UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DISEÑO Y DESARROLLO DE UN SISTEMA PARA EL CONTROL DISTRIBUIDO DE PROCESOS USANDO UN SISTEMA OPERATIVO EN TIEMPO REAL

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A :

ALEJANDRA LOPEZ RODRIGUEZ

DIRECTOR DE TESIS:
DR. ROGELIO ALCANTARA SILVA

MEXICO, D. F. 1995

FALLA DE ORIGEN

A todos los que contribuyeron
a hacer este sueño realidad.

ÍNDICE

INTRODUCCIÓN

El término *"tiempo real"* puede utilizarse para describir cualquier actividad de proceso de información que deba responder a estímulos de entrada generados externamente dentro de un intervalo de tiempo especificable, finito y breve. Sin embargo, este concepto de tiempo real ha creado controversias en lo que se refiere al intervalo de tiempo en que el sistema deba responder, ya que este variará de aplicación a aplicación, y será aceptable cuando pueda responder a cargas críticas.

Sin perder de vista la definición anterior, un sistema en tiempo real es un método computarizado que simultáneamente recibe información de varias partes geográficamente dispersas, procesa y regresa información en un período aceptable de tiempo.

De acuerdo con la definición de el Data Communications Dictionary de Sippl [SIP76] un programa en tiempo real se define como:

> *Aquel que opera concurrentemente con procesos externos los cuales son monitoreados o controlados, mientras que cubre los requerimientos del proceso con respecto al tiempo.*

Todavía en la década pasada el principal usuario de los Sistemas en Tiempo Real era la industria militar, sin embargo actualmente estos sistemas tienen gran popularidad, debido principalmente a la reducción en el costo del hardware y ahora tenemos diversas aplicaciones que incluyen:

- Control de procesos
- Investigación médica y científica
- Gráficos por computadora
- Comunicaciones locales y de largo alcance en sistemas aerospaciales
- Automatización industrial
- Pruebas asistidas por computadora y en instrumentación industrial.

Como cualquier otro sistema basado en computadora, un sistema en tiempo real integra hardware y software. Los sistemas en tiempo real generan alguna acción como respuesta a sucesos externos. Para ello, se realiza una adquisición de datos a alta velocidad bajo ciertas restricciones de tiempo y fiabilidad. Debido a que las restricciones son muy rigurosas los sistemas en tiempo real están frecuentemente dedicados a una sola aplicación, por lo que estamos hablando de sistemas dedicados.

El software que se desarrolla debe responder al mundo real, en un tiempo dictado por el dominio del problema, es por ello, que debe operar bajo requerimientos de rendimiento muy rigurosos, y su diseño esta determinado por la arquitectura del hardware, así como por las características del sistema operativo.

En un sistema en tiempo real es común que se module la aplicación en procesos que se ejecutarán de forma concurrente, para atender a las necesidades del mundo real. Estos procesos reclaman el uso de recursos de la computadora , así como el uso de canales de comunicación con el mundo exterior, al igual que deben comunicarse entre ellos para sincronizar actividades, como tener asignadas prioridades para su ejecución, por lo que son importantes las características y ventajas que nos ofrezca el sistema operativo que estemos utilizando para ejecutar el sistema.

Ante las necesidades de los sistemas en tiempo real, se han desarrollado los sistemas operativos de tiempo real, los cuales nos proveen de una serie de ventajas para el desarrollo y la ejecución de este tipo de sistemas, brindando todos sus servicios sin crear capas de software adicional.

Actualmente, aunque las aplicaciones de tiempo real son cada vez más populares, se hace necesario el que las personas que desarrollan aplicaciones de este tipo compartan sus experiencias en ello, ya que dentro de las carencias de esta área, encontramos la falta de una metodología de análisis y diseño que se ajuste a las necesidades actuales de desarrollo, involucrando las facilidades de que brindan los sistemas operativos de tiempo real.

El proyecto del que forma parte nuestro sistema, nace de la necesidad del hombre de interactuar con su entorno, a través de las herramientas que nos

proporciona la tecnología actual. El sueño del hombre de controlar y modificar su entorno, se hace posible a través de la electrónica, las comunicaciones y la computación, cualquier proceso o evento físico que pueda ser instrumentado electrónicamente es 100% monitoreable, y si podemos tener los actuadores necesarios para modificar las condiciones monitoreadas estaremos controlando el mundo real. Ante esta inquietud, el proyecto pretende tener una arquitectura abierta y flexible, en la que exista un control distribuido por medio de microcontroladores encargados del control de procesos o eventos físicos. Dichos microcontroladores trabajarán con sensores y actuadores, ejerciendo un control local sobre las variables sensadas; y a través de las comunicaciones los microcontroladores interactuarán con un sistema implementado en una computadora que monitoreará y controlará de forma central los diferentes procesos físicos, activando o desactivando los algoritmos de control locales de los diferentes microcontroladores, o bien, indicando el cambio de estado de los actuadores. Con la modularidad que se tiene del sistema, existe un gran número de procesos físicos que se pueden controlar, no importando que tan complejo o no sea su control, ya que para cada caso se tendría un microcontrolador con un algoritmo de control propio para ello, sin que la naturaleza del proceso físico afecte al resto del sistema. De esta forma, el proyecto puede crecer y mejorarse atendiendo a las necesidades que se presenten en un momento determinado.

