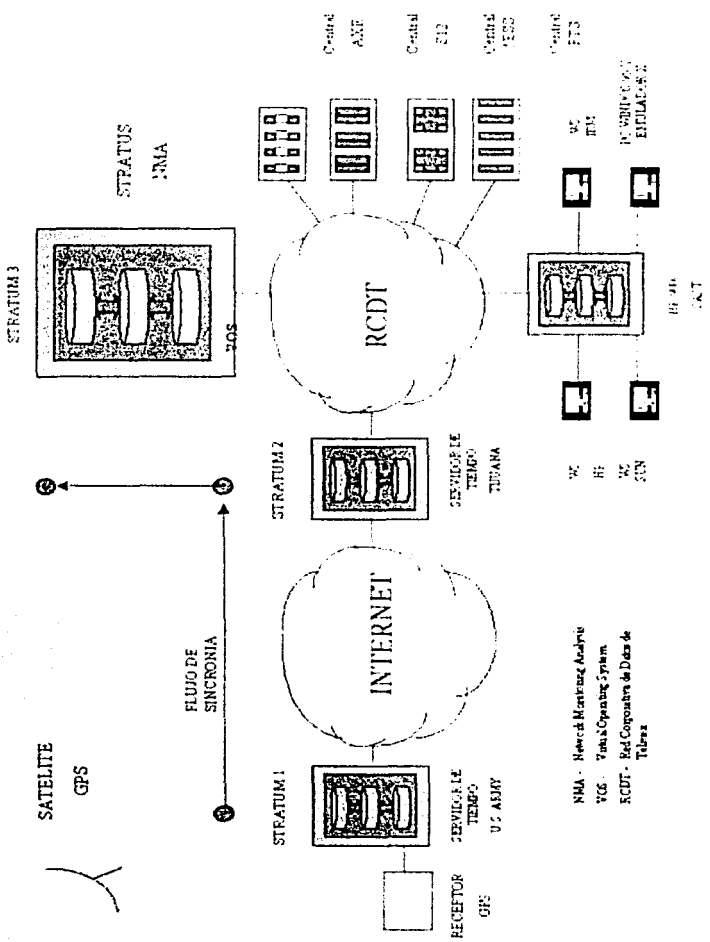


Fig. 4.3 Diagrama de gestión y flujo de sincronía.

De manera inicial, las pruebas preliminares del Sistema Automático de Sincronía de Tiempo se van a realizar sobre las centrales telefónicas de prueba (sin tráfico real), ubicadas en los laboratorios de TELMEX en la Ciudad de México. Estos laboratorios cuentan con todas las condiciones con las que cuenta la planta telefónica; es decir, una central telefónica de prueba de cada tipo de tecnología (AXE, S-12, 5ESS, PTS, etc) con las últimas versiones de *software* instaladas en la planta y equipadas con todos sus módulos de *hardware*, incluyendo los módulos de recolección de facturación.

Posteriormente, una vez que se haya observado el comportamiento del sistema en ambiente de laboratorio, se realizará una serie de pruebas piloto en centrales con tráfico real, con la presencia de personal responsable de la misma. Estas pruebas piloto se realizarán para cada uno de los tipos de centrales que va a contemplar el SAST. En el caso de que los resultados de este conjunto de pruebas haya sido satisfactorio, el siguiente paso será la implantación gradual del sistema en cada una de las divisiones geográficas en las que se encuentra dividida la planta telefónica, bajo un esquema de supervisión y monitoreo constante que permita prevenir cualquier imprevisto que pudiera afectar a la operación de las centrales.

4.4 Especificación Funcional del Sistema

Con la finalidad objeto de que el objetivo, los requerimientos y funcionalidades que habrá de cubrir el sistema queden asentados de una manera clara y precisa, desarrollamos la siguiente especificación funcional del sistema.

4.4.1 Objetivo

Desarrollo de un sistema automatizado montado sobre el NMA, que sea capaz de monitorear la hora de sistema de las centrales telefónicas de TELMEX, y de sincronizarlas dentro de un rango de +/- 2 segundos, tomando como referencia la hora del NMA.

4.4.2 Alcance

El sistema será implantado en el Centro Nacional de Supervisión (CNS) y en todos los Centros de Administración de la Red (CAR), en donde los técnicos especialistas serán los usuarios directos del mismo.

4.4.3 Restricciones

- La sincronización de las centrales PTS a través de este sistema no será factible, ya que estas no cuentan con el comando de sistema necesario para ajustar la hora empleando incrementos de tiempo.
- El muestreo y sincronización de centrales no conectadas al NMA no será factible.
- El muestreo y sincronización de centrales con versiones de *software* obsoletas, no será factible.
- Se tomará +/- 2 segundos como una desviación de tiempo aceptable.

4.4.4 Objetivos Generales

- Contribuir en gran medida a la solución de la problemática de errores en tiempos de facturación, atribuibles a deficiencias en la sincronía entre las centrales de conmutación.

- Contribuir en gran medida a la solución de la problemática de errores en tiempos de interconexión con otras empresas que ofrecen servicios de telefonía, atribuibles a problemas de sincronía.

- Evitar en la medida de lo posible, la adquisición e instalación de módulos de *hardware* y *software* de alto costo en cada una de las centrales telefónicas para su sincronización con un equipo servidor de tiempo.

4.4.5 Funcionalidades

El sistema deberá ofrecer al usuario las siguientes funcionalidades:

a) El usuario podrá accesar desde una estación de trabajo UNIX o desde una computadora personal con *software* de emulación de terminal.

b) Deberá tener un módulo para la creación y administración de usuarios del sistema.

c) Deberá contar con un esquema mínimo de seguridad; es decir, cada usuario deberá tener acceso mediante una clave de acceso y contraseña. Esta clave de acceso deberá de contar con la posibilidad de restringir al usuario por tipo de especialidad (AXE, S-12, 5ESS, PTS) y por tipo de movimiento (muestreo de la hora o sincronización).

d) Deberá de muestrear la hora de las centrales de manera periódica por medio de calendarizaciones en el NMA, de tal modo que en todo momento se cuente con información confiable.

e) Deberá de depositar todos las muestras de tiempo de las centrales en una base de datos local, para ser procesada posteriormente.

f) El sistema deberá generar los siguientes reportes:

- Reporte de Centrales en Sincronía
  Donde se listen todas las centrales cuyo reloj de sistema se encuentre dentro del rango de +/- 2 segundos de desviación con respecto a la hora de referencia.

- Reporte de Centrales Fuera de Sincronía
  Donde se listen todas las centrales cuyo reloj de sistema se encuentre fuera del rango de +/- 2 segundos de desviación con respecto a la hora de referencia.

- Eventos por Central y Fecha
  Donde se listen todos los movimientos realizados en una determinada central y en una fecha específica.

- Centrales con Muestreo
  Donde se listen todas las centrales de las cuales se cuente con muestras de tiempo recientes.

- Centrales sin Muestreo
  Donde se listen todas las centrales de las cuales no se tienen muestras de tiempo recientes.

- Bitácora de Movimientos
  Donde se listen todos los detalles de los movimientos realizados por los usuarios sobre el sistema.

g) Deberá contar con una pantalla de monitoreo en línea, donde podrá observarse una lista de centrales fuera de sincronía en tiempo real.

h) Generación automática de estadísticas de tiempo, con los parámetros que el sistema pone por default.

i) Generación manual de estadísticas de tiempo, con los parámetros que el usuario pone a su consideración.

j) Sincronización automática de centrales, con los parámetros que el sistema pone por default.

k) Sincronización manual de centrales, con los parámetros que el usuario pone a su consideración.

En el siguiente capítulo nos introduciremos al diseño del sistema, en base a el análisis que ya hemos desarrollado.

V. DISEÑO DEL SAST

En este capítulo presentaremos un diseño detallado de los diferentes módulos que habrán de conformar el Sistema Automático de Sincronía de Tiempo, resaltando su funcionalidad específica y las interrelaciones entre cada uno de ellos.

5.1 Definición de los Módulos

Como ya se había planteado, la solución propuesta para nuestra problemática es el desarrollo de un sistema distribuido, con módulos de programa corriendo tanto en el *mainframe Stratus* (sistema operativo VOS) como en un servidor UNIX externo al NMA. La comunicación entre estos módulos se llevará a cabo sobre un esquema cliente-servidor, empleando *sockets* orientados a conexión.

El Sistema Automático de Sincronía de Tiempo se va a conformar de los siguientes módulos de programa, cada uno de ellos con una función específica:

a) Módulo de Muestreo
b) Módulo de Recolección
c) Módulo de Monitoreo en Línea
d) Módulo de Muestreo y Sincronización Manual
e) Módulo de Reporteo
f) Módulo de Comunicaciones
g) Módulo de Construcción de Mensajes
h) Módulo de Administración

La figura 5.1 muestra los distintos módulos del sistema y la interrelación entre los mismos.

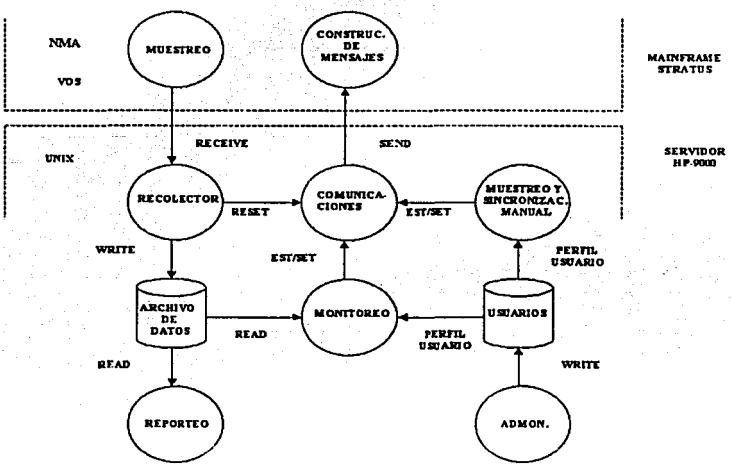


Fig. 5.1 Interrelación Modular del Sistema.

Las características de cada uno de estos módulos se presenta a continuación.

5.1.1 Módulo de Muestreo

Este módulo correrá en el *mainframe Stratus* y, su función principal será obtener muestras periódicas de la hora de sistema de cada una de las centrales de conmutación, para luego conformar los registros de información correspondientes en un formato específico, y direccionarlos a través de un *socket* o puerto lógico TCP/IP del NMA, de tal manera que puedan ser leídos por otros procesos a través de la red (En este caso, por el módulo de recolección).

El modulo de muestreo estará conformado por 2 programas distintos (uno para las tecnologías de central AXE, S-12 y 5ESS, y otro para las centrales PTS) desarrollados sobre el NMA, y escritos en un lenguaje de programación de propósito específico. Estos programas, denominados *scripts*, estarán calendarizados en el NMA, de tal forma que se ejecutarán cada hora sobre cada una de las centrales, tomando una muestra instantánea de tiempo, la cual se concatenará con la hora de referencia para conformar un registro que finalmente se direccionará por un *socket*.

La siguiente trama sirve para ejemplificar la forma en que este módulo escribirá los datos sobre el *socket:*

```
*
cdmxdfct13t
000203
12:42:28
12:42:31
3
NUL
AXE
#
*
cdmxdfct30w
000203
12:42:34
12:42:36
2
NUL
PTS
#
*
cdmxdfct31w
000203
12:42:46
12:42:47
1
NUL
PTS
#
```

Esta trama de información que fluye por el *socket* no es otra cosa que una serie de cadenas de caracteres escritas de tal forma que sea entendible para el módulo de recolección. Los caracteres "*" y "#" sirven para definir respectivamente donde inicia y donde termina un registro. Cada registro estará integrado por siete campos, cada uno de los cuales terminará con un carácter de salto de línea ("\n").

Pero, para entender mejor la estructura de estos registros de información, analizaremos a continuación uno de estos, presentando un formato tipo, donde se pueden observar los diferentes elementos que integrarán estos registros:

*	->	Carácter indicador de inicio de registro.
cdmxdfct31w	->	Clave de identificación de la central en el NMA (cllcode).
000203	->	Fecha de generación de la muestra en formato AAMMDD.
12:42:46	->	Hora de referencia en formato HH:MM:SS.
12:42:47	->	Hora instantánea de la central en formato HH:MM:SS.
1	->	Diferencia de tiempo en segundos calculada por el script.
NUL	->	Campo de relleno. Futura implementación.
PTS	->	Tipo de tecnología de la central (AXE, S-12, 5ESS, PTS).
#	->	Carácter indicador de final de registro.

5.1.2 Módulo de Recolección

Este módulo será el encargado de conectarse al *socket* arriba mencionado, con el objeto de leer cada uno de los registros de tiempo generados por el módulo de muestreo y depositarlos en un archivo de datos en el servidor UNIX, el cual estará disponible para realizar un procesamiento posterior del mismo.

El módulo de recolección escribirá los registros capturados por el *socket* en el archivo de datos centralizado, conservando la disposición de los campos y empleando una "," como separador. La trama que ejemplificamos en el módulo de muestreo quedaría de la siguiente forma:

```
cdmxdfct13t,000203,12:42:28,12:42:31,3,NUL,AXE
cdmxdfct30w,000203,12:42:34,12:42:36,2,NUL,PTS
cdmxdfct31w,000203,12:42:46,12:42:47,1,NUL,PTS
```

Por razones prácticas, existirá un archivo de datos por cada una de las divisiones que atiende el NMA. De tal forma que, cuando llega un nuevo registro de información, el módulo de recolección determinará en base al cllcode, a que división pertenece la central en cuestión y escribirá el registro en el archivo de datos correspondiente.

La figura 5.2 muestra esquemáticamente la forma en que el módulo de recolección recibe los registros provenientes del módulo de muestreo y los deposita en su correspondiente archivo de datos.

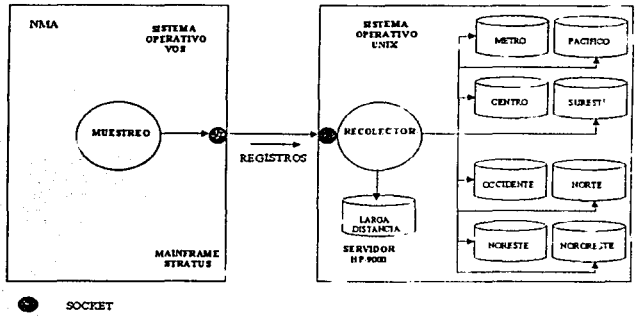


Fig. 5.2 Proceso de recolección de registros.

5.1.3 Módulo de Monitoreo

Este módulo, en base a un análisis de las muestras de tiempo (que consiste en la eliminación de tiempos pico y cálculo de la desviación de tiempo), contenidas en el archivo de datos correspondiente generado por el módulo de recolección, determinará cuales son las centrales que se encuentran fuera de sincronía; es decir, fuera del intervalo de +/-2 segundos con respecto a la hora de referencia, y las mostrará al usuario en una pantalla de monitoreo en tiempo real, con un tiempo de refresco de 10 segundos. Además, este módulo deberá proporcionar al usuario la posibilidad de poder generar muestras instantáneas de tiempo y de ejecutar el proceso de sincronización sobre cualquiera de las centrales que se muestran en esta pantalla de monitoreo.

Para tal fin, este programa enviará los comandos de muestreo y de sincronización al módulo de comunicaciones, quien será el encargado de enviarlos hacia el NMA para su inmediata ejecución.

5.1.4 Módulo de Muestreo y Sincronización Manual

A través de este módulo será posible que el usuario pueda generar muestras de tiempo y ejecutar el proceso de sincronización sobre cualquier central telefónica, proporcionando todos los parámetros necesarios de manera manual (cllcode, tecnología de central, canal virtual del NMA, número de muestras, desviación de tiempo, etc); es decir, sin tomar en cuenta los datos sugeridos por el sistema. Este procedimiento manual sobre las centrales telefónicas será perfectamente válido, ya que existe personal con tal grado de experiencia y conocimiento que no implicaría ningún problema para ellos.

De cualquier forma, cada uno de estos usuarios estará restringido en cuanto a permisos y catalogado por perfil de usuario, con el objeto de prevenir cualquier movimiento no autorizado.

Una vez que el usuario ha terminado de capturar todos los datos necesarios para la ejecución de un muestreo de tiempo o un ajuste de tiempo, el módulo de muestreo y sincronización manual los integra en la estructura movimiento y lo despacha por un *socket* hacia el módulo de comunicaciones, el cual a su vez lo envía por otro *socket* para su ejecución final en el NMA.

5.1.5 Módulo de Reporteo

Este módulo se conformará de las rutinas necesarias para la generación de distintos reportes en base al archivo de datos centralizado. Los reportes con los que podrá contar el usuario serán los siguientes:

a) Reporte de Centrales en Sincronía
b) Reporte de Centrales Fuera de Sincronía
c) Reporte de Eventos por Central y por Fecha
d) Reporte de Centrales con Muestreo Reciente
e) Reporte de Centrales sin Muestreo Reciente
f) Bitácora de Movimientos del Sistema

Todos los reportes se van a generar sobre el archivo de datos de la división a la cual pertenece el usuario, de tal manera que un usuario de la división metro podrá acceder a la información relativa a las centrales del área metropolitana, pero no podrá tener acceso a las centrales de la división occidente, pacífico, etc.

5.1.6 Módulo de Comunicaciones

Su función fundamental será servir como una interface de comunicación entre los módulos que se ejecutarán en el servidor UNIX y los que se ejecutarán en el *mainframe Stratus*. Se tratará de un programa servidor de sesiones TCP/IP, a través del cual se podrán enviar los comandos y mensajes necesarios al NMA para muestrear y/o ajustar la hora de sistema de determinada central telefónica. Este programa tendrá la capacidad de sostener varias sesiones de comunicación de manera simultánea, de interpretar la respuesta a cada comando ejecutado, y de enviar los resultados y códigos de error al programa cliente.

Este programa contará con dos *sockets*, uno para establecer la conexión con los programas cliente (*socket* de servicio) y otro para establecer la conexión con el NMA (*socket* de ejecución). Su ejecución será de manera permanente en el servidor UNIX (ejecución como demonio del sistema) y se mantendrá todo el tiempo en modo de escucha (***accept***); en el momento que un cliente haga una petición de conexión (***connect***), la sesión quedará establecida. Cuando el programa cliente quiera ejecutar un movimiento (***est*** o ***set***), el programa servidor habilitará la comunicación hacia el NMA por el *socket* de ejecución (hacia el puerto número 23 del *mainframe Stratus*), para poder enviar el comando a ejecución y obtener los resultados.

Los programas cliente que podrán conectarse al *socket* de servicio del módulo de comunicaciones, para poder enviar comandos al NMA, serán el módulo de muestreo y sincronización manual y el módulo de monitoreo.

Antes de poder enviar cualquier comando a ejecución, el módulo de comunicaciones debe de pasar ciertos parámetros al módulo de construcción de mensajes (residente en el *mainframe Stratus*), de manera que éste pueda conformar el mensaje completo, que habrá de servir como parámetro para la ejecución efectiva de cada comando de sistema.

La figura 5.3 muestra un mejor panorama de cómo se realiza la comunicación entre los módulos antes mencionados.

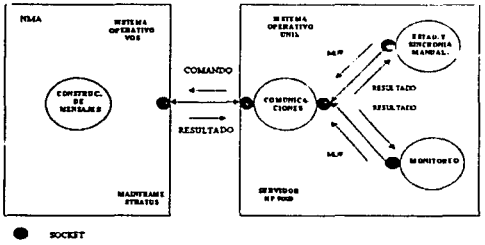


Fig. 5.3 Proceso de envío de comandos hacia el NMA.

5.1.7 Módulo de Construcción de Mensajes

Se trata de un pequeño programa en lenguaje C, que residirá en el *mainframe Stratus*, bajo sistema operativo VOS, y cuya función será recibir por un *socket* los datos necesarios para construir mensajes específicos en el NMA (sobre archivo de texto), los cuales servirán posteriormente como parámetro necesario para la ejecución de los comandos.

Estos datos serán proporcionados por el programa cliente (Módulo de Muestreo y Sincronización Manual o Módulo de Monitoreo, según sea el caso) a través del módulo de comunicaciones. Estos datos serán los siguientes:

a) Mensaje: Es el nombre del Script que habrá de ejecutarse según el tipo de tecnología de la central (**reloj2** para AXE, S-12, 5ESS y **show_time** para PTS).
b) Movimiento: Es el tipo de movimiento que se va a ejecutar sobre la hora de sistema de la central telefónica en cuestión (est: muestreo de tiempo, set: ajuste o sincronización).
c) Número: Este campo se empleará de dos maneras distintas según el tipo de movimiento que se ejecute. Cuando el tipo de movimiento es muestreo de tiempo (est), este campo contendrá el número de muestras consecutivas que se tomarán de la hora de la central, y cuando el tipo de movimiento es sincronización de tiempo (set),

este campo representará el incremento de tiempo (negativo o positivo) con el cual se ajustará la hora de la central.

d) Tecnología: Indicará el tipo de tecnología al que corresponde la central telefónica (AXE, S-12, 5ESS ó PTS). Este campo es importante ya que los mensajes del NMA para cada una de estas tecnologías difieren uno de otro.

e) Archivo: Denota el nombre del archivo en el cual se depositará el mensaje ya construido. Estos mensajes no son otra cosa que una cadena de caracteres que incluye los datos necesarios para la ejecución de un movimiento.

Para ejemplificar la forma en que van a estar estructurados estos mensajes, a continuación mostraremos un ejemplo de ellos por cada tipo de tecnología:

a) AXE:

Muestreo:	reloj2 est 10 `03<	->	Tomar 10 muestras de tiempo
Sincronización:	reloj2 set 3 `03<	->	Adelantar la hora 3 segundos

b) S-12:

Muestreo:	reloj2 est 20 `03	->	Tomar 20 muestras de tiempo
Sincronización:	reloj2 set -3 `03	->	Retrasar la hora 3 segundos

c) 5ESS:

Muestreo:	reloj2 est 5	->	Tomar 5 muestras de tiempo
Sincronización:	reloj2 set 7	->	Adelantar la hora 7 segundos

d) PTS:

Muestreo:	show_time set 5	->	Tomar 5 muestras de tiempo
Sincronización:	< No se contará con la capacidad de sincronización >		

En el ejemplo anterior, se puede notar que los mensajes difieren muy poco entre sí, sobre todo en los caracteres de terminación. Mientras que los mensajes para AXE requieren un número tres hexadecimal y un carácter "<" como cadena de terminación, los mensajes de S-12 sólo requieren el número hexadecimal. Los mensajes para 5ESS y PTS no requieren terminadores.

5.1.8 Módulo de Administración

Por medio de este programa, un usuario con cuenta de administrador podrá agregar, eliminar, modificar y visualizar cuentas de usuario del sistema. Cada cuenta de usuario contará con un *login* y un *password* para poder tener acceso y podrán personalizarse de manera que los usuarios podrán estar restringidos por tipo de tecnología de central (AXE, S-12, 5ESS, PTS) y por tipo de movimientos (est, estg, set, setg, estxd, etc.) realizados en las centrales.

Lo anterior quiere decir que un especialista en centrales AXE normalmente no podrá realizar movimientos en centrales S-12; y aún teniendo perfil de AXE, no podrá ejecutar comandos de ajuste de la hora si no cuenta con el permiso set. Sin embargo, esto no

quiere decir que un usuario no pueda contar con dos o más perfiles con distintos permisos de usuario para cada uno de ellos.

Todos los datos relativos a las cuentas de usuario se escribirán en un archivo de configuración de sistema al cual sólo los usuarios con categoría de administrador podrán tener acceso.

5.2 Estructura de Datos

Las principales estructuras de datos que se emplearán para implementar un manejo efectivo de la información entre los distintos módulos de programa, son las siguientes:

5.2.1 Estructura InfoReg

Esta será la estructura de datos principal del sistema, ya que se va a utilizar para dar forma a todas las muestras de tiempo tomadas de las centrales telefónicas. Los registros de información van a ser creados bajo esta estructura en el momento que el módulo de muestreo toma las muestras de tiempo de cada central y las direcciona por un *socket*, en donde el módulo de recolección las captura y los deposita en el archivo de datos correspondiente, de acuerdo a la división geográfica a la cual corresponda. Los módulos de monitoreo y de reporteo también harán uso de esta estructura, ya que las operaciones que realicen se ejecutarán sobre el archivo de datos centralizado.

La estructura es la siguiente:

```
#define L_CLLCODE     12        /* Longitud del campo Cllcode */
#define L_FECHA        7        /* Longitud del campo Fecha */
#define L_HORA         9        /* Longitud de los campos HoraRef y HoraCent */
#define L_DIF         10        /* Longitud del campo Dif */
#define L_MSG         15        /* Longitud del campo Msg */
#define L_TIPO        15        /* Longitud del campoTipo */
#define RANGO         20        /* Longitud del rango de holgura */

typedef struct {

  char Cllcode[L_CLLCODE+RANGO];      /* Clave única de la central telefónica */
  char Fecha[L_FECHA+RANGO];          /* Fecha de muestreo de tiempo */
  char HoraRef[L_HORA+RANGO];         /* Hora de referencia */
  char HoraCent[L_HORA+RANGO];        /* Hora de la central telefónica */
  char Dif[L_DIF+RANGO];              /* Dif. de tiempo (HoraRef-HoraCent) */
  char Msg[L_MSG+RANGO];              /* Campo no utilizado */
  char Tipo[L_TIPO+RANGO];            /* Tipo de tecnología de central */

} InfoReg;
```

5.2.2 Estructuras de Datos de Comunicación

Estas estructuras están diseñadas para contener los datos necesarios para establecer una comunicación entre procesos distintos a través de un *socket* y para contener las peticiones de los usuarios y las respuestas que da el sistema a las mismas.

