24-25

Universidad Nacional Autónoma de México

Facultad de Ingeniería

DESARROLLO DE PROTOCOLOS DE COMUNICACION PARA MAQUINAS CON SISTEMAS OPERATIVOS UNIX Y VMS.

T E S I S

Que para obtener el Título de

INGENIERO EN COMPUTACION

P r e s e n t a

Federico Alberto Morales Favila

Director: Act. Sergio Castro Resines

México, D. F. 1987

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO

PROLOGO

Al escoger el tema para realizar la tesis se decidió que ésta debía satisfacer dos requisitos primordiales:

1. Debía ser una tesis cuyo resultado resolviera un problema práctico y no fuera únicamente un trabajo teórico.

2. El tema seleccionado tenía que ser de actualidad.

En base a los criterios antes expuestos se decidió realizar los programas necesarios para emulación de terminales y transferencia de archivos entre las computadoras VAX 11/780 y Altos 986. La justificación reside en que dichos programas resolverían un problema del Centro de Cálculo de la Facultad de Ingeniería y versaban sobre un tema de actualidad: las redes de computadoras.

Por otra parte, la realización de este trabajo hubiera sido imposible sin el apoyo brindado por el CECAFI al autor por lo que quiero agradecer al Ing. Alejandro Jiménez García, al Ing. Sócrates A. Muñiz, al Ing. Néstor Gómez M. y en general a todas las personas que laboran en dicho centro por todas las facilidades brindadas para la realización de este trabajo.

Al Act. Sergio Castro R. le estoy agradecido por haberme dirigido en este trabajo así como por sus observaciones y sugerencias.

Al Ing. Eduardo Jallath C. también le estoy agradecido por sus consejos en beneficio de este trabajo.

Finalmente quisiera agradecer la gentil ayuda prestada por la Srita. Dolores A. Roldán en la revisión del texto.

Espero que el trabajo desarrollado sea de gran utilidad al Centro de Cálculo.

Federico A. Morales Favila

CAPITULO 1

INTRODUCCION.

El Centro de Cálculo de la Facultad de Ingeniería posee dos computadoras principales: la VAX 11/780 y la Altos 986. La primera está ubicada en el edificio principal en tanto que la segunda está ubicada en el edificio anexo. La distancia que separa a ambas computadoras así como la necesidad de intercambiar información entre ambas máquinas hizo que se planteara como proyecto el desarrollo de programas de comunicación tanto para emulación de terminales como para transferencia de archivos.

El propósito de esta tesis es proporcionar un conjunto de programas que permitan la emulación de una terminal de VAX cuando se trabaja en Altos y viceversa así como programas que permitan la transferencia de archivos en ambas direcciones. Se espera que estos programas sean de gran utilidad al Centro de Cálculo ya que en la actualidad no hay ningún mecanismo para intercambiar información entre ambas máquinas.

1.1 ORGANIZACION DE LA TESIS.

La justificación teórica del trabajo realizado se encuentra en la teoría de redes de computadoras. En base a esto, el Capítulo 2 tiene los antecedentes correspondientes a este tema. En particular se examina el modelo ISO para la arquitectura de una red.

El capítulo 3 examina la interfaz estándar RS-232-C la cual es utilizada para conectar una terminal con una computadora.

El Capítulo 4 examina lo que se denomina disciplina de línea la cual fija las reglas para intercambiar información entre dos máquinas. Se discuten las disciplinas de línea DDCMP y BYSINC.

El Capítulo 5 examina el multiplexor asíncrono DZ11 de la VAX 11/780 el cual es un dispositivo de hardware para conectar terminales.

El Capítulo 6 examina el sistema operativo VMS de la VAX 11/780 y que es uno de los sistemas operativos con los cuales se tuvo que trabajar.

El Capítulo 7 examina el sistema operativo UNIX de la Altos 986 y que es el otro sistema operativo con que se trabajó.

En el Capítulo 8 se explica la conexión que se realizó para comunicar ambas máquinas así como el funcionamiento de los programas y su relación con el modelo de ISO. Adicionalmente se proporcionan instructivos para hacer uso de estos programas.

Por último, en el Capítulo 9 se dan las conclusiones de este trabajo.

1.2 TERMINOLOGIA.

En este trabajo se decidió adoptar los términos en idioma inglés como si se trataran de nombres propios en virtud de lo siguiente:

1. La gran mayoría de los libros sobre computación están en el idioma inglés por lo que es mas fácil buscar mayor información conociendo los términos en este idioma.

2. No existe una terminología estándar en el idioma español.

Creo que seguir esta política facilitará la lectura y comprensión del tema y será beneficiosa para aquellos que deseen profundizar en el mismo.

CAPITULO 2

ANTECEDENTES.

2.1 INTRODUCCION.

Durante el Siglo XX una de las tecnologías claves ha sido la del tratamiento de la información y su distribución. Conforme nos aproximamos hacia el final del siglo, las diferencias entre recolectar, transportar, almacenar y procesar la información están desapareciendo rápidamente. Organizaciones que tienen cientos de oficinas alrededor del mundo tienen la capacidad de inspeccionar su oficina más remota con gran facilidad. El viejo modelo de tener una computadora sirviendo a toda una organización rápidamente está siendo reemplazado por otro en el que un vasto número de computadoras separadas geográficamente, pero interconectadas entre sí, realizan el trabajo. A éstos sistemas se les denomina redes de computadoras.

Al conjunto de computadoras autónomas que están interconectadas entre sí se le denomina red de computadoras. Por otra parte, se dice que dos computadoras están interconectadas si son capaces de intercambiar información.

2.2 OBJETIVOS DE UNA RED DE COMPUTADORAS.

Dos causas principales son las que están obligando a que una sola computadora sea reemplazada por una red. La primera es que muchas organizaciones tienen varias computadoras localizadas en diversos puntos geográficos. Al tener dichas computadoras en red es posible hacer que todos los programas, datos y otros recursos estén disponibles a los usuarios de la red sin importar su localización física.

La segunda causa tiene su origen en el deseo de lograr una mayor confiabilidad al tener varias fuentes de recursos. Cuando las computadoras no están conectadas y una máquina falla, los usuarios locales no pueden trabajar a pesar de que en algún otro lado haya una sustancial capacidad de cómputo. En una red, la pérdida temporal de una computadora no es tan seria porque es posible acomodar a los usuarios en algún otro lugar hasta que se reestablezca el servicio.

Otra razón importante tiene que ver con el costo del poder de cómputo comparado con el costo de comunicación. Hasta principios de los años setentas, las computadoras eran relativamente caras comparadas con los costos de comunicación. En la actualidad la situación se ha invertido. Además, es importante considerar el hecho de que la relación precio/rendimiento es superior en una computadora pequeña que en una grande. A grandes rasgos, puede decirse que una computadora grande es 10 veces más rápida que una computadora pequeña pero su costo es 100 veces mayor, lo que hace atractivo tener varias computadoras pequeñas en lugar de una grande.

Adicionalmente, al construir grandes sistemas acoplando un gran número de procesadores más pequeños, se tiene la posibilidad de un software más simple. Dentro de la red es posible dedicar uno o varios procesadores a una tarea específica; por ejemplo: a manejar una base de datos. En lugar de máquinas con multiprogramación, cada máquina puede realizar únicamente una tarea a la vez. Así, al eliminar la multiprogramación se elimina la complejidad asociada al software de máquinas multiprogramadas.

2.3 ESTRUCTURA DE UNA RED.

En cualquier red existe un conjunto de máquinas cuyo propósito es ejecutar programas de usuario. En la siguiente discusión se seguirá la terminología de una de las mayores redes: la red ARPANET y llamaremos a las máquinas "hosts". Las máquinas están comunicadas por la subred de comunicación o subred. El trabajo de la subred es llevar mensajes de una computadora a otra. Al separar los aspectos de comunicación (la subred) de los aspectos de aplicación (las computadoras), el diseño completo de la red se simplifica grandemente.

En casi todas las redes, la subred contiene dos elementos básicos: componentes de switcheo y líneas de transmisión. Los elementos de switcheo generalmente son computadoras especializadas que según ARPANET se denominan IMPs (Interface Message Processors). Las líneas de transmisión se denominan circuitos o canales.

A grandes rasgos, existen dos clases generales de subredes de comunicación:

1. Canales punto-a-punto.

2. Canales de difusión amplia.

En la primer estructura, la red tiene numerosos cables o líneas telefónicas rentadas, cada una conectando un par de IMPs. Si dos IMPs no comparten un cable y desean comunicarse deben hacerlo indirectamente vía otro IMP. Cuando se envía un mensaje de un IMP a otro utilizando IMPs intermedios, el mensaje es recibido por cada IMP, guardado hasta que es posible su transmisión y finalmente es enviado. Una subred que utiliza ese principio se llama punto-a-punto.

Cuando se utiliza una subred punto-a-punto, un problema de diseño importante es la topología de la red. Las redes locales que fueron diseñadas como tales normalmente tienen una topología simétrica. Por el contrario, las topologías irregulares son el resultado de conectar computadoras ya existentes a través del sistema telefónico. Algunas topologías comunes son: configuración en estrella, anillo, árbol, completa, de anillos que se intersectan e irregular.

En el segundo método existe un solo canal de comunicación que comparten todos los IMPs. En estos sistemas un mensaje transmitido por un IMP es recibido por todos los demás IMPs, por lo que alguna parte del mensaje debe especificar cual es el destinatario.

Algunas posibilidades para este tipo de estructura son la conexión por un bus común, la transmisión por radio o satélite, etc.

2.4 ARQUITECTURA DE UNA RED.

Las modernas redes de computadoras se diseñan de un modo estructurado. En las siguientes secciones se examinan las técnicas utilizadas para ello.

2.4.1 JERARQUIA DE LOS PROTOCOLOS.

Para reducir la complejidad de su diseño, la mayoría de las redes están organizadas como una serie de capas o niveles, cada una construida sobre su predecesor. El número de niveles varía de red a red. Sin embargo, en todas las redes el propósito de cada nivel es ofrecer ciertos servicios a los niveles superiores, apartando a éstos de detalles tales como la manera en que los servicios ofrecidos son implementados.

El nivel n de una máquina lleva a cabo una conversación con el nivel n de otra máquina. Las reglas y convenciones usadas en esta conversación se conocen como protocolo del nivel n. En realidad, ningún dato es transferido directamente de nivel a nivel. En cambio, cada nivel pasa datos e información de control al nivel inmediato inferior, hasta que se alcanza el nivel más bajo. En el nivel más bajo hay comunicación física con la otra máquina al contrario de la comunicación virtual utilizada por los niveles más altos.

Entre cada par de niveles adyacentes hay una interfaz. La interfaz define que operaciones y servicios ofrece cada nivel al nivel inmediato superior.

Al conjunto de niveles y protocolos se le llama Arquitectura de la Red.

2.4.2 EL MODELO DE REFERENCIA DE ISO.

El objetivo que ISO pretende al desarrollar su modelo de referencia es simplemente definir un conjunto de mecanismos que hagan posible la interconexión de sistemas informáticos heterogéneos, utilizando los medios públicos de transmisión de datos.

En sus documentos de trabajo ISO define un sistema abierto como: "Un sistema capaz de interconectarse con otros de acuerdo con unas normas establecidas". Por tanto, la interconexión de sistemas abiertos se ocupará del intercambio de información entre sistemas abiertos y su objetivo será la definición de un conjunto de normas que permitan a dichos sistemas cooperar entre sí.

La consecuencia de este planteamiento ha sido la definición, por parte de dicha organización, de un modelo de referencia para la interconexión de sistemas abiertos, el cual trata de presentar de una manera coherente, lo que denomina la arquitectura de interconexión de dichos sistemas.

2.4.3 LA ARQUITECTURA DEL SISTEMA DE INTERCONEXION.

En el análisis de un sistema de interconexión se utiliza habitualmente la metodología consistente en una estructuración según una jerarquía de niveles o estratos. Las definiciones de los elementos constitutivos de dicha jerarquía se dan a continuación:

1. El sistema de interconexión está formado por un conjunto de entes situados a diferentes niveles estructurales, denominados igualmente estratos.

2. Los entes de un determinado nivel "n" cooperan entre sí de acuerdo con un determinado protocolo "n".

3. Los entes de un nivel "n" utilizan los servicios (n-1) proporcionados por los entes de los niveles inferiores, mediante un acceso a ellos. La estructura de estos niveles inferiores es desconocida para el nivel "n" el cual nuevamente tiene en cuenta los servicios proporcionados por lo que se ha denominado bloque "n-1".

4. Los entes de un nivel "n" realizan unas determinadas funciones "n", utilizando los servicios de los entes del nivel "n-1" y proporcionando a su vez servicios a los entes del nivel "n+1".

Los elementos que constituyen el modelo de referencia se muestran en la Fig. 1.

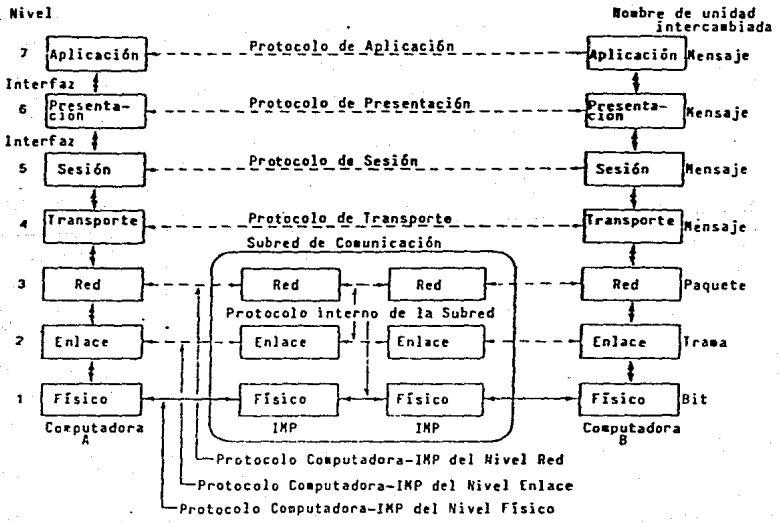


Figura 1

El modelo ISO para la interconexión de sistemas abiertos

2.4.3.1 Nivel 1: Físico. -

El nivel físico es el encargado de transmitir un conjunto de bits por un canal de comunicación. Los aspectos de diseño que se consideran tratan de garantizar que cuando un lado envía un bit 1, el otro lado lo recibe como un 1 y no como un 0. Las preguntas típicas son cuántos volts deben utilizarse para representar un 1 y un 0, cuántos microsegundos ocupa un bit, si la transmisión puede efectuarse en ambas direcciones de manera simultánea, etc.

2.4.3.2 Nivel 2: Enlace. -

La tarea de este nivel es tomar un canal de transmisión y presentarlo al nivel red como una línea libre de errores. Esto se logra rompiendo los datos de entrada en tramas de datos que son transmitidas secuencialmente y procesando las tramas de reconocimiento enviadas por el receptor. Dado que el nivel 1 simplemente acepta y transmite un conjunto de bits sin importar su significado o estructura, el nivel de enlace debe crear y reconocer los límites de cada trama. Esto puede llevarse a cabo añadiendo ciertos patrones especiales de bits al principio y al final de la trama. Estos patrones de bits pudieran ocurrir por accidente dentro de los mismos datos, por lo que debe tenerse especial cuidado para evitar confusiones.

Otra cuestión que debe resolver el nivel 2 es como evitar que un receptor lento pierda datos cuando está en contacto con un transmisor rápido. Es necesario proporcionar algún mecanismo que permita al transmisor conocer la capacidad del receptor. Típicamente este mecanismo y el manejo de errores se integran juntos.

2.4.3.3 Nivel 3: Red. -

El nivel red, que algunas veces se denomina nivel de comunicación de la subred, controla la operación de la subred. Entre otras cosas, determina las características principales de la interfaz computadora-IMP (Interface Message Processor), y cómo los paquetes, que son las unidades de información que se intercambian en este nivel, son encaminados dentro de la subred. Un problema de diseño clave es la división de trabajo entre las computadoras y los procesadores de comunicación, en particular quién garantiza que todos los paquetes son recibidos correctamente en su destino y en el orden apropiado. Lo que el software de este nivel hace es aceptar mensajes de la computadora fuente, convertirlos en paquetes y ver que los paquetes lleguen correctamente a su destino.

El problema clave de cómo encaminar el paquete es resuelto. Por otra parte, si hay muchos paquetes presentes dentro de la subred al mismo tiempo, pueden formarse cuellos de botella. El control de dicha congestión también corresponde a este nivel.

Finalmente, dado que los operadores de la subred esperan una remuneración por su trabajo, frecuentemente existe una función de contabilidad en este nivel.

2.4.3.4 Nivel 4: Transporte. -

La función básica del nivel de transporte es aceptar datos del nivel de sesión, dividirlo en unidades más pequeñas si es necesario, pasarlos al nivel de red y asegurarse que todas las piezas son recibidas correctamente en el otro lado.

Este nivel es un nivel de fuente-a-destino (end-to-end). En otras palabras, un programa en la computadora fuente lleva una conversación con un programa similar en la computadora destino utilizando los encabezados y mensajes de control.

2.4.3.5 Nivel 5: Sesión. -

Ignorando el nivel de presentación que simplemente realiza ciertas transformaciones en los datos, el nivel de sesión es la interfaz con el usuario dentro de la red. Con este nivel el usuario debe negociar para establecer una conexión con un proceso en otra máquina. Una vez que se ha establecido la conexión, el nivel de sesión puede manejar el diálogo de una manera ordenada.

Una conexión entre usuarios (técnicamente hablando, entre dos procesos del nivel de presentación) usualmente se llama sesión.

En algunas redes, los niveles de transporte y sesión se combinan en un solo nivel o el nivel de sesión está ausente, si todo lo que desea el usuario es servicio de comunicación.

2.4.3.6 Nivel 6: Presentación. -

El nivel de presentación realiza funciones que son solicitadas con suficiente frecuencia como para buscar una solución general para ellas, en lugar de permitir que cada usuario resuelva sus problemas. Estas funciones frecuentemente son realizadas por rutinas de biblioteca que el usuario invoca. Un ejemplo típico de un servicio de transformación es el de compresión de texto.

2.4.3.7 Nivel 7: Aplicación. -

El contenido del nivel de aplicación es individual para cada usuario. Cuando dos diferentes programas en diferentes máquinas se comunican entre sí, solamente ellos determinan el conjunto de mensajes permitidos y la acción que se toma a la recepción de cada uno de ellos.

CAPITULO 3

INTERFAZ EIA RS-232-C

3.1 REVISION HISTORICA.

En los primeros días de la comunicación de datos vía telefónica en los Estados Unidos, la compañía American Telegraph and Telephone (ATT) era, para fines prácticos, la única que proporcionaba éste servicio. Así, los modems diseñados por la compañía Bell Laboratories eran los estándares de la industria. Sin embargo, existía un gran número de fabricantes de equipo de cómputo que necesitaban conocer las características eléctricas de los modems fabricados por Bell. Adicionalmente, existían algunos fabricantes independientes de modems que necesitaban conocer las características de la interfaz utilizada entre la computadora y el equipo terminal de datos. Para resolver estos problemas, la Asociación de la Industria Eléctrica (EIA por sus siglas en inglés) con la cooperación de la Bell, los fabricantes independientes de modems y los fabricantes de computadoras, desarrollaron una Interfaz estándar para su utilización entre Equipo Terminal de Datos (DTE por sus siglas en inglés) y Equipo de Comunicación de Datos Utilizando Intercambio Serial Binario (DCE). Este estándar recibió el nombre de RS-232-C, el cual refleja la última revisión (C) hecha. Nuevos estándares, RS-422 y RS-423, han sido desarrollados y reemplazarán al anterior en el futuro. Sin embargo, en la actualidad RS-232-C es el estándar para el cual las interfaces de los modems son diseñadas.

Por otra parte, cualquier país diferente al de los Estados Unidos está en posición de dictar sus propias normas para regular la transmisión de datos. Esto impediría que equipos tales como modems, terminales y computadoras fueran fabricados fuera de su país de origen. Para permitir la manufactura económica de los modems, terminales, computadoras y para facilitar la comunicación por computadora entre países distintos, las oficinas gubernamentales de Correo, Telégrafo y Teléfono de las naciones afiliadas a la ONU han, en colaboración con el Comite Consultaif International Telephonique et Telegraphique (CCITT), promulgado estándares para interfaces de comunicación de datos y para otros aspectos de las telecomunicaciones bajo el nombre de 'recomendaciones'. Estas recomendaciones cubren todas las fases de las telecomunicaciones, desde procedimientos de operación hasta el control de interferencia en cables de telecomunicación. Estas recomendaciones se revisan en Asambleas Plenarias que se llevan a cabo aproximadamente cada cuatro años y los

resultados son publicados en un conjunto de volúmenes a los que se hace referencia por su color. Por ejemplo, la segunda Asamblea Plenaria se llevó a cabo en Nueva Delhi en 1960 y resultó en la publicación del Libro Azul. Las definiciones que se expondrán más adelante fueron tomadas del Libro Naranja que resultó de la sexta reunión plenaria llevada a cabo en 1976 en Geneva, Suecia. El volúmen relacionado con la Transmisión de Datos es el VIII y contiene recomendaciones que llevan el prefijo V o el prefijo X. La recomendación CCITT V.24 es el equivalente del estándar EIA RS-232-C.