En el presente trabajo, se expone el diseño y desarrollo del sistema residente en la computadora para el control distribuido que se mencionó en el párrafo anterior, el cual, por su funcionalidad entra dentro de la definición de un sistema en tiempo real. Dicho diseño además de cumplir con los requerimientos dictados por el proyecto del que forma parte, debe proveer de un grado de modularidad tal, que conforme el proyecto siga desarrollándose y creciendo, sea fácilmente mantenible, logrando que cada módulo sea transparente en su forma de operación al resto del sistema computacional.

Nuestro trabajo aquí presentado, hace una recopilación de las bases necesarias sobre la teoría de los sistemas operativos, con el objetivo de entender la importancia y el acoplamiento entre éstos, y las aplicaciones de tiempo real, capítulo 1. En el capítulo 2, se hace énfasis en las características principales que hacen útiles a los sistemas operativos denominados de tiempo real, proporcionando como ejemplo el sistema operativo que se utilizó como plataforma para el desarrollo del presente sistema. Para el diseño e implementación del sistema se hizo un estudio de las principales metodologías

empleadas para el desarrollo de este tipo de sistemas, y el resultado de esta investigación se expone en el capítulo 3. En el capítulo 4 presentamos de forma general la arquitectura de todo el proyecto dentro del que interactuará nuestro sistema, incluyendo la descripción de las arquitecturas de los microcontroladores que se encargan del control distribuido de los procesos. Finalmente en el capítulo 5, se detalla el análisis y desarrollo de la aplicación de software que presentamos en este trabajo.

CAPITULO 1

GENERALIDADES DE LOS SISTEMAS OPERATIVOS

Debido a los requerimientos de los sistemas que operan en tiempo real y de su interacción con el mundo exterior, se hace necesario una buena coordinación entre la máquina y su entorno. Cuando se diseña un sistema en tiempo real se identifican varias tareas o procesos que deben ejecutarse simultáneamente ya sea por un procesador único o por procesadores distribuidos, siendo necesario un buen entendimiento de lo que son los procesos, y de ahí la necesidad de una programación concurrente. Es por esto, que el sistema operativo seleccionado es una pieza clave dentro del desarrollo de estos sistemas; de una buena selección de éste, y las facilidades que nos brinde, nos permitirá tener un diseño más simple y nos garantizará un menor número de errores en su implementación.

Actualmente el sistema operativo es una capa de software, que nos permite interactuar con el hardware de la máquina de una forma más lógica, olvidándonos de la arquitectura misma de la computadora. Sin embargo, para poder entender cuál es la función del sistema operativo y dar una definición formal de éste es necesario dar un breve esbozo de la historia de los sistemas operativos.

1.1 HISTORIA DE LOS SISTEMAS OPERATIVOS

Así como el hardware de las computadoras ha evolucionado, han existido grandes cambios en el software, es por ello que en lo que se refiere a los sistemas operativos podemos identificar las siguientes genereciones [DEI84].

Generación cero

En la década de los 40's los sistemas de cómputo no tenían sistemas operativos, los usuarios tenían completo acceso al lenguaje de la máquina, por lo

que toda instrucción era codificada "manualmente". Es por ello que se perdía un tiempo considerable entre la terminación de un trabajo y el inicio del siguiente.

Primera generación

En los años 50's nacen los sistemas operativos, y el objetivo de los mismos era hacer más fluida la transición de un trabajo a otro. Este es el comienzo de los sistemas de procesamiento en lotes y de lo que fuera la primera generación de los sistemas operativos; en estos sistemas, los programas tenían completo dominio de la máquina mientras eran ejecutados, y cuando terminaban le devolvían el dominio al sistema operativo que se encargaba de limpiar la memoria, leer, cargar e iniciar el siguiente trabajo. Es así, como desde su nacimiento los sistemas operativos comienzan a administrar los recursos de la máquina.

Segunda generación

Durante la primera mitad de la década de los 60's se desarrollaron los primeros sistemas operativos multitarea, en los cuales varios programas comparten la memoria principal y el procesador, también nacen los sistemas con varios procesadores. Se establece la independencia de dispositivos, desarrollándose los sistemas de tiempo compartido.

Con la aparición de estos sistemas surgen los sistemas de tiempo real, desarrollándose las primeras aplicaciones de estos, principalmente para el control de procesos industriales y para usos militares.