- Estructura user

Para generar una conexión al *mainframe Stratus* para el envío de comandos por el *socket* número 23, es necesario suministrar ciertos datos como son la dirección IP del equipo, el *login* y el *password*. Para contener estos datos se emplea la siguiente estructura:

```
#define BUFSIZE   500        /* Longitud de buffer */

typedef struct {

   char   login_vos[BUFSIZE]      ;        /* Login de usuario  */
   char   password_vos[BUFSIZE] ;          /* Password de usuario */
   char   ipaddressvos[BUFSIZE]  ;         /* Dirección IP del mainframe Stratus */

} user ;
```

- Estructura petición

Todas las peticiones de movimientos generadas al módulo de comunicaciones deberán cumplir con un formato predefinido. La estructura de datos para estas peticiones es la siguiente:

```
typedef struct {

   char   no_peticion[3]          ;       /* Número consecutivo de petición */
   char   servicio[3]             ;       /* "00" muestreo, "01" ajuste, "02" desbloqueo */
   char   comando[ BUFSIZE ] ;            /* Comando a ser ejecutado */

} petición ;
```

- Estructura respuesta

El módulo de comunicaciones devolverá los resultados de la ejecución de cada comando empleando la siguiente estructura:

```
typedef struct {

   char   no_peticion[3]          ;       /* Número consecutivo de petición */
   char   error                   ;       /* '0' Ok, '1' error */
   char   descripcion[BUFSIZE] ;          /* Descripción del error */

} respuesta ;
```

5.2.3 Estructura de Movimientos

Esta estructura se empleará para concentrar todos los datos necesarios para la ejecución de un movimiento sobre una central telefónica a través de comandos del NMA. Esta estructura se muestra a continuación:

```
typedef struct {

   char    clicode[20];              /* Clave única de identificación de la central */
   char    canal[20];                /* Canal de comunicación del NMA hacia la central */
```

```
    char    tec[20];                /* Tipo de tecnología de la central */
    char    tipo_mov[20];           /* Tipo de movimiento (est o set) */
    char    msg_script[20];         /* Nombre del script a ejecutar */
    char    archivo_msg[20];        /* Archivo en el cual se escribirá el mensaje completo */
    int     num_estad;              /* Número de muestras de tiempo a tomar */
    int     delta;                  /* Desviación de tiempo */
    int     tipo_ejec;              /* Tipo de ejecución (manual o automática) */

} movimiento;
```

5.2.4 Estructura de Configuración

La siguiente estructura de datos va a ser empleada para contener todos los datos relativos a la configuración del sistema.

```
#define _R_ 100

struct SAST {

    char login_vos [ _R_ ]      ;       /* Login de VOS   */
    char password_vos [ _R_ ] ;         /* Password de VOS */
    char dir_ip [ _R_ ]         ;       /* Dir IP a NMA   */
    char pto [ _R_ ]            ;       /* Pto lógico  NMA */
    int  factor_str             ;       /* Factor corrección de la Stratus. */
    char div [ _R_ ]            ;       /* Nombre de la División          */
    int  ocurrencia             ;       /* # veces que ocurre una medición */
    char tabla_div[80]          ;       /* Nom. tabla de centrales de Div. */
    char nma[20]                ;       /* NMA que atiende a la División.  */
    char canal[20]              ;       /* Canal del NMA de la División.   */
} S ;
```

5.2.5 Estructura de Usuario del Sistema

Este Estructura va a ser empleada para contener todos los datos relativos a una cuenta de usuario del sistema.

```
typedef struct {

  char login[15];
  char password[15];
  char templateAXE[30];
  char templateS12[30];
  char template5ESS[30];
  char templatePTS[30];

} SAST_USER;
```

5.3 Diseño de la Interfaz de Usuario

Como es común en las aplicaciones desarrolladas sobre plataformas de sistema operativo basadas en UNIX, la interfaz de usuario del Sistema Automático de Sincronía de Tiempo (SAST) estará basada en menús y pantallas en modo texto, aunque aprovechando las facilidades que ofrece el CDE (DCE: *Common Desktop Environment*, Ambiente de Escritorio Común), el cual es un estándar con el que cuentan los fabricantes de equipo más importantes, como lo son Hewlett Packard, IBM y Sun Microsystems entre otros. El

CDE es un ambiente gráfico multiventanas que ofrece a los usuarios la posibilidad de contar con cuatro áreas de trabajo, de contar con un sistema de impresión, con una herramienta gráfica de administración de archivos y con íconos especiales para la hora de sistema, calendario y bloqueo de terminal.

Sobre este ambiente reposará la interfaz de usuario de nuestro sistema; de acuerdo a la figura 5.4;que se encuentra en la página siguiente, la cual estará diseñada con las siguientes etapas:

a) Acceso.
b) Menú Principal.
c) Reportes Estadísticos.
d) Monitoreo en Línea.
e) Sincronización por Central.
f) Estadísticas por Central.
g) Administración de Usuarios.

- Acceso.- La función principal de esta pantalla será la de proporcionar un grado de seguridad en el Acceso al Sistema, para controlar los usuarios que la utilizan, a través de una Clave de Acceso, como se muestra en la figura 5.5.

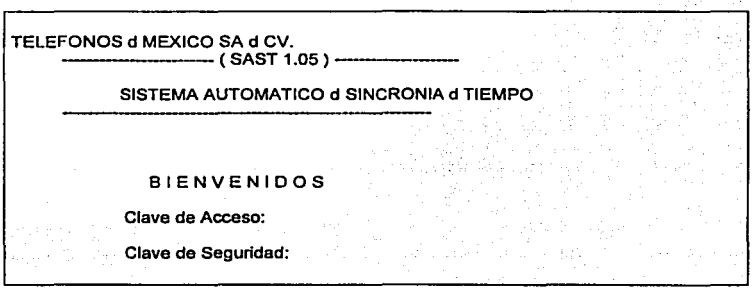


Fig. 5.5 Pantalla de Acceso.

CDE es un ambiente gráfico multiventanas que ofrece a los usuarios la posibilidad de contar con cuatro áreas de trabajo, de contar con un sistema de impresión, con una herramienta gráfica de administración de archivos y con íconos especiales para la hora de sistema, calendario y bloqueo de terminal.

Sobre este ambiente reposará la interfaz de usuario de nuestro sistema; de acuerdo a la figura 5.4;que se encuentra en la página siguiente, la cual estará diseñada con las siguientes etapas:

a) Acceso.
b) Menú Principal.
c) Reportes Estadísticos.
d) Monitoreo en Línea.
e) Sincronización por Central.
f) Estadísticas por Central.
g) Administración de Usuarios.

- Acceso.- La función principal de esta pantalla será la de proporcionar un grado de seguridad en el Acceso al Sistema, para controlar los usuarios que la utilizan, a través de una Clave de Acceso, como se muestra en la figura 5.5.

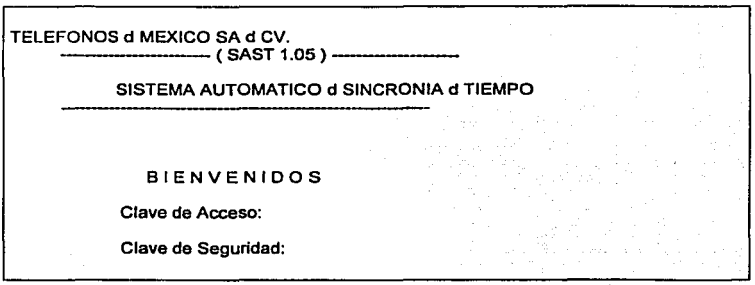


Fig. 5.5 Pantalla de Acceso.

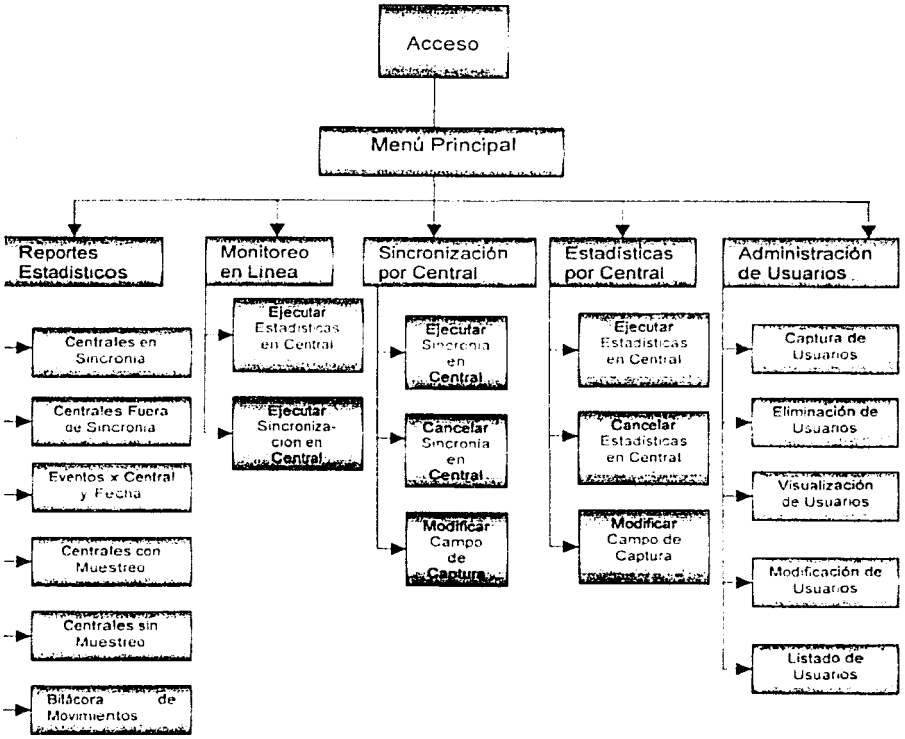


Fig. 5.4 Diseño de la Interfaz de usuario.

- Menú Principal.- En esta pantalla mostraremos todas las opciones que el cliente o usuario tiene para manipular el sistema, como se muestra en la figura 5.6.

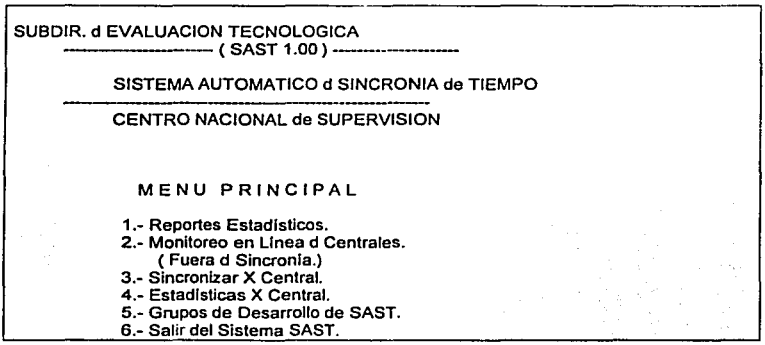


Fig. 5.6 Pantalla para el Menú Principal.

- Reportes Estadísticos.- Se trata de un submenú del menú principal. En este se presentarán al usuario todas las opciones de generación de reportes con que contará el sistema como se muestra en la figura 5.7.

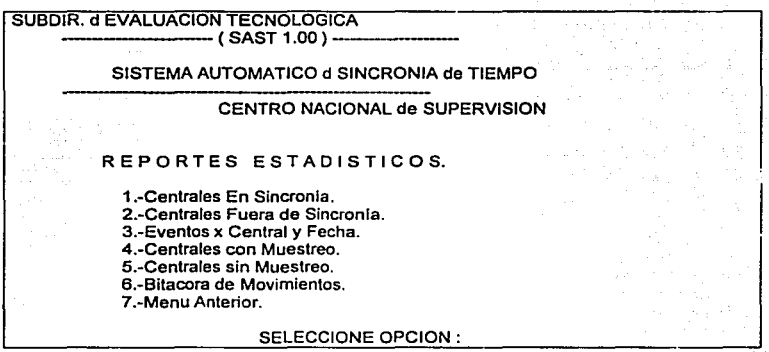


Fig. 5.7 Reportes Estadísticos.

a) Centrales en Sincronía: Esta será la disposición en la cual se desplegará al usuario la lista de Centrales en Sincronía al momento de generar el reporte,

(SISTEMA AUTOMATICO de SINCRONIA del TIEMPO.)

LABORATORIO d DESARROLLO d SW

TIPO DE REPORTE: Centrales en Sincronía

Código de la Central	Tipo de Central	Fecha de la Central	Hora de la Central	Hora del Mainframe Stratus	Tiempo de Desviación en segundos
Código1	XXX	000418	19:21:51	19:21:48	-3
Código2	YYY	000418	19:22:41	19:22:52	11

Fig. 5.8 Centrales en Sincronía.

b) Centrales Fuera de Sincronía: Esta será la disposición en la cual se desplegará al usuario la lista de centrales fuera de sincronía al momento de generar el reporte.

(SISTEMA AUTOMATICO de SINCRONIA del TIEMPO.)

LABORATORIO d DESARROLLO d SW

TIPO DE REPORTE: Centrales Fuera de Sincronía

Código de la Central	Tipo de Central	Fecha de la Central	Hora de la Central	Hora del Mainframe Stratus	Tiempo de Desviación en segundos
Código1	XXX	000418	19:21:51	19:21:48	-3
Código2	YYY	000418	19:22:41	19:22:52	11

Fig. 5.9 Centrales Fuera de Sincronía.

c) Eventos por Central y Fecha: En esta pantalla se van a listar todos los registros ocurridos en una determinada central telefónica y en determinada fecha.

(SISTEMA AUTOMATICO de SINCRONIA del TIEMPO.)

LABORATORIO d DESARROLLO d SW

TIPO DE REPORTE: POR CENTRAL Y FECHA

Código de la Central	Tipo de Central	Fecha de la Central	Hora de la Central	Hora del Mainframe Stratus	Tiempo de Desviación en segundos
Código1	XXX	000418	19:21:51	19:21:48	-3
Código2	YYY	000418	19:22:41	19:22:52	11

Fig. 5.10 Eventos x Central y Fecha.

d) Centrales con Muestreo: Aquí se mostrarán todas las centrales con registros de tiempo recientes.

SISTEMA AUTOMATICO d SINCRONIA de TIEMPO.

REPORTE: Ultimo registro por Fecha/Hora X Ctl del 000420.

	Número Central	Código de la Central	Fecha	Hora Ult. Reg.
:	1	Código1	000420	00:22:21
:	2	Código2	000420	00:22:20

Fig. 5.11 Centrales con Muestreo.

e) Centrales sin Muestreo: Aquí se mostrarán todas las centrales sin registros de tiempo recientes.

SISTEMA AUTOMATICOA d SINCRONIA de TIEMPO.

REPORTE: (Centrales sin Muestras en la Fecha 000420.)

#Reg	Código de la Central	Nombre de la Central	Dirección	Tecnología
1	CRNVXOBO22T	BORDA -CI/CTI	CUERNAVACA	AXE

Fig. 5.12Centrales sin Muestreo.

f) Bitácora de Movimientos: De esta forma se mostrarán los detalles de todos los movimientos realizados por los usuarios a través del sistema.

```
Usuario Conectado: rr9800891

      password: ********
   Dir. Stratus: 13.44.2.9  (Dirección IP de la mainframe desde donde
                             Se realizo el campo)
Central:          Código de la Central
Tecnologia:       Tipo de Tecnologia (AXE, 5ESS, S12..)
Tipo d Movimiento: est
Num. Estadisticas: 5
```

Fig. 5.13 Bitácora de Movimientos.

- Monitoreo en línea de centrales.- Esta será la forma en que se van a desplegar las centrales que se encuentren fuera de sincronía, en tiempo real y con un tiempo de refrescamiento de 10 segundos (figura 5.14). Permitiendo hacer un *scroll* sobre la misma e incluso se podrán generar peticiones de muestreo y de sincronización desde este punto.

```
MONITOR en TIEMPO REAL d CENTRALES FUERA d SINCRONIA

                              CENTRO NACIONAL de SUPERVISION

-------------------------------------------------------------------------------


No.  Código de  Tipo    Fecha   Hora     Hora del   Tiempo de
Reg. la Central de      de la   de la    Mainframe  Desviación
             Central    Central Central  Stratus    en segundos

0001 Código1    XXX     000418  19:21:51 19:21:48   -3
0002 Código2    YYY     000418  19:22:41 19:22:52   11


PAGINA [ Sig,Ant,Ini,Fin ]  Regresa  Estadisticas  siNcrononizar  Op?
```

Fig. 5.14 Monitoreo en línea de centrales.

- Sincronización por Central.- Esta será la pantalla en la cual el usuario capturará los datos necesarios para la sincronización manual del tiempo (figura 5.15) de una central determinada.

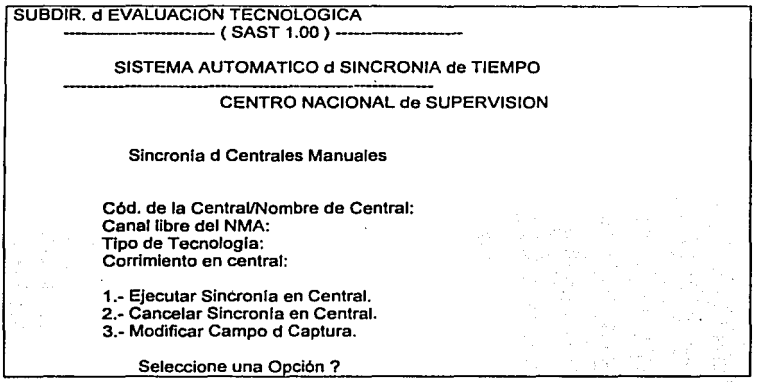


Fig. 5.15 Sincronización por Central.

a) Ejecutar Sincronía en Central: Actualizará el reporte de monitoreo en línea y reportes de centrales fuera de sincronía.

b) Cancelar Sincronía en Central: Suprime la ejecución del movimiento.

c) Modificar Campo de Captura: Servirá para modificar los datos erróneos acerca de la central deseada donde se ajustará la hora de su sistema.

- Estadísticas por Central.- Esta será la pantalla en la cual el usuario capturará los datos necesarios para la ejecución de un muestreo manual del tiempo de una central determinada (figura 5.16).

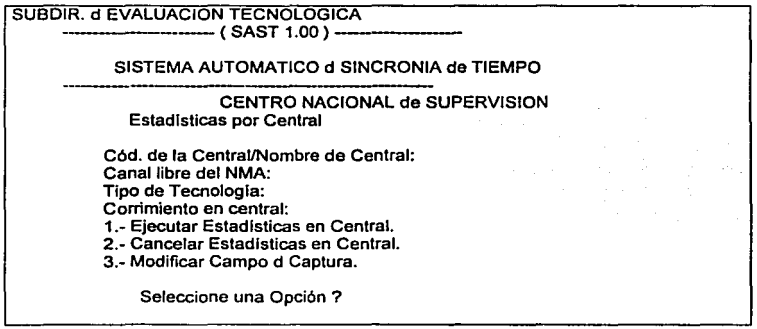


Fig. 5.16 Estadísticas por Central.

a) Ejecutar Estadísticas en Central: El resultado de esta opción se verá reflejado en el Reporte de Eventos por Central y Fecha.
b) Cancelar Estadisticas en Central: Esta opción servirá para evitar que se realicen las estadísticas.
c) Modificar Campo de Captura: Servirá para modificar los datos erróneos acerca de la central deseada donde se desean activar las estadísticas.

- Administración de Usuarios.- Presenta todas las opciones de movimientos posibles en cuanto a cuentas de usuario del sistema, como se muestra en la figura 5.17.

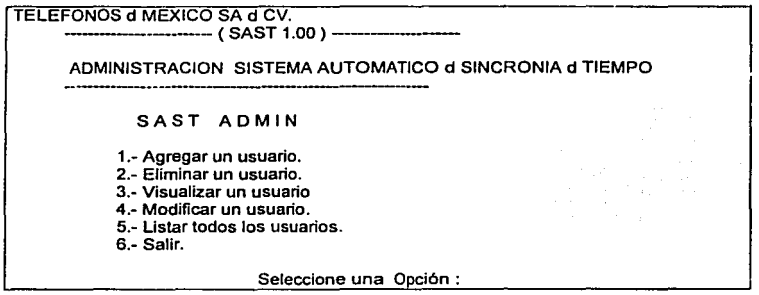


Fig. 5.17Administración del Sistema Automático de Sincronía de Tiempo.

a) Agregar a usuarios: De esta forma, un usuario con cuenta de administrador podrá dar de alta cuentas de usuario normales, otorgándoles los respectivos permisos para la ejecución de la sincronización (set), muestreo de estadísticas (est), estadísticas globales (estg) y/o sincronización de centrales de manera global (setg), como se muestra en la figura 5.18.

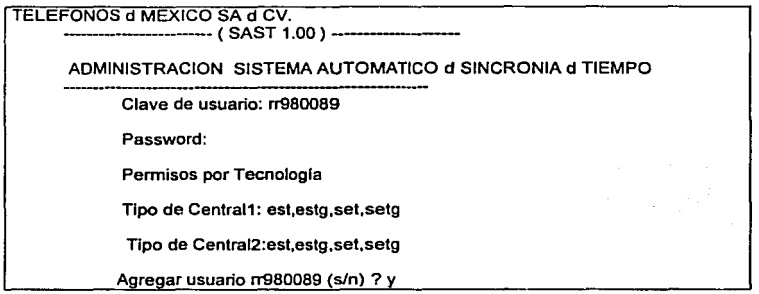


Fig. 5.18 Captura de Usuarios.

b) Eliminación de usuarios: A través de la siguiente pantalla, un usuario con cuenta de administrador podrá eliminar a un usuario normal del sistema de manera definitiva, tecleando los datos necesarios, como se muestra en la figura 5.19.

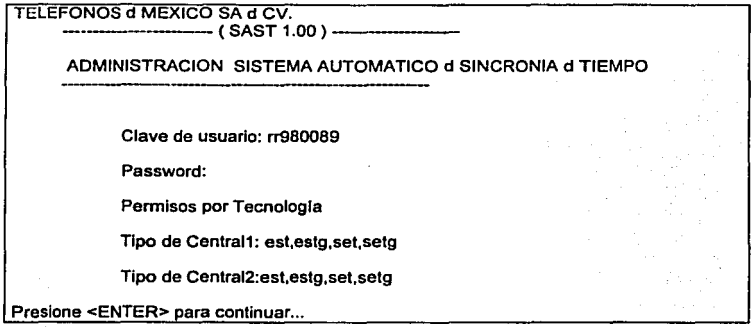


Fig. 5.19 Eliminación de usuarios.

c) Visualización de usuarios: En esta pantalla se podrán ver todos los permisos con que cuenta un usuario.

d) Modificación de usuarios: Aquí se muestra la forma en que se podrán modificar los permisos de una cuenta de usuario (figura 5.20).

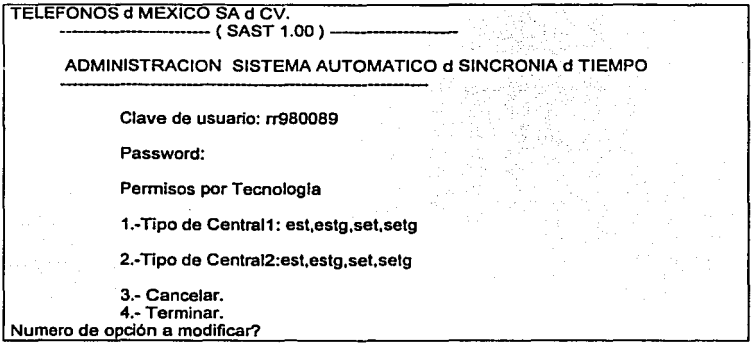


Fig. 5.20 Modificación de usuarios.

e) Listado de usuarios: Desplegará una lista completa de todos los usuarios dados de alta en el sistema, como se muestra en la figura 5.21.

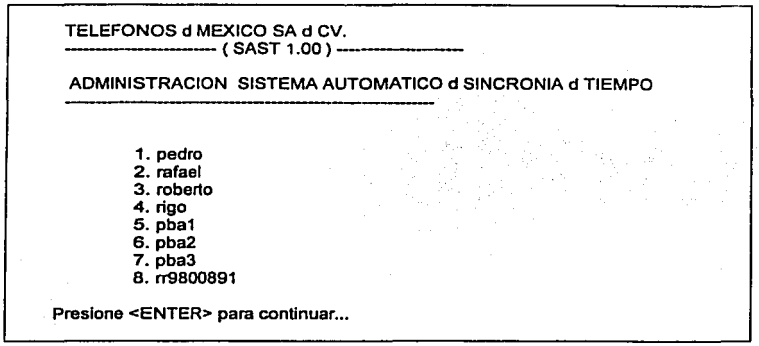


Fig. 5.21 Listado de usuarios.

En el siguiente capítulo traduciremos el diseño de los módulos de programa que hemos presentado en este apartado, en su código fuente correspondiente, es decir, pasaremos a la etapa de implementación del sistema.

VI. IMPLEMENTACIÓN DEL SAST

Después del diseño del sistema presentado en el capítulo anterior, el paso siguiente es la implementación de todos los módulos con que cuenta el sistema, codificándolos en el lenguaje de programación C, de manera que cada uno de ellos cumpla con la función específica para la que fue ideado. Como recordaremos los módulos del SAST son los siguientes:

- Módulo de Muestreo.
- Módulo de Recolección
- Módulo de Monitoreo en Línea
- Módulo de Muestreo y Sincronización Manual
- Módulo de Reporteo
- Módulo de Comunicaciones
- Módulo de Construcción de Mensajes
- Módulo de Administración

6.1 Módulo de Muestreo

El módulo de muestreo está conformado de 4 *scripts* desarrollados sobre el NMA (uno por cada tipo de tecnología de central). El objeto de estos *scripts* es el envío de los comandos necesarios para obtener la hora de la central y ajustarla cuando sea necesario. Estos *scripts* están desarrollados en un lenguaje de propósito específico.

Por cuestiones de espacio, presentaremos el código fuente de sólo uno de estos *scripts*; el que corresponde a las centrales telefónicas AXE.