3.2 DESCRIPCION DE LA INTERFAZ EIA RS-232-C.

Antes de comenzar, es importante notar con exactitud lo que este estándar contiene:

1. Una descripción funcional de los circuitos de intercambio(*).
2. Las características de las señales eléctricas.
3. Las características mecánicas de la interfaz.
4. Una lista de subconjuntos estándar de circuitos de intercambio, para grupos específicos de aplicaciones de comunicación de sistemas.

3.2.1 DESCRIPCION FUNCIONAL DE LOS CIRCUITOS DE INTERCAMBIO.

Las características eléctricas que a continuación se especifican se aplican a circuitos de intercambio con velocidades inferiores o iguales a 20,000 bits por segundo.

La Figura 1 muestra el circuito de intercambio equivalente con sus parámetros eléctricos, los cuales se definen a continuación:

Vo Es el voltaje del generador en circuito abierto.
Ro Es la resistencia total efectiva de D. C. asociada con el generador, medida en el punto de intercambio.
Co Es la capacitancia total efectiva asociada al generador, medida en el punto de intercambio.
Vl Es el voltaje en el punto de intercambio con respecto a la señal de tierra o retorno común.
Cl Es la capacitancia total efectiva asociada con la carga, medida

---

(*) Por circuito se entiende, en comunicaciones, el camino eléctrico completo que proporciona una comunicación en uno o dos sentidos entre dos puntos. Por punto de intercambio se entiende el lugar en donde las señales de la interfaz se transmiten entre equipo pot medio de interconexiones eléctricas.

en el punto de intercambio.
R1 Es la resistencia total efectiva de D. C. asociada a la carga, medida en el punto de intercambio.
E1 Es el voltaje de la carga a circuito abierto (polarización).

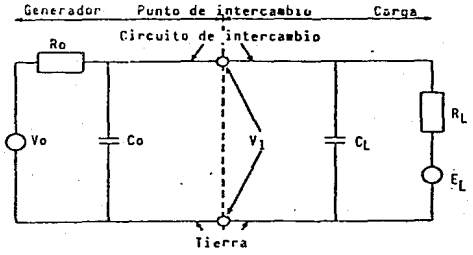


Figura 1
Circuito de Intercambio Equivalente y sus Parámetros Eléctricos

A continuación se presenta una lista que es una condensación de las reglas que rigen a éste circuito:

1. El voltaje a circuito abierto Vo no tendrá una magnitud mayor de 25 volts.

2. El circuito generador será capaz de sostener un corto circuito en cualquiera de sus cables sin que se dañe él mismo u otro equipo, la corriente de corto circuito no excederá 0.5 amperes.

3. Para las señales se considerará que están en el estado MARK ('1') cuando el voltaje V1 sea más negativo que -3 volts considerados con respecto al circuito AB (señal de tierra). Se considerará que las señales están en el estado SPACE ('0') cuando V1 sea más positivo que +3 volts con respecto a la señal de tierra. La región entre -3 y +3 volts se define como la región de transición, dentro de la cual el estado de la señal no está definido.

4. La impedancia de carga (R1 y C1) tendrá una resistencia R1 de D. C. que no será menor de 3000 ohms ni mayor de 7000 ohms.

5. Cuando la resistencia R1 cumple los requerimientos del punto anterior y el voltaje E1 sea cero, el voltaje V1 tendrá una magnitud de entre 5 y 15 volts.

6. El generador proporcionará un voltaje comprendido entre -5 y -15 volts relativos a la señal de tierra para representar la condición de MARK. Así mismo, proporcionará un voltaje comprendido entre +5 y +15 volts relativos a la señal de tierra para representar la condición de SPACE. Es interesante observar que esta regla en conjunción con la regla 3 nos da un margen de ruido de 2 volts.

7. El generador no cambiará su voltaje de salida a una tasa mayor de 30 volts por microsegundo, pero el tiempo requerido por una señal para pasar de -3 a +3 volts (región de transición) no excederá 1 milisegundo o 4% del lapso de duración de un bit, el que sea más pequeño.

8. La capacitancia shunt de C1 no excederá los 2500 picofarads, incluyendo la capacitancia del cable.

9. La impedancia del generador cuando está apagado será mayor a 300 ohms.

3.2.2 DEFINICION DE LAS FUNCIONES DE LOS CIRCUITOS DE LA INTERFAZ.

Las siguientes definiciones corresponden a aquellos circuitos que son necesarios para realizar la interfaz con un modem privado. Esto se debe a que, para nuestro estudio, éstos son los únicos circuitos que es necesario conocer.

1. Circuito AA: Protective Ground.
   Dirección: No aplicable.
   Este conductor será conectado eléctricamente a la máquina. Adicionalmente, puede ser conectado a alguna tierra externa.

2. Circuito AB: Signal Ground or Common Return.
   Dirección: No aplicable.
   Este conductor establece el potencial de referencia común de tierra para todos los circuitos de intercambio excepto para el circuito AA. Dentro de un equipo de comunicación de datos, este circuito puede ser conectado junto con el circuito AA.

3. Circuito BA: Transmitted data.
   Dirección: Hacia el equipo de comunicación de datos (DCE).
   Las señales de este circuito son generadas por el equipo terminal de datos (DTE) y son transferidas hacia el convertidor de señales local para la transmisión de datos hacia un DTE remoto.
   El DTE mantendrá el circuito BA en condición de MARK durante los intervalos entre caracteres o palabras y en cualquier tiempo en el cual no se estén transmitiendo datos.
   En todos los sistemas el DTE no transmitirá datos, a menos que los siguientes cuatro circuitos estén en ON (siempre y cuando estén implementados):

1. Circuito CA (Request to Send).

2. Circuito CB (Clear to Send).

3. Circuito CC (Data Set Ready).

4. Circuito CD (Data Terminal Ready).

Todas las señales que sean transmitidas a través de la interfaz en el circuito de intercambio BA durante el tiempo que la condición de ON se mantenga en los cuatro circuitos mencionados anteriormente, serán transmitidas por el canal de comunicación.

4. Circuito BB: Received Data.
Dirección: Desde el DCE.
Las señales de este circuito son generadas por el convertidor de señales que está recibiendo, en respuesta a las señales recibidas desde un DTE remoto vía el convertidor de señal remoto que transmite. El circuito BB se mantendrá en MARK todo el tiempo que el circuito CF (Received Line Signal Detector) esté en la condición de SPACE.
En un canal conectado en Half-Duplex, el circuito BB se mantendrá en la condición de MARK cuando el circuito CA (Request to Send) esté en condición de ON y durante el breve lapso que sigue a la transición de ON a OFF del circuito CA para permitir que la transmisión se complete.

5. Circuito CA: Request to Send.
Dirección: Hacia el DCE.
Este circuito es utilizado para condicionar la transmisión de datos vía el equipo local de comunicación de datos y, en un canal conectado en Half-Duplex, controlar la dirección de la transmisión de los datos.
En un canal duplex, la condición de ON mantiene al DCE en el modo de transmisión. La condición de OFF mantiene al DCE en un modo de No-Transmisión.
En un canal half duplex, la condición ON mantiene al DCE en modo de transmisión e inhibe el modo de recepción. La condición de OFF mantiene al DCE en el modo de recepción.
Una transición de OFF a ON instruye al DCE a entrar en el modo de transmisión. El DCE responde llevando a cabo aquellas acciones que sean necesarias para tal efecto y finalmente responde poniendo en ON el circuito CB (Clear to Send), indicando así al DTE que los datos pueden ser transferidos a través de la interfaz vía el circuito BA (Transmitted Data).
Una transición de ON a OFF instruye al DCE para que complete toda la transmisión que esté llevando a cabo y que, después, asuma un modo de No-Transmisión o un modo de recepción, según sea lo apropiado. El DCE responde a esta instrucción poniendo en OFF el circuito CB (Clear to Send) una vez que esté preparado para responder a una condición subsecuente de ON del circuito CA.

6. Circuito CB: Clear to Send.
Dirección: Desde el DCE.
Las señales en este circuito son generadas por el DCE para indicar si el data set(*) está listo para transmitir datos.
La condición de ON junto con la condición de ON en los circuitos CA, CC y CD (si están implementados) es una indicación al DTE de que las señales presentadas al circuito BA (Transmitted Data) serán transmitidas al canal de comunicación.
La condición de OFF es una indicación al DTE de que no debe transferir datos a través del circuito BA.
La ocurrencia de una condición de ON en el circuito CB es una respuesta a la ocurrencia de condiciones de ON simultáneas en los circuitos CC (Data Set Ready) y CA (Request to Send), y se retrasará el tiempo que sea necesario para que el DCE establezca un canal de comunicación con un DTE remoto.
Cuando el circuito CA (Request to Send) no está implementado en un DCE con capacidad para transmitir, se asumirá que el circuito CA está en condición ON todo el tiempo, y el circuito CB deberá responder de acuerdo a esta suposición.

7. Circuito CC: Data Set Ready.
Dirección: Desde el DCE.
Las señales en este circuito se utilizan para indicar el estado del modem local.
Este circuito presenta la condición de ON para indicar:

a) El DCE local está conectado al canal de comunicación y

b) El DCE local no está bajo prueba (local o remota) y

c) El DCE local ha completado, cuando sea aplicable:

   1. Cualquier función de conteo de tiempo requerida por el sistema de switcheo para completar el establecimiento de una llamada y

   2. La transmisión de cualquier tono discreto de respuesta cuya duración depende únicamente del data set local.

Cuando el DCE local no transmita un tono de respuesta o cuando la duración del tono de respuesta esté controlada por alguna acción del data set remoto, la condición de ON será presentada tan pronto como las otras condiciones (a, b y c-1) se satisfagan.
Este circuito se utiliza únicamente para indicar el estado del data set local. La condición de ON no debe ser interpretada ni como una indicación de que un canal de

---

(*) El término data set es equivalente a modem. A lo largo de este trabajo se utilizarán ambos términos de manera indistinta.

comunicación ha sido establecido con una estación remota de datos, ni como el estado del equipo de alguna estación remota.
La condición de OFF será una indicación de que el DCE ignorará cualquier señal que aparezca en cualquier circuito de intercambio, excepto para el circuito CE (Ring Indicator). La señal de OFF no impedirá el funcionamiento de los circuitos CE y CD.

8. Circuito CF: Received Line Signal Detector.
Dirección: Desde el DCE.
La condición de ON en este circuito se presenta cuando el DCE esta recibiendo una señal que satisface sus criterios de aceptabilidad. Estos criterios los establece el fabricante del DCE.
La condición de OFF indica que no se está recibiendo ninguna señal o que la señal recibida no es apropiada para ser demodulada.
La condición de OFF del circuito CF (Received Line Signal Detector) causará que el circuito BB (Received Data) pase a la condición de MARK.
En canales Half-Duplex, el circuito CF se mantiene en la condición OFF siempre que el circuito CA (Request to Send) esté en la condición de ON y durante un breve intervalo de tiempo seguido a la transición de ON a OFF del circuito CA.

3.2.3 CARACTERISTICAS MECANICAS DE LA INTERFAZ.

Un conector con 25 pines, dispuestos según se ilustra en la Figura 2, es el conector más ampliamente utilizado para el estándar RS-232-C. Un conector macho se utiliza en el equipo terminal de datos y un conector hembra se utiliza para el equipo de comunicación de datos.

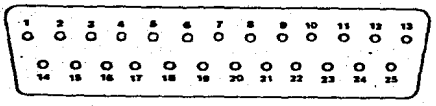


Figura 2
Vista frontal de un conector utilizado en un DTE

Para aproximadamente la mitad de los pines existe una asignación de circuitos internacionalmente aceptada, la cual es utilizada por todos los modems que implementan estos circuitos. Esta asignaciónes se muestran en la Tabla 1.

Tabla 1
Asignaciones Internacionalmente Aceptadas Para
Interfaz DTE/DCE Utilizando un Conector con 25 Pines

Pin	EIA	Nombre del circuito
1	AA	Protective Ground
2	BA	Transmitted Data
3	BB	Received Data
4	CA	Request to Send
5	CB	Clear to Send
6	CC	Data Set Ready
7	AB	Signal Ground
8	CF	Received Line Signal Detector
9	Usualmente utilizado como punto de prueba de poder del modem No conectar	
10	Usualmente utilizado como punto de prueba de poder del modem No conectar	
11		
12		
13		
14		
15	DB	Transmit Signal Element Timing. Originada en un DCE (Para modems síncronos únicamente)
16		
17	DD	Received Signal Timing Element. Originada en un DCE (Para modems síncronos únicamente)
18		
19		
20	CD	Data Terminal Ready
21		
22	CE	Ring/Calling Indicator
23		
24		
25		

Para los pines que no tienen una asignación internacionalmente aceptada existen variaciones entre los diferentes modems.

No se deben utilizar los pines que aún no tienen ninguna asignación. Varios fabricantes de equipo terminal del circuito de datos han puesto interfaces de 20 miliamperes en estos pines "no utilizados". Varios fabricantes de terminales han puesto niveles TTL de prueba en estos puntos.

3.3 PROBLEMAS QUE SE PRESENTAN AL UTILIZAR LA INTERFAZ RS-232-C.

Hace algunos años, cada diseñador de interfaces trató de cumplir con las especificaciones eléctricas antes vistas, con diferentes grados de éxito. En la actualidad, los fabricantes de los circuitos integrados han proporcionado circuitos que cumplen con las

especificaciones en términos de voltajes usados, rapidez de variación del voltaje, impedancias, etc. Sin embargo, existen ciertos aspectos de las características eléctricas que aún presentan problemas. A continuación se da una breve explicación de estos problemas:

1. Para que el generador de voltaje realmente proporcione voltajes de +5 y -5, el hardware dentro del cual están localizados estos circuitos tienen fuentes de poder que son más positivas de +6 y más negativas que -6. La razón de ésto es que los drivers(*) utilizan transistores para controlar los voltajes en la línea que se está utilizando; estos transistores están en serie con la potencia proporcionada por los circuitos integrados y una pérdida de un volt ocurre en estos transistores.

2. El segundo problema es el de la capacitancia del cable. El estándar de EIA especifica que la capacitancia del circuito que está siendo manejado (la carga) debe ser menor de 2500 picofarads, incluyendo al cable. Dado que 40-50 picofarads por pie de cable es muy común, un cable de 50 pies (15 metros aproximadamente) es el máximo permisible antes de violar el estándar. La consecuencia más obvia de violar ésta especificación es que la cantidad de tiempo necesaria para realizar una transición del estado MARK al estado SPACE y viceversa se incrementará por arriba del 4% permitido. Dado que también es muy probable que la resistencia de la circuitería del generador y del receptor sea diferente, se necesitará una cantidad de tiempo diferente para realizar las transiciónes MARK-SPACE y SPACE-MARK, lo que provoca que los circuitos receptores produzcan bits MARK que son mayores que los bits SPACE ("marking distortion") o bits SPACE que son mayores que los bits MARK ("spacing distortion"). Este tipo de distorsión puede causar que los caracteres sean recibidos incorrectamente, especialmente si existe distorsión en la velocidad del reloj o si existe ruido asociado con las transiciones entre los estados de la señal.

3. El tercer problema se refiere a la referencia de tierra. En las reglas enunciadas anteriormente, el voltaje V1 siempre se mide relativo a la señal de tierra, el circuito AB. Para que el valor de V1 no alcance valores arbitrariamente altos con respecto a la tierra lógica del receptor, es práctica común el conectar el circuito AB a la tierra lógica del dispositivo que contiene el circuito receptor. Desafortunadamente, dado que la tierra lógica del transmisor y la tierra lógica del receptor pueden diferir, es posible que fluya corriente a través del cable de señal de tierra. Dado que el cable tiene una resistencia diferente de cero, una caída de potencial ocurrirá a través suyo. Esta caída de potencial causa que el voltaje aplicado al circuito de intercambio por el driver,

---

(*) En esta discusión driver se utiliza como sinónimo del generador discutido en la sección 1.2.1

aparezca al receptor de manera diferente de lo que sería si ésta diferencia de potencial no existiese. Por ejemplo, en la Figura 3, el driver está proporcionando +5 volts pero el receptor sólo ve +3.
En la Figura 4, el driver está proporcionando -5 volts pero el receptor ve -7. La caída de 2 volts en el cable de "tierra" es el mismo no importa cual sea el voltaje que proporcione el driver porque esta caída se debe a la diferencia de potencial que existe en las tierras lógicas del transmisor y del receptor.

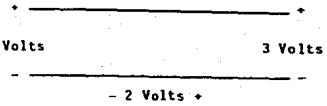


Figura 3
Efecto de la diferencia de potencial en las tierras sobre una señal de +5 volts

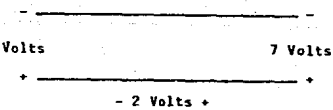


Figura 4
Efecto de la diferencia de potencial en las tierras sobre una señal de -7 volts

A pesar de la diferencia de potencial ejemplificada anteriormente, los datos aún serán interpretados correctamente, porque ésta diferencia es absorbida por el margen de ruido proporcionado por las especificaciones de la EIA. Sin embargo, considérese el mismo ejemplo con una diferencia de potencial de 9 volts; el receptor siempre verá -4 y -14 volts respectivamente en las dos figuras. Cada uno de estos voltajes será interpretado como MARK. De hecho, la diferencia de potencial no necesita ser tan grande para que ocurran pérdidas de información. Una diferencia de potencial de 3 volts es suficiente para que el receptor vea +2 volts en la Figura 3, y +2 volts está en la región indefinida de transición.

3.4 CONCLUSIONES.

El estándar RS-232-C fue diseñado como una interfaz para conectar computadoras y modems o terminales y modems para una distancia de hasta 15 metros. Trabajará moderadamente bien en distancias más grandes y puede trabajar sobre distancias mucho mayores que las especificadas

Por otra parte, los fabricantes se han preocupado por aumentar el alcance de la transmisión. Por ejemplo, la computadora DIGITAL VAX 11/780 del Centro de Cálculo de la Facultad de Ingeniería puede transmitir hasta una distancia de 300 metros sin ningún problema utilizando esta interfaz.

CAPITULO 4

DISCIPLINA DE LINEA

En este capítulo se examinan dos tipos de disciplina de línea comúnmente utilizados: BYSINC y DDCMP.

Una disciplina de línea es básicamente un conjunto de reglas para operar un sistema de comunicación. Las reglas son diseñadas para resolver los problemas operativos que se presentan en las siguientes áreas:

1. Reconocimiento del Patrón (Framing)-la determinación de cuáles grupos de ocho bits constituyen caracteres y, lo más importante, qué grupos de caracteres constituyen un mensaje.

2. Control de Errores-la detección de errores por medio de Chequeos por Redundancia Cíclica, Longitudinal o Vertical; la aceptación de mensajes correctos y la solicitud de retransmisión de mensajes erróneos.

3. Control de Secuencia-la numeración de los mensajes para evitar la duplicación o pérdida de mensajes y la identificación adecuada de mensajes que son retransmitidos por el sistema de control de errores.

4. Transparencia-la transmisión de información (tales como datos de un sistema de medición) que contiene patrones de bits que semejan caracteres de control utilizados para implementar las funciones 1, 2 y 3, sin que la estación receptora identifique aquellos patrones de bits como caracteres de control.

5. Control de Línea-la determinación, en el caso de que se utilice transmisión half-duplex o línea multipunto, de cuál estación va a transmitir y cuál estación(es) va(n) a recibir.

6. Casos Especiales-resolver el problema de que envia un transmisor cuando no tiene datos por enviar.

7. Control de Expiración del Tiempo-resolver el problema de que hacer si el flujo de mensajes repentinamente cesa.

8. Control de Inicio-el proceso de iniciar las transmisiones en un sistema de comunicación que ha estado inactivo.

Los protocolos pueden ser divididos en tres categorias dependiendo de la técnica que utilicen para reconocer los patrones de bits. Estos son: orientado a caracteres, orientado a un conteo de bytes y orientado a bits. En este capitulo se discute un protocolo orientado a caracteres (BYSINC) y un protocolo orientado a un conteo de bytes (DDCMP).

4.1 BYSINC Y LOS PROTOCOLOS ORIENTADOS A CARACTERES.

Uno de los protocolos más ampliamente utilizados en la industria es el protocolo BYSINC que fue desarrollado por IBM. BYSINC utiliza caracteres especiales para delinear los diversos campos que tiene un mensaje y para controlar las funciones asociadas del protocolo.
El formato general de un mensaje en BYSINC se muestra en la Figura 1.

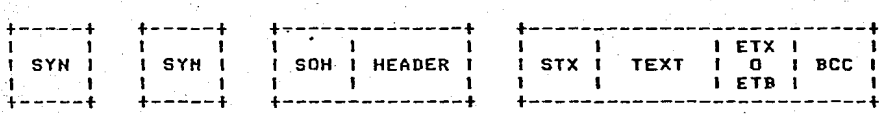


Figura 1
Formato de un Mensaje en BYSINC

El encabezado es opcional, pero si es utilizado deberá comenzar con SOH (Start Of Header) y terminar con un STX (Start Of Text). SOH y STX son caracteres especiales que se pueden encontrar en los códigos ASCII, EBCIDIC y Seis Bits, que son los códigos más comúnmente utilizados con BYSINC. El contenido del Encabezado (HEADER) lo define el usuario, excepto que el polling y direccionamiento para linea multipunto no se hacen por medio del encabezado, sino mediante un mensaje de control separado. La porción de Texto (TEXT) del mensaje es variable en longitud y puede contener "datos transparentes", es decir, bits que deben tratarse como datos (por ejemplo, datos generados por algún instrumento de medición) y no como caracteres. Esta característica requiere que la lógica de reconocimiento de caracteres del receptor sea apagada de tal manera que un patrón de bits que semeje ETX (End Of Text) o algún otro caracter de control no confunda al receptor. Para inhabilitar el reconocimiento de

caracteres en el receptor, los datos transparentes se delimitan por DLE STX (DLE = Data Link Escape) y DLE ETX o DLE ETB. No importa que el campo de texto termine con ETX, ETB, DLE ETX o DLE ETB, estos caracteres especiales indican el fin del campo de texto y el comienzo del último campo que únicamente contiene el Bloque de Chequeo de Caracteres (BCC).

BYSINC emplea un riguroso conjunto de reglas para establecer, mantener y terminar una secuencia de comunicación. Un intercambio típico entre una terminal y una computadora en una línea privada punto-a-punto se ilustra en la Figura 2.

Terminal	Computadora
1 La terminal envía un mensaje cuyo texto es un único carracter-ENQ. Esto significa 'tengo algunos datos para enviarte'.	
	2 La computadora recibe ENQ.
	3 La computadora confirma la presencia de la terminal respondiendo con un mensaje de 'adelante' (ACK0).
4 La terminal recibe 'adelante' (ACK0).	
5 La terminal envía un bloque de datos.	
	6 La computadora recibe el bloque de datos y checa la paridad. Si no hay error, seguir con el paso 8.
	Si ocurre un error, la computadora envía un caracter de control NAK que significa 'Por favor retransmite el mensaje'.
7 La terminal recibe NAK y retransmite el mensaje.	
	8 La computadora responde con un mensaje de reconocimiento ACK que significa 'Recibí correctamente, envía el siguiente mensaje'.
9 La terminal envía el siguiente bloque de datos o si la transmisión se ha completado, envía un caracter de control EOT que significa 'Ya terminé'.	
	10 La computadora recibe EOT y termina su secuencia de recepción.

Figura 2
Intercambio de datos típico utilizando BYSINC

Ahora se definen las abreviaciones de los caracteres de control:

SOH-Start Of Header.

STX-Start Of Text.

ETB-End of Transmission Block. ETB indica que el fin de un bloque de caracteres que comenzó con STX o SOH e indica que el bloque de chequeo sigue a continuación. ETB requiere una respuesta de la estación receptora indicando su estado: ACK0, ACK1, NAK, WACK o RVI.

ITB-End of Intermediate Transmission Block. ITB es utilizado para separar un mensaje en secciones para fin de detección de errores. La transmisión de ITB indica que el bloque de chequeo sigue inmediatamente. La estación receptora no contesta a la estación transmisora hasta que algún bloque termine con ETB o ETX. Excepto para el primer bloque intermedio, los bloques intermedios no necesitan comenzar con STX, excepto cuando se están transmitiendo datos transparentes. En este último caso, cada bloque debe comenza con DLE STX.

ETX-End Of Text. Termina un bloque de caracteres transmitidos que comenzó con SOH o STX. Su función es la misma que ETB, excepto que también significa que no hay más bloques de datos por transmitir.

EOT-End Of Transmission. Indica el final de la transmisión de un mensaje que puede contener un cierto número de bloques, incluyendo textos y encabezados. EOT también se utiliza para responder "nada para transmitir" a una secuencia de poll y también se utiliza como una señal para abortar.

NAK-Negative Acknowledgement. Indica que el bloque previo fue recibido con error.

DLE-Data Link Escape. Uno de los usos de DLE es en la creación de WACK, ACK0, ACK1 y RVI, que son secuencias de dos caracteres. El principal uso de DLE es en secuencias de control durante transferencias de datos transparentes. La secuencia DLE STX se utiliza para iniciar un texto de datos transparentes y DLE ETX, DLE ITB y DLE ETB se utilizan para terminar dicho texto. Adicionalmente, DLE ENQ, DLE DLE y DLE EOT también son utilizadas como secuencias de control durante la transmisión de textos de datos transparentes.

ENQ-Enquiry. Se utiliza para probar la línea cuando se utilizan conexiones punto-a-punto; indica el fin de una secuencia de poll. También se utiliza para solicitar la retransmisión de la respuesta ACK/NAK si la respuesta original fue confusa o no fue recibida cuando se le esperaba.

ACK0, ACK1-Affirmative Acknowledgement. Estas respuestas indican que no hubo error en el bloque previo y que el receptor está listo para para recibir el siguiente bloque. ACK0 se utiliza para confirmar una selección multipunto, una prueba de línea punto-a-punto y para bloques cuya numeración sea par. ACK1 se utiliza para confirmar bloques cuya numeración sea impar.

WACK-Wait before transmit positive Acknowledgement. Una respuesta WACK indica que el bloque previo fue aceptado sin error, pero que el receptor aún no está listo para recibir el siguiente bloque. La respuesta usual de la estación transmisora es ENQ y la estación receptora continúa respondiendo WACK hasta que esté lista para recibir.

RVI-Reverse Input. RVI es una confirmación positiva, pero a la vez solicita a la estación transmisora que termine la transmisión actual en virtud de que la estación receptora desea enviar un mensaje que tiene una alta prioridad.

TTD-Temporary Text Delay (STX ENQ). TTD es utilizado por una estación transmisora que aún no está lista para transmitir, pero que desea retener la línea. La estación receptora responde con NAK y la estación transmisora podrá nuevamente enviar TTD si aún no está lista.

Cuando se utiliza el código ASCII, BYSINC utiliza VRC en cada caracter para detectar los errores y LRC en todo el mensaje. En este caso, el bloque de chequeo consiste de un sólo caracter.

La operación del protocolo BYSINC puede apreciarse mejor a partir de un diagrama de estado como el que se ilustra en la Figura 3. Existen 5 estados para transmisión, dos de los cuales son aplicables para transmisión de datos ordinarios y tres son aplicables para la transmisión de datos transparentes. Unicamente se explicará el proceso de transmisión de datos ordinarios.

Para transmisión ordinaria de datos no transparentes, la transmisión comienza en el estado 3 para el envío de cualquier encabezado o para el envío de el caracter de control ENQ. Un encabezado comienza con SOH, el cual tiene asociadas ciertas reglas con él. El primer SOH o STX después de un cambio de línea (transmisor-receptor pasa a receptor-transmisor) causa que el BCC sea cero. Todos los siguientes SOH o STX (hasta el próximo cambio de línea) son incluidos en el BCC. Los caracteres ITB, ETB y ETX son incluidos en el BCC y son seguidos por el BCC. Cuando un delimitador STX o ITB ocurre en el estado 3, la transmisión pasa al estado 4, el modo de transmisión de texto. Cuando el transmisor ha enviado todo el

bloque de datos, la cuenta de caracteres alcanza cero y la transmisión regresa al estado 3 para la transmisión del siguiente bloque de datos.

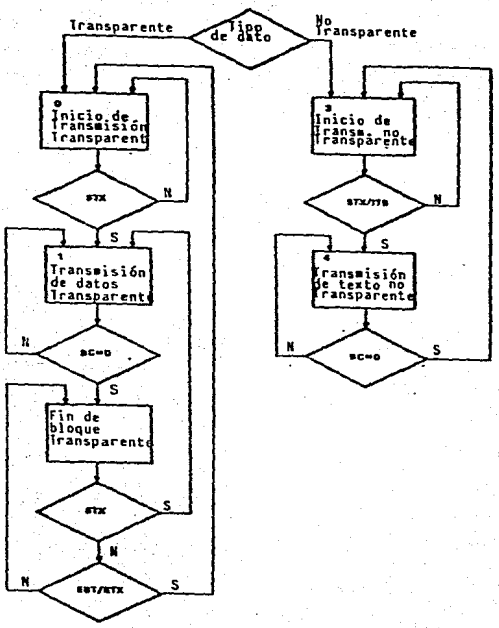


Figura 3
Diagrama de flujo para la transmisión utilizando BYSINC

La Figura 4 es el diagrama de flujo para el control del proceso de recepción de datos en BYSINC. Unicamente se explicará el proceso de recepción ordinaria.

Los estados 0 y 2 se utilizan para manejar la recepción de datos ordinarios mientras los estados 3, 4 y 5 se utilizan para manejar la recepción de datos transparentes. El estado 1 es un estado de transición entre recepción ordinaria y recepción transparente.

Mientras espera un mensaje, el receptor se encuentra en el estado 0. Cuando llega el primer caracter de control, la respuesta del receptor depende de cuál caracter de control se recibió:

ENQ-Genera una interrupción a la computadora para registrar el ENQ y preparar un buffer para recibir los datos. Permanece en el estado 0.

DLE-Va hacia el estado 1 (transición a transmisión transparente).

STX o SOH-Va hacia el estado 2 (recepción de datos ordinarios).

EOT-Genera una interrupción a la computadora que anuncia el fin del mensaje. Permanece en el estado 0.

NAK-Genera una interrupción a la computadora para informarle que el programa tendrá que retransmitir. Permanece en el estado 0.

En el estado 2 los siguientes caracteres tienen significado: ITB, ETB, ETX, ENQ y SYN. La recepción de cualquiera de los tres primeros es una indicación al receptor de que debe avisar a la computadora de que el bloque de chequeo viene a continuación. La recepción de ENQ es un error y la recepción de SYN se ignora.

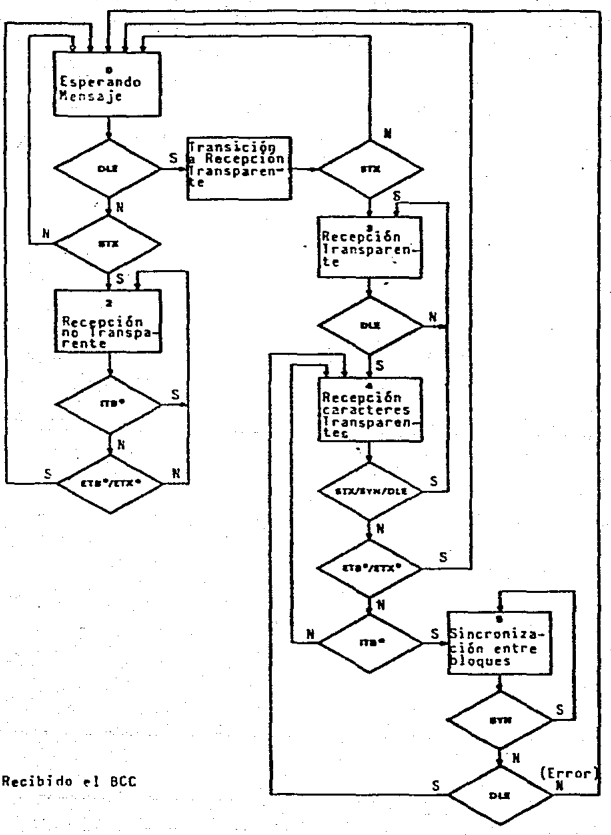


Figura 4
Diagrama de flujo para la recepción en BYSINC

4.2 DDCMP Y LOS PROTOCOLOS ORIENTADOS A LA CUENTA DE BYTES.

Un exámen de BYSINC revela que muchas de las complicaciones asociadas con este protocolo son un resultado de los procedimientos especiales utilizados durante la transmisión y recepción de datos transparentes.

Es posible diseñar un protocolo el cual, por medio de una cuenta de bytes, solucione los problemas de transparencia sin el uso de DLE u otros caracteres de control. Uno de estos protocolos que es utilizado ampliamente es el DDCMP, cuyo formato se muestra en la Figura 5.

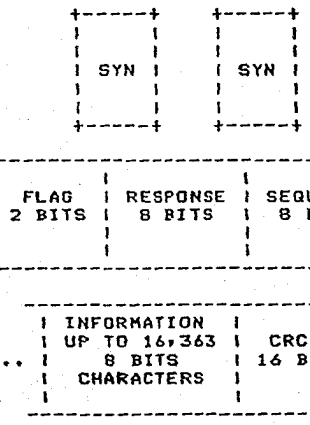


Figura 5
Formato de un mensaje en DDCMP

Como se muestra en dicha figura, el formato es parecido al de BYSINC en que el mensaje tiene dos partes: un encabezado conteniendo información de control y un cuerpo con el texto. Sin embargo, en este caso el encabezado no es opcional. Es la parte más importante del mensaje, ya que tiene la secuencia de numeración del mensaje y la cuenta de los caracteres, las dos características más importantes de DDCMP. Dada la importancia del encabezado, éste tiene su propio bloque de chequeo, indicado como CRC 1. Los mensajes que contienen datos en lugar de información de control, tienen una segunda sección que contiene cualquier número de caracteres de 8 bits (máximo 16,363) y un segundo bloque de chequeo indicado como CRC 2.

Antes de explicar el formato del mensaje, es necesario explicar el sistema de secuenciamiento de mensajes, ya que la mayor parte de la información del encabezado está directa o indirectamente relacionado con la operación de secuenciamiento.

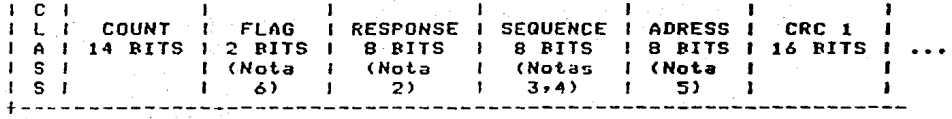
En el protocolo DDCMP, cualquier par de estaciones que intercambian mensajes los numeran secuencialmente comenzando con el mensaje 1. Cada mensaje sucesivo es numerado utilizando el siguiente número en secuencia módulo 256. Así, una larga secuencia de mensajes sería numerada 1, 2, 3,....., 254, 255, 0, 1,... La numeración se aplica a cada dirección de manera separada. Por ejemplo, la estación A podría estar enviando sus mensajes 6, 7 y 8 a la estación B, en tanto la estación B está enviando sus mensajes 5, 6 y 7 a la estación A.

Siempre que una estación transmite un mensaje a otra estación, le asigna el siguiente número secuencial y lo pone en el campo de "Sequence" en el encabezado. Adicionalmente, mantiene en un contador el número de mensajes recibidos de la otra estación. Este contador se actualiza cada vez que se recibe un mensaje cuyo número sea exactamente uno mayor que el mensaje recibido previamente. El contenido del contador de mensajes recibidos esta incluido en el campo "Response" del mensaje a enviar, para indicar a la otra estación el mayor mensaje secuenciado que se ha recibido.

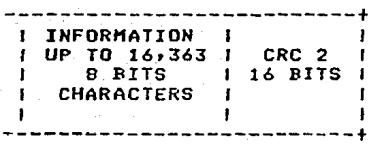
Cuando una estación recibe un mensaje erróneo, envía un mensaje con confirmación negativa (NAK) hacia la estación transmisora. DDCMP no requiere una confirmación para cada mensaje, dado que el número en el campo "Response" de un encabezado normal o en los mensajes especiales NAK o ACK, especifica el número de secuencia del último mensaje recibido correctamente.

Cuando una estación recibe un mensaje que está fuera de secuencia, no responde a ese mensaje. La estación transmisora detectará esta situación debido al campo "Response" de los mensajes que recibe, y si el temporizador de espera de respuesta expira antes de que la estación transmisora reciba una confirmación, enviará un mensaje REP. Si la estación receptora ha recibido correctamente el mensaje al que se refiere REP (así como los mensajes que le precedieron), responde a REP enviando una confirmación positiva (ACK). Si no ha recibido dicho mensaje, enviará una confirmación negativa (NAK) con el número del último mensaje correctamente recibido. La estación transmisora retransmitirá todos los mensajes después del especificado por NAK. El sistema de numeración de DDCMP permite que haya hasta 255 mensajes pendientes sin confirmación.

Ahora se examinarán en detalle los diversos formatos de los mensajes de DDCMP. Estos formatos se ilustran en la Figura 6.



	XXXXXXXX	XXXXXXXXXXXXXX	XX	XXXXXXXX	XXXXXXXX
Data Messages	10000001	Char. Count	QS	Resp #	Mess #
Acknoledgement	00000101	00000001000000	QS	Resp #	00000000
Negative Acknowledge	00000101	00000010------	QS	Resp #	00000000
Reasons:	BCC Header Error	000001			
	BCC Data Error	000010			
	Rep Response	000011			
	Buffer Unavailable	001000			
	Receiver Overrun	001001			
	Message Too Long	010000			
	Header Format Error	010001			
Reply Message	00000101	00000011000000	QS	00000000	LstMess#
Start Message	00000101	00000110000000	11	00000000	00000000
Start Acknowledgement	00000101	00000111000000	11	00000000	00000000
Maintenance Message	10010000	Char. Count	11	00000000	00000000



Nota 1

Figura 6
Formato detallado de un mensaje en DDCMP

NOTAS:

1. Solamente "Data Message" y "Maintenance Message" tienen cuenta de caracteres, por lo que solamente estos mensajes tienen el campo de información y el campo CRC 2 mostrado.

2. "Resp #" se refiere al número de respuesta. Este es el número del último mensaje recibido correctamente. Cuando se utiliza en una confirmación negativa, se asume que el siguiente mensaje numerado no fue recibido, fue recibido con errores o no fue aceptado por alguna otra razón. Ver "Reasons".

3. "Mess #" es el número secuencialmente asignado al mensaje. La numeración la asigna la estación transmisora módulo 256.

4. "LstMess#" es el número del último mensaje transmitido por la estación. Ver en el texto la discusión de los mensajes REP.

5. "Adress" es la dirección de la estación tributaria dentro de un sistema multipunto y es utilizada desde y hacia dicha estación. En operación punto-a-punto una estación envía la dirección "1" pero ignora dicho campo en la recepción.

6. "Q" y "S" se refieren a las banderas de "Quick sync" y "Select". Ver texto.

El primer caracter en un mensaje es el indicador de clase de mensaje. Existen tres clases de mensajes: de Datos, de Control y de Mantenimiento indicados por los caracteres SOH, ENQ y DLE respectivamente. Los dos siguientes caracteres del mensaje se componen de dos campos: uno de 14 bits y otro de 2 bits. El campo de 14 bits es utilizado en Datos y Mantenimiento para indicar el número de caracteres que seguirán al encabezado y que es la información del mensaje. En Mensajes de Control, los primeros 8 bits se utilizan para designar el tipo de mensaje de control y los últimos 6 bits normalmente son cero, excepto cuando se trata de un mensaje NAK en donde estos 6 bits especifican la causa del NAK. El campo de 2 bits contiene las banderas "Quick sync" y "Select".

La bandera "quick sync" se utiliza para informar a la estación receptora que el mensaje será seguido por caracteres de sincronía. La bandera de "select" se utiliza para indicar que este es el último mensaje que va a enviar la estación transmisora y que la estación direccionada puede transmitir. Esta bandera es útil en configuraciones half-duplex y multipunto.

El campo "Response" contiene el número del último mensaje recibido correctamente. Este campo se utiliza en los Mensajes de Datos y en los tipos de Mensaje de Control ACK y NAK.

El campo "Sequence" es utilizado en los Mensajes de Datos y en el tipo de Mensaje de Control REP. En un Mensaje de Datos, contiene el número de secuencia asignado por la estación transmisora. En un mensaje REP, éste es utilizado como parte de la pregunta: Has recibido correctamente todos los mensajes hasta el mensaje número (especifica)?

El campo "Adress" (campo de dirección) identifica a la estación tributaria en una configuración multipunto. En operación punto-a-punto, la estación envía "1" pero ignora el campo de dirección a la recepción.

Además de los tipos de mensajes de control REP, confirmación positiva y confirmación negativa, existen también mensajes de control "Start" (Comienza) y "Start acknowledge" (Comienza confirmación). Estos mensajes se utilizan para poner a la estación receptora en un estado conocido. En particular, inicializan los contadores de los mensajes, los temporizadores y otros contadores. El mensaje de comienza confirmación indica que estas acciones se han llevado a cabo.

El Mensaje de Mantenimiento típicamente contiene programas para cargar e inicializar la estación.

Finalizaremos la discusión haciendo un breve resumen de lo que es capaz DDCMP: puede correr en half-duplex o full-duplex; corre en muchas de las interfaces existentes actualmente; admite configuraciones punto-a-punto o multipunto; corre en sistemas síncronos o asíncronos y proporciona confirmaciones múltiples con un sólo mensaje de ACK (máximo 255).

CAPITULO 5

MULTIPLEXOR ASINCRONO DZ11 DE LA VAX 11/780

5.1 INTRODUCCION.

El DZ11 es un multiplexor asíncrono que proporciona una interfaz entre el procesador de la VAX 11/780 con 8 líneas asíncronas para transmisión en serie. Es posible operarlo a velocidades de hasta 9600 bauds utilizando la interfaz EIA RS232-C o utilizando ciclos de corriente de 20 mA.