6.1.1 Script para centrales AXE

```
SISTEMA AUTOMATICO DE SINCRONIA DE TIEMPO

SCRIPT PARA CENTRALES AXE

*II9210655              12/28/99 13:22 cst
*II9210655              12/21/99 18:53 cst
*II9210655              11/29/99 18:09 cst

perform-script 'amm_parse', result=parse_rc
if
      not-equal  parse_rc, 0
then
      exit-script
end-if

copy 'nma     ',ap_re
copy 'all     ',ap_gen
copy 'axe28    ',axe
copy 'none',axe_re
concatenate axe_rsp,'r',ap_re,ap_gen,'resp_time ',axe,axe_re
concatenate axe_rsp1,'r',ap_re,ap_gen,'r_ajust   ',axe,axe_re
* The WAIT-FOR-COMMAND will buffer util a channel is up
* I am added an hour conter to make sure if the message is recieved in
* one hour it will not execute an hour later (only \x0d will be sent)
```

```
            perform-script 'clock'
            copy hour,start_hour

copy 'no',mtp
send-to-ne '\x0d'
*sleep 5

wait-for-response axe_rsp,20

if
  not-equal @status,0
then
  exit-script
end-if

            perform-script  'clock'
            copy hour,stop_hour
            if
               greater stop_hour,start_hour
            then
               exit-script
            end-if

string-util utility='rtrim',
            result=operacion,
            arg-string=operacion

switch operacion
  case 'est','EST'
    copy number,numero
  case 'SET','set'
    copy 1,numero
  otherwise
    exit-script
end-switch

copy 1,veces
while
  less-or-equal veces,numero
do
  send-to-ne 'CACLP;\x0d'
  wait-for-response axe_rsp,60
  copy @status,aa
  if
    equal aa,00
  then
    perform-script 'clock'
    substring st_time,1,2,hr_2
    substring st_time,3,2,min_2
    substring st_time,5,2,sec_2
    ***** get time from 'TIME_STAMP' message
```

```
local index
array-size info,info_size
copy 1,index
while
  less index,info_size
do
  scan info[index],'DATE',pos
  if
    not-equal 0,pos
  then
    add 1,index
    if
      equal 11,index
    then
      substring info[index],17,2,sys_hr
      substring info[index],19,2,sys_min
      substring info[index],21,2,sys_sec
    else
      substring info[index],18,2,sys_hr
      substring info[index],20,2,sys_min
      substring info[index],22,2,sys_sec
    end-if
    substring info[index],9,6,sys_fecha
    copy info_size,index
  end-if
  add 1,index
end-while
copy sys_sec,b
copy sec_2,a
copy sys_min,bb
copy min_2,aa
multiply 60,bb
multiply 60,aa
add sys_sec,bb
add sec_2,aa
if
  greater aa,bb
then
  subtract bb,aa
  concatenate diff,aa
else
  subtract aa,bb
  if
    not-equal bb,0
  then
    concatenate diff,'-',bb
  else
    concatenate diff,bb
  end-if
end-if
```

```
switch operacion
  case 'est','EST'
    concatenate time_off,sys_hr,':',sys_min,':',sys_sec
    concatenate time_st,hr_2,':',min_2,':',sec_2
    string-util 'rtrim',co,neis_target_id
    concatenate info2,co,',',sys_fecha,',',time_off,',',
                time_st,',',diff,',',' ',',','AXE'

  case 'set','SET'
    scan diff,'-',pos
    if
      not-equal pos,0
    then
      add 1,pos
      substring main=diff,offset=pos,sub=n_valor
      copy n_valor,valor
    else
      copy diff,valor
    end-if
    scan main_string=number,
        sub_string='-',
          result=pos2
    if
      not-equal pos2,0
    then
      concatenate accion,'DEC'
      add 1,pos2
      substring main=number,offset=pos2,sub=n_valor2
      copy n_valor2,valor2
    else
      concatenate accion,'INC'
      copy number,valor2
    end-if
      copy valor2,num_jp
    if
      greater valor,valor2
    then
      subtract valor2,valor
      copy valor,diff2
    else
      subtract valor,valor2
      copy valor2,diff2
    end-if
    if
      less-or-equal diff2,4
    then
      switch accion
        case 'INC'
          concatenate command,'CACLC:TDSF=',num_jp,';\x0d'
          send-to-ne command
          wait-for-response axe_rsp,30
```

```
            send-to-ne ';\x0d'
            if
              not-equal mtp,'yes'
            then
              wait-for-response axe_rsp,30
              send-to-ne 'EXIT;\x0d'
            end-if
            wait-for-response axe_rsp1,80
            if
              equal @status,0
            then
              concatenate comando,'set',' ',number
            else
              concatenate comando,'set',' ','err'
            end-if
          case 'DEC'
            concatenate command,'CACLC:TDSB=',num_jp,';\x0d'
            send-to-ne command
            wait-for-response axe_rsp,30
            send-to-ne ';\x0d'
            if
              not-equal mtp,'yes'
            then
              wait-for-response axe_rsp,30
              send-to-ne 'EXIT;\x0d'
            end-if
            wait-for-response axe_rsp1,80
            if
              equal @status,0
            then
              concatenate comando,'set',' ',number
            else
              concatenate comando,'set',' ','err'
            end-if
          end-switch
        else
          concatenate comando,'ERROR',' ',number
        end-if
        concatenate time_off,sys_hr,':',sys_min,':',sys_sec
        concatenate time_st,hr_2,':',min_2,':',sec_2
        string-util 'rtrim',co,neis_target_id
        concatenate info2,co,',',sys_fecha,',',time_off.',',
            time_st,',',diff,',',comando,',','AXE'
        end-switch

***** use this to watch for problems *****
*post-log 1223,'scc',info2
* concatenate neis_raw_msg,info2
    *** send the information to a file in VOS!!!
    concatenate file,'%mex_city#m1_d03>nma_dblib>ufd>fg7701917>clk_axe_file'
    open-file file,port,1,3
```

```
    write-record port,info2
    close-file port
  else
    exit-script
  end-if
add 1,veces

get-sm-clli   target-id = neis_target_id,
          sm-id = rem_id,
          result = clli
gma-flatten into=gma_std_data
perform-script 'amm_build', result=buildpkt
if
      equal buildpkt.error, 0
then
      send-to-application  buildpkt.buf, 0
end-if

end-while
end-script
```

6.2 Módulo de Recolección

Es básicamente un programa que emplea dos sockets de *servicio*, uno para la recepción de los registros generados por el módulo de muestreo y otro para el envío de comandos de reinicialización de las terminales asociadas al socket en el NMA, a través del módulo de comunicaciones. Las partes de código fuente más representativas se muestran a continuación.

6.2.1 Archivos de Cabecera

- Archivo de Cabecera defs.h

```
/* Archivo: defs.h */

#if !defined (DEFS)
#define DEFS

/* Archivo "estado.txt" */
FILE *festatus;

/* Manejo de errores */
#define L_CAD_ERR 80
extern int errno;

/* Para manejo de errores */
#define L_CAD_AUX 250

#endif
```

- Archivo de Cabecera colector.h

```
/* Archivo colector.h */

/*Numero de archivos planos para recepcion de datos*/
#define NUM_BD     9

/* Trayectorias para archivos de datos con registro de 5 campos*/
#define TRAYECTO       "/home
#define DMETRO              TRAYECTO/metro/datos1.txt"
#define DMTY          TRAYECTO/mty/datos1.txt"
#define DGDL          TRAYECTO/gdl/datos1.txt"
#define DPBL          TRAYECTO/pba/datos1.txt"
#define DQRO          TRAYECTO/qro/datos1.txt"
#define DHMO          TRAYECTO/hmo/datos1.txt"
#define DMDA          TRAYECTO/mda/datos1.txt"
#define DCHI          TRAYECTO/chi/datos1.txt"
#define DCNS          TRAYECTO/cns/datos1.txt"

/* Trayectorias para archivos de datos con registro de 7 campos*/
#define TRAYECTO2      "/home
/*
#define DMETRO_OTE2     TRAYECTO/metroori/datos2.txt"
#define DMETRO_PTE2     TRAYECTO/metropon/datos2.txt"
#define DMETRO_SUR2     TRAYECTO/metrosur/datos2.txt"
*/
#define DMETRO2             TRAYECTO/metro/datos2.txt"
#define DMTY2          TRAYECTO/mty/datos2.txt"
#define DGDL2          TRAYECTO/gdl/datos2.txt"
#define DPBL2          TRAYECTO/pba/datos2.txt"
#define DQRO2          TRAYECTO/qro/datos2.txt"
#define DHMO2          TRAYECTO/hmo/datos2.txt"
#define DMDA2          TRAYECTO/mda/datos2.txt"
#define DCHI2          TRAYECTO/chi/datos2.txt"
#define DCNS2          TRAYECTO/cns/datos2.txt"

#define TABMETRO TRAYECTOT/tmetro.txt"
#define TABMTY              TRAYECTOT/tmty.txt"
#define TABGDL              TRAYECTOT/tgdl.txt"
#define TABPBL              TRAYECTOT/tpbl.txt"
#define TABQRO              TRAYECTOT/tqro.txt"
#define TABHMO              TRAYECTOT/thmo.txt"
#define TABMER              TRAYECTOT/tmer.txt"
#define TABCHI              TRAYECTOT/tchi.txt"
#define TABCNS              TRAYECTOT/tcns.txt"

#define ARCH_ESTATUS   "/home/reloj/estatus.txt"
#define ARCH_ERRORES   "/home/reloj/errores.txt"

/*Longitud de la linea leida del socket */
#define LONG_LINEA        200
```

```
/* Numero de mediciones de hora de central */
#define NUM_MEDS 2
/* Numero de tokens por linea leida del socket */
#define NUM_TOKENS      NUM_MEDS+6
/* Longitud de la linea en los archivos de entrada de datos */
#define LONG_DATOS      200

/* Constantes para la estructura de control*/
#define L_CLLCODE       12
#define L_FECHA         7
#define L_HORA          9
#define L_CMD           31
#define L_DIF           10
#define L_MSG           15
#define L_TIPO          15
#define RANGO           20

/* Estructura de control InfoReg. Longitud=92 char */
typedef struct {
            char Cllcode[L_CLLCODE+RANGO];
          char Fecha[L_FECHA+RANGO];
            char HoraRef[L_HORA+RANGO];
            char HoraCent[L_HORA+RANGO];
            char Dif[L_DIF+RANGO];
            char Msg[L_MSG+RANGO];
            char Tipo[L_TIPO+RANGO];
            } InfoReg;

/* Declaracion de Tablas */

/* Declaraciones para el manejo de las Tablas*/
#define L_METRO_OTE     58
#define L_METRO_PTE     44
#define L_METRO_SUR     33
#define L_METRO         137
#define L_MTY       62
#define L_GDL       78
#define L_CHI       27
#define L_QRO       57
#define L_PBL           76
#define L_MDA           42
#define L_HMO           39
#define L_CNS           42

typedef struct {
        char div[14];
        char cllcode[12];
        } ent;

ent Tab_CNS[L_CNS];
/*
```

```
ent Tab_Metro_ote[L_METRO_OTE];
ent Tab_Metro_pte[L_METRO_PTE];
ent Tab_Metro_sur[L_METRO_SUR];
*/
ent Tab_Metro[L_METRO];
ent Tab_Mty[L_MTY];
ent Tab_Gdl[L_GDL];
ent Tab_Hmo[L_HMO];
ent Tab_Pbl[L_PBL];
ent Tab_Mda[L_MDA];
ent Tab_Qro[L_QRO];
ent Tab_Chi[L_CHI];

/* variables para manejo del socket */
struct hostent *hent;
struct sockaddr_in fsock,sname;

/* Variables para el bloque de recepcion del telnet */

/* Variables globales para uso del telnet */
char telnet[30];
char tel_pto[10];

/* Variable para le manejo de la información proveniente del telnet*/
InfoReg Rx;

/* Arreglo de apuntadores a archivos de datos para archivos datos1.txt*/
FILE *fgrupo[NUM_BD];

/* Arreglo de apuntadores a archivos de datos para archivos datos2.txt*/
FILE *fgrupo2[NUM_BD];

/* Auxiliar */
FILE *faux;

/* Errores */
FILE *ferrores;

extern void exit();

compara_cllcode();
en_tabla();

/* Variables para el manejo de TRAYECTO's en la funcion: reg_a_disco */
enum{metro,mty,gdl,pbl,qro,hmo,mda,chi,cns};
/* Variables para chequeo de campos en la funcion: checa_campo */
enum{dcllcode,dfecha,dhora,ddif,dmsg,dtipo};

#define LONG_CAD_AUX 250

int NumCampos;
```

- Programa Principal reloj_cont_c.c

```
/* Archivo: reloj_cont_c.c */

#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <sgtty.h>
#include <fcntl.h>
#include <sys/signal.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/socket.h>
#include <setjmp.h>
#include "funcs.c"
#include "../hdr/defs.h"
#include "../hdr/telcontrol.h"
#include "../hdr/colector.h"

FILE *fprueba;

/* Variables globales para uso del telnet hacia NMA*/
char nma_sist[30];
char nma_sist_sock[10];

/* Variables para el bloque de recepcion del telnet */
char block_rx_nma[LONG_BLOCK_RX];

/* Descriptor del archivo alarmas.txt */
FILE *farch;

/*************************************************************************
a_disco()
*************************************************************************/
a_disco(division)
int division;
{
char linea[LONG_DATOS],cad_err[L_CAD_ERR];
int status;

        /* Se da formato del archivo de datos */

        /* (7 campos) Se da formato del archivo de datos */
        if(NumCampos==7){

        sprintf(linea,"%s,%s,%s,%s,%s,%s,%s\n",Rx.CIIcode,Rx.Fecha,Rx.HoraRef,R
x.HoraCent,Rx.Dif,Rx.Msg,Rx.Tipo);
                fprintf(festatus,"\nA disco datos2: %s ",linea);
```

```
                fflush(festatus);
                status=fputs(linea,fgrupo2[division]);
                if(status<0){
                        strerror_r(errno,cad_err,L_CAD_ERR);
                        fprintf(festatus,"No puedo escribir en archivo de datos2:
%s",cad_err);
                        fflush(festatus);
                        return MAL;
                }
                fflush(fgrupo2[division]);
        }
        return BIEN;
}
/*************************************************************************
reg_a_disco()
                Guarda el registro Rx en la base de datos de la division
                correspondiente.
*************************************************************************/
reg_a_disco()
{
        /* CNS */
        if(en_tabla(Tab_CNS,L_CNS)){
                fprintf(festatus,"\nen_tabla --> CNS ");
                fflush(festatus);
                a_disco(cns);
                return;
        }

        /* Metro */
         if(en_tabla(Tab_Metro,L_METRO)){
                  fprintf(festatus,"\nen_tabla --> METRO");
                  fflush(festatus);
                  a_disco(metro);
                  return;
         }

         /* Pba */
         if(en_tabla(Tab_Pbl,L_PBL)){
                  fprintf(festatus,"\nen_tabla --> PBL");
                  fflush(festatus);
                  a_disco(pbl);
                  return;
         }

         /* Qro */
         if(en_tabla(Tab_Qro,L_QRO)){
                fprintf(festatus,"\nen_tabla --> QRO");
                fflush(festatus);
                a_disco(qro);
                return;
          }
```

```
        /* Mda */
        if(en_tabla(Tab_Mda,L_MDA)){
                fprintf(festatus,"\nen_tabla --> MDA");
                fflush(festatus);
                a_disco(mda);
                return;
        }
        fprintf(festatus,"\nen_tabla  --> No encontrado!");
        fflush(festatus);
        fprintf(ferrores,"No encontrado en tablas: (%s) (%s) (%s) (%s) (%s) (%s)
(%s)",Rx.CIIcode,Rx.Fecha,Rx.HoraRef,Rx.HoraCent,Rx.Dif,Rx.Msg,Rx.Tipo);
        fflush(ferrores);

}

/*********************************************************************
void lee_msg(block,ixblock,lblock,msg)
*********************************************************************/
void lee_msg(block,ixblock,lblock,msg)
char *block,*msg;
int  *ixblock,*lblock;
{
char *aux;
register i=0;

        while(*ixblock<*lblock && i<LONG_MSG){
                *msg=block[*ixblock];
                if(*msg=='\n'){
                        *msg='\0';
                        /*msg++;*/
                        ++*ixblock;
                        return;
                }
                msg++;
                ++*ixblock;
                i++;
        }
        *msg='\0';
}

checa_campo(campo,tipo)
char *campo;
int tipo;
{
char *campo_aux;
char dif_aux[L_DIF+RANGO];
int i,signo=0;

        campo_aux=campo;
        switch(tipo){
```

```
case dcllcode:
        /*printf("\nChecando cllcode. campo_aux=(%s) ",campo_aux);*/
        for(i=0;*campo_aux;i++){
                if(!isalnum(*campo_aux)){
                        if(*campo_aux!='_') /* para maqueta_211 */
                                return MAL;
                }
                campo_aux++;
        }
        /*printf("  i= %d",i);*/
        if(i!=(L_CLLCODE-1))return MAL;
        return BIEN;
case dfecha:
        /*printf("\nChecando fecha. campo_aux=(%s)",campo_aux);*/
        for(i=0;*campo_aux;i++){
                if(!isdigit(*campo_aux))
                        return MAL;
                campo_aux++;
        }
        /*printf("  i= %d",i);*/
        if(i!=(L_FECHA-1))return MAL;
        return BIEN;
case dhora:
        /*printf("\nChecando hora. campo_aux=(%s) ",campo_aux);*/
        for(i=0;*campo_aux;i++){
                if(!isdigit(*campo_aux))
                        if(*campo_aux!=':')
                                return MAL;
                campo_aux++;
        }
        /*printf("  i=%d",i);*/
        if(i!=(L_HORA-1))return MAL;
        return BIEN;
case ddif:
        i=0;
        /*printf("\nChecando dif. campo_aux=(%s) ",campo_aux);*/
        if(*campo_aux=='-'){
                dif_aux[i++]='-';
                campo_aux++;
        }
        for( ;*campo_aux && i<L_DIF ;i++){
                if(!isdigit(*campo_aux))
                        break;
                dif_aux[i]=*campo_aux;
                campo_aux++;
        }
        dif_aux[i]='\0';
        if(i==0 || i>L_DIF)return MAL;
        strcpy(campo,dif_aux);
        return BIEN;
case dmsg:
```

```
                    if(strcmp(campo_aux," ")==0){
                        strcpy(campo_aux,"NUL");
                        return BIEN;
                    }
                    for(i=0;*campo_aux && i<L_MSG ;i++){
                        if(!isprint(*campo_aux))
                            return MAL;
                        campo_aux++;
                    }
                    return BIEN;
                case dtipo:
                    if(!isalnum(*campo_aux)){
                        NumCampos=5;
                        return BIEN;
                    }
                    NumCampos=7;
                    for(i=0;*campo_aux;i++){
                        if(!isalnum(*campo_aux)){
                            campo_aux='\0';
                            break;
                        }
                        campo_aux++;
                    }
                    return BIEN;
                    /*printf("  i= %d",i);*/
                    if(i<=(L_TIPO-1))return MAL;
                    return BIEN;
        }

}

/*************************************************************************
checa_reg(cllcode,fecha,hora_ref,hora_cent,dif,dif2)
*************************************************************************/
checa_reg(cllcode,fecha,hora_ref,hora_cent,dif,msg,tipo)
char *cllcode,*fecha,*hora_ref,*hora_cent,*dif,*msg,*tipo;
{
int res;
char tipo_cent[L_TIPO+RANGO],aux[L_TIPO+RANGO];

        aux[0]='\0';
        strcpy(tipo_cent,tipo);
        /* Cllcode */
        res=checa_campo(cllcode,dcllcode);
        if(res==MAL)return MAL;
        strcpy(Rx.Cllcode,cllcode);
        /*fprintf(festatus,"\nCAMPO Cllcode (%s) BIEN ",Rx.Cllcode);*/

        /* Fecha */
        res=checa_campo(fecha,dfecha);
        if(res==MAL)return MAL;
```

```
        strcpy(Rx.Fecha,fecha);
        /*fprintf(festatus,"\nCAMPO Fecha (%s) BIEN ",Rx.Fecha);*/

        /* Hora Ref */
        res=checa_campo(hora_ref,dhora);
        if(res==MAL)return MAL;
        strcpy(Rx.HoraRef,hora_ref);
        /*fprintf(festatus,"\nCAMPO HoraRef (%s) BIEN ",Rx.HoraRef);*/

        /* Hora Cent */
        res=checa_campo(hora_cent,dhora);
        if(res==MAL)return MAL;
        strcpy(Rx.HoraCent,hora_cent);
        /*fprintf(festatus,"\nCAMPO HoraCent (%s) BIEN ",Rx.HoraCent);*/

        /* Dif */
        res=checa_campo(dif,ddif);
        if(res==MAL)return MAL;
        strcpy(Rx.Dif,dif);
        /*fprintf(festatus,"\nCAMPO Dif (%s) BIEN ",Rx.Dif);*/

        /* Msg */
        if(strlen(msg)!=0){
                res=checa_campo(msg,dmsg);
                if(res==MAL)return MAL;
                strcpy(Rx.Msg,msg);
                /*fprintf(festatus,"\nCampo msg (%s) BIEN",msg);*/
        }
        else
                fprintf(festatus,"\nNo hay campo Msg.");
        /* Tipo */
        res=checa_campo(tipo_cent,dtipo);
        if(res==MAL) return MAL;
        limpia_token(tipo_cent,aux);
        strcpy(Rx.Tipo,aux);
        /*fprintf(festatus,"\nCAMPO Tipo (%s) BIEN ",Rx.Tipo);*/

        /*fprintf(festatus,"\nRegistro leido sin errores");*/
        fflush(festatus);
        return BIEN;
}

/*****************************************************************************
lee_reg(msg)
*****************************************************************************/
lee_reg(msg)
char *msg;
{
char c;
register i,numlineas,n;
char *aux;
```

```
char cad[LONG_LINEA+5];
char *delims=",\n";
char *token[NUM_TOKENS], *arg;
char dia[4];
int res;

        for(i=0; i<NUM_TOKENS ; i++ )
                token[i]="";
        arg=msg;
        for(i=0; token[i]=strtok(arg,delims) ; i++ )
                arg=NULL;
        limpia_Rx();
        /*limpia_token(token[6]);*/
        /*printf("\nAntes de checa_reg");*/
        res=checa_reg(token[0],token[1],token[2],token[3],token[4],token[5],token[6]);
        if(res==MAL) return MAL;
        fprintf(festatus,"\nRegistro leido: (%s) (%s) (%s) (%s) (%s) (%s)
(%s)",Rx.CIIcode,Rx.Fecha,Rx.HoraRef,Rx.HoraCent,Rx.Dif,Rx.Msg,Rx.Tipo);
        fflush(festatus);
        return BIEN;
}

abre_archivos()
{
FILE *arch;
char *archivos[NUM_BD];
short i;

        /* Se abren archivos para trayectoria de datos1 */
        archivos[0]=DMETRO;
        archivos[1]=DMTY;
        archivos[2]=DGDL;
        archivos[3]=DPBL;
        archivos[4]=DQRO;
        archivos[5]=DHMO;
        archivos[6]=DMDA;
        archivos[7]=DCHI;
        archivos[8]=DCNS;

       /*Se crean los archivos si no existen */
       for(i=0;i<NUM_BD;i++){
                if((fgrupo[i]=fopen(archivos[i],"a+"))==NULL)
                        if((fgrupo[i]=fopen(archivos[i],"w"))==NULL){
                                printf("\nFalla: i=%d archivos[i]=%s\n",i,archivos[i]);
                                perror("No puedo abrir.",archivos[i]);
                                exit(1);
                        }
       }

        /* Se abren archivos para trayectoria de datos2 */
        archivos[0]=DMETRO2;
```

```
        archivos[1]=DMTY2;
        archivos[2]=DGDL2;
        archivos[3]=DPBL2;
        archivos[4]=DQRO2;
        archivos[5]=DHMO2;
        archivos[6]=DMDA2;
        archivos[7]=DCHI2;
        archivos[8]=DCNS2;

        /*Se crean los archivos si no existen */
        for(i=0;i<NUM_BD;i++){
                if((fgrupo2[i]=fopen(archivos[i],"a+"))==NULL)
                        if((fgrupo2[i]=fopen(archivos[i],"w"))==NULL){
                                printf("\nFalla: i=%d archivos[i]=%s\n",i,archivos[i]);
                                perror("No puedo abrir.",archivos[i]);
                                exit(1);
                        }
        }

        /* Se crea archivo de errores, sino exite */
                if((ferrores=fopen(ARCH_ERRORES,"a+"))==NULL)
                        if((ferrores=fopen(ARCH_ERRORES,"w"))==NULL){
                                perror("No puedo abrir archivo errores.txt");
                                exit(1);
                        }

        /* Se crea archivo de estatus, si no exite */
                if((festatus=fopen(ARCH_ESTATUS,"a+"))==NULL)
                        if((ferrores=fopen(ARCH_ESTATUS,"w"))==NULL){
                                perror("No puedo abrir archivo estatus.txt");
                                exit(1);
                        }

}
/*************************************************************************
a_archivo()
a_archivo(msg)
char *msg;
{
int num;
char cad[LONG_MSG];

        if((farch=fopen("alarmas.aux","r+"))==NULL){
                printf("\nNo puedo abrir archivo alarmas.aux");
                perror("Error");
                return;
        }
        sprintf(cad,"%s\n",msg);
        num=fputs(cad,farch);
        if(num<=0)
                perror("No Puedo escribir en -alarmas.aux- ");
```

```
        fflush(farch);
        fclose(farch);
        system("cat alarmas.aux >> alarmas.txt");
        system("> alarmas.aux");

}
*******************/

main(argc,argv)
int argc;
char *argv[];
{
int     tty,res,s_nma,q_nma,lblock,ixblock,timeOut;
char    msg[LONG_MSG],msg_aux[LONG_MSG+1];
jmp_buf timeout_pont;

        signal(SIGINT,limpiar);
        signal(SIGKILL,limpiar);
        fflush(stdout);
        if(argc!=3) {
                fprintf(festatus,"\nUso: ");
                fprintf(festatus,"\n\n\t reloj_cont <telnet> <puerto>\n");
                fflush(festatus);
                exit();
        }

        abre_archivos();
        fprintf(festatus,"Iniciando sistema...");
        fflush(festatus);
        fprintf(festatus,"Iniciando sistema...");
        fflush(festatus);
        fprintf(festatus,"Iniciando sistema...");
        fflush(festatus);
        sube_tablas();
        strcpy(nma_sist,argv[1]);
        strcpy(nma_sist_sock,argv[2]);
        s_nma=crea_socket(nma_sist,nma_sist_sock);
        if(s_nma==MAL)
                s_nma=recupera_socket(s_nma,nma_sist,nma_sist_sock);
        system("clear");
        timeOut=TIMEOUT_TIME;
        while(1) {

        res=lee_block_rx(s_nma,nma_sist,nma_sist_sock,block_rx_nma,&ixblock,&l
block);
                if(res==TIMEOUT){
                        fprintf(festatus,"\nOcurrio timeout!. Reseteando socket...");
                        fflush(festatus);

        s_nma=recupera_socket(s_nma,nma_sist,nma_sist_sock,NO_DIRECTO);
                        continue;
```

```
                }
                if(res==SOCKET_DOWN){
                        fprintf(festatus,"\nSocket abajo!. Reseteando socket...");
                        fflush(festatus);

        s_nma=recupera_socket(s_nma,nma_sist,nma_sist_sock,NO_DIRECTO);
                        continue;
                }
                if((res=pantalla_nma(block_rx_nma,lblock))<0){
                        if(res==NMA_ACCESO){
                                fprintf(festatus,"\nRecibi una pantalla de NMA...");
                                fprintf(festatus,"\nEspero datos en un tiempo maximo de
%d segundos.",timeOut);
                                fflush(festatus);
                                continue;
                                }
                        else
                                {
                                fprintf(festatus,"\nNo      hay      terminales      virtuales
disponibles...");
                                fflush(festatus);

        s_nma=recupera_socket(s_nma,nma_sist,nma_sist_sock,DIRECTO);
                                continue;
                                }
                }
                while(ixblock<lblock){
                        lee_msg(block_rx_nma,&ixblock,&lblock,msg);
                        /*despliega_msg(msg);*/
                        strcpy(msg_aux,msg);
                        res = lee_reg(msg);
                        if(res==MAL){
                                fprintf(ferrores,"\n%s\n",msg_aux);
                                fflush(ferrores);
                                continue;
                        }
                        reg_a_disco();
                }
        }
}
```

6.3 Módulo de Monitoreo en Línea

Este módulo determina cuales son las centrales que se encuentran fuera de sincronía y las mostrará al usuario en una pantalla de monitoreo en tiempo real, con un tiempo de refresco de 10 segundos. Además, este puede generar muestras instantáneas de tiempo y ejecutar el proceso de sincronización sobre cualquiera de las centrales que se muestran en pantalla.