El DZ11 tiene varias funciones que proporcionan un control flexible de parámetros tales como baudaje, longitud de cada caracter, número de "stop bits" para cada línea, e interrupciones para transmisión-recepción.

5.2 CONFIGURACIONES DEL DZ11.

El DZ11 se fabrica en 6 diferentes versiones, cada una de ellas designada por una letra de la A a la F. Los DZ11-A y DZ11-B son dispositivos compatibles con el estándar EIA RS232-C con control parcial de modem. El DZ11-E es una combinación del DZ11-A y DZ11-B.

En la Fig. 1 se observan algunas aplicaciones comunes de este dispositivo.

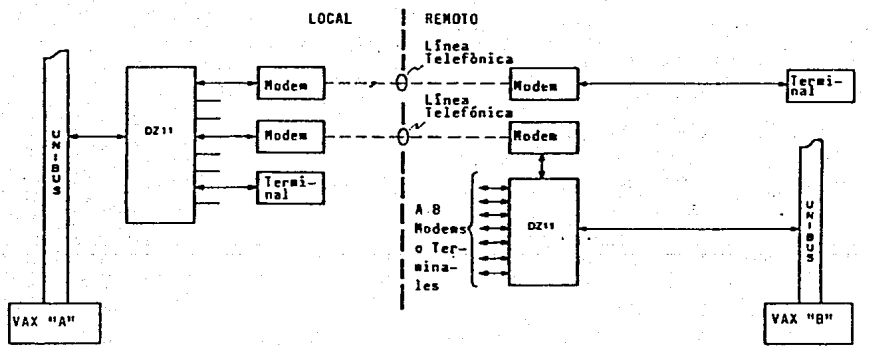


Figura 1
Aplicaciones comunes del DZ11

5.3 ESPECIFICACIONES GENERALES.

Los siguientes párrafos contienen especificaciones eléctricas, de comportamiento y de medio ambiente para todas las configuraciones del DZ11.

5.3.1 Parámetros De Comportamiento.

La Tabla 1 enumera los parámetros de su comportamiento.

Tabla 1
Parámetros de Comportamiento del DZ11

Parámetro	Descripción
Modo de Operación	Full Duplex
Formato del Dato	Asincrono, en serie bit a bit, 1 'start bit' y 1, 1.5 o 2 'stop bits' proporcionados bajo control de programa
Tamaño del Caracter	5, 6, 7 u 8 bits, seleccionable desde programa
Polaridades	Interfaz Salida EIA
Señal de Datos	High = 1 Low = 1 = Mark
	Low = 0 High = 0 = Space
Señal de Control	High = 1 Low = OFF
	Low = 0 High = ON
Orden de los Bits	Primero el bit menos significativo tanto para transmisión como para recepción
Baudaje	50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200 y 9600
Rendimiento	21,940 caracteres/segundo máximo.

5.3.2 Salidas.

Para las configuraciones A y B, los niveles de los voltajes de salida y los conectores utilizados cumplen con el estándar EIA RS232-C. Las señales que maneja son:

Circuito AA	Pin 1	Protective Ground
Circuito AB	Pin 7	Signal Ground
Circuito BA	Pin 2	Transmitted Data
Circuito BB	Pin 3	Received Data
Circuito CD	Pin 20	Data Terminal Ready
Circuito CE	Pin 22	Ring Indicator
Circuito CF	Pin 8	Carrier

5.3.3 Distorsión.

La máxima distorsión permitida para las transiciones 'space to mark' y 'mark to space' cuando se recibe un caracter es de 40%.

La máxima distorsión permitida en la velocidad de recepción es de 3.8% a 2000 bauds y 4% para todas las demás velocidades. Para la transmisión la máxima distorsión es de 2.2% a 2000 bauds, mientras que para las demás velocidades es de 2%.

5.3.4 Distancia.

La distancia recomendada entre la computadora y el DZ11 es de 15 metros máximo a 9600 bauds. La operación a más de 15 metros no está de acuerdo con la norma de la AIE, sin embargo, usualmente la operación es factible a distancias mayores dependiendo de la velocidad de transmisión, tipo de cable, medio ambiente, etc.

En la Tabla 2 se enumeran las distancias y velocidades a las que es posible trabajar. Esta Tabla no indica que la operación a dichas distancias y velocidades esté libre de errores en un momento dado.

Tabla 2
Distancias a las que es
posible trabajar utilizando el DZ11

DISTANCIA (METROS)	VELOCIDAD (BAUDS)
90	9600
300	4800
300	2400
900	1200
1500	300

5.4 DESCRIPCION FUNCIONAL.

La Figura 2 presenta un diagrama de bloques que representa las unidades funcionales de que consta el DZ11: Interfaz al Bus, Lógica de Control e Interfaz con la Línea.

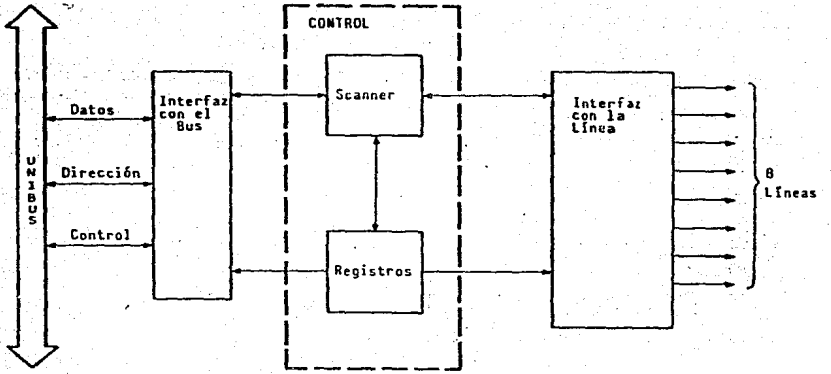


Figura 2
Unidades funcionales del DZ11

5.4.1 Interfaz Con El Bus.

La interfaz con el Bus maneja todas las transacciones entre la lógica de control del DZ11 y el Bus. Esta interfaz realiza tres funciones: manejo de los datos, reconocimiento de su dirección y control de interrupciones.

En la función de manejo de datos, la interfaz lleva los datos desde y hacia los varios registros que existen dentro de la lógica de control y proporciona el acondicionamiento de voltaje necesario para recibir y transmitir datos desde o hacia el Bus.

La lógica de reconocimiento de direcciones y la lógica de control activan las señales apropiadas de lectura y carga de datos cuando en el Bus se reconoce la dirección asignada al DZ11. Estas señales son utilizadas por la función de manejo de datos para llevar la información que llega o sale a la dirección deseada.

La función de control de interrupciones inicia y controla el procesamiento de interrupciones entre el DZ11 y el procesador de la VAX.

5.4.2 Lógica De Control.

La lógica de control proporciona las señales temporizadoras y de control requeridas para manejar todas las operaciones de transmisión y recepción. Esta lógica puede dividirse en dos partes: el scanner y los registros.

El scanner continuamente examina cada una de las líneas y basado en la información proporcionada por la interfaz con la línea, genera señales que ocasionan un flujo de datos desde o hacia la línea apropiada. El scanner tiene un reloj que trabaja a 5.08 MHz y un buffer FIFO de 64 palabras.

5.4.3 Interfaz Con La Línea.

Dos de las más importantes operaciones dentro del DZ11 son las conversiones de formato en serie a formato en paralelo para cada dato. Estas conversiones se requieren en virtud de que el DZ11 está localizado entre el Bus (datos en paralelo) y terminales o líneas telefónicas (datos en serie). Estas conversiones se llevan a cabo mediante la utilización de UARTs (Universal Asynchronous Receiver Transmitter).

Otro componente de la interfaz con la línea es el receptor de línea, el cual convierte los niveles de voltaje TTL que existen dentro del DZ11 a niveles que corresponden a los existentes en las líneas de entrada externas (modem o terminal).

CAPITULO 6

EL SISTEMA OPERATIVO VMS.

6.1 INTRODUCCION.

Las computadoras VAX son una familia de computadoras de 32 bits con multiprogramación y memoria virtual fabricadas por Digital Equipment Corporation. Los sistemas VAX son una extensión de la popular familia de computadoras PDP-11.

Conforme los costos de hardware continúan declinando y las tendencias hacia la descentralización de los equipos de cómputo se siguen reforzando, el mercado para sistemas con gran funcionalidad a un costo de minicomputadora continúa creciendo. Esto ha cambiado la naturaleza de las minicomputadoras; de máquinas relativamente simples a sistemas complejos que admiten multiprogramación y diversos modos de operación tales como batch, tiempo real, etc. Estas capacidades del sistema operativo solo era posible encontrarlas en las grandes computadoras de hace unos cuantos años.

Esta funcionalidad en las minicomputadoras ha traído consigo la necesidad de una mayor capacidad en el almacenamiento virtual. DEC desarrolló la arquitectura VAX-11 como una respuesta a esta necesidad.

El objetivo principal de la arquitectura VAX-11 es proporcionar un significativo incremento en el tamaño de almacenamiento virtual disponible en la PDP-11 (hasta 128 Kbytes de almacenamiento virtual).

6.2 METAS EN EL DISEÑO DE LA VAX.

Varias metas influenciaron el diseño de la familia de computadoras VAX:

- La arquitectura debería tener una prolongada vida útil debido a las fuertes inversiones que deben realizarse tanto por el vendedor como por los usuarios al momento de diseñar tanto el hardware como el software.

- La arquitectura debería proporcionar un gran espacio para almacenamiento virtual.

- Debería proporcionar un medio ambiente adecuado para la eficiente compilación y ejecución de programas escritos en lenguajes de alto nivel.

- La arquitectura debería ser similar a la de la PDP-11 para que los usuarios de estas máquinas encontraran conveniente adquirir una VAX.

- La arquitectura debería soportar una familia completa de máquinas con diferente comportamiento, pero con funciones compatibles.

- Para proporcionar una ejecución eficiente del Sistema Operativo, la arquitectura debería incluir instrucciones que facilitasen la ejecución de tareas frecuentemente utilizadas por el Sistema Operativo. (Los diseñadores incluyeron instrucciones que proporcionan soporte por hardware de colas, campos de bits de longitud variable, y almacenamiento y recuperación del contexto de un programa).

6.3 COMPATIBILIDAD CON PDP-11.

La meta de compatibilidad entre la VAX-11 y la PDP-11 se logró de diversas maneras. Ambas máquinas utilizan el mismo formato para sus datos y los mismos formatos para E/S. Así, los archivos de datos producidos en una máquina pueden ser leídos en la otra.

La sintaxis y los mnemónicos del lenguaje ensamblador son básicamente los mismos. Así, los compiladores que generan código ensamblador de PDP-11 pueden adaptarse para fácilmente generar código ensamblador de VAX-11.

El Sistema Operativo VAX/VMS (Virtual Memory System) es similar en su estructura al sistema operativo PDP-11 RSX. Muchas de las áreas funcionales clave de ambos sistemas operativos, tal como el sistema de archivos, son idénticas.

La VAX-11 tiene un modo compatible por hardware, el cual puede ejecutar muchos de los programas de PDP-11 directamente. Un paquete denominado "Applications Migration Executive" ayuda a convertir programas a VAX utilizando una emulación de los servicios proporcionados por RSX.

6.4 INSTRUCCIONES Y ALMACENAMIENTO.

La VAX tiene un único y potente conjunto de instrucciones. Una instrucción puede tener hasta 37 bytes de longitud y hasta seis operandos. Existen 6 formatos diferentes para las instrucciones aritméticas. Se proporcionan instrucciones para la eficiente manipulación de palabras hasta de 32 bits y para strings hasta de 64 Kbytes.

Varias instrucciones proporcionan una funcionalidad que difícilmente se encuentra disponible en otras máquinas:

- EMOD realiza una reducción extremadamente precisa en números de punto flotante. Es útil para normalizar argumentos de funciones trigonométricas.
- POLY calcula el valor de un polinomio de la forma:

  $$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

  lo cual incrementa fuertemente la velocidad en el cálculo de funciones matemáticas.
- INDEX examina el valor de un índice y calcula la localidad correspondiente dentro de un arreglo.
- CASE implementa directamente la instrucción GOTO calculado con un chequeo automático de rango.
- CRC realiza chequeos por redundancia cíclica directamente. Esto es útil en sistemas de comunicación de datos para detectar posibles errores en la transmisión y/o recepción.

El manejo de memoria virtual se ha incorporado tanto en el hardware como en el Sistema Operativo. Múltiples programas de un usuario pueden ser soportados de manera simultánea, cada uno con su propio espacio virtual.

Cuatro niveles jerárquicos de protección han sido proporcionados. Estos son: Kernel, Executive, Supervisor y User. La protección se lleva a cabo a nivel de página y cada página tiene 512 bytes. Las tablas de páginas ayudan a mapear direcciones virtuales en direcciones físicas además de que especifican el modo de acceso (ninguno, solo lectura y lectura/escritura) para cada uno de los modos de protección.

6.5 LA VAX-11/780.

La VAX-11/780 fué el primer miembro de la familia VAX. Consiste de una unidad central de proceso, subsistema de memoria principal, subsistema de I/O y un subsistema de consola (Fig. 1). El procesador, el subsistema de almacenamiento y el subsistema de I/O están interconectados por el SBI (Synchronous Backplane Interconnect); un bus de alta velocidad que es el camino primario del sistema para control y datos. El Unibus proporciona un camino de 1.5 Mbytes/seg para manejar una gran variedad de periféricos. El Massbus es utilizado para proporcionar caminos de 2.0 Mbytes/seg para transferencias de disco o cinta magnéticas. Se admite un máximo de 4 Massbus.

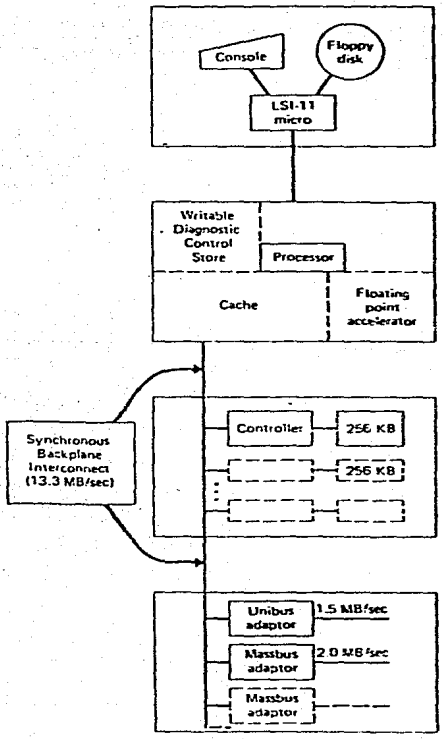


Figura 1
La VAX-11/780

El conjunto de instrucciones está implementado en microcódigo. Una memoria cache de 8 Kbytes proporciona acceso ultra-rápido a datos utilizados frecuentemente. Otra memoria cache de 128 bytes es utilizada para facilitar la traducción de direcciones físicas. Un buffer de instrucción es utilizado de tal manera que el procesador puede acceder y decodificar la siguiente instrucción a ejecutar mientras la instrucción actual está ejecutándose. Otro buffer, colocado en la interfaz SBI-CPU, permite al CPU efectuar escrituras al buffer, iniciar la transferencia hacia alguna unidad de almacenamiento y continuar con la ejecución sin tener que esperar a que se complete la escritura.

Se proporciona un conjunto sustancial de capacidades de detección y correción de errores que efectivamente mejora la confiabilidad y facilidad de mantenimiento del sistema. El Sistema Operativo inclusive puede mapear alrededor de áreas defectuosas de almacenamiento y continuar su operación normal.

La consola inteligente contiene una microcomputadora LSI-11 con RAM, ROM y unidad de floppy. La unidad de floppy se utiliza tanto como para la instalación del sistema como para hacer actualizaciones en las versiones del software.

6.6 ADMINISTRACION DE LA MEMORIA.

El administrador de memoria en VMS consiste de dos partes, el "Pager" o paginador y el "Swapper". El paginador es responsable de leer una página referenciada cuando ocurre una falta de página, y de escribir hacia disco una página residente cuando es necesario crear espacio para una página que llega. El paginador se ejecuta en el contexto del proceso que lo llama y es una subrutina compartida por todos los procesos dentro del sistema.

El swapper remueve o carga procesos enteros de memoria principal. Trabaja estrechamente junto con el "Scheduler" para determinar qué procesos intercambiar entre disco y memoria principal, y viceversa.

6.6.1 PAGINACION.

Cada proceso tiene un límite, llamado el "Resident Set Limit", para el número de páginas en memoria que puede ocupar. Aquellas páginas que un proceso tiene en memoria en un momento dado constituyen el "Resident Set" (también llamado "Working Set" dentro de la documentación de VMS). La lista del "resident set" señala hacia estas páginas.

Cuando una falta de página ocurre, el paginador adquiere el control. Dentro del espacio virtual, localiza la posición dentro de la tabla de páginas de la página faltante. Esta posición contiene información que ayuda al paginador a localizar la página en disco. El paginador localiza una página vacía en memoria e inicia su lectura desde disco. La posición correspondiente dentro de la tabla de páginas es modificada para indicar la posición en memoria de dicha página. Cuando la página es leída, el paginador regresa el control al programa del usuario en la instrucción que generó la falta.

Conforme el proceso continúa con su ejecución, genera faltas de página cada vez que referencia una página que no está en su resident set. Si el proceso aún no alcanza su límite máximo de páginas permitidas cuando ocurre una falta de página, la nueva página es leída a memoria. Si el límite se ha alcanzado, entonces el proceso debe desprenderse de una de sus páginas para dejar espacio a la nueva página.

Es interesante observar que VAX no utiliza un esquema sofisticado para reemplazar una página. En cambio, las páginas se reemplazan siguiendo un esquema de "First-In-First-Out". Sin embargo, como se verá a continuación, el costo de eliminar una página que está por ser referenciada (una posibilidad real dentro de un esquema FIFO), se mantiene bajo mediante otros mecanismos proporcionados por VMS.

El sistema mantiene dos listas de páginas físicas; la lista de páginas libres y la lista de páginas modificadas. Cuando un proceso necesita una página física, toma la página que está al principio de la lista de páginas libres. El sistema trata de mantener siempre algunas páginas libres pero ésto no siempre es posible.

Cuando un proceso debe desprenderse de una página, ésta es puesta a la cabeza de la lista de páginas libres o de la lista de páginas modificadas, dependiendo si se trata de una página modificada o no. Las páginas modificadas no se escriben en disco hasta que la lista alcanza cierto tamaño. Las páginas sin modificar permanecen en la lista de páginas libres hasta que alguna necesite ser asignada. Una página en cualquiera de estas listas puede ser reclamada por un proceso que referencie dicha página. La "reclamación de página" hace que una página referenciada regrese a memoria a un costo relativamente bajo, al mismo tiempo que ayuda a mantener bajo el número de accesos a disco provocadas por la paginación. De hecho, este esquema es un tanto injusto con procesos relativamente estables. En efecto, los procesos que constantemente tienen faltas de página tienden a extender su resident set por medio de las listas de páginas libres y modificadas.

Las páginas son escritas a disco en conjuntos llamados 'Clusters' para minimizar la entrada/salida asociada con la paginación. Por supuesto, la reclamación de páginas remueve muchas de ellas de la lista de páginas modificadas, lo que reduce aún más las operaciones de entrada/salida.

Con varias páginas dentro de la lista de páginas modificadas, el sistema puede tratar de escribir páginas contiguas de los mismos procesos en localidades contiguas de disco. Esto permite que futuras lecturas traigan clusters de páginas adyacentes si así se desea.

Los procesos que comienzan su ejecución tienden a experimentar un número elevado de faltas de página. VMS trata de minimizar esta actividad de paginación haciendo lecturas de un gran número de páginas para estos nuevos procesos.

VMS ajusta dinámicamente el resident set size de un proceso basado en la tasa de faltas de página que está experimentando. Lo incrementa para procesos con una tasa elevada y lo disminuye si la tasa es baja. Esta estrategia tiende a igualar la tasa de faltas de página de todos los procesos.

6.6.2 SWAPPING.

Además de la paginación, VMS mueve por completo el resident set del 'Balance Set' (es decir, del conjunto de procesos que están en memoria principal) hacia disco y viceversa. Este movimiento se denomina 'swapping' y es realizado por un proceso llamado 'swapper'.

El swapper intercambia resident sets con otros que están en disco en espera de ser ejecutados. VMS difiere de otros sistemas de memoria virtual en este aspecto. Muchos otros sistemas requieren que los procesos no residentes demanden memoria una página a la vez. En cambio, VMS intercambia procesos completos, lo que reduce significativamente los accesos a disco provocados por la paginación.

VMS garantiza que un proceso que acaba de ser instalado en memoria recibirá al menos un quantum antes de que sea intercambiado otra vez a disco. La manera de determinar el siguiente proceso a intercambiar es directa: se escoge al proceso residente en disco con la más alta prioridad. En este punto, el swapper debe encontrar un número suficiente de páginas para almacenar el proceso. Dichas páginas se obtienen de diversas maneras:

- Pueden tomarse de la lista de memoria libre.