El código de la rutina principal del módulo de monitoreo es el que se lista a continuación:

```
/* monitor.c - Archivo de implementación para monitoreo en linea */

/* Retorna el numero de registros en el archivo monitoreado */
int lee_archivo(nom_arch)
char *nom_arch;
{
   int       abre_archivo();
   int       numreg = 0;
   int       ctrl();
   char       readbuffer[MAXLEN];


   if ( abre_archivo(nom_arch) == -1 )
   {
        perror("abriendo archivo");
        exit(0);
   }

   numreg = 0;
   fgets(readbuffer, MAXLEN, rep_file);
   while ( numreg < MAXREG && !feof(rep_file) )
   {
        if ( readbuffer[0] != ' ' && readbuffer[0] != '-' &&
           strlen(readbuffer) > 0 && !isupper(readbuffer[0]) &&
           strchr(readbuffer, ':') != NULL )
     {
           strcpy(tabla[numreg++], readbuffer);
           if ( numreg == MAXREG )
        {
                printf("tabla de registros desbordada...\n");
                sleep(3);
                break;
        }

        }
     fgets(readbuffer, MAXLEN, rep_file);
   }
   fclose(rep_file);

   return(numreg);
}


int abre_archivo(nom_arch)
{
   if ( (rep_file=fopen(nom_arch, "r")) == NULL )
   {
    perror("Abriendo archivo");
    return(-1);
   }
```

```
   return(0);
}


/* Retorna -1 si ocurrio un timeout o Retorna 1 en salida normal */
int monitorear(nom_arch)
        char *nom_arch;
{
   void imprime_reporte1(), imprime_encab() ;
   void construye_mensaje();
   int  lee_archivo(), k, i, v, inicio, tope, reg_sel ;
   static int NUMREGS = 10 ; /*Num. de Reg. por Desplegar.*/
   char op = 'I', buffer[100], cllcode_sel[100], borra[100] ;
   char s[2], auxbuf[20], temporal[100], arc_datos[100] ;
   movimiento v_mov, *mov;

   REFRESH:

   reg_sel = 0;
   inicio = 0;
   tope = 0;
   k = 0;


        sprintf ( temporal,"cp datos2.txt %d.tmp.dat%c",getpid(),'\0') ;
        system  ( temporal) ;
        sprintf ( arc_datos,"%d.tmp.dat%c",getpid(),'\0') ;

        memset ( nom_arch,'\0',100) ;
        leer_archivo ( arc_datos,1,0,nom_arch) ;

   tope = lee_archivo(nom_arch);

   while ( op != 'R' )
   {

      system("clear");
        imprime_reporte1();
        imprime_encab();

      if ( tope <= 0 )
        {
         printf("\n\n\n\t NO HAY CENTRALES FUERA d SINCRONIA, EXCELENTE,
EXCELENTE ....");
         printf("\n\n\n\t[ Enter ] ... para continuar");
         fflush(stdin);
         gets(s);
         break;
      }
        else
        {
```

```
      for (k=inicio;k<(inicio+NUMREGS);k++)
      {
          if ( tabla[k][0] != ' ' && tabla[k][0] != '-' &&
               !isupper(tabla[k][0]) )
             printf("%04d  %s", k+1, tabla[k] );

          if ( k == tope-1 ) break;
      }
    }
       printf("\n");
    for (i=0; i<76; i++) printf("%c", '-');

    printf("\n\nPAGINA [ Sig,Ant,Ini,Fin ] Regresa  Estadisticas  siNcrononizar  Op?
");

      v = t_gets(buffer, 20);

      switch ( v )
      {
         case -1: return(0);  /* Ocurrio EOF */
                  break;
         case -2: goto REFRESH;
                  printf("\nActualizando datos...");  /* Ocurrio timeout */
                  sleep(1);
                  return(-1);
                  break;
    }

      op = toupper(buffer[0]);

    switch ( op )
    {
        case 'I' :
                  inicio = 0;
                  break;

      case 'F' :
                  if ( tope >= NUMREGS )
                     inicio = tope - NUMREGS;
                  break;

      case 'S' :
                  if ( inicio < tope -NUMREGS )
                     inicio += NUMREGS;
                  break;

      case 'A' :
                  if ( inicio >= NUMREGS )
                     inicio -= NUMREGS;
             else
                        inicio = 0;
```

```
                    break;
        case 'E' :
                    printf("\n                    Numero d Registro d Central ? ") ;

                    do {
                        memset(auxbuf, '\0', 20);
                      fflush(stdin);
                      gets(auxbuf);
                      reg_sel = atoi(auxbuf) ;
                } while ( reg_sel <=  0 || reg_sel > tope );

                    mov = &v_mov;
                construye_mensaje(mov, 'E', tabla[reg_sel-1]);
                est_manual(mov);
                    /* mensaje(mov); */
                    goto REFRESH;
                    break;
        case 'N' :
                    printf("\n              Dame #reg de la Central ? ");
                    do {
                        memset(auxbuf, '\0', 20);
                        fflush(stdin);
                        gets(auxbuf);
                        reg_sel = atoi(auxbuf);
                } while ( reg_sel <= 0 || reg_sel > tope );

                    mov = &v_mov;
                construye_mensaje(mov, 'N', tabla[reg_sel-1]);
                sinc_manual(mov);
                    /* mensaje(mov); */
                    goto REFRESH;
                    break;
    }
  } /* Fin de while */

        sprintf ( borra,"rm %s %c", nom_arch, '\0') ;
        system ( borra) ;

        sprintf ( temporal,"rm %d.tmp.dat%c", getpid(),'\0') ;
        system ( temporal) ;

        return ( 1) ;   /* Salida normal (Quit) */
}

void construye_mensaje(mov, op, regbuffer)
movimiento *mov;
char      op;
char      *regbuffer;
{
 char *auxbuf;
 int  k;
```

```
  auxbuf = strtok(regbuffer, " ,");
  strcpy(mov->cllcode,  auxbuf);

  mov->tipo_ejec = AUTO;

  if ( op == 'E' )       /* Generar Estadisticas */
    strcpy(mov->tipo_mov, "est");
  else if ( op == 'N' )  /* Sincronizar */
    strcpy(mov->tipo_mov, "set");

  /* Obtener datos descomponiendo por tokens */
  for ( k=0; k<5; k++)
  {
    auxbuf = strtok(NULL, " ");
    if ( k == 0 ) strcpy(mov->tec, auxbuf);
    if ( k == 4 ) mov->delta =  atoi(auxbuf);
  }


  get_msg_script(mov, mov->tec);
  /* strcpy(mov->msg_script, "reloj2"); */
  /*
  if ( strncmp(mov->tec, "PTS", 3) == 0 )
    strcpy(mov->msg_script, "show_time");
  */
  sprintf(mov->archivo_msg, "MSG%d%s", getpid(), mov->tec);

  if ( op == 'N' )
    mov->num_estad = mov->delta;


  /* strcpy(mov->canal, "ldtc"); */
  strcpy(mov->canal, S.canal);
}
```

6.4 Módulo de Muestreo y Sincronización Manual

Este módulo puede generar muestras de tiempo y ejecutar el proceso de sincronización sobre cualquier central telefónica, proporcionando todos los parámetros necesarios de manera manual. Este procedimiento manual sobre las centrales telefónicas será perfectamente válido.

La rutina principal es la siguiente:

```
/* Programa est_sinc_manual.c */

int sinc_manual( sincronia )
        movimiento *sincronia ;
{
        void strupper();
```

```
        char cen[100], canal[100], tecnologia[100], cor[100] ;
        char aux[12];
        int corrimiento , op, n_campo ;

        system("clear");
        header() ;

        memset(cen,'\0',100) ;
        memset(canal,'\0',100) ;
        memset(tecnologia,'\0',100) ;
        memset(cor,'\0',100) ;

        if ( sincronia->tipo_ejec == AUTO )
        {
          strcpy(cen, sincronia->cllcode) ;
       strcpy(canal, sincronia->canal) ;
          strcpy(tecnologia, sincronia->tec) ;
          corrimiento = sincronia->delta ;
      }

        system("clear");
        header() ;
        printf("\n\n");
        printf("                    Sincronia d Centrales Manuales");
        printf("\n\n\n");

        printf("              (1) - ( Clli/Nombre d Central ) -> %s ",cen)  ;
        if ( sincronia->tipo_ejec == MANUAL ) scanf("%s",cen);
        else printf("\n");
        printf("              (2) - ( Canal libre d NMA     ) -> %s ",canal) ;
        if ( sincronia->tipo_ejec == MANUAL ) scanf("%s",canal) ;
        else printf("\n");
        printf("              (3) - ( Tipo d Tecnologia     ) -> %s ",tecnologia) ;
        if  ( sincronia->tipo_ejec  ==  MANUAL  )  {  scanf("%s",tecnologia)  ;
        strupper(tecnologia); }
        else printf("\n");
        if ( sincronia->tipo_ejec == AUTO ) sprintf(cor, "%d", corrimiento);
        else sprintf(cor, "%s", "");
        printf("              (4) - ( Corrimiento en central)  -> %s ",cor ) ;
        if ( sincronia->tipo_ejec  ==  MANUAL  ) { scanf("%s",&cor) ;  corrimiento =
atoi(cor); }
        else printf("\n");
        printf("\n") ;
        printf("              (5) - ( Ejecutar Sincronia en Central ) ") ;
        printf("\n") ;
        printf("              (6) - ( Cancelar Sincronia en Central ) ") ;
        printf("\n") ;
        printf("              (7) - ( Modificar Campo d Captura ) ") ;

        while( 1) {
```

```
printf( "\n\n                    Selecciona Opcion ? ");

        gets(aux);
        fflush(stdin);
        op = atoi(aux);

        switch(op) {
                case 5: /* Ejecutar Sincronia en Central */
                        strcpy(sincronia->cllcode,cen) ;
                        strcpy(sincronia->canal,canal) ;
                        strcpy(sincronia->tec,tecnologia) ;
                        sincronia->delta = corrimiento  ;
                        strcpy(sincronia->tipo_mov,"set") ;
                        /* strcpy(sincronia->msg_script,"reloj2"); */
                        get_msg_script(sincronia, sincronia->tec);
                        sprintf(sincronia->archivo_msg,"MSG%d%s",
                                                getpid(),
                                                sincronia->tec) ;

                        mensaje( sincronia );

                        return( EJE_SINC ) ;
                case 6: /*  Cancelar Sincronia en Central */
                        strcpy(sincronia->cllcode,cen) ;
                        strcpy(sincronia->canal,canal) ;
                        strcpy(sincronia->tec,tecnologia) ;
                        sincronia->delta = corrimiento  ;
                        return( CAN_SINC ) ;
                case 7: /*  Modificar Campo d Captura */
                        printf( "                Numero d Campo a Modificar ? ");
                        scanf("%d",&n_campo) ;
                        switch ( n_campo) {
                        case 1:
                                cen[0]='\0' ;
                                printf("\n              (1) - ( Clli/Nombre d Central ) -
> %s ",cen)   ;
                                scanf("%s",cen);
                                break ;
                        case 2:
                                canal[0] = '\0' ;
                                printf("\n              (2) - ( Canal libre d NMA     ) ->
 %s ",canal) ;
                                scanf("%s",canal) ;
                                break ;
                        case 3:
                                tecnologia[0] = '\0' ;
                                printf("\n              (3) - ( Tipo d Tecnologia     ) ->
 %s ",tecnologia) ;
                                scanf("%s",tecnologia) ;
                                strupper(tecnologia);
                                break ;
```

```
                        case 4:
                                cor[0] = '\0' ;
                                printf("\n              (4) - ( Corrimiento en central)  -
> ") ;
                                gets(cor) ;
                                corrimiento = atoi(cor);
                                break ;
                        default :
                                desapliega(cen,canal,tecnologia,corrimiento) ;
                                break ;
                        }
                default :
                        desapliega(cen,canal,tecnologia,corrimiento) ;
                        break ;
                }
        }
        return ;
}

int est_manual( sincronia )
        movimiento *sincronia ;
{
        void strupper();
        char cen[100], canal[100], tecnologia[100], cor[100] ;
        char aux[12];
        int corrimiento = 5, op, n_campo ;

        system("clear");
        header()  ;

        memset(cen,'\0',100) ;
        memset(canal,'\0',100) ;
        memset(tecnologia,'\0',100) ;
        memset(cor,'\0',100) ;

        if ( sincronia->tipo_ejec )
        {
          strcpy(cen, sincronia->cllcode) ;
          strcpy(canal, sincronia->canal) ;
          strcpy(tecnologia, sincronia->tec) ;
          strcpy(cor,'\0',100) ;
      }

        system("clear");
        header()  ;
        printf("\n\n");
        printf("                Estadisticas d Centrales Manuales");
        printf("\n\n\n");

        printf("            (1) - ( Clli/Nombre d Central )  -> %s ",cen)   ;
        if ( sincronia->tipo_ejec == MANUAL ) scanf("%s",cen);
```

```
else printf("\n");
printf("              (2) - ( Canal libre d NMA    ) -> %s ",canal) ;
if ( sincronia->tipo_ejec == MANUAL ) scanf("%s",canal) ;
else printf("\n");
printf("              (3) - ( Tipo de Tecnologia   ) -> %s ",tecnologia) ;
if  ( sincronia->tipo_ejec  ==  MANUAL  )  {  scanf("%s",tecnologia)  ;
strupper(tecnologia); }
else printf("\n");
if ( sincronia->tipo_ejec ) sprintf(cor, "%d", corrimiento);
else sprintf(cor, "%s", "");
printf("              (4) - ( Numero d Estadisticas )  -> %s ",cor  ) ;
if ( sincronia->tipo_ejec == MANUAL ) scanf("%s",&cor) ;
else printf("\n");
printf("\n") ;
printf("              (5) - ( Ejecutar Estadisticas en Central ) ") ;
printf("\n") ;
printf("              (6) - ( Cancelar Estadisticas en Central ) ") ;
printf("\n") ;
printf("              (7) - ( Modificar Campo d Captura ) ") ;

while( 1) {

printf( "\n\n                    Selecciona Opcion ? ");
        gets(aux);
        fflush(stdin);
        op = atoi(aux);

        switch(op) {
                case 5: /*  Ejecutar Estadisticas en Central */
                        strcpy(sincronia->cllcode,cen) ;
                        strcpy(sincronia->canal,canal) ;
                        strcpy(sincronia->tec,tecnologia) ;
                        sincronia->num_estad = atoi( cor)  ;
                        strcpy(sincronia->tipo_mov,"est") ;
                        /* strcpy(sincronia->msg_script,"reloj2") ; */
                        get_msg_script(sincronia, sincronia->tec);
                        /*
                        if ( strncmp(sincronia->tec, "PTS", 3) == 0 )
                          strcpy(sincronia->msg_script, "show_time");
                      */

                        sprintf(sincronia->archivo_msg,"MSG%d%s",
                                                        getpid(),
                                                        sincronia->tec) ;

                        mensaje( sincronia );

                        return( EJE_SINC ) ;
                case 6: /* Cancelar Estadisticas en Central */
                        strcpy(sincronia->cllcode,cen) ;
                        strcpy(sincronia->canal,canal) ;
```

```
                        strcpy(sincronia->tec,tecnologia) ;
                        sincronia->delta = atoi( cor) ;
                        return( CAN_SINC ) ;
                case 7: /* Modificar Campo d Captura */
                        printf( "                Numero d Campo a Modificar ? ");
                        scanf("%d",&n_campo) ;
                        switch ( n_campo) {
                        case 1:
                                cen[0]='\0' ;
                                printf("\n                (1) - ( Clli/Nombre d Central ) -
> %s ",cen)  ;
                                scanf("%s",cen);
                                break ;
                        case 2:
                                canal[0] = '\0' ;
                                printf("\n                (2) - ( Canal libre d NMA    ) ->
%s ",canal) ;
                                scanf("%s",canal) ;
                                break ;
                        case 3:
                                tecnologia[0] = '\0' ;
                                printf("\n                (3) - ( Tipo d Tecnologia    ) ->
%s ",tecnologia) ;
                                scanf("%s",tecnologia) ;
                                strupper(tecnologia);
                                break ;
                        case 4:
                                cor[0] = '\0' ;
                   printf("\n                (4) - ( Numero d Estadisticas ) -> %s
",cor ) ;
                                scanf("%s",&cor) ;
                                break ;
                        default :
                                desp_letrero( cen, canal, tecnologia, cor) ;
                                break ;
                        }
                default :
                        desp_letrero ( cen, canal, tecnologia, cor) ;
                        break ;
                }
        }
        return ;
}

desp_letrero ( cen, canal, tecnologia, cor)
        char *cen, *canal, *cor ;
{

        system("clear");
        header() ;
        printf("\n\n");
```

```
        printf("              Estadisticas d Centrales Manuales");
        printf("\n\n\n");
        printf("            (1) - ( Clli/Nombre d Central )  -> %s ",cen)   ;
        printf("\n") ;
        printf("            (2) - ( Canal libre d NMA     )  -> %s ",canal) ;

        printf("\n") ;
        printf("            (3) - ( Tipo d Tecnologia     )  -> %s ",tecnologia) ;
        printf("\n") ;
        printf("            (4) - ( Numero d Estadisticas )  -> %s ",cor  ) ;
        printf("\n\n") ;
        printf("            (5) - ( Ejecutar Estadisticas en Central )") ;
        printf("\n") ;
        printf("            (6) - ( Cancelar Stadisticas en Central ) ") ;
        printf("\n") ;
        printf("            (7) - ( Modificar Campo d Captura ) ") ;
        return ;
}

desapliega(cen,canal,tecnologia,cor)
        char *cen, *canal;
        int  cor ;
{
        system("clear");
        header() ;
        printf("\n\n");
        printf("              Sincronia d Centrales Manuales");
        printf("\n\n\n");
        printf("            (1) - ( Clli/Nombre d Central )  -> %s ",cen)   ;
        printf("\n") ;
        printf("            (2) - ( Canal libre d NMA     )  -> %s ",canal) ;
        printf("\n");
        printf("            (3) - ( Tipo d Tecnologia     )  -> %s ",tecnologia) ;
        printf("\n") ;
        printf("            (4) - ( Corrimiento en central)  -> %d ",cor  ) ;
        printf("\n\n") ;
        printf("            (5) - ( Ejecutar Sincronia en Central ) ") ;
        printf("\n") ;
        printf("            (6) - ( Cancelar Sincronia en Central ) ") ;
        printf("\n") ;
        printf("            (7) - ( Modificar Campo d Captura ) ") ;
        return ;
}
```

6.5 Módulo de Reporteo

Este módulo se conforma de las rutinas necesarias para la generación de distintos reportes en base al archivo de datos centralizado. Todos los reportes se van a generar sobre el archivo de datos de la división a la cual pertenece el usuario.

Las rutinas principales en la generación de los reportes son las siguientes:

```
void menu_reportes()
{
     char op, ar[100] ,fecha[100] ;
       char cmdbuf[100], macro[100] ;
       char tmp[150], s[10], temporal[100], arc_datos[100] ;

       do {

          system("clear");
          header() ;
          printf("\n              R E P O R T E S   E S T A D I S T I C O S.\n") ;
          printf("\n                (1) - Centrales En Sincronia.");
          printf("\n                (2) - Centrales Fuera de Sincronia.");
          printf("\n                (3) - Eventos x Central y Fecha.");
          printf("\n                (4) - Centrales con Muestreo.");
          printf("\n                (5) - Centrales sin Muestreo.");
          printf("\n                (6) - Bitacora de Movimientos.");
          printf("\n                (7) - Menu Anterior.");
          printf("\n\n");
          printf("\n                                  GCIA. d DESARROLLO d SOFTWARE.") ;
          printf("\n          ------------------------------------------------------------");
          printf("\n\n                                        SELECCIONE OPCION : ");

          fflush(stdin);
          switch ( op = getchar() )
          {
             case '1':
                       sprintf( temporal,"cp datos2.txt %d.tmp.dat%c",
                                                        getpid(),'\0') ;
                       system( temporal) ;
                       sprintf( arc_datos,"%d.tmp.dat%c",getpid(),'\0') ;
                     leer_archivo( arc_datos,2,1,ar) ;
                       sprintf( temporal,"rm %d.tmp.dat%c",
                                                        getpid(),'\0') ;
                       system( temporal) ;
                     break;
          case '2':
                       sprintf(temporal,"cp datos2.txt %d.tmp.dat%c",
                                                        getpid(),'\0') ;
                       system( temporal) ;
                       sprintf( arc_datos,"%d.tmp.dat%c",getpid(),'\0') ;

                     leer_archivo( arc_datos,1,1,ar) ;
                       sprintf(temporal,"rm %d.tmp.dat%c",
                                                        getpid(),'\0') ;
                       system( temporal) ;
                     break;
          case '3':
                       sprintf(temporal,"cp datos2.txt %d.tmp.dat%c",
                                                        getpid(),'\0') ;
```

```
                system( temporal) ;
                sprintf( arc_datos,"%d.tmp.dat%c",getpid(),'\0') ;

              leer_archivo( arc_datos,0,1,ar) ;
                sprintf(temporal,"rm %d.tmp.dat%c",
                                                    getpid(),'\0') ;
                system( temporal) ;
              break;
case '4':

      fecha[0] = '\0' ;
      obtener_tiempo ( fecha) ;
      sprintf(temporal,"cp datos2.txt %d.tmp.dat%c",getpid(),'\0') ;
      system( temporal) ;
      sprintf( arc_datos,"%d.tmp.dat%c",getpid(),'\0') ;

      sprintf(tmp,"%d_tmp",getpid ()) ;
      sprintf(macro,"grep %s %s | sort -o %s",
                        fecha,arc_datos,tmp,'\0' ) ;
      system( macro) ;
      sprintf( macro,"sast_audita %s %s S%c",S.tabla_div,tmp,'\0') ;
      system( macro) ;
      sprintf(macro,"rm %s%c",tmp,'\0') ;
      system(macro) ;

      sprintf(temporal,"rm %d.tmp.dat%c", getpid(),'\0') ;
      system( temporal) ;
      break;

case '5':

      fecha[0] = '\0' ;
      obtener_tiempo ( fecha) ;
      sprintf(temporal,"cp datos2.txt %d.tmp.dat%c",getpid(),'\0') ;
      system( temporal) ;
      sprintf( arc_datos,"%d.tmp.dat%c",getpid(),'\0') ;

      sprintf(tmp,"%d_tmp",getpid ()) ;
      sprintf(macro,"grep %s %s | sort -o %s",
                        fecha,arc_datos,tmp,'\0' ) ;
      system( macro) ;
      sprintf( macro,"sast_audita %s %s N%c",S.tabla_div,tmp,'\0') ;
      system( macro) ;
      sprintf(macro,"rm %s%c",tmp,'\0') ;
      system(macro) ;

      sprintf(temporal,"rm %d.tmp.dat%c", getpid(),'\0') ;
      system( temporal) ;
      break;

case '6':
```

```
                    sprintf(cmdbuf, "view %s", LOG_FILE);
                    system(cmdbuf) ;
                  break;
          case '7':
                  return;
                  break;
        }
   } while ( op != '7' );
}


void leer_archivo( archivo,opcion, monitoreo,ar)
        char *archivo ;
        short int opcion, monitoreo ;
        char *ar ;
{
        FILE *fopen(), *fp ;
        char buffer[_DIM_BUF_], central[100], fecha[100], _archivo_[100] ;
        char _archivo1_[100], _archivo2_ [100], op[10] ;
        char _arc_salida_[100], _arc_[100], *p, borra[100] ;
        short int  f_e_sinc ;

        memset(_arc_salida_,'\0',100) ;
        sprintf(_arc_salida_,"_%d_.%d.sal%c",getpid (),monitoreo,'\0') ;
        sprintf(ar,"%d.%d.txt%c",getpid (),monitoreo,'\0') ;

        fp = fopen( archivo,"r") ;
        if( fp ==NULL)
                printf("\n Error Base de Datos = (%s)\n", archivo) ;

        switch( opcion ) {
                case 0:
                        sprintf(_archivo1_,"%d.txt%c",getpid (),'\0') ;

                      fp_salida = fopen(_archivo1_,"w") ;
                        if( fp_salida ==NULL)
                                printf("\n Error Archivo salida\n") ;
                        imprime_reporte() ;
                        imprime_header("POR CENTRAL y FECHA") ;
                        printf("\n CLLI de la CENTRAL :") ;
                        fflush(stdin) ;
                        gets( central) ;
                        printf("FECHA de BUSQUEDA (AAMMDD) :") ;
                        fflush(stdin) ;
                        gets( fecha) ;

                        while( fgets( buffer, _DIM_BUF_, fp)) {
                                p = strstr(buffer,"::") ;
                                if( p !=NULL) continue ;

                                p = strchr(buffer,',') ;
```

```
                if( p ==NULL) continue ;
                sast_reportes(buffer,central,fecha,0) ;
                }
        fclose( fp_salida) ;
        sprintf(_archivo2_,"vi %s%c", _archivo1_,'\0') ;
        system(_archivo2_) ;
        break ;
case 1:

        sprintf(_archivo1_,"%d.%d.txt%c",
                                getpid (),monitoreo,'\0') ;
      fp_salida = fopen(_archivo1_,"w") ;
        if( fp_salida ==NULL)
                printf("\n Error Archivo salida\n") ;
        imprime_reporte() ;
        imprime_header("Centrales Fuera de Sincronia") ;
        obtener_tiempo ( fecha) ;
        sprintf(_arc_,"grep %s %s | sort -o %s",
                        fecha,archivo,_arc_salida_,'\0' ) ;
        system(_arc_) ;
        f_e_sinc = 1 ;
        leer_sincronia( _arc_salida_,f_e_sinc) ;
        sprintf(_archivo2_,"vi %s%c", _archivo1_,'\0') ;

        fclose( fp_salida) ;
        if( monitoreo == 1)
                system(_archivo2_) ;
        break ;
case 2:

        sprintf(_archivo1_,"%d.txt%c",getpid (),'\0') ;
      fp_salida = fopen(_archivo1_,"w") ;
        if( fp_salida ==NULL)
                printf("\n Error Archivo salida\n") ;
        imprime_reporte() ;
        imprime_header("Centrales En Sincronia") ;
        obtener_tiempo ( fecha) ;
        sprintf(_arc_,"grep %s %s | sort -o %s",
                        fecha,archivo,_arc_salida_,'\0' ) ;
        system(_arc_) ;
        f_e_sinc = 0 ;
        leer_sincronia( _arc_salida_,f_e_sinc) ;
        sprintf(_archivo2_,"vi %s%c", _archivo1_,'\0') ;
        fclose( fp_salida) ;
        system(_archivo2_) ;
        break ;
case 3:
        sprintf(_archivo1_,"%d.txt%c",getpid (),'\0') ;
      fp_salida = fopen(_archivo1_,"w") ;
        if( fp_salida ==NULL)
                printf("\n Error Archivo salida\n") ;
```

```
                    imprime_reporte() ;
                    imprime_header("HISTORICO por TODAS las CENTRALES.") ;
                    central[0] = '\0' ;
                    fecha[0]   = '\0' ;

                    while( fgets( buffer, _DIM_BUF_, fp)) {
                            p = strstr(buffer,"::") ;
                            if( p !=NULL) continue ;
                            p = strchr(buffer,',') ;
                            if( p ==NULL) continue ;
                            sast_reportes(buffer,central,fecha,0) ;
                    }
                    fclose( fp_salida) ;
                    sprintf(_archivo2_,"vi %s%c", _archivo1_,'\0') ;
                    system(_archivo2_) ;
                    break ;

            default:
                    break ;
    }


    if( monitoreo!= 0) {

            /* BORRA ARCHIVOS TEMPORALES */

            sprintf( borra, "rm %s%c",_archivo1_,'\0') ;
            system ( borra) ;
            sprintf( borra, "rm %s%c",_arc_salida_,'\0') ;
            system ( borra) ;
    }

    if( monitoreo== 0) {
            sprintf( borra, "rm %s%c",_arc_salida_,'\0') ;
            system ( borra) ;
    }

    fclose( fp) ;
}
```

6.6 Módulo de Comunicaciones

Este módulo sirve como una interface de comunicación entre los módulos que se ejecutarán en el servidor UNIX y los que se ejecutarán en el *mainframe* Stratus. Un programa servidor de sesiones TCP/IP, será el encargado de enviar los comandos y mensajes necesarios al NMA para muestrear y/o ajustar la hora de sistema de determinada central telefónica.

6.6.1 Archivo de cabecera

```
/* Proyecto Sistema Automático de Sincronía de Tiempo */
```

```
/* Archivo de cabecera para comunicación cliente-servirdor */
/* Esta versión supone que todos los comandos serán recibidos ya listos
   para su ejecución en VOS */


#define BUFSIZE   500   /* longitud del buffer de lectura/escritura */
#define MAX_LINE  20000 /* longitud del buffer de lectuta/escritura en VOS */

#define LOGOUT       "logout\x0D\x0A"
#define TRUE              1

#define ENTER        "\x0D\x0A"

/* Estructura para contener los datos de la cuenta de usuario */

typedef struct {

   char   login_vos[BUFSIZE]     ;
   char   password_vos[BUFSIZE]       ;
   char   ipaddressvos[BUFSIZE];

} user ;

/* Estructura de datos de peticiones del cliente */

typedef struct {

   char   no_peticion[3]          ;
   char   servicio[3]             ;
   char   comando[ BUFSIZE ]    ;

} petición ;

/* Estructura de datos de la respuesta del servidor */
typedef struct {

   char   no_peticion[3]          ;
   char   error                   ;
   char   descripcion[BUFSIZE]       ;

} respuesta ;


/* Descriptor de Archivo para registro de errores */
#define ERROR_FILE      "ssso.error"
FILE   *errorlog;


/* Descriptor de Archivo para bitácora de comandos */
#define LOG_FILE      "ssso.log"
FILE   *logfile;
```

```
/* Display Flag */
extern int Debug = 1;
```

6.6.2 Programa Principal

```
/* Programa ssso.c – Programa principal de comunicación a S.O. VOS */

#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include "ssso.h"

void sast_con_vos()         ;
void sast()                 ;
void sast_eje_com_vos()     ;
int  sast_prompt_vos()      ;

#include "sast_conexion.c"
#include "sast_eje_com_vos.c"
#include "sast_posproceso.c"
#include "debug.c"

main (argc, argv)
int argc;
char *argv[];
{
        signal(SIGCHLD, SIG_IGN); /* Evita la generación de zombis */

        if ( argc != 3 )
        {
          printf("uso: ssso <puerto> <debug>\n");
          exit(1);
      }

        Debug = (strncmp(argv[2], "debug", 7) == 0) ? 1 : 0;

        sast(argv) ;

}

void sast(argv)
char *argv[];
{
        int ls,sock_cli,addrlen,sock_vos,resp ;
```

```
        struct sockaddr_in myaddr ;
        respuesta v_respuesta      ;
        peticion  v_peticion       ;
        user      v_user           ;
        char r_com[ MAX_LINE]    ;
        void sast_parsear() ;

   DEBUG("Este es el programa servidor...\n");

   if ( (errorlog=fopen(ERROR_FILE, "a")) == NULL )
   {
    perror("creando el archivo de errores");
    exit(0);
   }

   if ( (logfile=fopen(LOG_FILE, "a")) == NULL )
   {
    perror("creando el archivo de bitácora");
    exit(0);
   }

   /* get a listen socket */
   if ((ls = socket(AF_INET, SOCK_STREAM, 0)) ==-1) {

        DEBUG("server socket call: ") ;
        fprintf(errorlog, "PROGRAMA: ssso.c   RUTINA: sast()   Error: creando el
socket: %s\n", strerror(errno) );
        exit( -1) ;
   }

   /* Server and client have agreed to communicate on this port */
   myaddr.sin_port = htons(atoi(argv[1]));

   /* Server is willing to accept requests that come from any card */
   myaddr.sin_addr.s_addr = INADDR_ANY ;

   /* Using internet domain */
   myaddr.sin_family = AF_INET ;

   /* Bind to the address */
   if ( bind(ls, &myaddr, sizeof(struct sockaddr_in)) == -1)
   {
      DEBUG("binding socket: ");
        fprintf(errorlog, "PROGRAMA: ssso.c  RUTINA: sast()  Error: binding socket:
%s\n", strerror(errno) );
        exit(-1) ;
   }

   /* Listen for a connection */
   if ( listen(ls, 1000) == -1) {
```

```
        DEBUG("Server listen call: ");
        fprintf(errorlog, "PROGRAMA: ssso.c  RUTINA: sast()  Error: server listen
call: %s\n", strerror(errno) );
        exit(-1);
    }


    while ( TRUE ) {

      /* Accept connection request */

      addrlen = sizeof(struct sockaddr_in);
      if ( (sock_cli= accept(ls, &myaddr, &addrlen)) == -1) {

          DEBUG("server accept call: ");
          fprintf(errorlog, "PROGRAMA: ssso.c  RUTINA: sast()  Error: server accept
call: %s\n", strerror(errno) );
          exit( -1) ;
      }
      if ( fork() == 0) {

resp = rec_login_cliente( &v_user,    sock_cli) ;

          fprintf(logfile, "Nuevo usuario conectado:\nlogin_vos: %s password_vos:
%s ipaddress_vos: %s\n",
                        v_user.login_vos,
                          v_user.password_vos,
                          v_user.ipaddressvos);

          fflush(logfile);


          while( TRUE ) {

              resp = rec_datos_cliente( &v_peticion, sock_cli) ;
              if( resp ==-1) {
                 fprintf(errorlog,  "PROGRAMA:  ssso.c    RUTINA:  sast()    Error:
recibiendo datos del cliente: %s\n", strerror(errno) );
                        close( sock_cli);
                        exit( -1) ;
              }

            fprintf(logfile, "no_peticion: %s, servicio: %s, comando: %s\n",
                          v_peticion.no_peticion,
                          v_peticion.servicio,
                          v_peticion.comando);

              fflush(logfile);

              if ( strlen(v_peticion.comando) == 0 ||
                 strncmp(v_peticion.comando, "close_socket", 12 ) == 0 )
```

```
        {
                DEBUG("Cerrando sesión del cliente.");
                close( sock_cli);
                exit(-1);
            }

            resp = sast_prompt_vos( &v_user, 23, &sock_vos) ;
            sast_eje_com_vos( v_peticion.comando,r_com,sock_vos) ;


            if ( strncmp(v_peticion.comando, "ping", 4) == 0 )
              sast_parsear( 0, r_com, &v_respuesta) ;

        else if ( strncmp(v_peticion.comando, "stop_lockers", 12) == 0 )
              sast_parsear(1, r_com, &v_respuesta) ;

        else if ( strncmp(v_peticion.comando, "amp_free_terminals", 18) == 0 )
              sast_parsear(2, r_com, &v_respuesta) ;

        else if ( strncmp(v_peticion.comando, "amp_init_terminals", 18) == 0 )
              sast_parsear(3, r_com, &v_respuesta) ;

            resp = env_datos_cliente( sock_cli, &v_respuesta) ;
            if( resp ==-1) {
               fprintf(errorlog, "PROGRAMA: ssso.c RUTINA: sast() Error:
enviando datos del cliente: %s\n", strerror(errno) );
                close( sock_cli);
                exit( -1) ;
            }
        }
    }
    else {

      /* Close the connection */
      close( sock_cli);
    }
  }
}

int env_datos_cliente( sock_cli,v_respuesta)
      int sock_cli ;
      respuesta *v_respuesta ;

{
    char sendbuf[ MAX_LINE] ;
      int resp ;

      if ( Debug ) printf("server sent: (%s) (%c) (%s)\n",
                                    v_respuesta->no_peticion ,
                                    v_respuesta->error ,
                                    v_respuesta->descripcion );
```

```
        memset( sendbuf,'\0',sizeof( sendbuf)) ;
        memcpy( &sendbuf[0], v_respuesta, sizeof( respuesta)) ;

        if((resp = send( sock_cli,sendbuf,sizeof( respuesta),0)) == -1) {
              DEBUG("\n Error al enviar info al cliente\n");
                return( -1) ;
        }

        return( 1) ;
}

int rec_datos_cliente( v_peticion, sock_cli)
        peticion *v_peticion ;
        int sock_cli ;
{
        char recvbuf[ MAX_LINE] ;
        int resp ;

        /* Get data from the client */

        while( TRUE )
        {

        DEBUG("\n ESPERO PETICION DEL CLIENTE \n") ;

          memset( recvbuf, '\0', sizeof( recvbuf)) ;
          if (( resp = recv(sock_cli, recvbuf, MAX_LINE-1,0))==-1)
          {
              DEBUG("\nEL CLIENTE EN ESTADO DE DESCONEXION.\n") ;
              return( -1) ;
          }

          *v_peticion = *((peticion *)recvbuf) ;

          if ( Debug ) printf("\n no_peticion = (%s) servicio = (%s) comando = (%s)\n",

                                              v_peticion->no_peticion,
                                              v_peticion->servicio   ,
                                              v_peticion->comando    );

          if ( v_peticion->no_peticion != 0 )
              break ;

        }

        return(1) ;
}

int rec_login_cliente( v_user, sock_cli)
        user *v_user ;
```

```
        int sock_cli ;
{
        char recvbuf[ MAX_LINE] ;
        int resp ;

        /* Get login, password and host from the client */

        while( TRUE )
        {

        DEBUG("\n ESPERO PETICION DEL CLIENTE \n") ;

          memset( recvbuf, '\0', sizeof( recvbuf)) ;
          if (( resp = recv(sock_cli, recvbuf, MAX_LINE-1,0))==-1)
          {
               perror("\nEL CLIENTE EN ESTADO DE DESCONEXION.\n") ;
               return( -1) ;
          }

          *v_user = *((user *)recvbuf) ;

          if ( Debug ) printf("\n login_vos = (%s) password_vos = (%s) ipaddressvos
= (%s)\n",

                                                v_user->login_vos   ,
                                                v_user->password_vos ,
                                                v_user->ipaddressvos );

          if ( v_user->login_vos != 0 )
               break ;

        }

        return(1) ;
}
```

6.6.3 Rutina de Conexión hacia VOS

```
/* Programa sast_conexion */

void sast_con_vos( dir_nma, num_sock, sock_vos)
        char *dir_nma ;     /* Direccion del host */
        int num_sock  ;     /* Numero de puerto   */
        int *sock_vos ;     /* Descriptor de socket */
{
        struct sockaddr_in sock ;
        struct hostent *hentt ;
        int _LEN_ADDR_ = sizeof (struct sockaddr_in) ;
        int len ;

        if(!( hentt = gethostbyname( dir_nma))) {
```

```
                DEBUG(" ERROR GETHOSTBYNAME: ") ;
                fprintf(errorlog,       "PROGRMA:       sast_conexion()         RUTINA:
sast_con_vos()  Error: gethostbyname: %s\n", strerror(errno));
                exit( 1);
        }

        if((*sock_vos= socket( AF_INET,SOCK_STREAM,0)) <0) {
                DEBUG(" ERROR SOCKET AL NMA: ");
                fprintf(errorlog,       "PROGRMA:       sast_conexion()         RUTINA:
sast_con_vos()  Error: creando el socket: %s\n", strerror(errno));
                exit(1);
        }
        sock.sin_family = AF_INET ;
        sock.sin_addr.s_addr = *(long *)hentt->h_addr ;
        sock.sin_port = htons( num_sock) ;

        len = connect( *sock_vos,(struct sockaddr *)&sock,_LEN_ADDR_) ;
        if( len ==-1) {
                DEBUG(" ERROR CONNECT: ") ;
                fprintf(errorlog,       "PROGRMA:       sast_conexion()         RUTINA:
sast_con_vos()  Error: conectando al socket: %s\n", strerror(errno));
                close( *sock_vos) ;
                exit( 1) ;
        }
        return ;
}

int sast_prompt_vos( v_user, pto, sock_vos)
                user *v_user ;
                int  pto, *sock_vos ;
        {

        int len ;
        char buffer[ MAX_LINE] ;
        char catbuffer[ MAX_LINE * 10];

        /* Checar estado socket */

        sast_con_vos( v_user->ipaddressvos, pto, sock_vos) ;

        memcpy(catbuffer, '\0', strlen(catbuffer));
        do {
                len = recv( *sock_vos, buffer, MAX_LINE -1, 0) ;
                if ( Debug ) printf("<%s", buffer);
                strcat(catbuffer, buffer);

                if ( coinciden(catbuffer, "PAUSE") )
                {
                /* Envia un <ENTER> a VOS */
                len = send ( *sock_vos                  ,
                      ENTER  ,
```

```
                    strlen ( ENTER ) ,
                    0 ) ;
            }

    } while ( !coinciden(catbuffer, "login")) ;

    /*
    memcpy(catbuffer, '\0', strlen(catbuffer));
    do {
            len = recv( *sock_vos, buffer, MAX_LINE -1, 0) ;
            if ( Debug ) printf("<%s", buffer);
            strcat(catbuffer, buffer);

    } while ( !coinciden(catbuffer, "login")) ;
*/

    /* Envia el login */
    len = send ( *sock_vos                    ,
            v_user->login_vos       ,
            strlen ( v_user->login_vos) ,
            0 ) ;

    memcpy(catbuffer, '\0', strlen(catbuffer));
    do {
            memset( buffer,'\0',MAX_LINE) ;
            len = recv( *sock_vos, buffer, MAX_LINE -1, 0) ;
            if ( Debug ) printf("<%s", buffer);
            strcat(catbuffer, buffer);

} while ( !coinciden(catbuffer, "Password")) ;

    /* Envia el password */
    len = send ( *sock_vos,
            v_user->password_vos,
            strlen(v_user->password_vos),
            0 ) ;

    memcpy(catbuffer, '\0', strlen(catbuffer));
    do {
            memset( buffer,'\0',MAX_LINE) ;
            len = recv( *sock_vos, buffer, MAX_LINE -1, 0) ;
            if ( Debug ) printf("<%s", buffer);
            strcat(catbuffer, buffer);

    } while ( !coinciden(catbuffer, "ready") );

    printf("\nready  ") ;
    return ( 1);
}
```

```
/* Retorna uno si encuentra coincidencia */

int coinciden( cad1, cad2)
        char *cad1 ;
        char *cad2 ;
{
        char *res  ;

        return((res = strstr( cad1, cad2)) == NULL ? 0:1) ;
}
```

6.6.4 Rutina de Envio de comandos

```
/* Programa: sast_eje_com_vos.c */

void sast_eje_com_vos( com, buf_aux, sock_vos)
        char *com;     /* Comando que va a ser ejecutado */
        char *buf_aux;
        int sock_vos ; /* Descriptor de socket al S.O. destino */
{
        char buffer[MAX_LINE];     /* Buffer de recepcion */
        char s[MAX_LINE] ;         /* Buffer de envio     */
        int i, res, indice =0 ;
        void respuesta_com()  ;

        memset(buf_aux,MAX_LINE,'\0');

        sprintf( s,"%s\x0D\x0A%c",com,'\0') ;
        res = send( sock_vos, s, strlen ( s),0) ;
        if( res ==-1) {
                DEBUG("\nError al enviar comando\n") ;
                return ;
        }

        do {
                memset( buffer,'\0',MAX_LINE) ;
                res= recv( sock_vos, buffer, MAX_LINE -1, 0) ;
                if ( Debug ) printf("<%s", buffer);
                respuesta_com( buf_aux,buffer,&indice,res ) ;
        } while( !coinciden(buffer,"ready")) ;

        buf_aux[indice] = '\0' ;

        /* Cierra sesion de VOS */
        res = send( sock_vos,LOGOUT,strlen( LOGOUT),0) ;


        shutdown( sock_vos, 2) ;
        close( sock_vos) ;
        return ;
```

```
}


void respuesta_com( b, buffer, indice, longitud)
        char *b, *buffer ;
        int *indice ;
{
        int i ;

        for( i = 0; i <longitud; ++i) {
                b[*indice] = buffer[ i] ;
                *indice += 1 ;
        }
        return ;
}
```

6.6.5. Rutina de Postprocesamiento de Comandos

```
/* Programa: sast_postproceso.c */

void sast_parsear( num_com, r_com,r)
        int num_com  ;
        char  *r_com ;
        respuesta *r ;
{
        void parsear_com_ping () ;
        void parsear_com_stop_lockers () ;
        void parsear_coms_amp () ;

        switch( num_com) {

                case 0  :
                        parsear_com_ping( r_com,r) ;
                        break ;
                case 1  :
                        parsear_com_stop_lockers(r_com,r) ;
                        break;
                case 2  :
                        parsear_coms_amp(r_com,r);
                        break;
                case 3  :
                     parsear_coms_amp(r_com,r) ;
                        break ;
                default :
                        break ;
        }
}

void parsear_com_ping (r_com, r)
        char *r_com  ;
```

```
        tresp *r ;
{
        int i = 0 ;
        char *token ;
        char *lista_token[200] ;

        printf("r_com= <<< %s >>>\n", r_com);

        token = strstr( r_com, "min/avg/max") ;
        printf("\n token = (%s)",token) ;

        lista_token[ i] = strtok( token," /") ;

        for( i =1; lista_token[ i] = strtok( NULL," /"); ++i) ;
        memset( r, '\0', sizeof( respuesta)) ;
        strcpy(r->no_resp, "1") ;
        strcpy(r->t_min, lista_token[4]) ;
        strcpy(r->t_avg, lista_token[5]) ;
        strcpy(r->t_max, "\0") ;
        strcpy(r->t_man, r->t_max) ;
        strcpy(r->packets_lost, "0\0");

        return ;
}

/* Parsea el comando stop_lockers */
void parsear_com_stop_lockers(r_com, r)
char    *r_com;
respuesta  *r;
{
        printf("Parseando comando stop_lockers: <<< %s >>>\n", r_com);

        strcpy(r->no_peticion, "1");
        r->error = (strstr(r_com, "File is NOT locked") == NULL) ? '0' : '1';
        if ( r->error == '1' )
       strcpy(r->descripcion, "File is NOT locked");
     else
       strcpy(r->descripcion, "Ok");
}

/* Parsea los comandos amp_free_terminals y amp_init_terminals */
void parsear_coms_amp(r_com, r)
char    *r_com;
respuesta  *r;
{
        printf("Parseando comandos amp: <<< %s >>>\n", r_ccm) ;

        strcpy(r->no_peticion, "1");
        r->error = ( strstr(r_com, "Error in 'terminal_id'.") == NULL ||
                     strstr(r_com, "failed") == NULL ) ? '0' : '1';
        if ( r->error == '1' )
```

```
      strcpy(r->descripcion, "bad terminal name");
    else
      strcpy(r->descripcion, "Ok");
}
```

6.7 Módulo de Construcción de Mensajes

El módulo de construcción de Mensajes se constituye de un solo programa muy simple, cuya función es el conformar un mensaje sobre una cadena de caracteres y escribirla sobre un archivo de texto, el cual va a ser tomado en cuenta como parámetro en la ejecución de un comando posterior del NMA. Cabe mencionar que este programa fue desarrollado en lenguaje C bajo sistema operativo VOS y residirá en cada uno de los mainframe Stratus (México y Guadalajara).

Como este programa es muy pequeño, se presenta el código completo:

display_file construye_msg.c

```
/* Programa construye_msg.c - Escribe un mensaje dado desde linea de
   comandos a un archivo de texto.
*/

#include <stdio.h>      /* Declara archivo de cabecera */

main(argc, argv)      /* Declara parametros por linea de comando */
int argc;
char *argv[];
{
 FILE *p;
 char buffer[10] ;


 if ( argc != 6 )                /* Si el número de parámetros es incorrecto termina
programa */
 {
  printf("USO:   escribe   <mensaje>   <movimiento>   <numero>   <tecnologia>
<archivo>\n");
  exit();
 }

 if ( (p=fopen(argv[5], "w")) == NULL )    /* Abre archivo especificado para escritura
*/
 {
  perror("Abriendo archivo de escritura");
  exit();
 }

 /* Si el tipo de central es diferente de PTS o 5ESS, el buffer de terminación es igual
al
    caracter 3 hexadecimal */
 if ( strncmp(argv[4], "PTS", 3) != 0 && strncmp(argv[4], "5ESS", 3) != 0)
```

```
   strcpy( buffer,"\x03") ;

  /* Si el tipo de central es AXE, concatena el buffer de terminación con "<" */
  if ( strncmp(argv[4], "AXE", 3) == 0 )
    strcat( buffer,"<") ;

  /* Escribe el mensaje al archivo en el siguiente formato */
  fprintf(p, "%s %s %d %s", argv[1],
                                argv[2],
                                  atoi(argv[3]), buffer);

  fclose(p);    /* cierra el archivo */

}
```

6.8 Módulo de Administración

Este módulo también se conforma de un solo programa, el cual realiza operaciones de altas, bajas, visualización y modificaciones de registros en un archivo de datos sin formato. Cada uno de estos registros contiene los datos concernientes a una cuenta de usuario. Dada la naturaleza de estos movimientos, sólo un usuario con cuenta de administrador podrá contar con los permisos de ejecución sobre este programa.