- La lista de páginas modificadas puede escribirse a disco para crear más páginas libres.
- Un proceso de igual o menor prioridad puede ser pasado a disco.

Cuando un proceso es muy grande puede ser necesario sacar de memoria a varios procesos.

6.7 SCHEDULING.

VMS utiliza un esquema de prioridad para incluir los procesos ejecutables dentro del balance set. Los procesos normales se conocen como procesos de 'timesharing' o de 'background', los procesos de tiempo real se conocen como 'time-critical'.

VMS utiliza 32 niveles de prioridad para incluir los procesos. Las prioridades de la 0 a la 15 son repartidas a los procesos normales; prioridades en el rango 16-31 son repartidas a los procesos de tiempo real. Mientras mayor es un número, mayor es la prioridad. Los procesos de tiempo real se incluyen estrictamente de acuerdo a su prioridad. Las prioridades de los procesos normales se varian dinámicamente para alcanzar un máximo de utilización del CPU y de procesamiento de entrada/salida. Cada evento dentro del sistema tiene asociado un incremento en la prioridad. Cuando un evento ocurre, y si ocasiona que el proceso se vuelva ejecutable, el scheduler modifica la prioridad y la hace igual a la prioridad base mas la prioridad del evento. Cada vez que un proceso es incluido en el balance set, su prioridad va disminuyendo pero nunca por debajo de su prioridad base.

6.8 ENTRADA/SALIDA.

El sistema de Entrada/Salida de VMS está organizado en varias capas como se ilustra en la Figura 2. Desde el nivel más alto hasta el mas bajo, las operaciones de Entrada/Salida proceden de las más lógicas a la mas físicas. En los niveles más altos, VMS maneja archivos, registros y campos. En los niveles más bajos, maneja la operación física y la organización de los diversos dispositivos de Entrada/Salida.

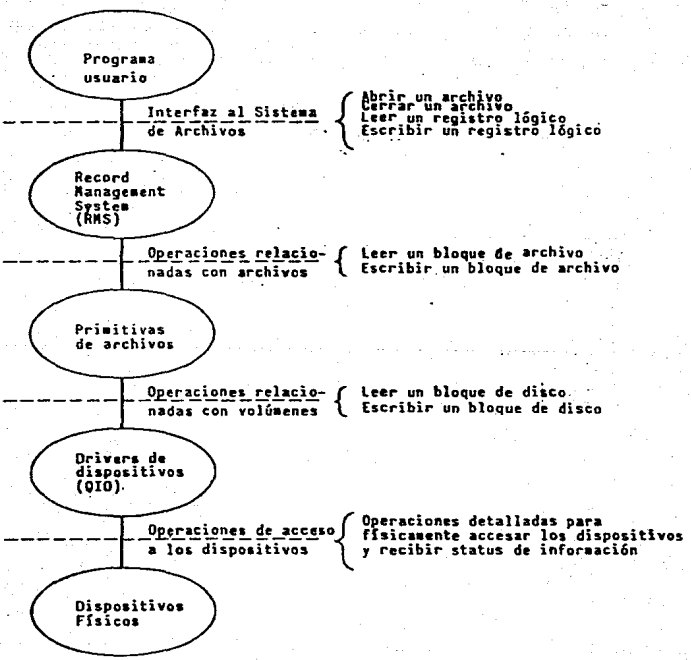


Figura 2
Capas que componen el Sistema de Entrada/Salida

La Entrada/Salida normalmente es asincrona. Sin embargo, los procesos pueden explícitamente esperar a que una operación de Entrada/Salida termine o solicitar una notificación de este evento. Los procesos que desean esperar a que termine una operación de Entrada/Salida llaman a una rutina del sistema operativo solicitando que sean suspendidos. Es importante notar que en VMS dicha suspensión no ocurre automáticamente como resultado de una solicitud de E/S.

Estructuras de datos son utilizadas para describir el estado de E/S del sistema en cualquier momento. Bloques de control especiales describen cada:

- Adaptador al bus.
- Controlador.
- Dispositivo.
- Solicitud de E/S incompleta.

Los diversos bloques de control están ligados para formar una estructura de datos que representa la topología de hardware del sistema de E/S. Existen seis distintos bloques de control cuyas funciones son:

- Device Data Block (DDB)- Contiene información común a los varios dispositivos de un determinado tipo que están conectados a un solo controlador.
- Unit Control Block (UCB)- Contiene las características y estado de un dispositivo individual y el contexto del proceso que lo controla.
- I/O Request Packet (IRP)- Contiene la información que describe una solicitud de E/S en particular.
- Channel Request Block (CRB)- Contiene información que describe el estado de un controlador. También tiene la identificación del dispositivo que en el momento está transfiriendo datos, y las identificaciones de los dispositivos que están a la espera de que el controlador esté disponible.
- Interrupt Dispatch Block (IDB)- Contiene información acerca de las características y estado del adaptador Massbus y del adaptador Unibus.

El contenido de todos estos bloques de control indican el estado de todo el hardware de E/S del sistema en un momento dado.

6.8.1 COMPONENTES DEL SISTEMA VMS I/O

E/S en VMS implica tres componentes a saber:

- o System Service Queue I/O (QIO).
- o Drivers del dispositivo.
- o Rutinas de post-proceso de E/S.

QIO es un procedimiento llamado por el programa del usuario que solicita una operación de E/S. QIO valida los argumentos dados por el usuario y construye un IRP describiendo la operación a realizar. El IRP contiene los datos que necesita el driver del dispositivo para realizar la transferencia física de datos.

La UCB apunta hacia una lista de solicitudes de E/S que están en espera. QIO inserta el nuevo IRP dentro de las colas apropiadas para el dispositivo apropiado, regresando de nuevo al programa del usuario. Dicho programa puede continuar o solicitar su suspensión hasta que se complete la operación.

El driver del dispositivo controla la operación de éste. Este proceso se ejecuta con una alta prioridad para garantizar una alta utilización de los recursos. La UCB define un proceso driver en la misma forma que el Process Control Block define a cualquier otro proceso. Cada dispositivo tiene asociado su UCB que indica el estado del dispositivo, así como el estado del driver de dicho dispositivo.

Las rutinas de post-proceso de E/S completan estas operaciones. Regresan el status final y los datos hacia la memoria del proceso del usuario.

6.8.2 CONTROL DE FLUJO DE E/S.

En esta sección, el flujo de una sola solicitud de E/S en VMS se seguirá en detalle.

Cada solicitud de E/S se inicia cuando un proceso llama a QIO. QIO valida la solicitud, construye un IRP, lo encola dentro del UCB apropiado y regresa el control al usuario. El IRP define la solicitud de E/S al sistema. En particular, contiene toda la información requerida para permitir que la solicitud sea atendida de manera asíncrona, para posteriormente avisar de su terminación.

La secuencia precisa de eventos a partir de este momento se muestra en la Fig. 3. El usuario llama a la rutina QIO que opera dentro del propio proceso del usuario, pero en modo Kernel. QIO localiza la UCB del dispositivo y checa cualquier parámetro que sea independiente del dispositivo dentro de la llamada. Llama a diversas subrutinas del driver para validar aquellos parámetros que sean dependientes del dispositivo. Entonces, construye la IRP y la encola dentro de la UCB, donde permanece hasta que el driver está listo para procesarla.

El driver remueve de la cola la solicitud y comienza la operación de E/S indicada. Después, voluntariamente suspende su ejecución poniendo su UCB en una cola de espera, permitiendo que se continúe con otros procesos mientras el dispositivo completa la operación. Cuando ésto ocurre, se genera una interrupción al Sistema Operativo que entonces reinicia el proceso driver. El driver copia el status resultante de la operación de E/S de los registros del dispositivo y del controlador y coloca esta información dentro del IRP. Este es colocado en una cola en espera de las rutinas de post-proceso de E/S.

La rutina de post-proceso checa el IRP para determinar la manera de completar la E/S. Copia el status hacia la memoria del usuario y notifica al proceso que la operación de E/S ha terminado.

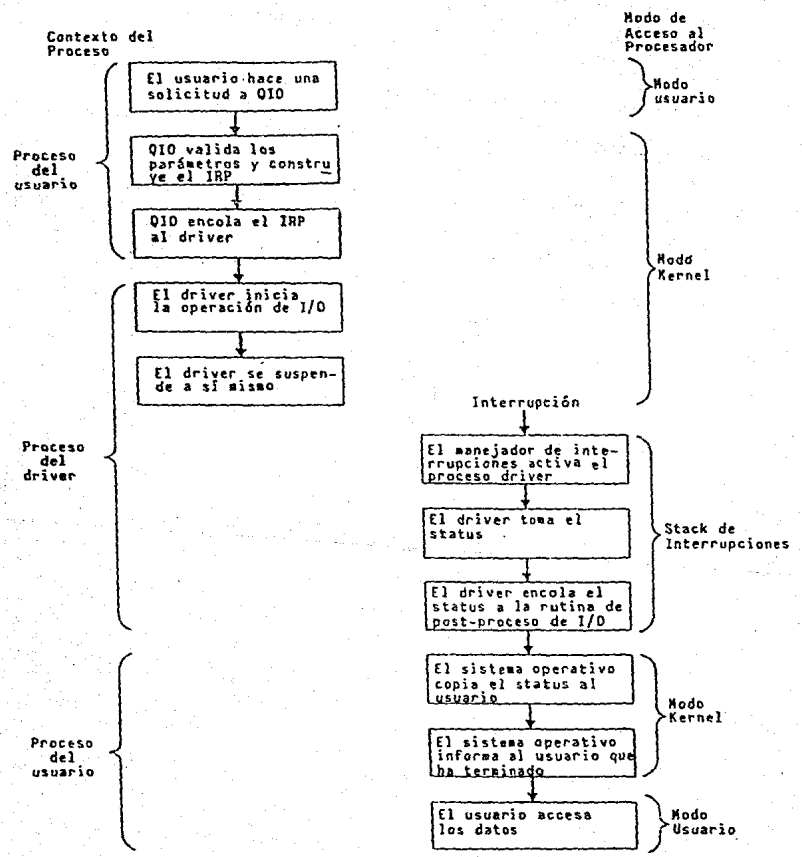


Figura 3
Flujo en el procesamiento de una solicitud de E/S

6.9 SINCRONIZACION E INTERCOMUNICACION ENTRE PROCESOS.

VAX/VMS proporciona una serie de facilidades para la intercomunicación de procesos para sincronizar su ejecución, enviarse mensajes y compartir datos entre diversos procesos cooperativos. Cuatro diferentes técnicas de comunicación existen:

- o Banderas comunes de eventos (Common Event Flags).
- o Mailboxes.
- o Areas comunes de almacenamiento.
- o Archivos compartidos.

Estas técnicas ofrecen varias velocidades y grados de generalidad a procesos que desean comunicarse unos con otros.

6.9.1 BANDERAS COMUNES DE EVENTOS.

La bandera de eventos es el mecanismo de intercomunicación de procesos más simple que existe en VMS. Una bandera de eventos es un bit de estado que puede encenderse o apagarse indicando la ocurrencia de un evento. Varios mecanismos para su proceso se proporcionan:

- o Enciende o apaga una bandera específica.
- o Prueba el estado actual de una bandera.
- o Coloca un programa en estado de espera hasta que una bandera o grupo de banderas se encienden.

Cada bandera reside en un cluster de eventos de 32 bits. Cada proceso tiene acceso a 4 clusters numerados del 0 al 3. Dos de los 4 clusters son para funciones de proceso locales y dos son para intercomunicación de procesos.

Los clusters 2 y 3 se llaman clusters comunes de banderas de eventos. Son creados dinámicamente mediante llamadas a los System Services y pueden compartirse con otros procesos. Los procesos deben asociarse con un cluster común; el primer proceso en hacerlo ocasiona que el cluster sea creado, los demás procesos en hacerlo tienen acceso a las banderas.

VMS proporciona seis servicios generales para realizar operaciones en banderas de eventos locales y comunes:

- Enciende una bandera de evento.
- Apaga una bandera de evento.
- Lee una bandera de evento.
- Espera por una bandera de evento.
- Espera por un OR lógico de banderas de eventos.
- Espera por un AND lógico de banderas de eventos.

Tres servicios son específicos de las banderas comunes de eventos:

- Asocia un cluster común de banderas de eventos.
- Disasocia un cluster común de banderas de eventos.
- Borra un cluster común.

6.9.2 MAILBOXES.

Las banderas de eventos son útiles para esperar eventos y determinar cuando éstos han ocurrido, pero no permiten el paso de datos entre procesos. Para este fin, la VAX proporciona un mecanismo más general llamado comunicación por "mailbox". Un mailbox es un dispositivo virtual que puede utilizarse para la intercomunicación de procesos. Se escribe o se lee de un mailbox como si éste fuera un archivo normal.

Los mensajes se insertan en un lado y se lee por el otro, de tal manera que los mensajes más viejos se leen primero. Un proceso que lee de un mailbox recibe el mensaje más viejo o una indicación de que el mailbox está vacío. Un proceso puede solicitar que le sea notificado cuando un mensaje sea colocado en el mailbox.

El creador de un mailbox especifica el tamaño y número máximo de mensajes que el mailbox puede tener. El creador puede controlar cuales procesos tienen derecho a escribir sobre el mailbox. Los mailboxes están protegidos de la misma manera que lo está un archivo común.

Cuando un proceso crea otro proceso, puede especificar un mailbox que recibirá el status de terminación cuando el proceso creado termine.

6.9.3 AREAS COMUNES DE ALMACENAMIENTO.

El más general de los mecanismos de intercomunicación es el área común de almacenamiento. Los procesos pueden compartir memoria dentro de su espacio de direccionamiento a través de secciones globales. Una sección global es un área de almacenamiento que puede ser direccionada por varios procesos.

Los procesos que utilizan almacenamiento compartido para intercomunicación de procesos utilizan su propio protocolo para sincronizarse, proporcionando un mecanismo de comunicación muy general. Los procesos que utilizan este mecanismo frecuentemente usan las banderas de eventos así como las áreas comunes.

Algunas secciones globales están instaladas permanentemente dentro del sistema; otras son creadas dinámicamente. Cuando una sección global dinámica es referenciada, es mapeada hacia el proceso que hace referencia y es borrada cuando no hay más referencias hacia ella.

6.9.4 ARCHIVOS COMPARTIDOS.

Los procesos pueden comunicarse a través de archivos compartidos. Los archivos de disco pueden ser compartidos por un número cualquiera de programas que están leyendo (pero no escribiendo) el archivo. Archivos secuenciales pueden ser accesados por un solo escritor o compartido entre muchos lectores. Archivos relativos e indexados pueden ser compartidos por múltiples escritores y lectores.

6.10 SYSTEM SERVICES.

Los system services son procedimientos incorporados en y usados por el sistema operativo para controlar los recursos disponibles para los procesos, para proporcionar comunicación entre los procesos y para realizar operaciones básicas del sistema operativo tal como la coordinación de las operaciones de entrada/salida.

A pesar de que la mayoria de los system services son básicamente utilizados por el sistema operativo, muchos están disponibles a los usuarios y proporcionan técnicas que pueden ser utilizadas en programas de aplicación. Por ejemplo, cuando un usuario entra al sistema se llama al system service "Create Process" para crear un proceso al usuario. El usuario a su vez puede utilizar este system service para creur un subproceso.

Los system services están organizados de la siguiente manera:

- Servicios para banderas de eventos.
- Asynchronous System Traps.
- Servicios de Nombres lógicos.
- Servicios de Entrada/Salida.
- Servicios de control de procesos.
- Servicios de temporización y conversión de tiempo.
- Servicios de manejo de condiciones.
- Servicios de manejo de memoria.

CAPITULO 7

EL SISTEMA OPERATIVO UNIX Y LA ALTOS 986.

7.1 INTRODUCCION.

El campo de los sistemas operativos ha sufrido muchos fracasos frustrantes y costosos, pero al mismo tiempo, muchos avances significativos han ocurrido. El sistema operativo UNIX es uno de estos éxitos.

Los sistemas UNIX fueron diseñados para ser sistemas adecuados para el desarrollo de programas. Tienen un simple pero poderoso lenguaje de comandos y un sistema de archivos que es independiente de los dispositivos.

Cientos de sistemas UNIX están en uso a través de la compañía Bell, y es muy popular dentro de las universidades y la industria. La configuración mínima para soportar un sistema multiusuario es un procesador de 16 bits con 256 Kbytes de memoria principal y un disco de acceso rápido.

7.2 HISTORIA.

Durante los años 1965-1969, los Laboratiorios Bell participaron, junto con General Electric (más tarde Honeywell) en el proyecto MAC del MIT (Massachusetts Institute of Technology) para desarrollar el sistema Multics. Originalmente diseñado para correr en la poderosa computadora GE-645, Multics es un sistema extremadamente complejo. Es un sistema de propósito general construído para satisfacer las necesidades de una gran diversidad de usuarios.

En 1969, los Laboratorios Bell abandonaron este proyecto. Algunos miembros del equipo de investigadores comenzaron a trabajar en un proyecto menos ambicioso llamado UNIX. El grupo dirigido por Ken Thompson, se esforzó por crear un ambiente que facilitase la investigación y desarrollo de programas. La primera versión de UNIX se produjo para una PDP-7 y fué escrito en lenguaje ensamblador.

El sistema UNIX llamó la atención de Dennis Ritchie que trabajaba en la traducción de programas al lenguaje C alrededor del año de 1972. Esto ayudó a hacer el software de UNIX más transportable y entendible. El código de máquina creció en aproximadamente una tercera parte comparado contra la versión de lenguaje ensamblador, pero parte de este incremento se debió a la adición de soporte para multiprogramación y a la habilidad de compartir procedimientos reentrantes.

La PDP-11 ya era popular en las universidades cuando las noticias del sistema UNIX comenzaron a extenderse. En 1973, Western Electric accedió a dar licencia de uso del sistema a organizaciones no lucrativas. Esto hizo que UNIX estuviera al alcance de varias docenas de instituciones educacionales.

Para 1975, los sistemas UNIX ya eran muy populares dentro de las universidades y se había creado una organización de usuarios llamada USENIX. La primera versión de UNIX para el mercado fué la versión V. Esta versión más parecía el resultado de un proyecto de investigación que el resultado de un proyecto de desarrollo de un producto comercial. La versión VI apareció en 1975 y aún es ampliamente utilizada. La versión VII, dada a conocer en 1979, refleja un gran esfuerzo para pulir el sistema y convertirlo en un producto comercial viable.

7.3 VERSIONES DEL SISTEMA UNIX.

A pesar de haber sido diseñado para la familia PDP-11 de minicomputadoras, el sistema UNIX ha sido implementado exitosamente en muchos otros sistemas. En 1977 fué implementado en una Interdata 8/32. En la actualidad está disponible para una gran variedad de macrocomputadoras. La UNIVAC 1100 fué la primer macrocomputadora en adoptar el sistema UNIX. También se encuentra disponible para las computadoras VAX.

Varias versiones del sistema UNIX están en uso actualmente. En las siguientes secciones se discuten dos de las versiones más populares.

7.3.1 UNIX ESTANDAR.

Los sistemas UNIX estándar son sistemas con multiprogramación y tiempo compartido. Proporcionan soporte hasta de 40 usuarios y son los sistemas más populares en la actualidad. Tienen un sistema jerárquico de archivos con protección completa, independientes de los dispositivos, con volúmenes desmontables y características que facilitan la simpleza en la programación. Cualquier programa o grupo de programas pueden ejecutarse asincronamente en "foreground" (modo interactivo) o "background" (modo no interactivo) sin ningún cambio.

Los sistemas UNIX no distinguen entre los programas del usuario y los programas del sistema en cuanto a capacidades o uso, excepto por las restricciones impuestas por la protección de archivos. Se utilizan "buffers" para la Entrada/Salida, la distribución de memoria principal y disco las maneja automáticamente el sistema y son invisibles para el usuario. El código generado por el compilador C es reentrante y compartible.

Los sistemas UNIX permiten al usuario direccionar la salida de un programa y convertirla en la entrada de otro. Esto facilita la solución de problemas de programación en gran escala al poder utilizar directamente varios programas pequeños, en lugar de desarrollar un programa completamente nuevo.

Los sistemas UNIX estándar incluyen un conjunto de programas tales como un editor de texto, un intérprete de comandos programable, varios compiladores para lenguajes populares, un ensamblador, depuradores, formateadores de texto (con capacidades matemáticas), procesadores de texto, una utilería de sort, comunicación interusuario, programas administrativos y de mantenimiento, librerías estándar del sistema y del usuario y un paquete de juegos.

El código fuente para todos los programas (excepto el paquete de juegos) se proporciona junto con el sistema. Gran parte de la documentación se encuentra disponible a los usuarios de manera interactiva.

7.3.2 UNIX/V7: LA SEPTIMA EDICION DEL SISTEMA.

La séptima edición del sistema, para usar en una PDP-11/45, contiene un número significativo de mejoras sobre las versiones anteriores. Los archivos pueden ser hasta de 1,000 millones de bytes. El sistema se ha hecho más portable. El lenguaje C se ha extendido. Se ha incluído un Shell (intérprete de los comandos del usuario) más poderoso que incluye variables string, programación estructurada y otras funciones.

La séptima versión de UNIX refleja los esfuerzos de los Laboratorios Bell para satisfacer las necesidades de un número cada vez mayor de usuarios.

7.4 METAS DE DISEÑO.

Dos aspectos del origen de los sistemas UNIX han contribuído grandemente a su diseño único:

- La primera versión de UNIX fué construída en unos cuantos años y básicamente por dos personas.
- La personas que construyeron UNIX eran los mayores usuarios del sistema.

La coherencia de UNIX es atribuible a estos factores.

Es posible para un programador de sistemas experimentado comprender todo el sistema en cuestión de semanas en lugar de los meses o años que toma aprender algunos de los grandes sistemas operativos de hoy.

La primer meta de diseño fué mantener el sistema operativo simple. El usuario es quien tiene la tarea de proporcionar una auténtica sofisticación.

La segunda meta fué la generalidad. Un método único debería servir a una variedad de propósitos. La generalidad se manifiesta en diversas áreas del sistema:

- Las mismas llamadas del sistema se utilizan para leer o escribir archivos, dispositivos y mensajes entre procesos.
- Los mismos nombres, alias y mecanismos de protección se aplican a archivos de datos, directorios y dispositivos.
- El mismo mecanismo es utilizado para manejar los traps de software y del procesador.

La tercer meta es crear un ambiente en el cual grandes tareas pueden llevarse a cabo combinando programas pequeños ya existentes en lugar de desarrollar nuevos programas.

7.5 CONTROL DE PROCESOS.

El corazón del sistema UNIX es el Kernel. Consiste de aproximadamente 10,000 líneas de código en C y otras 1,000 líneas de código ensamblador. Del código en ensamblador, la mayor porción ejecuta funciones de hardware que no son convenientes o prácticas en C. El resto es código ejecutado frecuentemente y fué escrito en ensamblador para eficiencia en su ejecución. El Kernel representa una pequeña parte del sistema operativo (entre 5 y 10% del código).

El estado actual de la pseudocomputadora de un usuario individual se llama imagen. Un proceso es la ejecución de una imagen. Una imagen contiene:

- Una imagen de memoria.
- Valores de los registros generales.
- El directorio de trabajo.

y otra información. La imagen de un proceso reside en memoria principal durante la ejecución de dicho proceso. Una imagen puede ser intercambiada a disco si un proceso de mayor prioridad necesita ejecutarse y necesita memoria.

Cada imagen se divide en tres segmentos lógicos:

1. Segmento de procedimientos reentrantes (comenzando en la localidad cero del espacio de almacenamiento virtual).
2. Segmento de datos.
3. Segmento de stack.

Los segmentos de procedimientos reentrantes pueden ser compartidos entre procesos; solamente una copia de un segmento compartido se mantiene en memoria principal. El segmento de datos comienza después y puede crecer hacia abajo (hacia direcciones de memoria mayores). El segmento de stack comienza en la dirección mayor del espacio de almacenamiento virtual y crece hacia arriba conforme la información es almacenada debido a llamadas a subrutinas o interrupciones.

Los segmentos de texto de solo-lectura están controlados en la tabla de texto (Fig. 1). Cada entrada contiene tanto direcciones en memoria primaria como secundaria del segmento. Cuando un proceso primero ejecuta un segmento de texto compartido, ese segmento es colocado en memoria secundaria, y la tabla de texto es creada con las direcciones de memoria adecuadas y un contador indicando el número de procesos que comparten el segmento. Cuando este contador llega a cero, la entrada es liberada y la memoria principal y secundaria ocupada por el segmento se libera.

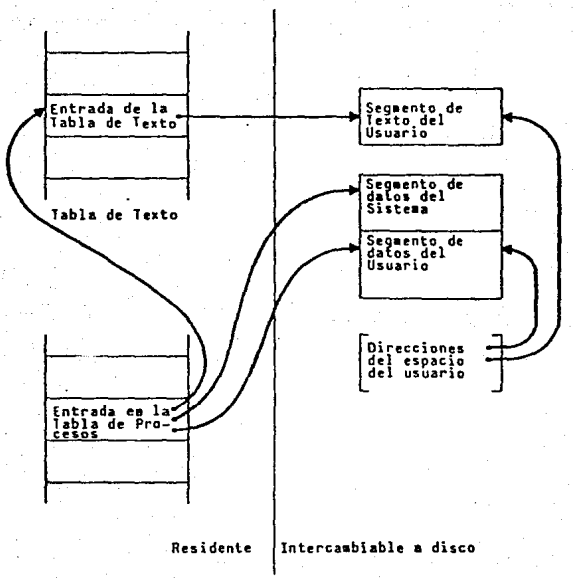


Figura 1
Tablas de Control de Procesos

El segmento de datos de la imagen contiene datos privados para lectura o escritura del proceso. Los datos del sistema asociados con este proceso se mantienen en un segmento separado de tamaño fijo. Este segmento de datos del sistema (system data segment) es intercambiado junto con el proceso. Este segmento del sistema contiene datos acerca del proceso activo tales como:

- o Valor de los registros.
- o Descriptores de archivos abiertos.
- o Datos de contabilización.
- o Area de datos temporales (Scratch Data Area).
- o Stack para la fase de proceso del sistema.

Un segmento de datos del sistema no es direccionable directamente desde el proceso al cual corresponde.

Cada proceso tiene una tabla de proceso (process table) conteniendo los datos necesitados por el sistema cuando el proceso no esta activo. La tabla del proceso contiene el nombre del proceso, la localización de sus segmentos y otra información. Existe durante la vida del proceso y es direccionable por el Kernel.

7.5.1 CREACION DE PROCESOS.

Los nuevos procesos se crean mediante la instrucción fork. Esta llamada al sistema causa que el proceso actual se divida en dos procesos independientes y concurrentes llamados proceso padre y proceso hijo. Los dos procesos no comparten memoria principal, pero comparten todos los archivos que estén abiertos. Copias de todos los segmentos escribibles de datos se realizan para el hijo. La llamada a fork devuelve un valor de tal manera que cada proceso puede determinar si es el padre o el hijo.

7.5.2 EXEC.

La instrucción exec permite a un proceso ejecutar un archivo. Esto causa que los segmentos de texto y datos actuales sean intercambiados por los nuevos segmentos especificados por el archivos a ejecutar. Los segmentos anteriores se pierden. Esto efectivamente cambia el programa que el proceso está ejecutando, pero no el proceso en si mismo. Los archivos que hayan sido abiertos antes de exec permanecen abiertos.

7.5.3 SINCRONIZACION DE PROCESOS.

La sincronización de procesos se lleva a cabo mediante un mecanismo de eventos. Los procesos esperan que ocurran los eventos. Tablas de proceso se asocian con los eventos. Los eventos se representan como las direcciones de las tablas correspondientes. Un proceso padre esperando a que termine uno de sus procesos hijo espera por un evento que es la dirección de su propia tabla de proceso. La señalización de un evento que nadie está esperando no tiene efecto.

En cualquier momento, todos excepto uno de los procesos han llamado a un "Espera-Evento" dentro del Kernel. El otro proceso es el proceso que está ejecutándose; cuando llama a "Espera-Evento", se despacha otro proceso cuyo evento ha sido señalizado.

7.5.4 INCLUSION DE PROCESOS.

Los procesos pueden correr en cualquiera de dos estados: User State y System State. En user state, un proceso ejecuta programas de usuario y accesa el segmento de datos del usuario. En system state, un proceso ejecuta código del Kernel y accesa el segmento de datos del sistema.

El principal objetivo de la inclusión de procesos en UNIX es proporcionar una rápida respuesta a los usuarios interactivos. Los procesos se incluyen de acuerdo a prioridades. Las prioridades iniciales las asigna el código del Kernel que procesa los eventos de espera. Los eventos de disco reciben una alta prioridad. Eventos de las terminales, eventos de la hora del día y eventos de los procesos de los usuarios reciben progresivamente menores prioridades.

Los procesos de los usuarios reciben prioridades basados en la cantidad de tiempo de procesador que han recibido. Procesos que han recibido una gran cantidad de tiempo reciben menor prioridad y viceversa. Este método asegura una rápida respuesta a los usuarios interactivos. Todos los procesos del sistema tienen una prioridad mayor que los de los usuarios. Un proceso recibe un quantum de CPU de típicamente 1 segundo. Este termina ya sea porque el proceso termina voluntariamente antes de que expire el quantum o porque es suspendido al terminar el quantum. Procesos de igual prioridad se incluyen de acuerdo a una política de round-robin. El mecanismo para incluir procesos tiende a ser justo con todos los procesos.

7.5.5 SWAPPING.

Los procesos son intercambiados desde y hacia memoria secundaria según sea necesario. Tanto la memoria primaria como secundaria es distrubuída utilizando una política de first-fit. Si un proceso necesita crecer o solicita mayor memoria, se le da una nueva sección de almacenamiento lo suficientemente grande y todo el contenido de la sección anterior es copiado. El área anterior es entonces liberada.

Si no existe memoria suficiente al momento de la expansión, el proceso es intercambiado a memoria secundaria y es intercambiado a memoria primaria cuando exista suficiente memoria disponible.

El proceso swapper controla todos los intercambios de todos los demás procesos. Cuando decide intercambiar hacia memoria principal un proceso que está listo a ejecutarse, le proporciona suficiente memoria principal y lee sus segmentos. Este proceso entonces compite con todos los demás procesos por el tiempo de CPU. Si no existe suficiente memoria principal, el swapper examina la tabla de procesos para determinar que procesos pueden ser intercambiados a disco. El intercambio de procesos que están listos para ser ejecutados se realiza siguiendo una política first-in-first-out, es decir, el proceso que tenga más tiempo en disco es el que primero es llevado a memoria principal. Los procesos residentes en memoria principal son llevados a disco teniendo en cuenta el tiempo que han estado ocupándola, así como el hecho de que se esté a la espera de eventos lentos como E/S.

7.5.6 TERMINACION DE PROCESOS.

Los procesos pueden terminar voluntariamente ejecutando la instrucción exit, o involuntariamente como resultado de acciones ilegales o señales y traps generadas por el usuario. Las traps usualmente son ocasionadas por fallas del programa tales como referencias a direcciones ilegales o a intentos de ejecutar códigos de operaciones desconocidas. La terminación involuntaria provoca que la imagen del proceso sea escrita en un archivo. Esta imagen puede ser examinada más tarde con un programa depurador para determinar la causa de la terminación.

La instrucción interrupt puede ser utilizada para terminar un programa. La instrucción quit funciona de manera semejante a interrupt excepto que también escribe la imagen del proceso en un archivo.

7.6 SISTEMA DE ENTRADA/SALIDA.

El sistema de E/S consiste de dos diferentes componentes:

- El sistema estructurado de E/S o E/S por bloques.
- El sistema no estructurado de E/S o E/S por caracter.

En UNIX la Entrada/Salida es manejada básicamente por cinco llamadas al sistema denominadas: open, close, read, write y seek. Otras tres llamadas, gtty, stty y stat son utilizadas para obtener y fijar información acerca de archivos y terminales.

Para abrir un archivo se utiliza:

```
fd = open( filename, mode)
```

donde mode indica si se va a leer, escribir o ambos y fd es llamado descriptor de archivo (file descriptor) para ser utilizado en siguientes referencias al archivo.

La lectura y escritura se lleva a cabo mediante:

```
n_read = read( fd, buffer, n_deseados)
```

y

```
n_written = write( fd, buffer, n_deseados)
```

En el caso de la lectura existen tres posibilidades, cada una de las cuales involucra lectura secuencial:

- Si es la primer lectura, entonces la lectura es secuencial a partir del principio del archivo.
- Si el archivo acaba de ser leído, entonces la lectura actual obtiene los datos que siguen a los que fueron leídos anteriormente.
- Si se ejecutó seek, entonces la lectura es secuencial a partir del desplazamiento especificado en la llamada a seek.

Lo mismo ocurre para la escritura. Toda la lectura y escritura es secuencial, pero el efecto de un acceso directo se logra utilizando una llamada a seek para ajustar el desplazamiento dentro del archivo de acuerdo con:

```
seek(fd, desplazamiento, tipo)
```

en el que tipo significa:

- Si el desplazamiento es relativo o absoluto.
- Si el desplazamiento es en bytes o en bloques de 512 bytes.

La operación seek también trabaja bien en cinta magnética. Por supuesto, programas con "acceso directo" trabajan mucho más lentamente cuando se utiliza cinta magnética. Los archivos se cierran ejecutando:

```
close(fd)
```

Para usuarios que estén familiarizados con sistemas operativos mas grandes, parecería que UNIX es un juguete. Sin embargo, grandes sistemas administradores de datos se han construido con estas funciones. El problema real no es de capacidad funcional, sino de comportamiento. Ambientes comerciales de procesamiento de datos frecuentemente demandan una mucho mayor capacidad que la proporcionada por UNIX.

Las mismas funciones se usan para realizar E/S sobre dispositivos físicos y se utilizan exactamente de la misma manera. Los dispositivos físicos se representan dentro de UNIX como archivos especiales dentro de la estructura de archivos.

Normalmente, los programas no tienen que abrir su entrada y salida estándar, las cuales estan normalmente asignadas a la terminal del usuario. La terminal es abierta automáticamente por el comando login.

7.7 SISTEMA DE ARCHIVOS.

Un archivo del sistema UNIX es una colección de caracteres direccionables de manera aleatoria. Su tamaño es exactamente el número de caracteres que contiene, hasta un máximo de aproximadamente 1,000 millones. Un archivo contiene cualquier clase de datos que el usuario desee poner en él y no tiene ninguna estructura diferente a la impuesta por el propio usuario.

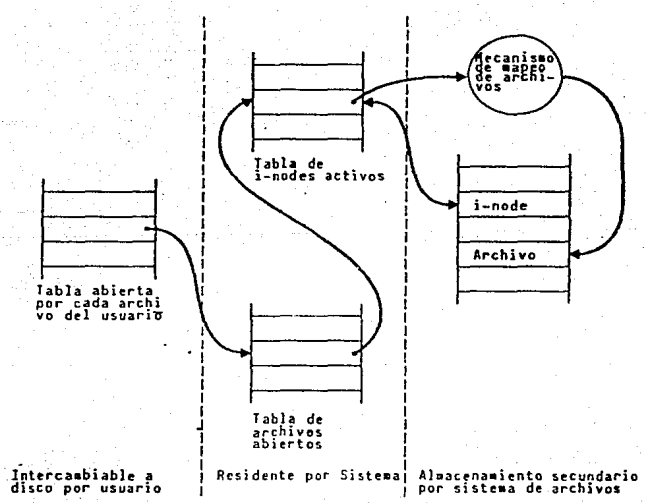


Figura 2
Estructura de Datos del Sistema de Archivos

El sistema de archivos (Fig. 2) reside principalmente en discos compuestos por bloques de 512 bytes cada uno. Un archivo divide al disco en 4 regiones (Fig. 3):

- Un bloque no usado.
- Un super bloque que contiene el tamaño del disco y los límites de las otras regiones.
- La lista "i-list" que contiene una lista de descripciones de archivos llamada "i-nodes".
- Áreas de almacenamiento para el contenido del archivo.

Un i-node contiene:

- Identificación del usuario.
- Grupo al que pertenece el usuario.
- Bits de protección.
- Espacio físico para almacenar el contenido del archivo.
- Tamaño del archivo.
- Hora de creación.
- Hora de última utilización.
- Hora de la última modificación.
- Número de ligas al archivo.
- Indicación de si el archivo es un directorio, un archivo ordinario o un archivo especial.

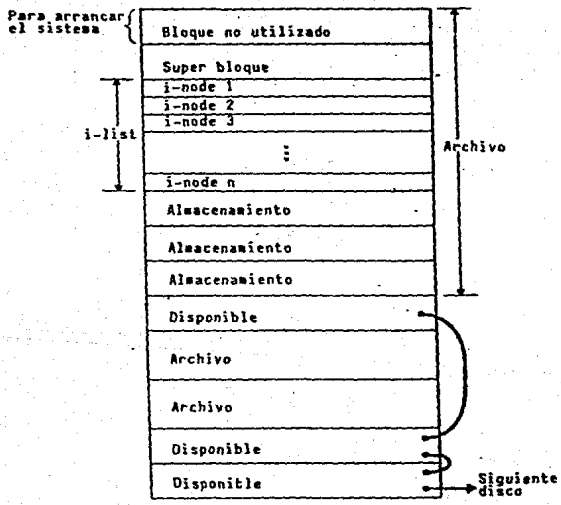


Figura 3
Estructura de archivos en un disco

En seguida de la i-list están bloques de almacenamiento disponibles para guardar el contenido del archivo. El restante espacio dentro del disco se mantiene por medio de una lista ligada de bloques disponibles.

El sistema de archivos de UNIX es una estructura de datos residente en disco y que contiene un super bloque que define el sistema de archivos, un arreglo de i-nodes que definen todos los archivos dentro del sistema, los archivos en si mismos y una colección de bloques disponibles. Toda la distribución se realiza con bloques de tamaño fijo.

Cada archivo está definido de manera única mediante un número mayor de dispositivo, un número menor de dispositivo y un índice llamado i-number (es un índice del archivo i-node dentro del arreglo de i-nodes). Cuando un driver de dispositivo es llamado, el número mayor de dispositivo es utilizado como índice dentro de un arreglo que contiene la dirección del driver. El número menor de dispositivo es utilizado por el driver para seleccionar uno de varios dispositivos iguales (por ejemplo discos).

Los archivos directorio listan los nombres de los archivos y proporcionan un mapeo entre estos nombres y los archivos en si. Los directorios son gráficas dirigidas con una estructura de árbol. Los archivos regulares y especiales pueden ser colocados en cualquier lugar de la gráfica. Programas no privilegiados no pueden escribir sobre los directorios, pero con los permisos adecuados pueden leer directorios. Los directorios no pueden ser ligados.

Existen muchos directorios que el sistema UNIX tiene para su propio uso. Uno de estos es el directorio root; la base para toda la estructura de directorios a través de la cual se localiza un archivo. Otros directorios del sistema UNIX contienen los programas y comandos disponibles para los usuarios y los archivos de los dispositivos.

Los nombres de archivo se construyen especificando una secuencia de directorios separados por una diagonal "/" y que desembocan en una hoja de un árbol particular. Los nombres de archivo sin esta diagonal implican que el camino comienza en el directorio actual de trabajo. Nombres de archivo que comienzan con "../" implican que el camino comienza con el directorio que es el padre del directorio actual. El nombre de archivo "datos" implica el archivo "datos" dentro del directorio actual. El nombre de archivo "/user1/federico/datos" busca dentro del directorio root el directorio user1, dentro de éste busca el directorio federico y dentro de éste busca el archivo datos. Este ejemplo ilustra la estructura jerárquica típica del sistema de archivos.

Un archivo que no sea directorio puede aparecer en muchos directorios diferentes bajo distintos nombres. Esto se llama 'linking'. Las entradas dentro de los directorios se llaman "links" o ligas. En UNIX, todas las ligas tienen igual estado. Un archivo no existe dentro de un directorio. En cambio, un archivo existe independientemente de las entradas en los directorios y las ligas en los directorios apuntan a los archivos físicos. Un archivo desaparece cuando la última liga hacia él desaparece. Una liga puede tener bits de protección diferentes a los que tiene el archivo en si. Esto resuelve el problema de restringir el acceso a ciertas personas.

Cada dispositivo tiene asociado uno o más archivos especiales. E/S utilizando archivos especiales es igual a la utilizada con archivos normales, pero estas intrucciones causan la activación de los dispositivos asociados. Cada archivo especial usualmente reside en el directorio /dev. Se pueden hacer ligas hacia archivos especiales de la misma manera que se hace con los archivos normales.

Toda la jerarquía del sistema de archivos no necesita residir en un mismo dispositivo como root. La llamada al sistema mount permite la incorporación de archivos que están en un volúmen removible dentro de la jerarquía del sistema de archivos.

Un número de identificación del usuario y 10 bits de protección se utiliza para implementar la protección de archivos. Nueve de estos bits implementan el acceso de lectura, escritura y ejecución para el dueño del archivo, otros miembros del mismo grupo y para los demás usuarios del sistema.

7.8 SHELL.

El shell es el mecanismo dentro de UNIX para comunicación entre el usuario y el sistema. Es un intérprete que lee líneas tecleadas por el usuario y causa la ejecución de las funciones solicitadas. El shell no reside permanentemente en memoria como el Kernel.

El lenguaje de comandos completo que interpreta el shell es grande y algo complejo, pero la mayoría de los comandos son fáciles de usar y relativamente consistentes. Un comando consiste de un nombre de comando seguido de una lista de argumentos separados por blancos. El shell divide la línea en sus componentes. El archivo especificado por el comando se carga y los parámetros se hacen disponibles a él. Se ejecuta fork creando un proceso hijo que ejecuta el comando mientras el proceso padre espera a que la ejecución termine. Cuando esto ocurre, el shell despliega su prompt en la terminal para indicar al usuario que puede continuar.