El código de fuente es relativamente corto, por lo cual se muestra el programa completo a continuación:

```
$cat sast_admin.c

/* Programa: sast_admin.c */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define MAXLEN        200
#define MAX_USERS      50
#define TERM_FILE     "system/terminal.txt"
#define USERS_DATA_BASE "system/password"
#define TEMP_DATA_BASE  "system/aux.dat"

#define ON  1
#define OFF 0

typedef struct {

  char login[15];
  char password[15];
  char templateAXE[30];
  char templateS12[30];
  char template5ESS[30];
  char templatePTS[30];
```

```
  char templateDMS[30];
  char templateBOSTON[30];
  char templateDISP1[30];
  char templateDISP2[30];
  char macros[30];

} SAST_USER;

void read_sast_user();
void agregar_sast_user();
void list_sast_user();
void eliminar_sast_user();
void visualizar_sast_user();
void modificar_sast_user();
int  existe_sast_user();
void eco();
void menu();
void header();

main()
{
 SAST_USER a;
 char op;

 do {

  menu();
  op = getchar();

  switch ( op )
  {
   case '1':
                agregar_sast_user();
                break;
   case '2':
                eliminar_sast_user();
                break;
   case '3':
                visualizar_sast_user();
                break;
   case '4':
                modificar_sast_user();
                break;
   case '5':
                list_sast_user();
                break;
  }

 } while ( op != '6' );

}
```

```
void agregar_sast_user()
{
   FILE *p;
   SAST_USER u;
   char op;

   system("clear");
   header();

   if ( (p=fopen(USERS_DATA_BASE, "a")) == NULL )
   {
    perror("abriendo archivo de usuarios");
    exit(0);
   }

read_sast_user(&u);

   printf("\n\t\tAgregar usuario %s (s/n) ? ", u.login);
   do
    op = tolower(op=getchar());
   while ( op != 's' && op != 'n' );

   if ( op == 's' )
    fwrite(&u,sizeof(SAST_USER),1,p);

   fclose(p);
}

void eliminar_sast_user()
{
   FILE     *p, *aux;
   SAST_USER u;
   SAST_USER tabla[MAX_USERS];
   char     remov_user[30];
   int      j, k = 0;
   int      user_flag;
   char     s[10];
   char     op;

   system("clear");
   header();

   if ( (p=fopen(USERS_DATA_BASE, "r")) == NULL )
   {
    perror("abriendo archivo de usuarios");
    exit(0);
   }

   printf("\n");
```

```
printf("\n\n\t\t  Usuario a remover: ");
fflush(stdin);
gets(remov_user);

user_flag = 0;
fread(&u,sizeof(SAST_USER),1,p);
while ( !feof(p) )
{
 if ( strncmp(u.login, remov_user, strlen(u.login) ) != 0 )
   tabla[k++] = u;
 else
     user_flag = 1;
 fread(&u,sizeof(SAST_USER),1,p);
}

fclose(p);

if ( user_flag != 0 )
{

 printf("\n\n\t\t  Remover usuario %s (s/n) ? ", remov_user);
 do
     op = tolower(op=getchar());
 while ( op != 's' && op != 'n' );

 if ( op == 's' )
 {

  if ( (aux=fopen(TEMP_DATA_BASE, "w")) == NULL )
  {
   perror("abriendo archivo de usuarios");
   exit(0);
  }

  for ( j=0; j<k; j++)
   fwrite(&tabla[j],sizeof(SAST_USER),1,p);

  fclose(aux);

  remove(USERS_DATA_BASE);
  rename(TEMP_DATA_BASE, USERS_DATA_BASE);

  printf("\n\n\t\t  Usuario %s removido.\n", remov_user);
 }
 else
  printf("\n\n\t\t  Usuario %s no removido.\n", remov_user);
}
else
{
 printf("\n\n\t\t  No se encontro usuario %s.\n", remov_user);
}
```

```
    printf("\n\n\t  Presione <ENTER> para continuar...");
    fflush(stdin);
    gets(s);
}

void visualizar_sast_user()
{

    FILE     *p;
    SAST_USER u;
    SAST_USER tabla[MAX_USERS];
    char     view_user[30];
    int      j, k = 0;
    int      user_flag;
    char     s[10];

    system("clear");
    header();

    if ( (p=fopen(USERS_DATA_BASE, "r")) == NULL )
    {
     perror("abriendo archivo de usuarios");
     exit(0);
    }

    printf("\n");
    printf("\n\n\t\t  Usuario a visualizar: ");
    fflush(stdin);
    gets(view_user);

    user_flag = 0;
    fread(&u,sizeof(SAST_USER),1,p);
    while ( !feof(p) )
    {
     if ( strncmp(u.login, view_user, strlen(u.login) ) == 0 )
     {
         user_flag = 1;
         break;
     }
     fread(&u,sizeof(SAST_USER),1,p);
    }

    fclose(p);

    if ( user_flag != 0 )
    {
     system("clear");
     header();
```

```
    printf("\n");
    printf("\n\t\t     Clave de usuario: %s\n", u.login);
    printf("\n\t\t           Password: %s\n", "**********");
    printf("\n\t\t     << Permisos por Tecnologia >>\n");
    printf("\n\t\t       AXE: %s", u.templateAXE);
    printf("\n\t\t       S12: %s", u.templateS12);
    printf("\n\t\t      5ESS: %s", u.template5ESS);
    printf("\n\t\t       PTS: %s", u.templatePTS);
    printf("\n\t\t     MACROS: %s", u.macros);
  }
  else
  {
   printf("\n\n\t\t  No se encontro usuario %s.\n", view_user);
  }


  printf("\n\n\t  Presione <ENTER> para continuar...");
  fflush(stdin);
  gets(s);
}

void modificar_sast_user()
{

  FILE    *p, *aux;
  SAST_USER u;
  SAST_USER tabla[MAX_USERS];
  char    modify_user[30];
  int     j, k = 0;
  int     user_flag;
  int     modify_flag;
  char    s[10];
  char    op;

  system("clear");
  header();

  if ( (p=fopen(USERS_DATA_BASE, "r")) == NULL )
  {
   perror("abriendo archivo de usuarios");
   exit(0);
  }

  printf("\n");
  printf("\n\n\t\t  Usuario a modificar: ");
  fflush(stdin);
  gets(modify_user);

  user_flag = 0;
  fread(&u,sizeof(SAST_USER),1,p);
  while ( !feof(p) )
```

```
{
 if ( strncmp(u.login, modify_user, strlen(u.login) ) == 0 )
 {
     user_flag = 1;
     break;
 }
 else
     tabla[k++] = u;
 fread(&u,sizeof(SAST_USER),1,p);
}

fclose(p);

if ( user_flag != 0 )
{
 do {

   modify_flag = 1;

     if ( (aux=fopen(TEMP_DATA_BASE, "w")) == NULL )
     {
       perror("abriendo archivo de usuarios");
       exit(0);
   }

   system("clear");
   header();

   printf("\n");
   printf("\n\t\t     Clave de usuario: %s\n", u.login);
   printf("\n\t\t(1)          Password: %s\n", "**********");
   printf("\n\t\t     << Permisos por Tecnologia >>\n");
   printf("\n\t\t(2)      AXE: %s", u.templateAXE);
   printf("\n\t\t(3)      S12: %s", u.templateS12);
   printf("\n\t\t(4)     5ESS: %s", u.template5ESS);
   printf("\n\t\t(5)      PTS: %s", u.templatePTS);
   printf("\n\t\t(6)   MACROS: %s", u.macros);
   printf("\n");
   printf("\n\t\t(7)      Cancelar.");
   printf("\n\t\t(8)      Terminar.");

   printf("\n\n\t\tNumero de opcion a modificar? ");

   do
       op = getchar();
   while ( !isdigit(op) );

   switch ( op )
   {
       case '1':
      do {
```

```
        memset(u.password, '\0', 15);
        printf("\n\t\t(1)          Password: ");
          fflush(stdin);
          eco(OFF);
          gets(u.password);
          eco(ON);

        if ( strlen(u.password)<=0 || strlen(u.password)>10 )
        {
          printf("\n\n\t\tLongitud de clave de seguridad incorrecta.\n");
          printf("\n\t\tPresione <ENTER> para continuar...\n");
          fflush(stdin);
          gets(s);

          system("clear");
          header();
        }
      } while ( strlen(u.password)<=0 || strlen(u.password)>10 );
          break;
       case '2':
      printf("\n\t\t(2)      AXE: ");
          fflush(stdin);
          gets(u.templateAXE);
       break;
       case '3':
      printf("\n\t\t(3)      S12: ");
          fflush(stdin);
          gets(u.templateS12);
          break;
       case '4':
      printf("\n\t\t(4)     5ESS: ");
          fflush(stdin);
          gets(u.template5ESS);
          break;
       case '5':
      printf("\n\t\t(5)      PTS: ");
          fflush(stdin);
          gets(u.templatePTS);
          break;
       case '6':
      printf("\n\t\t(6)   MACROS: ");
          fflush(stdin);
          gets(u.macros);
          break;
       case '7':
          modify_flag = 0;
          break;
      }
} while ( op != '7' && op != '8' );

if ( modify_flag )
```

```
    {
      for ( j=0; j<k; j++)
        fwrite(&tabla[j],sizeof(SAST_USER),1,aux);

      /* Agregar usuario modificado */
      fwrite(&u,sizeof(SAST_USER),1,aux);

      fclose(aux);

      remove(USERS_DATA_BASE);
      rename(TEMP_DATA_BASE, USERS_DATA_BASE);

      printf("\n\n\t\t  Usuario %s modificado.\n", modify_user);
    }
    else
      printf("\n\n\t\t  Usuario %s no modificado.\n", modify_user);
  }
  else
  {
    printf("\n\n\t\t  No se encontro usuario %s.\n", modify_user);
    printf("\n\n\tPresione <ENTER> para continuar...\n");
    fflush(stdin);
    gets(s);
  }

 return;
}

void read_sast_user(u)
SAST_USER *u;
{
 char s[10];

 do {
   memset(u->login, '\0', 15);
   printf("\n");
   printf("\n\t\t     Clave de usuario: ");
   fflush(stdin);
   gets(u->login);

   if ( strlen(u->login)<=0 || strlen(u->login)>10 )
   {
     printf("\n\n\t\tLongitud de clave de usuario incorrecta.\n", u->login);
     printf("\n\t\tPresione <ENTER> para continuar...\n");
     fflush(stdin);
     gets(s);

     system("clear");
     header();
   }
```

```
    if ( existe_sast_user(u->login) )
    {
      printf("\n\n\t\tEl usuario %s ya existe en el sistema.\n", u->login);
      printf("\n\t\tPresione <ENTER> para continuar...\n");
      fflush(stdin);
      gets(s);

      system("clear");
      header();
    }
  } while ( existe_sast_user(u->login) || strlen(u->login) <= 0 ||
        strlen(u->login) > 10 );

  do {
    memset(u->password, '\0', 15);
    printf("\n\t\t          Password: ");
    fflush(stdin);
    eco(OFF);
    gets(u->password);
    eco(ON);

    if ( strlen(u->password)<=0 || strlen(u->password)>10 )
    {
      printf("\n\n\t\tLongitud de clave de seguridad incorrecta.\n");
      printf("\n\t\tPresione <ENTER> para continuar...\n");
      fflush(stdin);
      gets(s);

      system("clear");
      header();
    }

  } while ( strlen(u->password)<=0 || strlen(u->password)>10 );
  printf("\n\t\t    << Permisos por Tecnologia >>\n");
  printf("\n\t\t      AXE: ");
  fflush(stdin);
  gets(u->templateAXE);
  printf("\n\t\t      S12: ");
  gets(u->templateS12);
  fflush(stdin);
  printf("\n\t\t     5ESS: ");
  fflush(stdin);
  gets(u->template5ESS);
  printf("\n\t\t      PTS: "),
  fflush(stdin);
  gets(u->templatePTS);
  printf("\n\t\t    MACROS: ");
  fflush(stdin);
  gets(u->macros);
  return;
}
```

```
int existe_sast_user(username)
char *username;
{
   FILE *p;
   SAST_USER u;
   int  k = 1;
   char s[10];

   if ( (p=fopen(USERS_DATA_BASE, "r")) == NULL )
   {
    perror("abriendo archivo de usuarios");
    exit(0);
   }

   fread(&u,sizeof(SAST_USER),1,p);
   while ( !feof(p) )
   {
    if ( strncmp(username, u.login, strlen(u.login) ) == 0 )
    {
        fclose(p);
        return(1);
    }
    fread(&u,sizeof(SAST_USER),1,p);
   }

   fclose(p);

   return(0);

}

void list_sast_user()
{
   FILE *p;
   SAST_USER u;
   int  k = 1;
   char s[10];

   system("clear");
   header();

   if ( (p=fopen(USERS_DATA_BASE, "r")) == NULL )
   {
    perror("abriendo archivo de usuarios");
    exit(0);
   }

   printf("\n\n");

   fread(&u,sizeof(SAST_USER),1,p);
   while ( !feof(p) )
```

```
    {
      printf("\t\t\t%d. %s\n", k++, u.login );
      fread(&u,sizeof(SAST_USER),1,p);
    }

    fclose(p);

    printf("\n\tPresione <ENTER> para continuar...\n");
    fflush(stdin);
    gets(s);
}

void menu()
{
        system("clear");

        header() ;
        printf("\n                    S A S T   A D M I N\n");
        printf("\n                  (1) - Agregar un usuario.");
        printf("\n                  (2) - Eliminar un usuario.");
        printf("\n                  (3) - Visualizar un usuario") ;
        printf("\n                  (4) - Modificar un usuario.") ;
        printf("\n                  (5) - Listar todos los usuarios.") ;
        printf("\n                  (6) - Salir.");
        printf("\n");
        printf(
        "\n                              GCIA. d DESARROLLO d SOFTWARE.") ;
        printf("\n          ----------------------------------------------------------");
        printf("\n                                        Seleccione Opcion : ");
}

void header() {
char encabe[80];

        printf("\n     ") ;

        sprintf(encabe,"     TELEFONOS d MEXICO SA d CV.") ;
      printf("%s",encabe);
        printf("\n          ---------------------- ( SAST 1.00 ) ----------------------\n");
        printf("\n            ADMINISTRACION  SISTEMA AUTOMATICO d SINCRONIA d
TIEMPO") ;
        printf(
"\n          ----------------------------------------------------------\n");
        return ;
}

void eco(status)
int status;
{
  FILE *p;
  char readbuffer[MAXLEN], cmd[MAXLEN], *terminal;
```

```
 memset(cmd ,'\0', MAXLEN);
 sprintf(cmd, "who am i > %s\0", TERM_FILE);
 system(cmd);

 if ( (p=fopen(TERM_FILE, "r")) == NULL )
 {
  perror("abriendo archivo");
  exit(0);
 }

 fgets(readbuffer, MAXLEN, p);
 while ( !feof(p) )
 {
  if ( strlen(readbuffer) > 0 ) break;
  fgets(readbuffer, MAXLEN, p);
 }

 terminal = strtok(readbuffer, " \t");
 terminal = strtok(NULL,       " \t");

 memset(cmd ,'\0', MAXLEN);

 if ( status == OFF )
  sprintf(cmd, "stty -echo < /dev/%s\0", terminal);
 else if ( status == ON )
  sprintf(cmd, "stty  echo < /dev/%s\0", terminal);

 system(cmd);
 fclose(p);
 return;
}
```

Una vez concluida la implementación del sistema, daremos paso al proceso de pruebas, liberación y administración del mismo, según se verá en el siguiente capítulo.

VII. PRUEBAS, LIBERACIÓN Y ADMINISTRACIÓN DEL SISTEMA

En este capítulo describiremos el conjunto de pruebas aplicadas a los módulos de programa ya terminados, el proceso de liberación del sistema en el ámbito nacional y los procedimientos de administración y mantenimiento posterior que se le darán al mismo.

7.1 Pruebas al Sistema

Antes de determinar el método de prueba de *software* que pudiera aplicar mejor al sistema que hemos desarrollado a lo largo de este trabajo, presentaremos un resumen de los distintos métodos que existen, y finalmente tomaremos una decisión.

7.1.1 Métodos de Prueba de Software

- Prueba de la Caja Blanca

La prueba de caja blanca, denominada a veces prueba de caja de cristal, es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca, el ingeniero de *software* puede tener casos de prueba que (1) garantizan que se ejercite por lo menos una vez todos los caminos independientes de cada módulo; (2) que se ejerciten todas las decisiones lógicas de sus vertientes verdadera y falsa; (3) que se ejecuten todos los bucles en sus límites y con sus límites operacionales, y (4) que se ejerciten las estructuras internas de datos para asegurar su validez.

Dado lo anterior, se puede plantear una pregunta razonable. ¿Por qué emplear tiempo y energía preocupándose de (y probando) los detalles lógicos cuando podríamos emplear mejor el esfuerzo que se han alcanzado los requisitos del programa? La respuesta se encuentra en la naturaleza misma de los defectos del software:

a) Los errores lógicos y las suposiciones incorrectas son inversamente proporcionales a la probabilidad de que se ejecute un camino del programa. Los errores tienden a introducirse en nuestro trabajo cuando diseñamos e implementamos funciones, condiciones o controles que se encuentran fuera de lo normal. El procedimiento habitual tiende a hacerse más comprensible (y bien examinado), mientras que el procesamiento de "casos especiales" tiende a caer en el caos.

b) A menudo creemos que un camino lógico tiene pocas posibilidades de ejecutarse cuando, de hecho, se puede ejecutar de forma normal. El flujo lógico de un programa a veces no es nada intuitivo, lo que significa que nuestras suposiciones intuitivas sobre el flujo de control y los datos nos pueden llevar a tener errores de diseño que sólo se descubren cuando comienza la prueba de ese camino.

c) Los errores tipográficos son aleatorios. Cuando se traduce un programa a código fuente en un lenguaje de programación, es muy probable que se den algunos errores de escritura. Muchos serán descubiertos por los mecanismos de control de sintaxis, pero otros permanecerán sin detectar hasta que comience la prueba. Es igual de probable que haya un error tipográfico en un oscuro camino lógico que en un camino principal.

Cada una de estas razones nos da un argumento para llevar a cabo las pruebas de caja blanca.

- Prueba del Camino Básico

La técnica del camino básico es una técnica de caja blanca propuesta inicialmente por Tom McCabe. El método de camino básico permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución.

Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

- Prueba de la Estructura de Control

La técnica de prueba del camino básico descrita anteriormente es una de las muchas técnicas para las pruebas de estructura de control. Aunque la prueba del camino básico es sencilla y altamente efectiva, no es suficiente por si sola, ya que necesita de la realización de varias pruebas, como lo son la prueba de condición, la prueba de flujo de datos y la prueba de bucles.

- Prueba de la Caja Negra

Las pruebas de la caja negra se centran en los requisitos funcionales del *software*. O sea, la prueba de la caja negra permite al Ingeniero de Software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. La prueba de la caja negra no es una alternativa a las técnicas de prueba de caja blanca. Más bien se trata de un enfoque complementario que intenta descubrir diferentes tipos de errores que los métodos de caja blanca.

La prueba de la caja negra intenta encontrar errores de las siguientes categorías: (1) funciones incorrectas o ausentes, (2) errores de interfaz, (3) errores en estructuras de datos o en acceso a bases de datos externas, (4) errores de rendimiento y (5) errores de inicialización y terminación.

A diferencia de la prueba de caja blanca, que se lleva a cabo previamente en el proceso de prueba, la prueba de caja negra tiende a aplicarse durante fases posteriores a la prueba. Ya que la prueba de caja negra ignora intencionalmente la estructura de control, centra su atención en el campo de la información. Las pruebas se diseñan para responder a las siguientes preguntas:

- ¿Cómo se prueba la validez funcional?
- ¿Qué clases de entrada compondrán unos buenos casos de prueba?
- ¿Es el sistema particularmente sensible a ciertos valores de entrada?
- ¿De qué forma están aislados los límites de una clase de datos?
- ¿Qué volúmenes y niveles de datos tolerara el sistema?
- ¿Qué efectos sobre la operación del sistema tendrán combinaciones específicas de datos?

Mediante las pruebas de la técnica de la caja negra se obtiene un conjunto de casos de prueba que satisfacen los siguientes criterios: (1) casos de prueba que reducen, en un

coeficiente que es mayor que uno, el número de casos de prueba adicionales que se deban diseñar para alcanzar una prueba razonable (2) casos de prueba que nos dicen algo sobre la presencia o ausencia de clases de errores en lugar de errores asociados solamente con la prueba que estamos realizando.

- Pruebas de Entornos y Aplicaciones Especializadas

A medida que el software se ha hecho más complejo, ha crecido la necesidad de enfoques de pruebas especializadas. Los métodos de prueba de caja blanca y de caja negra tratados con anterioridad son aplicables a todos los entornos, arquitecturas y aplicaciones pero a veces se necesitan unas directrices y enfoques únicos para las pruebas. Estos métodos de pruebas para entornos especiales se enlistan a continuación:

- Prueba de Interfaces Gráficas de Usuario

Las Interfaces Gráficas de Usuario (IGU) presentan interesantes desafíos para los ingenieros de software. Debido a los componentes reutilizables provistos como parte de los entornos de desarrollo de las IGU, la creación de la interfaz de usuario se ha convertido en menos costosa en tiempo y más exacta. Al mismo tiempo, la complejidad de las IGU ha aumentado, originando más dificultad en el diseño y ejecución en los casos de prueba.

Dado que las IGU modernas tienen la misma apariencia y filosofía, se puede obtener una serie de pruebas estándar. Las pruebas de las IGU se centran en la correcta operación de las ventanas, de los menús, de las operaciones con el ratón y de la entrada de datos.

- Pruebas de documentación y de ayuda

El término "prueba de software" hace imaginarnos gran cantidad de casos de prueba preparados para ejecutar programas de computadora y los datos que manejan. Es importante darse cuenta de que la prueba debe extenderse al tercer elemento de la configuración del software: la documentación.

Los errores en la documentación pueden ser tan destructivos para la aceptación del programa como los errores en los datos o en el código fuente. Nada es más frustrante que seguir fielmente el manual de usuario y obtener resultados o comportamientos que no coinciden con los anticipados con el documento. Por esta razón la prueba de documentación debería ser una parte importante de cualquier plan de pruebas de software.

La prueba de documentación se puede enfocar en dos fases. La primera fase, la revisión técnica formal, examina el documento para comprobar la claridad editorial, la segunda la prueba en vivo, utiliza la documentación junto al uso del programa real. El manejo del programa se sigue entonces con la documentación intentando responder las siguientes interrogantes.

- ¿Describe con exactitud la documentación como seguir cada modo de empleo?
- ¿Es exacta la descripción de cada secuencia de interacción?
- ¿Son exactos los ejemplos?

- ¿Son consistentes con el programa real la terminología, las descripciones del menú y las respuestas del sistema?
- ¿Es relativamente fácil localizar ayuda en la documentación?
- ¿Se pueden solucionar problemas fácilmente con la documentación?
- ¿Son exactos y completos la tabla de contenido y el índice?
- ¿Facilita el diseño del documento (distribución, tipos de letra, indentación, gráficas) la compresión y rápida asimilación de la información?
- ¿Están descritos con gran detalle los mensajes de error para el usuario en el documento?
- Si se utilizan enlaces de hipertexto ¿son exactos y completos?

La única manera viable es que un equipo independiente (por ejemplo, usuarios seleccionados) pruebe la documentación en el contexto del empleo del programa. Se anotan todas las discrepancias y se definen las áreas de ambigüedad o puntos débiles del documento para reescribirlas.

- Prueba de sistemas en tiempo real

La naturaleza asíncrona y dependiente de muchas aplicaciones de tiempo añade un nuevo y potencialmente difícil elemento a la complejidad de las pruebas: el tiempo. El responsable del diseño de casos de prueba, no sólo tiene que considerar los casos de prueba de caja blanca y de caja negra, sino también el tratamiento de sucesos (por ejemplo, proceso de interrupciones), la temporización de los datos y el paralelismo de las tareas (procesos) que manejan los datos. En muchas ocasiones, los datos de prueba proporcionados al sistema de tiempo real cuando se encuentran en un determinado estado darán un proceso correcto, mientras que al proporcionárselos en otro estado llevarán a un error.

Además la estrecha relación que existe entre el software de tiempo real y su entorno de *hardware* también puede introducir problemas en la prueba. Las pruebas de software deben considerar el impacto de los fallos de *hardware* sobre el proceso del software. Puede ser muy difícil simular de forma realista esos fallos.

Todavía han de evolucionar mucho los métodos generales de diseño de casos de prueba para sistema de tiempo real. Sin embargo, se puede proponer una estrategia en cuatro pasos:

a) Prueba de tareas. El primer paso de la prueba de sistemas en tiempo real consiste en probar cada tarea en forma independiente. Es decir, se diseñan pruebas de caja blanca y de caja negra, y se ejecutan para cada tarea. Durante estas pruebas, cada tarea se ejecuta independientemente. La prueba de la tarea ha de descubrir errores en la lógica y el funcionamiento, pero no los errores de temporización o de comportamiento.

b) Prueba de comportamiento. Utilizando modelos del sistema creados con herramientas CASE (*Computer Assisted Software Engineering*, Ingeniería de Software Asistida por Computadora) es posible simular el comportamiento del sistema en tiempo real y examinar su comportamiento como consecuencia de sucesos externos. Estas actividades de análisis como base del diseño de casos de prueba que se llevan a cabo cuando se ha construido el software de

tiempo real. Utilizando una técnica parecida a la partición equivalente, se puede categorizar los sucesos (por ejemplo: interrupciones, señales de control, datos) para la prueba. Por ejemplo, los sucesos para la fotocopiadora pueden ser interrupciones del usuario (por ejemplo: reinicialización del contador), interrupciones mecánicas (por ejemplo: atasco de papel), interrupciones del sistema (por ejemplo: bajo nivel de tinta) y modos de fallo (por ejemplo: rodillo excesivamente caliente). Se prueba cada uno de esos sucesos individualmente y se examina el comportamiento del sistema ejecutable para detectar errores que se produzca como consecuencia del proceso asociado a esos sucesos. Se puede comparar el comportamiento del modelo del sistema (desarrollado durante el análisis) con el *software* ejecutable para ver si existe concordancia. Una vez que se ha probado cada clase de sucesos, al sistema se le presentan sucesos en un orden aleatorio y con una frecuencia aleatoria. Se examina el comportamiento del software para detectar errores de comportamiento.

c) Pruebas de intertareas. Una vez que se han aislados los errores en las tareas individuales y en el comportamiento del sistema las pruebas se dirigen hacia los errores relativos al tiempo. Se prueban las tareas asíncronas que se sabe que comunican con otras, con diferentes tazas de datos y cargas de procesos para determinar si se producen errores de sincronismos entre las tareas. Además se prueban las tareas que se comunican mediante colas de mensajes o almacenes de datos, para detectar errores en el tamaño de esas zonas de almacenamiento de datos.

d) Prueba del sistema. El *software* y *hardware* están integrados, por lo que se lleva a cabo una serie completa de pruebas del sistema para intentar descubrir errores en la interfaz *software – hardware.*

La mayoría de los sistemas en tiempo real procesan interrupciones. Por lo tanto, es esencial la prueba del manejo de estos sucesos lógicos. Usando el diagrama estado – transición y la especificación de control, el responsable de la prueba desarrolla una lista de todas las posibles interrupciones y del proceso que ocurre como consecuencia de la interrupción. Se diseñan después pruebas para valorar las siguientes características del sistema:

- ¿Se han asignado y gestionado apropiadamente las prioridades de interrupción?
- ¿Se gestiona correctamente el procesamiento para todas las interrupciones?
- ¿Se ajusta a los requisitos el rendimiento (por ejemplo: tiempo de proceso) de todos los procedimiento de gestión de interrupciones?
- ¿Crea problemas de funcionamiento o rendimiento la llegada de gran volumen de interrupciones en momentos críticos?

Además se deberían de probar las áreas de datos globales que se usan para transferir información como parte del proceso de una interrupción para valorar el potencial de generación de efectos colaterales.

- Pruebas de Arquitectura Cliente/Servidor

Las arquitecturas cliente/servidor representan un desafío significativo para los responsables de la prueba de software. La naturaleza distribuida de los entornos

cliente/servidor, los aspectos de rendimiento asociados con el proceso de transacciones, la presencia potencia de diferentes plataformas de *hardware* y *software*, las complejidades de las comunicaciones de red, la necesidad de servir a múltiples clientes desde una base de datos centralizada (o en algunos casos distribuida) los requisitos de coordinación impuestos al servidor se combinan todos para hacer las pruebas de las arquitecturas cliente/servidor y del software residente en ellas, considerablemente más difíciles que la prueba de aplicaciones individuales. De hecho, los estudios recientes sobre la industria indican un significativo aumento en el tiempo invertido y los costos de la prueba cuando se desarrollan entornos cliente/servidor.

La naturaleza distribuida de los sistemas cliente/servidor plantea un conjunto de problemas específicos para los encargados de la prueba de software, por lo cual se sugiere poner mayor énfasis en las siguientes zonas de interés:

- Consideraciones de la Interfaz Gráfica de Usuario del Cliente
- Consideraciones de la diversidad de Plataformas de *hardware* y *software*
- Consideraciones de Bases de Datos Distribuida
- Consideraciones de Procesamiento en Paralelo
- Consideraciones de que el extremo remoto pudiera no ser robusto
- Consideraciones en los Protocolos de Comunicación
- Consideraciones de ambientes multiusuarios

En general, la comprobación de software cliente/servidor se produce en tres niveles diferentes: (1) las aplicaciones del cliente individuales se prueban en modo "desconectado" (el funcionamiento del servidor y de la red subyacente no se consideran); (2) las aplicaciones de software de cliente y del servidor se prueban de manera simultánea, pero no se ejercitan las operaciones de red; (3) se comprueba la arquitectura completa de cliente/servidor, incluyendo el rendimiento de funcionamiento de la red.

Frecuentemente se encuentran los siguientes enfoques de comprobación para arquitecturas cliente/servidor:

a) Comprobaciones de Función de Aplicación. Se comprueba la funcionalidad de las aplicaciones de cliente empleando los métodos descritos anteriormente. En esencia, la aplicación se comprueba en solitario en un intento de descubrir errores en su funcionamiento.

b) Comprobaciones de servidor. Se comprueban la coordinación y las funciones de gestión de datos del servidor.

c) Comprobaciones de Bases de Datos. Se comprueba la integridad y precisión de los datos almacenados en el servidor. Se examinan las transacciones entre las aplicaciones cliente y las aplicaciones del servidor para asegurar que los datos se almacenen, actualicen y recuperen adecuadamente.

d) Comprobaciones de transacciones. Se cree una serie de comprobaciones adecuada para comprobar que todas las clases de transacciones se procesen de acuerdo con los requisitos. Las comprobaciones hacen especial hincapié en a corrección de procesamiento, y también en los temas de rendimiento (p. ej.:

tiempo de procesamiento de transacciones y comprobación de volúmenes de transacción).

e) Comprobaciones de comunicación a través de la red. Estas comprobaciones verifican que la comunicación entre los nudos de la red se produzcan correctamente, y que el paso de mensaje, las transacciones y el tráfico de red relacionado tenga lugar sin errores. También se pueden efectuar comprobaciones de seguridad de la red como parte de esta actividad de comprobación.

Luego de haber analizado los distintos métodos que existen para prueba de software, encontramos que nuestro sistema puede considerarse como una aplicación especializada, y que precisamente existe un método de pruebas orientado a arquitecturas cliente/servidor, esquema bajo el cual se encuentra diseñado el SAST. En conclusión, el método de prueba que vamos a utilizar es el método de prueba de arquitecturas cliente/servidor, desde un punto global del sistema; y además, aplicaremos técnicas de caja negra para probar la funcionalidad de cada módulo del sistema de manera individual.

7.1.2 Pruebas aplicadas al módulo de muestreo

Las pruebas que se efectuaron para verificar la correcta operación del módulo de muestreo fueron las siguientes:

a) Se comprobó que el script de muestreo y sincronización se encontrara activo en la central correspondiente, por el canal adecuado del NMA y para el tipo de tecnología de central correcto.
b) Se comprobó que los scripts se encontraran calendarizados en el NMA para ejecutarse de manera periódica (cada hora) sobre cada una de las centrales conectadas al NMA.
c) Se comprobó que el script enviara el comando correcto a la central telefónica para el muestreo de la hora de sistema, según el tipo de tecnología de que se trate.
d) Se comprobó que el script enviara el comando correcto a la central telefónica para el ajuste de la hora de sistema, según el tipo de tecnología de que se trate.
e) Se comprobó que el script obtuviera la hora de sistema del NMA (hora de referencia) de manera correcta.
f) Se comprobó que el script obtuviera la hora de sistema de la central telefónica de manera correcta.
g) Se comprobó que el script calculara la desviación de tiempo (hora de referencia - hora de la central) de manera correcta.
h) Se comprobó que el script conformara los registros de muestras de tiempo en el formato especificado.
i) Se comprobó que el script direccionara los registros de muestras de tiempo por el socket especificado y de manera correcta.