El shell porporciona a cada programa que ejecuta tres archivos abiertos: uno para entrada, otro para salida y otro para salida de errores. Estos archivos normalmente están asignados a la terminal del usuario pero pueden ser fácilmente redireccionados de la siguiente manera: si en la línea un argumento es precedido de "<" entonces se está haciendo referencia a un archivo para tomar la entrada. Un archivo con el prefijo ">" es para recibir la salida producida por el comando. El prefijo ">>" indica que la salida debe añadirse al archivo en lugar de reemplazarlo. La importancia de esto es que la entrada y la salida pueden ser redireccionadas sin modificar el programa.

El lenguaje de comandos del shell incluye caracidades de control de flujo tales como if-then-else, case, while y for.

Facilidades para concordancia de patrones se proporcionan para contruir nombres o grupos de nombres de archivos. El asterisco (*) indica que cualquier combinación de caracteres y el signo de interrogación indica un caracter solo.

7.8.1 PIPES Y FILTROS.

Una de las mas importantes contribuciones del sistema UNIX es el concepto de "pipe". Un pipe es un archivo abierto que conecta a dos procesos. La información escrita en un lado del pipe puede ser leída por el otro lado (Fig. 4). La sincronización, scheduling y los buffers son manejados automáticamente por el sistema. Utilizando la noción de pipe entre pares de procesos un usuario puede crear una cadena de procesos conectando varios de ellos en forma lineal. El usuario especifica una cadena de procesos al shell mediante una serie de nombres de archivos ejecutables separados por barras verticales. La salida del programa a la izquierda de la barra es la entrada del proceso a la derecha de la barra. Los procesos pueden ser interconectados de una manera más general.

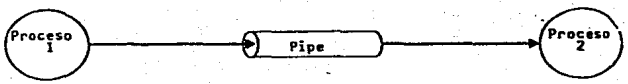


Figura 4
El concepto de pipe
El proceso 1 escribe al pipe. El proceso 2 lee del pipe

Un filtro (filter) es un programa que procesa una entrada y produce una sola salida. Por ejemplo, los procesadores de lenguaje proporcionados con UNIX no producen listados. En su lugar, su salida se alimenta en un filtro que imprime todos los listados de salida en un formato uniforme. Los filtros y los pipes pueden utilizarse para producir muchas salidas interesantes.

Supóngase que se desea determinar cuantos usuarios están actualmente en el sistema. Esto puede hacerse alimentando la salida del comando who dentro de la utileria que determina el número de líneas a su entrada. La línea:

```
who ! wc -l
```

causa que la salida del comando who, que puede ser parecida a:

```
zack      console     Aug 27 18:40
sbn       tty0        Aug 27 16:25
famf      tty1        Aug 27 19:20
suem      tty2        Aug 27 17:19
asa       tty3        Aug 27 20:01
```

sea enviada al programa wc que significa "word count". La opción -l indica que únicamente se imprimirá el número de líneas. Así, este comando imprime el número de usuarios en el sistema:

```
$ who ! wc -l
5
$
```

El usuario puede crear un archivo que tenga la línea de comandos antes vista y nombrarlo "users". Tecleando users, es posible conocer el número de usuarios en el sistema.

7.8.2 MULTITASKING.

El usuario puede especificar múltiples tareas terminando un comando con un ampersand (&). En este caso, el shell no espera a que termine de ejecutarse el comando. En cambio, inmediatamente está listo para recibir un nuevo comando mientras el comando anterior se ejecuta en "background". El comando especificado puede ser un archivo especificando otros comandos por lo que de esta manera el usuario puede hacer procesos batch en background. Por ejemplo, el comando:

```
(cc programa; a.out) &
```

provoca que los dos comandos (compila y ejecuta) sean ejecutados en background. El punto y coma es un separador de comandos. Indica que los comandos separados por él deben ejecutarse secuencialmente. Los parentesis se usan para agrupar comandos, de tal manera que el ampersand afecta a ambos comandos.

7.9 LA COMPUTADORA ALTOS 986.

Dentro de la nueva generación de computadoras de bajo costo y alto rendimiento basadas en el Sistema Operativo UNIX, la Altos 986 es una computadora apropiada para negocios pequeños, fabricantes de software, centros de investigación y laboratorios.

El mueble de la 986 contiene una unidad de disco flexible, un disco duro de 33 Mbytes, 1 Mbyte de memoria principal y 9 puertos serie. El Sistema Operativo se llama XENIX que es la versión que fabrica la compañía Microsoft de la Séptima Versión de UNIX.

La 986 realmente es una computadora multiproceso teniendo tres diferentes procesadores. El procesador que ejecuta todos los programas de usuario es un Intel 8086. Otro procesador, el Intel 8089 controla las interfaces hacia la unidad de disco y por último, dos procesadores Z-80 se encargan de manejar los puertos serie.

7.9.1 CARACTERISTICAS DE LOS PUERTOS.

La velocidad de transmisión de los puertos serie puede determinarse por medio de software a cualquiera de las velociadades estándar desde 110 hasta 19,200 bits por segundo.

Por otra parte, deficiencias en el software hacen sumamente difícil el utilizar modems. Otro problema con los puertos de comunicación es que los puertos son incapaces de aceptar una entrada de datos a una velocidad alta de manera sostenida. La máxima velocidad a la que pueden leer datos es 2,400 bps. Altos sostiene que esto se debe a la manera en que los buffers de E/S son manejados por XENIX.

Como conclusión es posible decir que la Altos 986 es una buena máquina de costo razonable. Sin embargo, deben mencionarse dos desventajas:

1. Las fallas en el software que maneja los modems hace que la computadora no sea adecuada para aplicaciones en las que se piense utilizar la máquina por vía telefónica.

2. No es una máquina recomendable para personas que desean desarrollar software debido a que la documentación proporcionada por Altos tiene errores y omisiones.

CAPITULO 8

DESCRIPCION DE LA CONEXION FISICA Y LOS PROGRAMAS.

8.1 INTRODUCCION.

En este capitulo se examina la conexión física y los programas utilizados para emular terminales y transferir archivos. Se sigue el modelo de referencia de ISO planteado en el capitulo de antecedentes.

8.2 DESCRIPCION DE LA CONEXION FISICA.

La conexión física corresponde al nivel 1 del modelo de referencia de ISO. Para llevarla a cabo, se utilizó la interfaz RS-232-C conectando un puerto de la VAX a un puerto de la Altos en el CECAFI Anexo. La conexión se ilustra en la Fig. 1.

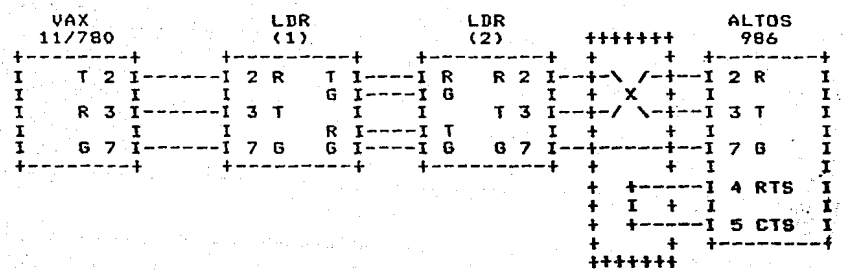


Figura 1

Conexión física entre la VAX y la Altos

En la figura anterior, la letra T significa transmisión, R recepción y la G significa tierra. Los números asociados a estas letras indican el número de pin utilizado en el conector.

Los dispositivos marcados con LDR son 'Line Drivers'. Un 'Line Driver' no es más que un amplificador que se utiliza en lugar de un modem para transmitir a distancias no mayores a 9.6 km. transmitiendo a 2400 bps. La ventaja de este dispositivo es que es mucho más barato que un modem y es muy adecuado para líneas privadas, como es el caso de las líneas que van desde la VAX hasta el CECAFI Anexo.

La ubicación de cada dispositivo se da a continuación:

- La VAX 11/780 está ubicada en el CECAFI Principal.
- El primer 'line driver' también está ubicado en CECAFI Principal.
- El segundo 'line driver' está en CECAFI Anexo.
- La Altos 986 está en CECAFI Anexo.

La conexión que va de la VAX hacia el primer 'line driver' se realiza utilizando solamente tres hilos: Transmisión, Recepción y Tierra. El motivo por el que se conectan los pines 2 con 2, 3 con 3 y 7 con 7 es que internamente el 'line driver' tiene invertidos los pines de transmisión y recepción. Esto es para simplificar la conexión al usuario.

La conexión que va del primer 'line driver' al segundo 'line driver' es utilizando cuatro hilos. En esta conexión se conectan las respectivas tierras y la transmisión con la recepción y la recepción con la transmisión. Es interesante notar que el 'line driver' maneja la Transmisión y Recepción con dos tierras independientes.

La conexión que va del segundo 'line driver' a la Altos es utilizando tres hilos. La parte que está encerrada en un cuadro es la conexión que el usuario debe realizar para establecer la comunicación. Es interesante notar que se hace un cruce del pin 3 del 'line driver' al pin 2 de la Altos, y del pin 2 al 3. Esto se debe a que los puertos serie de la Altos no respetan la convención RS-232-C sino que internamente intercambia los pines de transmisión y recepción. Por otra parte, el puente que se hace entre los pines 4 y 5 se debe a que el puerto de la Altos sí maneja estas señales. Cuando la Altos desea transmitir un dato, enciende la señal Request To Send que a su vez enciende la señal Clear To Send. Con este mecanismo la Altos cree que puede transmitir en cualquier momento.

8.3 DESCRIPCION DE LOS PROGRAMAS DE COMUNICACION Y TRANSFERENCIA DE ARCHIVOS.

Existen cuatro programas de comunicación, que son:

1. Programa que emula una terminal de VAX en Altos.

2. Programa que emula una terminal de Altos en VAX.

3. Programa que transfiere un archivo residente en Altos a la VAX.

4. Programa que transfiere un archivo residente en VAX a la Altos.

Los programas que corren en VAX fueron hechos en Fortran 77 haciendo uso de los System Services. Debe notarse que estos programas no funcionan con una clave normal sino que deben ejecutarse utilizando una clave privilegiada. Esto se debe a que los System Services que asignan los puertos de comunicación no pueden ser utilizados por cualquier usuario.

Los programas que corren en Altos fueron hechos en C. El código utiliza únicamente funciones estándar de C por lo que los programas deben correr en cualquier sistema UNIX. Lo que debe investigarse para utilizar estos programas en otra máquina es la manera de configurar el puerto que se desea utilizar para transmitir. Los programas deben de ser ejecutados desde la clave de superusuario ya que es la única clave que puede configurar los puertos de comunicación.

La lógica que utilizan los programas de emulación de terminal y de transferencia de archivos es la misma.

A continuación se da el pseudocódigo de los programas que emulan una terminal:

```
WHILE (El usuario desee seguir utilizando este programa)
    IF (Hay datos para leer en el puerto de comunicación)
        S Lee los datos
        S Reemplaza los caracteres indeseados por blancos
        S Escribe en la terminal los datos leídos
    ELSE
        IF (Hay datos para leer en la terminal del usuario)
            S Lee los datos
            S Envía los datos por el puerto de comunicación
        ENDIF
    ENDIF
ENDWHILE
```

Ahora se da el pseudocódigo de los programas que transfieren archivos:

```
S Lee el nombre del archivo a transferir
S Lee el nombre con el que se desea quede en la otra máquina
S Abre los archivos en la máquina fuente y destino
WHILE (Haya datos en el archivo)
    S Lee un registro
    WHILE (Dato en puerto diferente de LINEFEED)
        S No hagas nada
```

```
        ENDWHILE
        S Envia el registro por el puerto de comunicación
    ENDWHILE
```

8.3.1 DESCRIPCION DETALLADA DE LOS PROGRAMAS EN ALTOS.

En primer lugar, se analiza el programa que emula una terminal de VAX en Altos. Para ello, a continuación se muestra el código fuente:

```
#include "defines.c"

#include <stdio.h>

#include <signal.h>

#define  ESC '\033'

main()
    {
    char buffer_in [256];          /* Buffer asociado a la lectura      */
         buffer_out [80];          /* Buffer asociado a la escritura    */
    int  n_read          ,         /* Numero de caracteres leidos       */
         n_written       ,         /* Numero de caracteres escritos     */
         status          ,         /* Status de la ejecucion de system  */
         sigue_proc ()   ,         /* Direccion para alarm              */
         (*istat) ()     ,         /* Guarda la direccion de signal()   */
         fd              ;         /* File descriptor tty10             */


    /* Ahora abre tty10 */
    fd = open("/dev/tty10", 2);
    if (fd == -1)
        {
        printf("No pudo abrir tty10\n");
        exit(1);
        }
    endif

    /* Logica principal */
    limpia_buffer(buffer_in, 256);
    istat = signal(SIGALRM, sigue_proc);
    alarm(1);
    n_read = 0;
    while ((n_read = read(fd, buffer_in, 256)) != 0)
        {
        if ((n_read != -1) && (n_read != 0))
            {
            /* Si leyo datos del puerto, los escribe en pantalla*/
            istat = signal(SIGALRM, SIG_IGN);
```

```
            quita_esc(buffer_in, 256);
            printf("%s", buffer_in);
            limpia_buffer(buffer_in, 256);
            n_read = 0;
            }
        else
            {
            /* No hay datos en el puerto, lee de la terminal */
            n_written = 0;
            limpia_buffer(buffer_out, 80);
            istat = signal(SIGALRM, sigue_proc);
            alarm(1);
            n_written = getline(buffer_out, 80);
            if (n_written != 0)
                {
                istat = signal(SIGALRM, SIG_IGN);
                n_written = n_written + 1;
                n_written = write(fd, buffer_out, n_written);
                }
            endif
            }
        endif
        istat = signal(SIGALRM, sigue_proc);
        alarm(4);
        }
    endwhile
    }
endmain

int sigue_proc()
    {
    /* Funcion nula para manejo de interrupciones */
    }
endfunc

limpia_buffer(s, lim)
    char s[];
    int  lim;
    {
    /* Limpia el buffer que se le pasa como parametro */
    int i;
    for ( i = 0; i < lim; i++)
        {
        s[i] = '\0';
        }
    endfor
    }
endfunc

getline( s, lim)
    char s[];
    int  lim;
    {
    /* Lee una linea de la terminal del usuario */
    int c,
```

```
        i;

    i = 0;
    while (--lim > 0 && (c = getchar()) != EOF && c != '\n')
        s[i++] = c;
    endwhile
    s[i] = '\0';
    }
endfunc
```

Leyendo el código se observa que la mayor parte del tiempo el programa está leyendo del puerto. Esto se debe a que la Altos es incapaz de manejar entradas de datos sostenidas a velocidades mayores de 2400 bps. Las constantes lecturas sobre el puerto son para evitar pérdidas de información. Además, se observa que si después de 4 segundos no hay datos en el puerto, se procede a leer la terminal del usuario. Si después de un segundo no hay datos por parte del usuario, se procede a leer el puerto de entrada y así sucesivamente. Cuando sí hay datos en el puerto de entrada, éstos se leen y se procede a quitar los caracteres <ESC> (ESCAPE) debido a que representan secuencias de control en una terminal de VAX. Sin embargo, estas secuencias no son reconocidas por la terminal de Altos. Cuando sí hay datos en la terminal del usuario, éstos se leen y se transmiten por el puerto de comunicación.

El segundo programa en Altos es el de transferencia de archivos. Su código es el siguiente:

```
#include "defines.c"

#include <stdio.h>

#include <signal.h>

#define TAMANO 133

#define CTRLZ '\032'

main(argc, argv)
    int argc;
    char *argv[];
    {

    FILE *fopen(),
         *fp     ;              /* File pointer del archivo a transferir */

    int  fd,                    /* File descriptor tty10                 */
         status,                /* Status de la ejecucion de instrucc.   */
         num_transf,            /* Numero de registros transferidos      */
         quita_cr();            /* Funcion para quitar <CR>              */
```

```
        dame_tamano(),          /* Da la longitud en bytes de un string  */
        sigue_proc(),           /* Rutina para el alarm                  */
        (*istat) (),            /* Guarda la direccion de signal         */
        n_written,              /* Numero de bytes escritos              */
        n_read,                 /* Numero de bytes leidos                */
        i;                      /* Contador auxiliar                     */

    char buffer_out[TAMANO],    /* Buffer para transf. de archivo        */
         buffer_in[TAMANO],     /* Buffer para recibir los line-feed     */
         c              ;       /* Caracter auxiliar                     */

    if ((argc < 3) || (argc > 3))
        {
        printf("Uso: tranvax <archivo altos> <archivo vax>\n");
        exit(1);
        }
    endif

    fp = fopen(*++argv, "r");
    if (fp == NULL)
        {
        printf("tranvax: no pudo abrir %s\n",*argv);
        exit(1);
        }
    endif

    fd = open("/dev/tty10", 2);
    if (fd == -1)
        {
        printf("tranvax: puerto # 10 no disponible\n");
        exit (1);
        }
    endif

    strcpy(buffer_out, "CREATE ");
    *++argv;
    strcat(buffer_out, *argv);
    n_written = dame_tamano(buffer_out, TAMANO);
    n_written = write(fd, buffer_out, n_written);

    num_transf = 0;
    limpia_buffer(buffer_out, TAMANO);
    while ((fgets(buffer_out, TAMANO, fp)) != NULL)
        {
        n_written = quita_cr(buffer_out, TAMANO);
        n_written = write(fd, buffer_out, n_written);
        istat = signal(SIGALRM, sigue_proc);
        alarm(2);
        n_read = read(fd, buffer_in, TAMANO);
        printf("%s", buffer_in);
        printf("%s\n",buffer_out);
        limpia_buffer(buffer_in, TAMANO);
        limpia_buffer(buffer_out, TAMANO);
        num_transf++;
        }
```

```
    endwhile

    c = CTRLZ;
    n_written = write(fd, &c, 1);
    printf("Se escribieron %d registros\n", num_transf);
    }
endmain

int quita_cr(buffer, lim)
    char buffer[];
    int lim;
    {
    int i;
    for (i = 0; ((buffer[i] != '\n') && (i < lim)); i++)
        ;
    buffer[i] = '\0';
    return(i+1);
    }
endfunc

limpia_buffer(s, lim)
    char s[];
    int lim;
    {
    int i;
    for (i = 0; i < lim; i++)
        s[i] = '\0';
    endfor
    }
endfunc

int dame_tamano( s, lim)
    char s[];
    int lim;
    {
    int i;
    for (i = 0; (i < lim) && (s[i] != '\0'); i++)
        ;
    endfor
    return(i+1);
    }
endfunc

int sigue_proc()
    {
    }
endfunc
```

La primer acción del programa es abrir el archivo en Altos y enviar el mensaje "CREATE <NOMBRE DEL ARCHIVO EN VAX>". Con ello crea el archivo en VAX. A continuación se lee un registro del archivo en Altos, se transmite, se espera dos segundos, se lee otro registro y se transmite, etc. El motivo por el cual se espera dos segundos es el siguiente: cuando la VAX recibe un <CR> (CARRIAGE RETURN) contesta con un <LF> (LINE FEED). Lo óptimo sería enviar un registro, leer el <LF> de la VAX y continuar. El problema es que la Altos es incapaz de reconocer el <LF> como terminador. El único terminador que reconoce es el <CR>. Esperando 2 segundos se garantiza que la VAX está lista para recibir otro mensaje. Cuando se termina de leer el archivo en Altos se procede a enviar un CTRL-Z. Esto es para cerrar el archivo en VAX.