7.1.3 Pruebas aplicadas al módulo de recolección

Las pruebas aplicadas al módulo de recolección se enlistan a continuación:

a) Se comprobó que la conexión hacia el socket del módulo de muestreo se estableciera de manera exitosa.
b) Se verificó la correcta recepción de los registros de tiempo a través del socket.

c) Se verificó que los registros de tiempo recibidos por el socket contuvieran información coherente y en el formato especificado.
d) Se verificó que los registros de tiempo leídos por el socket se escribieran de manera correcta al archivo de datos de la división geográfica correspondiente.
e) Se verifico que los comandos de desbloqueo del puerto del NMA llegasen hasta el módulo de comunicaciones y se ejecutaran de manera exitosa.
f) Se comprobó que la conexión hacia el socket de recepción se reestableciera después de los comandos de desbloqueo, y en caso negativo, que estos comandos se enviaran nuevamente.

7.1.4 Pruebas aplicadas al módulo de monitoreo en línea

Las pruebas ejecutadas sobre módulo de monitoreo en línea fueron:

a) Se comprobó que existan registros de tiempo recientes en el archivo de datos de la división geográfica correspondiente.
b) Se comprobó que cada uno de los registros que aparecen en esta pantalla de monitoreo en línea correspondiera con la realidad. Se pueda acceder a la central telefónica de manera manual a través de una pantalla del NMA para ver su hora de sistema y compararla con la que se muestra en la pantalla de monitoreo en línea y de esta manera determinar si la información que nos presenta el sistema es confiable.
c) Se comprobó la posibilidad de poder navegar a través de la pantalla de monitoreo en línea empleando las opciones 'I' para ir al inicio del reporte, 'F' para saltar al final del reporte, 'S' para pasar al bloque siguiente de centrales, y 'A' para regresar al ver las centrales del bloque anterior.
d) Se comprobó que la opción 'E' para la generación de muestras instantáneas de tiempo funcionara correctamente.
e) Se comprobó que la opción 'F' para la sincronización de la hora de la central funcionara correctamente.

7.1.5 Pruebas aplicadas al módulo de muestreo y sincronización manual

Las pruebas que se aplicaron al módulo de muestreo y sincronización manual son las que siguen:

a) Se verificó que los datos proporcionados por el usuario en la pantalla de captura de generación de muestras de tiempo de manera manual, fueran coherentes.
b) Se verificó que los datos proporcionados por el usuario, en la pantalla de captura de sincronización de tiempo de manera manual, fueran coherentes.
c) Se comprobó que la conexión con el socket de servicio del módulo de comunicaciones se estableció exitosamente.
d) Se comprobó que el registro tipo movimiento se estructuró de manera correcta.
e) Se comprobó que el registro tipo movimiento se enviara por el socket con éxito.
f) Se comprobó que la pantalla de monitoreo en línea realizó el refrescamiento de pantalla después de cada movimiento que hace el usuario, o después de cada 10 segundos en caso de inactividad.

7.1.6 Pruebas aplicadas al módulo de Generación de Reportes

Dentro de los reportes entregados por el sistema se verificaron los siguientes aspectos:

a) Se comprobó que la información que se muestra en el Reporte de Centrales en Sincronía correspondiera con la realidad. Comparando la desviación de tiempo que se muestra en el registro de cada central con la hora de sistema que se obtiene al accesar a esa misma central a través del NMA.
b) Se comprobó que la información que se muestra en el Reporte de Centrales Fuera de Sincronía correspondiera con la realidad. Se comparó la desviación de tiempo que se muestra en el registro de cada central con la hora de sistema que se obtiene al accesar a esa misma central a través del NMA.
c) Se comprobó que la información que se muestra en el Reporte de Eventos por Central y Fecha se desplegara de manera correcta. Se ejecutaron varios movimientos sobre una central en específico para posteriormente verificar que se vean reflejados en el reporte.
d) Se comprobó que la información que se muestra en el Reporte de Centrales con Muestreo correspondiera con la realidad. Se ejecutaron varias operaciones de muestreo de tiempo sobre una central en específico, que en ese momento no contaba con muestras recientes, para posteriormente verificar que se vieran reflejados en este reporte.
e) Se comprobó que la información que se muestra en el Reporte de Centrales sin Muestreo correspondiera con la realidad. Consultando el Reporte de Eventos por Central y Fecha en alguna central en específico, verificando que en efecto no se encuentran registros de tiempo recientes.
f) Se comprobó que todos los movimientos ejecutados por los usuarios quedarán registrados en el archivo de bitácora del sistema, verificando que todos los datos correspondieran fielmente con el movimiento realizado. Una forma de hacer esto fue ejecutar una serie de movimientos de muestreo y sincronización sobre una central específica, e inmediatamente consultar la bitácora del sistema y asegurar que se registraron.

7.1.7 Pruebas aplicadas al módulo de comunicaciones

Para la comprobación de la correcta operación del módulo de comunicaciones fue necesario realizar el siguiente conjunto de pruebas:

- Programa Principal

a) Se comprobó que el programa se encontrara en ejecución. Para esta prueba se utilizó el comando **ps** de UNIX.
b) Se verificó que el programa creara el socket de manera exitosa y se encontrara en estado de escucha de peticiones de conexión (***LISTEN***), por el número de puerto predefinido. Se empleó el comando ***netstat -rn*** de UNIX.
c) Se comprobó que el programa aceptara peticiones de conexión de los programas cliente a través del socket; es decir, se verificó que cuando un programa cliente haciera una solicitud de conexión a este programa servidor, mediante la función ***connect***, se estableciera una conexión entre ambos puntos y el socket quedara en el estado ***ESTABLISHED***. También se puede emplear el comando ***netstat -rn***.
d) Se comprobó que el programa servidor recibiera los registros a través del socket y en el formato especificado, observando la pantalla donde se ejecuta este proceso o consultando el archivo de bitácora del sistema, donde se registran todos los movimientos.

- Rutina de Conexión a VOS

a) Se comprobó que el programa creara el socket de envío de comandos de manera exitosa y que se encontrara en estado de conexión (**ESTABLISHED**) hacia el NMA, por el socket número 23 del mainframe Stratus correspondiente (México o Guadalajara). Se utilizó el comando **netstat -rn** de UNIX.
b) Se verificó que la apertura de sesiones hacia VOS se realizará sólo si el nombre de usuario y la contraseña correspondientes fueran los correctos.
c) Se comprobó que esta rutina recibiera la respuesta proveniente del mainframe Stratus para cada comando enviado.

- Rutina de Envío de Comandos

a) Se comprobó que cada comando enviado por los programas cliente hacia el programa servidor, fuera enrutado hacia los equipos Stratus para su ejecución.
b) Se verificó que la función **send** se ejecutará sin retornar un código de error.

- Rutina de Postprocesamiento de Comandos

a) Se verificó que cada uno de los comandos enviados a los mainframe Stratrus fueran ejecutados con éxito.
b) Nos aseguramos que la respuesta a cada uno de los comandos fuera interpretada y los códigos de error fueran enviados de vuelta al programa cliente.

7.1.8 Pruebas aplicadas al módulo de construcción de mensajes

Se aseguró la correcta operación del módulo de construcción de mensajes mediante las siguientes pruebas.

a) Se comprobó que el programa construyera los mensajes de manera correcta para cada uno de los cuatro tipos de centrales (AXE, S-12, 5ESS, PTS).
b) Se comprobó que estos mensajes ya construidos sobre una cadena de caracteres, fueran depositados sobre el archivo de datos con el nombre especificado.

7.1.9 Pruebas aplicadas al módulo de administración

Las pruebas que se aplicaron al módulo de muestreo y sincronización manual fueron.

a) Se comprobó que los datos proporcionados por el operador en la pantalla de captura fueran consistentes y coherentes.
b) Se comprobó que las cuentas de usuario creadas con la opción de agregar un usuario se generaran de manera correcta.
c) Se comprobó que las cuentas de usuario creadas en la opción de agregar un usuario se hayan generado con el perfil de usuario y permisos correctos.
d) Se comprobó que las cuentas de usuario sean efectivamente canceladas mediante la opción de eliminar un usuario.
e) Se verificó que la información presentada en la opción de visualizar un usuario fuera la correcta.
f) Se comprobó que las modificaciones sobre las cuentas de usuario se realizaran de forma correcta.
g) Se comprobó que todos los usuarios fueran listados de forma correcta.

De manera inicial, las pruebas preliminares del Sistema Automático de Sincronía de Tiempo se realizaron sobre las centrales telefónicas de prueba (sin tráfico real), ubicadas en los laboratorios de Telmex en la Ciudad de México. Estos laboratorios cuentan con todas las condiciones con las que cuenta la planta telefónica; es decir, una central telefónica de prueba de cada tipo de tecnología (AXE, S-12, 5ESS, PTS, etc) con las últimas versiones de *software* instaladas en la planta y equipadas con todos sus módulos de *hardware*, incluyendo los módulos de recolección de facturación.

Posteriormente, una vez que se observó el comportamiento del sistema en ambiente de laboratorio, se realizaron una serie de pruebas piloto en centrales con tráfico real, con la presencia de personal responsable de la misma. Estas pruebas ejecutadas en vivo fueron las siguientes:

a) Habilitación del módulo de muestreo para la obtención de muestras de la hora de sistema de todas las centrales telefónicas conectadas al NMA.
b) Generación de muestras de tiempo y de forma manual de una central en específico con tipo de tecnología AXE.
c) Generación de muestras de tiempo y de forma manual de una central en específico con tipo de tecnología S-12.
d) Generación de muestras de tiempo y de forma manual de una central en específico con tipo de tecnología 5ESS.
e) Generación de muestras de tiempo y de forma manual de una central en específico con tipo de tecnología PTS.
f) Generación de muestras de tiempo y de forma automática de una central en específico con tipo de tecnología AXE.
g) Generación de muestras de tiempo y de forma automática de una central en específico con tipo de tecnología S-12.
h) Generación de muestras de tiempo y de forma automática de una central en específico con tipo de tecnología 5ESS.
i) Generación de muestras de tiempo y de forma automática de una central en específico con tipo de tecnología PTS.
j) Sincronización de una central en específico tecnología AXE.
k) Sincronización de una central en específico tecnología S-12.
l) Sincronización de una central en específico tecnología 5ESS.
m) Sincronización de una central en específico tecnología PTS.
n) Generación de Reporte de Centrales en Sincronía.
o) Generación de Reporte de Centrales Fuera de Sincronía.
p) Generación de Reporte de Movimientos por Central y Fecha.
q) Generación de Reporte de Centrales sin Muestreo.
r) Generación de Reporte de Centrales con Muestreo.
s) Revisión de la bitácora del sistema.
t) Observación de la Pantalla de Monitoreo en Línea.

Debido en gran parte a que el sistema completo se había probado en ambiente de laboratorio, todas las funcionalidades del sistema se ejecutaron de manera exitosa, presentándose únicamente pequeños contratiempos que fueron solucionados mediante mínimas adecuaciones, no sobre el código fuente del sistema, sino sobre los archivos de configuración del mismo. Estos pequeñas adecuaciones consistieron en lo siguiente:

a) Configuración de la dirección IP del mainframe Stratus que atendía a la división geográfica correspondiente.
b) Configuración del nombre del canal del NMA por el cual se envían los comandos por defecto.
c) Configuración de la ruta dentro del servidor UNIX, donde se encuentra el archivo de datos correspondiente a la división geográfica bajo monitoreo.

7.2 Liberación del Sistema

Una vez que todo el proceso de pruebas en su conjunto resultó satisfactorio, el siguiente paso consistió en la implantación gradual del sistema a nivel nacional. Debido a que el diseño del SAST no requiere de la instalación de programas cliente en la terminal del usuario, el proceso de liberación del consistió en el otorgamiento de cuentas de usuario del sistema, a los administradores de sistemas de cada uno de los CAR y el CNS, los cuales están ubicados en cada una de las divisiones geográficas en las ciudades de México, Puebla, Querétaro, Mérida, Guadalajara, Monterrey, Hermosillo y Chihuahua. Estos administradores (con cuentas de administrador en el SAST), pueden a su vez, otorgar cuentas de usuario normal a los técnicos especialistas de cada CAR y del CNS, los cuales accesarán al sistema desde sus estaciones de trabajo bajo plataforma UNIX o PC's con software de emulación de terminal, a través de la RCDT.

Dado que el personal usuario del sistema está ya familiarizado con el uso de sistemas de software para la operación, supervisión y mantenimiento de la planta telefónica, la capacitación de los mismos en el empleo del SAST se limitó a la distribución del manual de usuario (el cual se anexa en el apéndice al final de este trabajo) vía correo electrónico a los directivos y administradores de sistemas de los CAR y del CNS. Al mismo tiempo, el grupo de desarrollo del SAST brindó asesorías de manera telefónica a los usuarios en los casos que fue requerido.

7.3 Administración y Mantenimiento

El grupo de desarrollo del SAST y los administradores de sistemas de los CAR y el CNS, serán los encargado de la administración del sistema a nivel nacional. El grupo de desarrollo proveerá con cuentas de nivel de administrador a estos ingenieros en sistemas, quienes a un segundo nivel, se encargarán de la administración regional del sistema, atendiendo a toda la división geográfica a la que corresponden. Además de administrar las cuentas de usuario, también colaborarán en la capacitación de los especialistas en centrales telefónicas para el uso del SAST con base en el manual de operación elaborado por el grupo de desarrollo.

Las tareas de mantenimiento del sistema serán nuestra responsabilidad exclusiva, ya que requiere de un conocimiento integral del sistema a nivel de módulos de programa, archivos de datos, archivos de configuración y un dominio casi completo del sistema operativo UNIX. El mantenimiento del SAST consistirá fundamentalmente en las siguientes tareas:

- Mantenimiento de los *file systems* (sistemas de archivos), los cuales eventualmente se llegan a saturar al ir creciendo los archivos de datos, los archivos temporales y las bitácoras del sistema.

- Desbloqueo de terminales de sistema del NMA asociadas a sockets, las cuales se bloquean con frecuencia por inestabilidad de la red y provocan el corte de la comunicación entre procesos.

- Reestablecimiento de la comunicación mediante socket, los cuales a menudo se van a estado de desconexión por inestabilidad de la red o por inactividad prolongada.

- Administración de las cuentas de usuario, aunque solamente a nivel de administradores, ya que es tarea de estos últimos el control de las cuentas normales de usuario.

- Supervisión y levantamiento de los distintos procesos que conforman el sistema, ya que en ocasiones llegan a salirse de ejecución provocando graves problemas de operación en el mismo.

- Generación periódica de respaldos del sistema, lo cual implica el respaldo en cinta o en otro servidor del código fuente, de los archivos de datos, de los archivos de configuración y de las bitácoras del sistema.

- Desarrollo de adecuaciones o de nuevas funcionalidad que requiera el sistema.

- Asesorías rápidas vía telefónica a los usuarios en cuanto a la utilización del sistema.

- Generación de reportes especiales sobre la base de los archivos de datos del sistema.

Una vez que se ha concluido con toda la parte activa, en el desarrollo del sistema, en el siguiente capítulo presentaremos los resultados y conclusiones obtenidas del mismo.

VIII. RESULTADOS Y CONCLUSIONES

Una vez que se han concluido las etapas de análisis, diseño, implementación y liberación del Sistema Automático de Sincronía de Tiempo, analizaremos los resultados obtenidos y estableceremos cuales son las conclusiones finales del desarrollo de este trabajo.

8.1 Resultados

El Sistema Automático de Sincronía de Tiempo se ha convertido en una realidad. Actualmente se encuentra en operación a nivel nacional, monitoreando y ajustando la hora de sistema de las centrales telefónicas, de todas las divisiones geográficas que integran la planta telefónica de TELMEX.

El SAST fue liberado en un servidor *HP-9000*, ubicado en la ciudad de Monterrey, N.L., desde donde es accesado a través de la RCDT por personal del CNS y de los CAR, quienes son los encargados de explotar la información generada por el mismo; y en base a ella, ejecutar los comandos de sincronización necesarios. El SAST ha facilitado enormemente este proceso de sincronización de tiempo, ya que anteriormente los operadores dedicaban grandes espacios de tiempo monitoreando la hora de sistema de las centrales telefónicas y enviando los comandos de ajuste de tiempo (según su propio criterio), a través de distintas pantallas del NMA.

A través del SAST, un solo operador, sin conocimientos en el manejo del NMA ni de comandos de la central, puede monitorear y sincronizar todas las centrales telefónicas de una división geográfica en un tiempo promedio de treinta minutos, lo cual se traduce, en la posibilidad de poder dedicar al personal a actividades de mayor prioridad, por ejemplo, la atención de alarmas y soporte técnico.

Luego de alrededor de dos meses de haber entrado en operación, el SAST ha arrojado resultados alentadores. Por ejemplo, los reportes semanales de centrales fuera de sincronía indican que de un total de 558 centrales telefónicas, monitoreadas por el sistema, únicamente un promedio de 10 se encuentran fuera de sincronía en una muestra instantánea; es decir, fuera del intervalo de +/- 2 segundos con respecto a la hora de referencia. Además, este número promedio de centrales que se detectan fuera de sincronía, no se encuentran desviadas mas allá del rango de +/- 5 segundos.

Para hacer una evaluación más directa del sistema en cuanto a su desempeño, haremos un análisis del mismo tomando en cuenta los siguientes indicadores:

a) Estabilidad

   El sistema ha mostrado tener un comportamiento muy estable, ya que se ha mantenido en operación de manera continua, con pequeñas interrupciones generadas por causas externas al sistema, tales como mantenimiento de *hardware* en servidores, retardos en la red, y actualizaciones de versiones de sistema operativo en el NMA.

b) Confiabilidad

   Los reportes generados por el SAST han resultado ser 100% confiables (siempre y cuando la hora de los mainframe *Stratus* se encuentre en sincronía vía NTP), ya que en múltiples ocasiones se comprobó que la información mostrada en los mismos correspondiera con la realidad de la situación en la planta telefónica, a través de pruebas de verificación de la misma mediante pantallas del NMA.

c) Efectividad

La efectividad del sistema, en cuanto al envío de comandos de muestreo y sincronización de tiempo sobre las centrales telefónicas, es también muy alta. Las causas de que un comando de esta naturaleza no se ejecute con éxito obedecen sobre todo a errores del usuario o a problemas de saturación de canales virtuales de las centrales hacia el NMA, y no a deficiencias inherentes al sistema.

Como todo sistema de estas características, el SAST también cuenta con puntos críticos, los cuales hemos detectado en su corto tiempo de operación y merecen de una atención especial para asegurar su correcta operación. Estos puntos críticos son:

a) La adecuada operación de los *sockets* o puertos lógicos TCP/IP, a través de los cuales se establece la comunicación entre los distintos módulos que conforman el sistema. Esto quiere decir, que la comunicación a través de los *sockets*, siempre debe estar disponible para la transferencia de datos y envío de comandos.

b) La correcta operación de las macros de sistema, que se encuentran calendarizadas en el NMA para el muestreo de la hora de sistema de las centrales cada 60 minutos. Cuando hay cambio de parámetros de sistema operativo o actualizaciones de *software* en los equipos *Stratus*, estas calendarizaciones quedan desactivadas, siendo necesaria su reprogramación.

c) El correcto funcionamiento de la rutina de desbloqueo del socket de direccionamiento de registros en el NMA.

d) La exactitud de la hora de los *mainframe Stratus*, ya que esta es tomada como hora de referencia para fines de sincronización. De hecho, basta con que, los equipos *Stratus* se encuentren sincronizados con el servidor de tiempo a nivel de segundos, ya que los relojes de sistema de las centrales telefónicas no cuentan con capacidad de manejo de milisegundos, y además, el sistemas considera una desviación de tiempo como aceptable, si se encuentra dentro del rango de +/- 2 segundos.

e) La ejecución de forma continua y en paralelo de los módulos de muestreo y recolección, para asegurar que en todo momento, el SAST cuente con información reciente y confiable.

El desarrollo del SAST requirió de cuatro programadores trabajando de tiempo completo por seis meses, siguiendo una metodología de desarrollo basada en diseño por fases. En realidad, en este tiempo de desarrollo, está incluido un tiempo inicial de investigación de los comandos de sistema específicos para cada tipo de central y en la creación de los *scripts* correspondientes en el NMA.

El costo económico del desarrollo de este sistema ha sido mínimo, ya que se emplearon recursos humanos, de *hardware* y de *software* con los cuales ya contaba la empresa. Cabe mencionar que el desarrollo del SAST le ha permitido a TELMEX eliminar la necesidad de adquirir equipo y *software* de alto costo.

El SAST ha tenido una buena aceptación por parte de sus usuarios y el reconocimiento y la confianza por parte de altos directivos de la empresa, quienes reconocen la aportación del mismo en el desempeño de la planta telefónica.

8.2 Conclusiones

Tomando en cuenta los objetivos que fueron definidos al inicio del presente trabajo, podemos deducir las siguientes conclusiones:

- A nivel nacional, los relojes de sistema de las centrales telefónicas de conmutación de TELMEX, de los tipos AXE, S-12, 5ESS y PTS, se encuentran sincronizados con la hora del NMA (hora de los *mainframe Stratus*), tomando +/- 2 segundos como una desviación de tiempo aceptable.

- El SAST contribuye en gran medida a la solución de la problemática de errores en tiempos de facturación de llamadas.

- El SAST contribuye en gran medida a la solución de la problemática de errores en tiempos de interconexión de llamadas.

- Se eliminó la necesidad de adquirir módulos de *hardware* y *software* de alto costo, en cada una de las centrales telefónicas para su sincronización con un equipo servidor de tiempo.

De manera más general, y luego de la experiencia adquirida en el desarrollo del presente trabajo podemos agregar lo siguiente:

- Todas las funciones que fueron especificadas para el SAST en un inicio, pudieron ser implementadas, salvo la sincronización automática de las centrales PTS (que se ejecuta de manera manual desde el CNS a través del NMA, tomando como base los datos generados por el SAST), cuyo sistema operativo no cuenta con el comando necesario para el ajuste de la hora por incremento de tiempo en segundos.

- El SAST requiere de una supervisión constante para asegurar su correcta operación, ya que depende de factores externos como lo pueden ser los disturbios en la red, la sincronización a través de la red Internet, e incluso la disponibilidad del NMA.

- Como el SAST reside sobre el NMA, cualquier actualización o modificación de esta aplicación o del sistema operativo VOS podría afectar su operación y requerir de adecuaciones futuras.

- Este esquema de programación cliente-servidor empleado en el desarrollo del SAST, pudiera ser aplicado en al desarrollo de otros sistemas semejantes para la solución de otros problemas que presenta la planta telefónica de TELMEX.

Finalmente, nosotros, como parte del grupo de trabajo que desarrolló el Sistema Automático de Sincronía de Tiempo (SAST), para Teléfonos de México, creemos que ha sido una oportunidad formidable de: reafirmar los conocimientos académicos con los que ya contábamos de antemano, de aprender nuevos e interesantes conceptos en éste dinámico campo de la informática, de valorar el poder del trabajo en equipo, de conocer la empresa para la cual trabajamos y contribuir con su mejor desempeño.

IX. BIBLIOGRAFÍA

Thomas W. Madron, *"Redes de Area Local"*, Editorial Megabyte, Grupo Noriega Editores, México 1992.

Uyless Black, *"Redes de Computadoras, Protocolos, Normas e Interfaces"*, Editorial Macrobit, México 1990.

Howard W. Sams & Company, *"Handbook of Computers Comunications V.2"*, Local Network Standars Editions.

RAD Data Comunications, *"Catálogo de Productos para la comunicación de datos"*, México 1994.

Gary C. Kessler, David A. Train, *"Metropolitan Area Networks. Concepts, Standars and Services"* McGraw-Hill, Inc., 1991.

Cisco Systems de México S.A. de C.V., *"Interconexión de Términos y Acrónimos."* Cisco Systems, Inc., 1992.

M. Morris Mano, *"Arquitectura de Computadores"* Prentice-Hall Hispanoamericana, S.A., México, 1985.

Herbert Schildt, *"Manual de Referencia C"*, McGraw-Hill, España, 1990.

Chris Brown, *"Distribuiting Programing in UNIX"*, Ed. Prentice Hall, Estados Unidos, 1994.

Guillermo Levine, *"Estructuras Fundamentales de la Computación"*, Ed. McGraw-Hill, México, 1996.

Byron S. Gottfried, *"Programación en C"*, Ed. McGraw-Hill, España, 1999.

Richard Fairley, *"Ingeniería de Software"*, Ed. McGraw-Hill/Interamericana de México, S.A. de C.V., México, 1998.

Roger S. Pressman, *"Ingenierái del Software un Enfoque Práctico"*, Ed. McGraw-Hill, México, 1998.

Börje Langefords, *"Los ordenadores"*, Ed. Salvat, España, 1973.

APÉNDICE

OBJETIVO.

Mostrar al usuario la forma de uso del Sistema Automático de Sincronía de Tiempo con el objeto de lograr su máximo aprovechamiento en beneficio de la planta telefónica.

ALCANCE.

Este documento aplica al personal de la Gerencia de Desarrollo de *Software* Aplicación a Planta, de los Centros de Administración de Red (CAR) y del Centro Nacional de Supervisión (CNS).

INTRODUCCIÓN.

El Sistema Automático de Sincronía de Tiempo (SAST) surge como alternativa de solución a la problemática que tiene TELMEX en cuanto a errores en tiempos de facturación.

Dado lo anterior, el Centro Nacional de Supervisión (CNS) solicitó formalmente a la Gerencia de Desarrollo de *Software* de Aplicación a la Planta (Led), el desarrollo de un sistema capaz de monitorear y sincronizar automáticamente las centrales de larga distancia, ya que hasta ese momento, la sincronización se llevaba a cabo semanalmente y de forma manual en el CNS y en los CAR's tomando +/- 10 segundos como una desviación aceptable.

Es así como el grupo de Desarrollo de la Gerencia de Desarrollo de *Software* (Led), luego de un análisis detallado y múltiples pruebas en su laboratorio logró un sistema que a través de algoritmos estadísticos y módulos de programa en C bajo UNIX y VOS, es capaz de monitorear las centrales que están fuera de sincronía y enviarles los comandos de sincronización correspondientes (según el tipo de tecnología) a través del NMA, tomando +/- 2 segundos como una desviación aceptable y la hora de la Stratus como referencia.

Es importante mencionar que la implantación del Sistema Automático de Sincronía del Tiempo implica los siguientes beneficios a la empresa:

- Contribuirá en gran medida a la solución de la problemática de errores en los tiempos de facturación de llamadas detectada por Auditoría.
- Evitará la necesidad de adquirir e instalar módulos de *hardware* y *software* de alto costo en cada una de las centrales telefónicas para sincronización con un equipo maestro.
- Funciona únicamente con recursos con los que ya cuenta la empresa.
- El rango de desviación de tiempo se reducirá a +/- 2 segundos.
- Los procesos estadísticos y de sincronía se ejecutan de manera automática.

Favor de Referirse al Anexo I "Diagrama a Bloques del SAST" para obtener un mejor panorama del funcionamiento del sistema.

DESARROLLO.

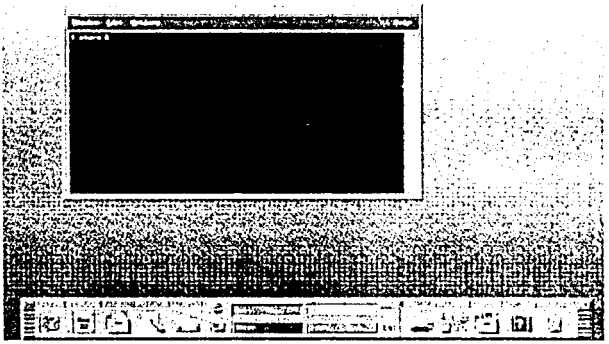
- Acceso al Sistema Automático de Sincronía de Tiempo

Este sistema puede ser accesado desde cualquier estación de trabajo bajo plataforma UNIX o desde una PC que cuente con algún *software* de emulación de terminal (Extra! X, Kea! X, Exceed, Extra Personal Client, etc).

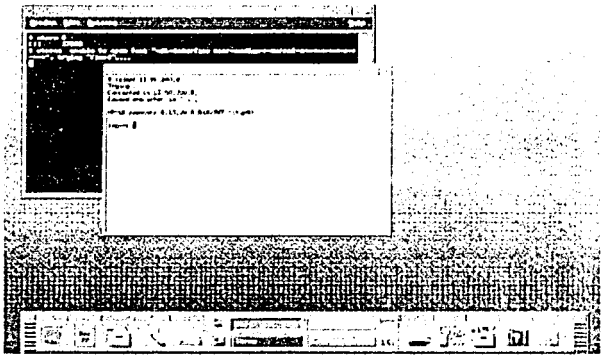
Para accesar al sistema solo hay que seguir los siguientes pasos:

a) Abrir una ventana xterm tecleando el siguiente comando

S xterm &

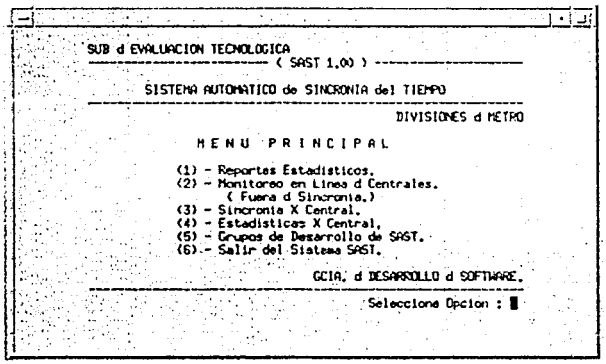


b) Abrir una sesion telnet a la siguiente direccion 13 50 200 8



Utilizar el ***login*** y ***password*** proporcionados por la Gerencia de Desarrollo de *Software* de Aplicación a la Planta (Led)

c) Una vez que se ha iniciado la sesión, el sistema mostrará de manera automática el siguiente menú.



En este punto se debe teclear el número correspondiente a la opción que requerimos del sistema.

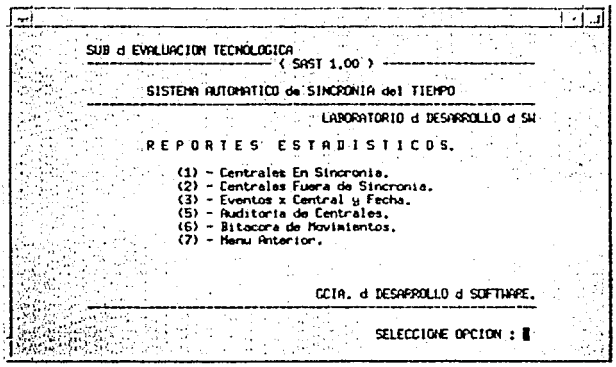
- Reportes Estadisticos

El Sistema Automático de Sincronia de Tiempo cuenta con la facilidad de ofrecer distintos reportes estadisticos tales como

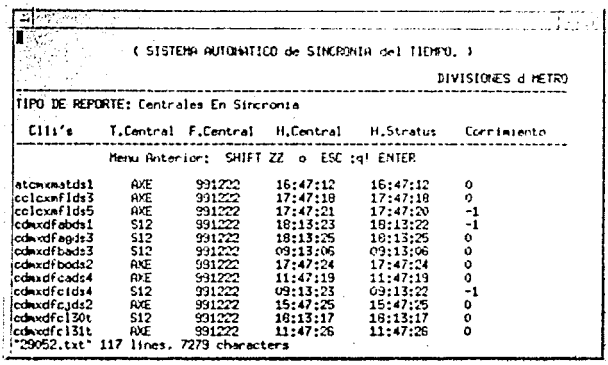
1. Centrales en Sincronia
2. Centrales Fuera de Sincronia.
3. Eventos por Central y Fecha.
4. Eventos secuenciales
5. Bitácora de Movimientos.

Para hacer uso de estos reportes solo hay que seguir los siguientes pasos:

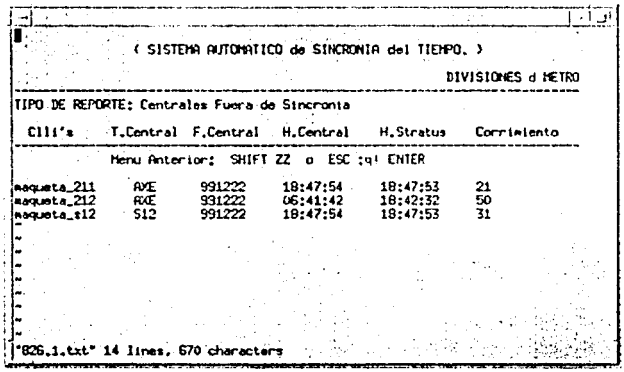
a) Elegir la opción (1) del menú principal.
b) Aparecerá un submenú con las siguientes opciones:



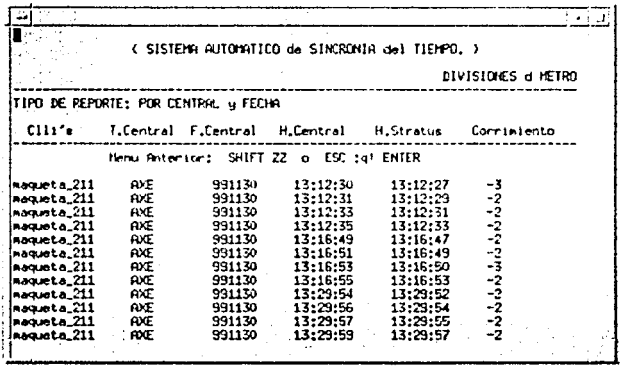
c) Si elegimos la opción (1) obtendremos un reporte completo de Centrales en Sincronía (desviación dentro de +/- 2 seg.). Un ejemplo de este reporte se muestra a continuación:



d) Si elegimos la opción (2) obtendremos un reporte completo de Centrales Fuera de Sincronia. (Desviación mayor a +/- 2 seg.) Un ejemplo de este reporte se muestra a continuación.

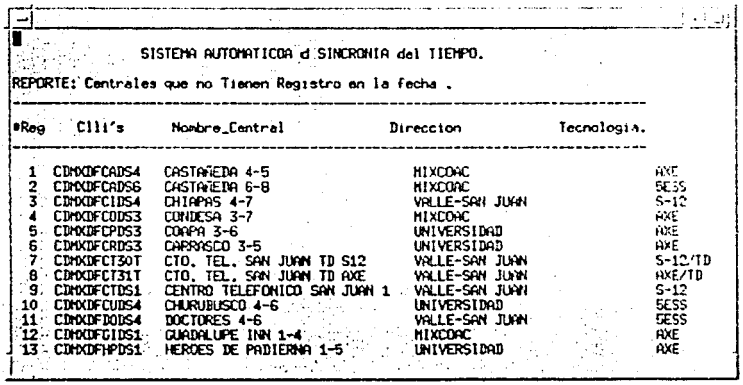


e) Si elegimos la opción (3) obtendremos un reporte completo de Eventos por Central y Fecha. Un ejemplo de este reporte se muestra a continuación:



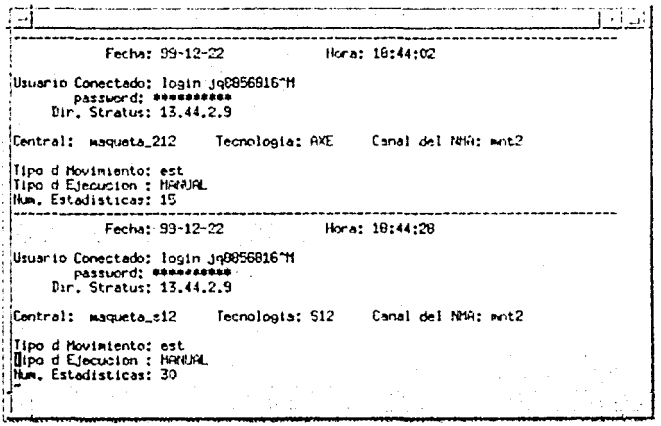
```
( SISTEMA AUTOMATICO de SINCRONIA del TIEMPO. )

                                                    DIVISIONES d METRO
TIPO DE REPORTE: POR CENTRAL y FECHA

  Cili's      T.Central  F.Central   H.Central    H.Stratus    Corrimiento

            Menu Anterior:  SHIFT ZZ  o  ESC :q! ENTER

maqueta_211      AXE        991130     13:12:30     13:12:27      -3
maqueta_211      AXE        991130     13:12:31     13:12:29      -2
maqueta_211      AXE        991130     13:12:33     13:12:31      -2
maqueta_211      AXE        991130     13:12:35     13:12:33      -2
maqueta_211      AXE        991130     13:16:49     13:16:47      -2
maqueta_211      AXE        991130     13:16:51     13:16:49      -2
maqueta_211      AXE        991130     13:16:53     13:16:50      -3
maqueta_211      AXE        991130     13:16:55     13:16:53      -2
maqueta_211      AXE        991130     13:29:54     13:29:52      -2
maqueta_211      AXE        991130     13:29:56     13:29:54      -2
maqueta_211      AXE        991130     13:29:57     13:29:55      -2
maqueta_211      AXE        991130     13:29:59     13:29:57      -2
```

f) La opción (4) se refiere a Auditoria de Centrales. En este reporte se listan todas las centrales de las cuales no hay estadisticas de tiempo a la fecha.



```
                SISTEMA AUTOMATICOA d SINCRONIA del TIEMPO.

REPORTE: Centrales que no Tienen Registro en la fecha .

#Reg   Cili's      Nombre_Central                Direccion              Tecnologia.

  1   CDMXDFCADS4  CASTAÑEDA 4-5                 MIXCOAC                AXE
  2   CDMXDFCADS6  CASTAÑEDA 6-8                 MIXCOAC                5ESS
  3   CDMXDFCIDS4  CHIAPAS 4-7                   VALLE-SAN JUAN         S-12
  4   CDMXDFCODS3  CONDESA 3-7                   MIXCOAC                AXE
  5   CDMXDFCPDS3  COAPA 3-6                     UNIVERSIDAD            AXE
  6   CDMXDFCRDS3  CARRASCO 3-5                  UNIVERSIDAD            AXE
  7   CDMXDFCT30T  CTO. TEL. SAN JUAN TD S12     VALLE-SAN JUAN         S-12/TD
  8   CDMXDFCT31T  CTO. TEL. SAN JUAN TD AXE     VALLE-SAN JUAN         AXE/TD
  9   CDMXDFCTDS1  CENTRO TELEFONICO SAN JUAN 1  VALLE-SAN JUAN         S-12
 10   CDMXDFCUDS4  CHURUBUSCO 4-6                UNIVERSIDAD            5ESS
 11   CDMXDFDODS4  DOCTORES 4-6                  VALLE-SAN JUAN         5ESS
 12   CDMXDFGIDS1  GUADALUPE INN 1-4             MIXCOAC                AXE
 13   CDMXDFHPDS1  HEROES DE PADIERNA 1-5        UNIVERSIDAD            AXE
```

g) Si elegimos la opción (5) podremos revisar la bitácora de movimientos registrados (Archivo *log* del sistema) Un ejemplo de este reporte se muestra a continuación



h) Si elegimos la opción (6) regresaremos al menú principal.

i) Es importante mencionar que estos reportes son manejados a través del editor de texto *vi*, y la forma de navegar a través de los mismos es empleando las secuencias de teclas de este editor. Por ejemplo:

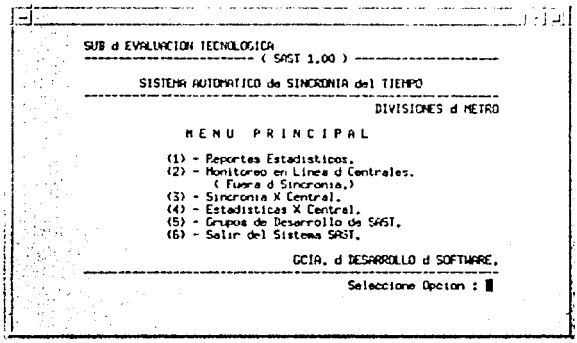
↑ Moverse un renglón hacia arriba.
↓ Moverse un renglón hacia abajo.
← Moverse un carácter a la izquierda
→ Moverse un carácter a la derecha

ESC-SHIFT-:S Ir al final del archivo.
ESC-SHIFT- se un Enumerar el archivo.
ESC-SHIFT se nonu Quitar la numeración del archivo
ESC-SHIFT-:N Ir a la N-ésima línea del reporte.
ESC-SHIFT /palabra Búsqueda de una palabra en el reporte.
ESC-SHIFT-:ZZ Salir del reporte

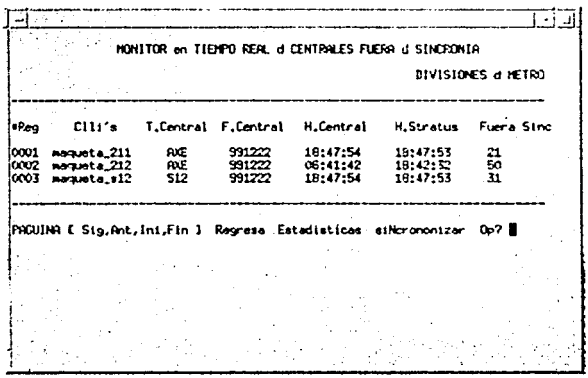
- Monitoreo en Línea de Centrales Fuera de Sincronía.

El Sistema Automático de Sincronía del Tiempo cuenta con un módulo que permite monitorear centrales fuera de sincronía en tiempo real. Los pasos para ejecutar este monitoreo son los siguientes

a) Elegir la opción (2) del menú principal



b) Aparecerá una pantalla como la que se muestra a continuación:

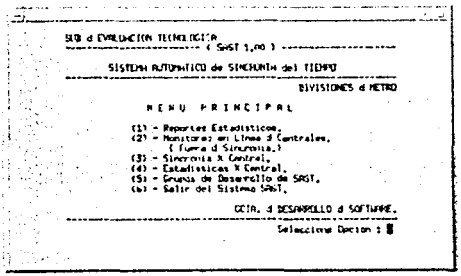


c) Como se puede ver, la lista de centrales fuera de sincronía se presenta en bloques de 10 en 10 por razones de espacio, sin embargo, se cuenta con la capacidad de hacer scroll sobre el reporte a través de las siguientes teclas, según se muestra en el pequeño de la parte inferior de la pantalla:

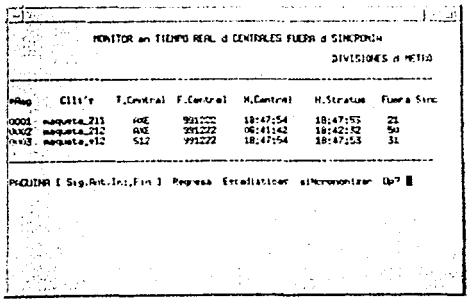
S – Siguiente bloque.
A – Bloque Anterior
I – Inicio
F – Final
E – Estadísticas (Ver punto 3.4 para más detalle)
N – Sincronía. (Ver punto 3.5 para más detalle).
R – Regresar al menú principal

- Generación Automática de Estadísticas de Tiempo.

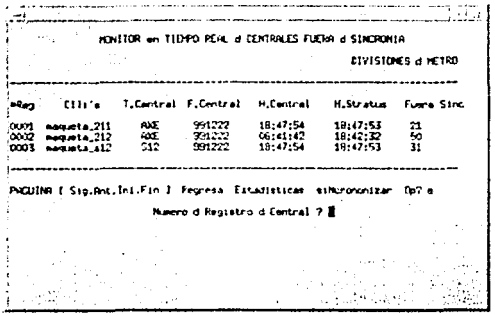
a) Elegir la opción (2) del menú principal



b) El sistema nos presentará la lista de centrales fuera de sincronía numerados en forma consecutiva por #Reg de la siguiente forma



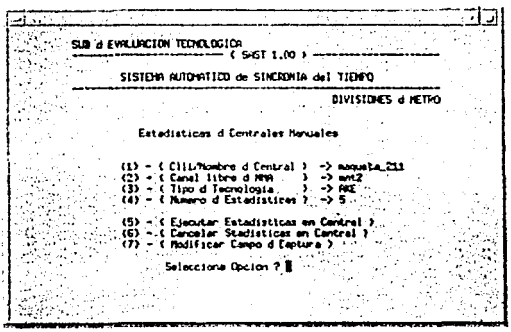
c) Presionar E <ENTER> para entrar a la opción de estadísticas. El sistema preguntará por el #Reg de la central sobre la cual se quieren generar estadísticas de tiempo



en este punto hay que teclear el #Reg correspondiente (número consecutivo que se muestra en la primera columna del reporte) y dar un <ENTER>.

NOTA: No es necesario escribir los ceros que van a la izquierda.

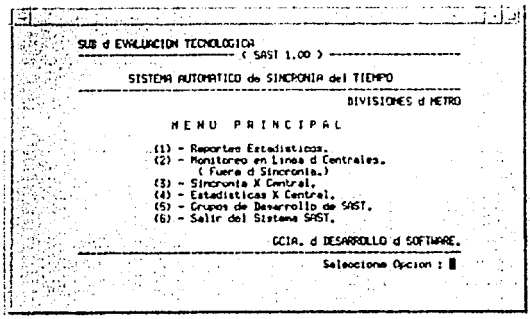
d) La siguiente pantalla mostrará todos los datos con los cuales se van a ejecutar las estadísticas.



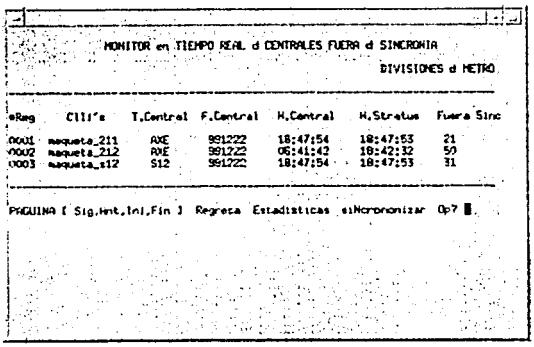
e) Si estamos de acuerdo en ejecutar la generación de estadísticas con los datos que el sistema genera de manera automática, seleccionamos la opción (5). si deseamos cancelar la ejecución de las estadisticas seleccionamos la opción (6), y por último, si queremos modificar algún campo de captura lo podemos hacer seleccionando la opción (7)

- Sincronización Automática de Centrales.

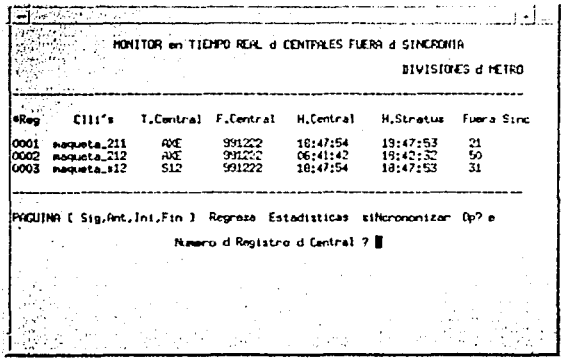
a) Entrar a la opción de monitoreo en línea (opción (2) del menú principal)



b) El sistema nos presentará la lista de centrales fuera de sincronia numeradas en forma consecutiva por #Reg de la siguiente forma:



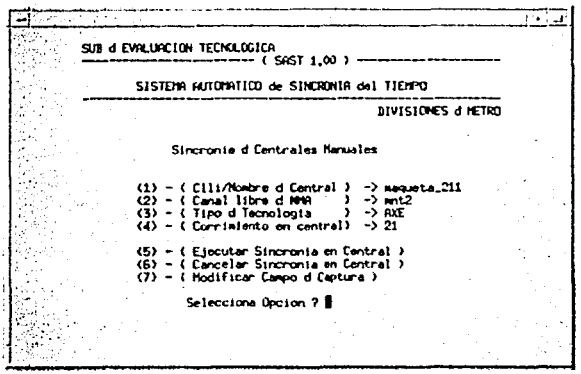
c) Presionar N <ENTER> para entrar a la opción de sincronizar. El sistema preguntará por el #Reg de la central que va a ser sincronizada.



en este punto hay que teclear el #Reg correspondiente (número consecutivo que se muestra en la primera columna del reporte) y dar un <ENTER>.

NOTA: No es necesario escribir los ceros que van a la izquierda.

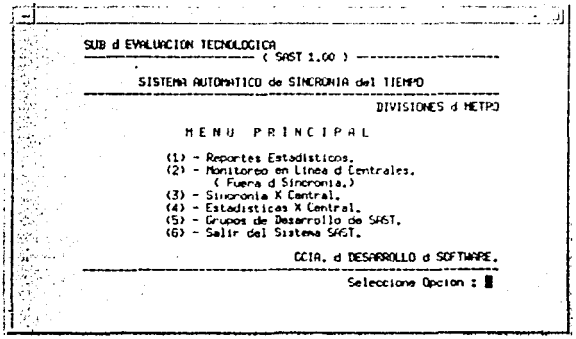
d) La siguiente pantalla mostrará todos los datos con los cuales se van a ejecutar las estadísticas.



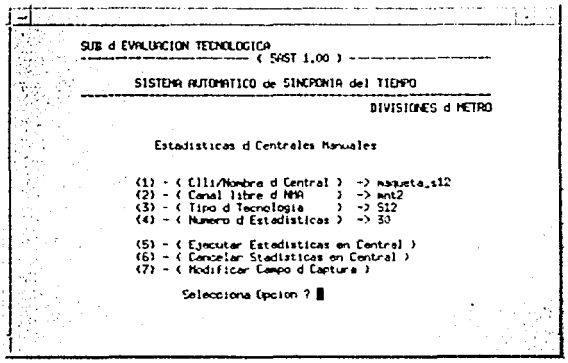
e) Si estamos de acuerdo en ejecutar la sincronización de la central con los datos que el sistema genera de manera automática, seleccionamos la opción (5), si deseamos cancelar la ejecución del proceso de sincronización seleccionamos la opción (6), y por último, si queremos modificar algún campo de captura lo podemos hacer seleccionando la opción (7)

- Generación Manual de Estadísticas de Tiempo

a) Entrar a la opción de monitoreo en línea (opción (2) del menú principal).



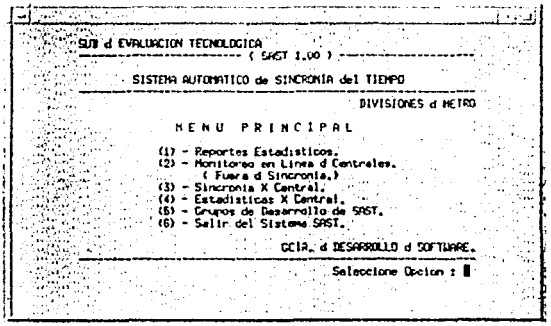
b) El sistema nos presentará la siguiente pantalla para llenar manualmente los campos necesarios para ejecutar la generación de estadísticas de tiempo.:



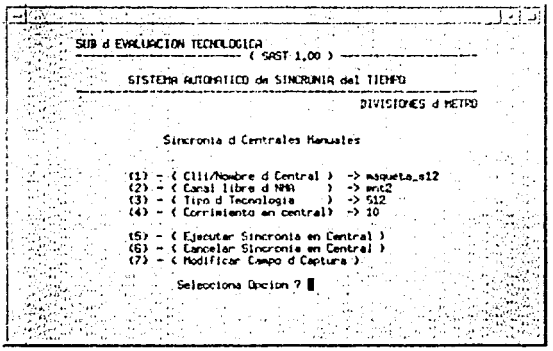
c) Si estamos de acuerdo en ejecutar la generación de estadísticas con los datos que el sistema genera de manera automática, seleccionamos la opción (5), si deseamos cancelar la ejecución de las estadísticas seleccionamos la opción (6), y por último, si queremos modificar algún campo de captura lo podemos hacer seleccionando la opción (7)

- Sincronización Manual de Centrales.

a) Entrar a la opción de Sincronia X Central (opción (3) del menú principal).



b) El sistema nos presentará la siguiente pantalla para llenar los campos necesarios para ejecutar el proceso de sincronización de la central:



c) Si estamos de acuerdo en ejecutar la sincronización de la central con los datos que el sistema genera de manera automática, seleccionamos la opción (5). si deseamos cancelar la ejecución del proceso de sincronización seleccionamos la opción (6). y por último, si queremos modificar algún campo de captura lo podemos hacer seleccionando la opción (7).

DIAGRAMA DE BLOQUES DEL SISTEMA AUTOMATICO DE SINCRONIA DE TIEMPO

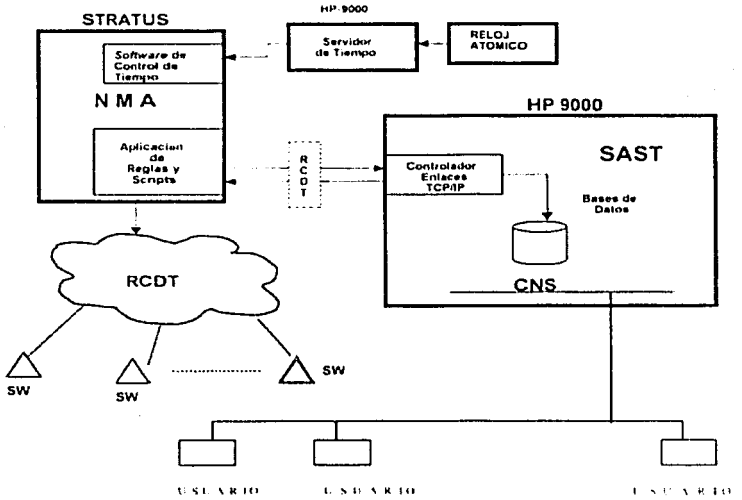


TABLA DE FIGURAS

TABLA DE TABLAS

Pág.