8.3.2 DESCRIPCION DETALLADA DE LOS PROGRAMAS EN VAX.

Primero se analiza el programa que emula una terminal. Su código es el siguiente:

```
      PROGRAM TERMINAL_ALTOS_986.
C
C     F A C U L T A D   D E   I N G E N I E R I A .
C     PROYECTO: COMUNICACION DE LAS COMPUTADORAS
C               VAX 11/780 Y ALTOS 986
C     REALIZO:  FEDERICO A. MORALES FAVILA
C
C     ESTE PROGRAMA EMULA UNA TERMINAL DE ALTOS UTILIZANDO
C     EL PUERTO TTA3 DE VAX.
C
      IMPLICIT        NONE

      CHARACTER       TERMINAL_ALTOS*5 /'TTA3:'/,
     1                TERMINAL_LEC*2 /'TT'/,
     1                DATO_ENTRADA*132,
     1                COMANDO*80

      EXTERNAL        IO$_READVBLK,
     1                IO$_WRITEPBLK,
     1                IO$M_TIMED,
     1                SS$_TIMEOUT

      INTEGER*2       CANAL_ALTOS,
     1                TEXT_IOSB(4),
     1                TERM_CHAN_LEC,
     1                POSICION

      INTEGER*4       SYS$ASSIGN,
     1                SYS$QIOW,
     1                SYS$QIO,
     1                SYS$DASSGN,
```

```
     1              STATUS,
     1              IO_FUNC,
     1              TERMINADOR_CR(2)

     LOGICAL        FIN_DEL_PROGRAMA

C    SE ASIGNA EL CANAL DE ALTOS
     STATUS = SYS$ASSIGN (TERMINAL_ALTOS,
     1                    CANAL_ALTOS,,)
     IF (.NOT. STATUS) CALL LIB$STOP (%VAL(STATUS))

C    SE ASIGNA EL CANAL DE LA TERMINAL DEL USUARIO VAX
     STATUS = SYS$ASSIGN (TERMINAL_LEC,
     1                    TERM_CHAN_LEC,,)
     IF (.NOT. STATUS) CALL LIB$STOP (%VAL(STATUS))

C    LA FUNCION DE LECTURA ES: LEE O ABANDONA LA LECTURA
C    DESPUES DE UN PERIODO ESPECIFICO DE TIEMPO
     IO_FUNC = %LOC(IO$_READVBLK) .OR.
     1         %LOC(IO$M_TIMED)

C    EL UNICO TERMINADOR VALIDO PARA TERMINAR UN MENSAJE
C    DEL USUARIO ES EL <CR>
     TERMINADOR_CR(1) = 0
     TERMINADOR_CR(2) = '00002000'X

C    LOGICA PRINCIPAL
     FIN_DEL_PROGRAMA = .FALSE.
     DO WHILE ( .NOT. FIN_DEL_PROGRAMA)
C        SE LEE SI HAY RESPUESTA DEL COMANDO
         STATUS = SYS$QIOW (, %VAL(CANAL_ALTOS),
     1                      %VAL(IO_FUNC),
     1                      TEXT_IOSB,,,
     1                      %REF(DATO_ENTRADA),
     1                      %VAL(132), %VAL(4),
     1                      ,,)
         IF (.NOT. STATUS) CALL LIB$STOP (%VAL(STATUS))

         IF (TEXT_IOSB(2) .NE. 0) THEN
             IF (TEXT_IOSB(1) .NE. %LOC(SS$_TIMEOUT)) THEN
                 POSICION = TEXT_IOSB(2) + 1
                 DATO_ENTRADA(POSICION:POSICION) = CHAR(13)
             ELSE
                 POSICION = TEXT_IOSB(2)
             ENDIF
             CALL QUITA_ESC(DATO_ENTRADA, POSICION)
             STATUS = SYS$QIOW (,
     1                          %VAL(TERM_CHAN_LEC),
     1                          IO$_WRITEPBLK,,,,
     1                          %REF(DATO_ENTRADA),
     1                          %VAL(POSICION),,,,)
             IF (.NOT. STATUS) CALL LIB$STOP (%VAL(STATUS))
             DATO_ENTRADA = ' '
         ELSE
C            SE LEE EL NOMBRE DEL COMANDO DE LA TERMINAL DE VAX
```

```
            COMANDO = ' '
            STATUS = SYS$QIOW (,%VAL(TERM_CHAN_LEC),
     1                         IO$_READVBLK,
     1                         TEXT_IOSB,,,
     1                         %REF(COMANDO),
     1                -        %VAL(80),,
     1                         %REF(TERMINADOR_CR),,)
            IF (.NOT.STATUS) CALL LIB$STOP(%VAL(STATUS))
            IF ((COMANDO(1:3) .NE. 'VAX') .AND.
     1      (COMANDO(1:3) .NE. 'vax')) THEN
C               SE MANDA EL COMANDO A LA ALTOS
                POSICION = TEXT_IOSB(2) + 1
                COMANDO(POSICION:POSICION) = CHAR(13)
                STATUS = SYS$QIOW (, %VAL(CANAL_ALTOS),
     1                             IO$_WRITEPBLK,,,,
     1                             %REF(COMANDO),
     1                             %VAL(POSICION),,,,)
                IF (.NOT. STATUS) CALL LIB$STOP (%VAL(STATUS))
            ELSE
                FIN_DEL_PROGRAMA = .TRUE.
                STATUS = SYS$DASSGN(%VAL(CANAL_ALTOS))
                IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
            ENDIF
        ENDIF
    ENDDO
    STOP ' Termina emulacion de Altos.'
    END

    SUBROUTINE QUITA_ESC(DATO, LONGITUD)
        CHARACTER*132 DATO
        INTEGER LONGITUD
C
C       ESTA SUBRUTINA QUITA LOS <ESC> QUE ENVIA LA
C       ALTOS.
C
        DO I = 1,LONGITUD
            IF (DATO(I:I) .EQ. CHAR(27)) THEN
                DATO(I:I) = ' '
            ENDIF
        ENDDO
        RETURN
    END
```

Como se observa, el programa es muy similar al utilizado en Altos. La diferencia está en que aquí se utilizan los System Services.

El programa de transferencia de archivos se muestra a continuación:

```
      PROGRAM TRANSFIERE_VAX_ALTOS
C
C     F A C U L T A D   D E   I N G E N I E R I A .
C     PROYECTO: COMUNICACION ENTRE LAS COMPUTADORAS
C               VAX 11/780 Y ALTOS 986.
C     REALIZO: FEDERICO A. MORALES FAVILA.
C
C     ESTE PROGRAMA TRANSFIERE UN ARCHIVO RESIDENTE EN
C     VAX A LA ALTOS USANDO EL PUERTO TTA3
C
C     PARA TRANSFERIR ARCHIVOS ES NECESARIO ESTAR EN
C     SESION EN ALTOS EN LA CLAVE DONDE SE DESEA
C     QUE QUEDE EL ARCHIVO.
C
      IMPLICIT          NONE

      CHARACTER         TERMINAL_ALTOS*5 /'TTA3:'/,
     1                  TERMINAL_LEC*2 /'TT'/,
     1                  NOMBRE_ARCHIVO*80,
     1                  PROMPT_PIDE*34
     1                  /'DAME EL NOMBRE DEL ARCHIVO EN VAX:'/,
     1                  DATO_ENTRADA*132,
     1                  DATO_ALTOS*133,
     1                  CTRL_D*1


      EXTERNAL          IO$_READVBLK,
     1                  IO$_READPROMPT,
     1                  IO$_WRITEPBLK,
     1                  IO$M_TIMED,
     1                  SS$_TIMEOUT

      INTEGER*2         CANAL_ALTOS,
     1                  TEXT_IOSB(4),
     1                  TERM_CHAN_LEC,
     1                  ICHAR,
     1                  NOMBRE_ARCHIVO_L,
     1                  POSICION

      INTEGER*4         TERMINADOR_LF(2),
     1                  SYS$ASSIGN,
     1                  SYS$QIOW,
     1                  SYS$QIO,
     1                  STATUS,
     1                  IO_FUNC,
     1                  DAME_LONGITUD,
     1                  NUM_TRANSF

      WRITE(6,13)
```

```
13      FORMAT(//,'  PROGRAMA PARA TRANSFERIR ARCHIVOS DE '
        1'VAX A ALTOS ',//)
C       SE ASIGNA EL CANAL A ALTOS
        STATUS = SYS$ASSIGN (TERMINAL_ALTOS,
        1                    CANAL_ALTOS,,)
        IF (.NOT. STATUS) CALL LIB$STOP (%VAL(STATUS))

C       SE ASIGNA EL CANAL DE LA TERMINAL DEL USUARIO VAX
        STATUS = SYS$ASSIGN (TERMINAL_LEC,
        1                    TERM_CHAN_LEC,,)
        IF (.NOT. STATUS) CALL LIB$STOP (%VAL(STATUS))

        TERMINADOR_LF(1) = 0 !INDICA QUE SE USE LA MASCARA
        TERMINADOR_LF(2) = '00000400'X !MASCARA DEL LINE-FEED

C       LA FUNCION DE LECTURA ES: LEE O ABANDONA LA LECTURA
C       DESPUES DE UN PERIODO ESPECIFICO DE TIEMPO
        IO_FUNC = %LOC(IO$_READVBLK) .OR. %LOC(IO$M_TIMED)

C       SE LEE EL NOMBRE DEL ARCHIVO EN VAX DE LA TERMINAL DE VAX
        STATUS = SYS$QIOW (, %VAL(TERM_CHAN_LEC),
        1                    IO$_READPROMPT,
        1                    TEXT_IOSB,,,
        1                    %REF(NOMBRE_ARCHIVO),
        1                    %VAL(80),,,
        1                    %REF(PROMPT_PIDE),
        1                    %VAL(34))
        IF (.NOT. STATUS) CALL LIB$STOP (%VAL(STATUS))
        NOMBRE_ARCHIVO_L = TEXT_IOSB(2)

C       SE ABRE EL ARCHIVO EN VAX QUE SE VA A
C       TRANSMITIR A ALTOS
        NOMBRE_ARCHIVO(NOMBRE_ARCHIVO_L+1
        1                    :NOMBRE_ARCHIVO_L+1) = ' '!QUITA CR
        OPEN(   UNIT = 2,
        1       FILE = NOMBRE_ARCHIVO,
        1       STATUS = 'OLD')

C       SE LEE EL NOMBRE DEL ARCHIVO PARA ALTOS DE LA
C       TERMINAL DE VAX
        PROMPT_PIDE = 'DAME EL NOMBRE DEL ARCHIVO ALTOS'
        STATUS = SYS$QIOW (, %VAL(TERM_CHAN_LEC),
        1                    IO$_READPROMPT,
        1                    TEXT_IOSB,,,
        1                    %REF(NOMBRE_ARCHIVO),
        1                    %VAL(80),,,
        1                    %REF(PROMPT_PIDE),
        1                    %VAL(33))
        IF (.NOT. STATUS) CALL LIB$STOP (%VAL(STATUS))

        NOMBRE_ARCHIVO_L = TEXT_IOSB(2)

C       SE DA UN CAT EN LA ALTOS
        DATO_ALTOS(1:6) = 'cat > '
        DATO_ALTOS(7:7+NOMBRE_ARCHIVO_L) = NOMBRE_ARCHIVO
```

```
      POSICION = 7+NOMBRE_ARCHIVO_L
      DATO_ALTOS(POSICION:POSICION) = CHAR(13) !RETURN
      STATUS = SYS$QIOW (, %VAL(CANAL_ALTOS),
     1                        IO$_WRITEPBLK,,,,
     1                        %REF(DATO_ALTOS),
     1                        %VAL(POSICION),,,,)
      IF (.NOT. STATUS) CALL LIB$STOP (%VAL(STATUS))


      DATO_ENTRADA = ' '
      NUM_TRANSF = 0

C     SE LEE Y TRANSMITE EL ARCHIVO COMPLETO
      DO WHILE (.TRUE.)
C           SE LEE EL ARCHIVO DE VAX
            READ(2,11,END = 99)DATO_ALTOS
11          FORMAT(A132)
            POSICION = DAME_LONGITUD(DATO_ALTOS, 133)
            WRITE(6,14)DATO_ALTOS
14          FORMAT('$',A<POSICION>)
            POSICION = POSICION + 1
            DATO_ALTOS(POSICION:POSICION) = CHAR(13)
C           SE MANDA LA LINEA A ALTOS
            STATUS = SYS$QIOW (, %VAL(CANAL_ALTOS),
     1                              IO$_WRITEPBLK,,,,
     1                              %REF(DATO_ALTOS),
     1                              %VAL(POSICION),,,,)
            IF (.NOT. STATUS) CALL LIB$STOP (%VAL(STATUS))

            STATUS = SYS$QIOW (, %VAL(CANAL_ALTOS),
     1                        %VAL(IO_FUNC),
     1                        TEXT_IOSB,,,
     1                        %REF(DATO_ENTRADA),
     1                        %VAL(132),
     1                        %VAL(1),
     1                        %REF(TERMINADOR_LF),,)
            IF (.NOT. STATUS) CALL LIB$STOP (%VAL(STATUS))
            NUM_TRANSF = NUM_TRANSF + 1
      ENDDO
99    CONTINUE

C     SE ENVIA UN CTRL-D PARA CERRAR EL ARCHIVO EN ALTOS
      CTRL_D = CHAR(4)
      STATUS = SYS$QIOW (, %VAL(CANAL_ALTOS),
     1                IO$_WRITEPBLK,,,,
     1                %REF(CTRL_D),
     1                %VAL(1),,,,)
      IF (.NOT. STATUS) CALL LIB$STOP (%VAL(STATUS))

C     ANULA EL ECO EN EL PUERTO DE ALTOS
      DATO_ALTOS(1:10) = 'stty -echo'
      DATO_ALTOS(11:11) = CHAR(13)
      STATUS = SYS$QIOW (, %VAL(CANAL_ALTOS),
     1                IO$_WRITEPBLK,,,,
     1                %REF(DATO_ALTOS),
```

```
      1                %VAL(11),,,,)


      WRITE(6,19)NUM_TRANSF
19    FORMAT('$',' SE TRANSFIRIERON ',I5,' REGISTROS.')
      END

      INTEGER FUNCTION DAME_LONGITUD(ARREGLO, TAMANO)
          CHARACTER ARREGLO*(*)
          INTEGER TAMANO
C
C         'ESTA FUNCION DEVUELVE EL TAMANO DEL STRING QUE SE
C         LE PASA COMO PARAMETRO.
C
          INTEGER I
          I = TAMANO
          DO WHILE ((ARREGLO(I:I) .EQ. ' ') .AND. (I .GT. 1))
              I = I - 1
          ENDDO
          DAME_LONGITUD = I
          RETURN
      END
```

Nuevamente se aprecia que el programa es muy similar al utilizado en Altos.

8.4 RELACION ENTRE LOS PROGRAMAS Y EL MODELO ISO.

Ya se examinó la parte correspondiente al Nivel 1 (Físico).

La relación entre el Nivel 2 y los programas es la siguiente: En primer lugar se sabe que la transmisión no tiene errores. Esto se debe a la cercanía de las instalaciones, a la velocidad relativamente baja de transmisión (2400 o 1200 bps) y a la ayuda proporcionada por los 'Line Drivers' por lo que los programas no tienen ninguna provisión especial para verificar los datos. En segundo lugar está el problema de pérdida de datos debido a la incapacidad del receptor para leerlos. Se garantiza que no habrá pérdidas de información utilizando retrasos de tiempo cada vez que el transmisor desea enviar un mensaje.

Los Niveles 3 y 4 (Red y Transporte) no tienen sentido en esta aplicación. Esto se debe a que se trata de una conexión punto a punto que no requiere ningún software especial para determinar como encaminar paquetes. Tampoco se lleva una contabilidad especial, ya que se trata de líneas privadas que son utilizadas a voluntad por el Centro de Cálculo. Por último, los mensajes no son divididos en unidades más pequeñas porque se trata de programas que emulan el funcionamiento de una terminal o transfieren archivos que obligan a que los mensajes sean transmitidos sin ninguna modificación.

Tampoco existe Nivel 6 (Presentación) en esta aplicación, ya que no se llevan a cabo transformaciones especiales sobre los datos.

Los niveles 5 y 7 (Sesión y Aplicación) son los programas en sí. Es decir, el usuario debe negociar con estos programas para comunicarse con la otra computadora. Además, los programas tienen la lógica que permite que se lleve a cabo la aplicación, es decir, manejan los mensajes de tal forma que la otra computadora cree que está manejando a una de sus terminales.

8.5 INSTRUCTIVOS DE OPERACION

A continuación se detallan los procedimientos necesarios para utilizar los programas.

8.5.1 INSTRUCTIVO PARA EMULAR UNA TERMINAL DE VAX DESDE ALTOS.

El procedimiento que debe seguirse es el siguiente:

1. Entrar a sesión en Altos con la clave normal del usuario.

2. Convertirse en el superusuario y teclear
   ```
   # disable tty10
   # setmode /dev/tty10 2400 -echo
   ```

3. Abandonar la clave de superusuario.

4. Entrar a sesión en VAX en la terminal TTA3 con la clave del usuario.

5. Teclear la instrucción:
   ```
   $ set term/speed=2400/noecho
   ```

6. Desconectar los puertos TTA3 de VAX y tty10 de Altos y realizar la conexión física.

7. Por último, en Altos ejecutar el programa:
   ```
   $ /usr/federico/tesis.dir/dcl
   ```

8. Para terminar de utilizar el programa simplemente oprímase la tecla <DEL> del teclado de la terminal en Altos.

9. Una vez terminada la sesión, convertirse en superusuario y teclear:
   ```
   # setmode /dev/tty10 9600 echo
   # enable tty10
   ```

10. Abandonar la clave de superusuario.

8.5.2 INSTRUCTIVO PARA TRANSFERIR ARCHIVOS DE ALTOS A VAX.

El procedimiento a seguir es:

1. Realizar el procedimiento para emular una terminal (Ver sección anterior).
2. Abandonar el programa que emula una terminal con la tecla <DEL>.
3. Ejecutar el programa. En el siguiente ejemplo "datos" es el nombre del archivo en Altos y "datos.dat" es el nombre del archivo en VAX.
   ```
   $ /usr/federico/tesis.dir/tranaltos datos datos.dat
   ```
4. Al terminar la transferencia, volver a ejecutar el programa que emula una terminal para continuar con la sesión o abandonarla.

8.5.3 INSTRUCTIVO PARA EMULAR UNA TERMINAL DE ALTOS EN VAX.

El procedimiento a seguir es:

1. Entrar a sesión en VAX con una clave privilegiada.
2. Dar la intrucción:
   ```
   $ set term/perm/speed=2400/noecho TTA3:
   ```
3. En el puerto tty10 de Altos entrar a sesión con la clave normal de usuario. Seguidamente teclear:
   ```
   $ stty 2400 -echo
   ```
4. Realizar la conexión física entre los puertos.
5. Ejecutar el programa con:
   ```
   $ run dba1:[federico.tesis.pruebas]shell
   ```
6. Para terminar de utilizar el programa de emulación teclear:
   ```
   $ vax
   $ set term/perm/speed=4800/echo tta3:
   ```

8.5.4 INSTRUCTIVO PARA TRANSFERIR ARCHIVOS DE VAX A ALTOS.

Para transferir archivos se sigue el siguiente procedimiento:

1. Entrar en sesión de acuerdo a lo descrito en la sección anterior.

2. Abandonar el programa que emula terminal.

3. Ejecutar la transferencia de archivos con:
   ```
   $ run dba1:[federico.tesis.pruebas]tranaltos
   ```
   y contestar a las preguntas que hace el programa.

4. Al terminar la transferencia ejecutar nuevamente el programa de emulación de terminal y continuar o terminar la sesión.

CAPITULO 9

CONCLUSIONES.

Antes de enunciar las conclusiones es necesario hacer las siguientes observaciones:

1. La Altos 986 no es una computadora adecuada para desarrollar software ni para utilizarla en aplicaciones que involucren comunicaciones como fué el caso. Esta afirmación se apoya en el hecho de que la documentación es pobre, el sistema operativo no es 100% confiable y a que sus puertos son incapaces de manejar una entrada sostenida de datos a velocidades mayores a 2400 bps.

2. La Altos 986 no es una máquina adecuada para manejar 9 usuarios a la vez. Esto se debe a que tiene un procesador Intel 8086 el cual simplemente no tiene el suficiente poderío para atender a 9 usuarios al mismo tiempo. En opinión del autor, la configuración máxima debería ser la Altos 586 la cual atiende a un máximo de 5 usuarios.

A pesar de los inconvenientes arriba mencionados es posible enunciar las siguientes conclusiones:

1. Los programas desarrollados cumplieron con su objetivo de proporcionar comunicación e intercambio de información entre ambas máquinas.

2. Es posible utilizar los programas de manera local o remota a través de modem ya que la interfaz RS-232-C así lo permite.

3. En virtud de que la comunicación es a través de un puerto asíncrono es posible utilizar los programas para emular terminales para establecer comunicación con prácticamente cualquier máquina.

4. Los programas de transferencia de archivos fácilmente pueden ser modificados para ser utilizados con otras máquinas.

5. Los programas desarrollados en Altos fueron hechos exclusivamente utilizando funciones estándar de 'C' por lo que su transportación a otros sistemas UNIX no debe representar un gran problema. Lo único que necesita conocerse en otro sistema es la manera de configurar los puertos serie de manera apropiada.

6. Los programas sirven para transferir archivos de datos y programas. No se recomienda su utilización para transferir archivos con código ejecutable ya que se bloquearía la terminal.

BIBLIOGRAFIA

1.- Alabau Muñoz Antonio y otros. Teleinformática y Redes de Computadoras. Publicaciones Marcombo, México, 1982, 181 pp. Serie: Mundo Electrónico.

2.- IEEE. Revista Computer, Network Protocols. September 1979.

3.- Fundación Arturo Rosenblueth. Aspectos Fundamentales en el Procesamiento Remoto de Datos. Sedeco, México, 1982.

4.- IEEE. Revista Computer, Communications Networks, Proceedings, November 1972.

5.- Denning J. P. Operating Systems Principles for Data Flow Networks, Computer, Vol. 11, No. 7. July 1978.

6.- Andrew S. Tanenbaum. Computer Networks. Prentice Hall Inc., 1981.

7.- Harvey M. Deitel. An Introduction to Operating Systems. Addison Wesley Publishing Co., 1984.

8.- Henry Mc Gilton y Rachel Morgan. Introducing the UNIX System. Mc Graw Hill, 1984.

9.- Brian W. Kernigham y Dennis M. Ritchie. The C Programming Language. Prentice Hall Inc., 1978.

10.- Brian W. Kernigham y otros. UNIX Programmers Manual, Volume 2: Supplementary Documents. Bell Laboratories, 1979.

11.- VAX/VMS System Services Reference Manual. Digital Equipment Corporation, 1980.

12.- Fortran User's Guide. Digital Equipment Corporation, 1980.

13.- DZ11 User's Guide. Digital Equipment Corporation, 1979.