Tercera generación

En esta etapa que abarcó de la segunda mitad de la década de los 60's hasta mediados de los 70's se diseñan sistemas para usos generales, con lo cual los sistemas operativos se les denominó de modos múltiples, ya que algunos de ellos soportaban el procesamiento en lotes, tiempo compartido, procesamientos de

tiempo real y multitarea. Al introducirse una mayor complejidad en los sistemas computacionales se creó una capa de software entre el usuario y el hardware.

Cuarta generación

Esta abarca de finales de la década de los 70's hasta nuestros días, siendo esta generación caracterizada por las redes de computadoras y el procesamiento en línea, con lo que los sistemas operativos contemplan estos requerimientos en su operación.

1.2 CONCEPTO DE SISTEMA OPERATIVO

Actualmente los sistemas operativos son demasiado complejos, en un principio, en la década de los 60's pudo haberse definido al sistema operativo como el software que controlaba el hardware. Si bien esta definición nos sirve como punto de partida para entender lo que es un sistema operativo deben analizarse también sus funciones.

Deitel [DEI84] nos define al sistema operativo como:

"...La serie de programas, ya sea dispuesto en el software o en la memoria fija, que hacen al hardware utilizable. El hardware provee de poder computacional básico. Los sistemas operativos ponen este poder convenientemente a disposición del usuario... administrando al hardware para que logre una buena ejecución."

Dentro de las funciones que el sistema operativo debe ofrecer tenemos entre otras:

- La ejecución de programas

- Las operaciones de entrada/salida
- El sistemas de manipulación de archivos
- La detección de errores

Al ser una interface entre el usuario y la máquina el sistema operativo debe ofrecer un intérprete de comandos, este puede ser una intérprete en línea de comandos o un shell como en el caso de Unix.

Visto como administrador de recursos el sistema operativo se encarga de:

- Administrar la memoria principal y secundaria
- Administrar los procesos
- Administrar el sistema de E/S
- Administrar el sistema de archivos
- Administrar el procesador

Dentro de nuestro estudio nos enfocaremos particularmente a aquellos sistemas operativos que permiten la multitarea, estudiando solamente en el presente trabajo al sistema operativo como administrador de procesos y la planeación del procesador (scheduling).

1.3 ADMINISTRACIÓN DE PROCESOS

Los procesos son las entidades "vivas" en un sistema multitarea. Su concepto nace en la década de los 60's, y es usado originalmente por los diseñadores del sistema Multics [ORG72], y como todo término computacional su concepto ha ido evolucionando. Aunque las definiciones son múltiples podemos decir que un proceso es un programa en ejecución, sin embargo, un programa por si sólo no es un proceso, los programas son entidades pasivas, mientras que los procesos son entidades activas.

Los procesos son identificados por el sistema operativo por un bloque de control del proceso. Como entidades activas los procesos demandan recursos como tiempo de procesador, memoria, archivos, dispositivos de E/S, etc.

1.3.1 BLOQUE DE CONTROL DEL PROCESO

El bloque de control del proceso es la estructura por medio de la cual el sistema operativo identifica al proceso, conteniendo la información referente a sus principales características que lo identifican y hacen único. Este bloque contiene básicamente la siguiente información:

- Identificador del proceso
- Estado del proceso
- Prioridad del proceso
- Apuntadores a la memoria asignada al proceso
- Estado de las operaciones de E/S
- Copia de los registros del cpu

1.3.2 CICLO DE VIDA DE UN PROCESO

Un proceso se crea, se ejecuta y se destruye, y a lo largo de su vida el proceso pasa por varios estados, la transición de estados no sólo depende de él, sino de una planificación establecida e interrupciones del sistema.

Creación de un proceso

Un proceso es creado al ejecutarse un programa, el proceso así creado puede crear nuevos procesos haciendo una llamada al sistema durante su ejecución. El proceso creador es conocido como proceso padre y los procesos creados se denominarán procesos hijos. Estos nuevos

procesos pueden a su vez crear nuevos procesos hijos, creando con ello una estructura jerárquica, en donde cada proceso hijo tiene uno y sólo un padre, mientras que un proceso padre puede tener uno o más procesos hijos.

Ejecución de los procesos

Cuando un proceso hijo es creado puede ejecutarse secuencial o concurrentemente con su padre. En el primer caso el padre espera a que su hijo termine su ejecución, mientras que en el segundo caso el padre continua ejecutándose junto con su hijo.

Una característica común de los procesos es la necesidad de recursos para completar su ejecución, en este caso el proceso hijo puede obtener los recursos directamente del sistema operativo o bien, el proceso padre puede compartir todos los recursos con su hijo o una parte de ellos.

Terminación de los procesos

Un proceso termina al ejecutarse la instrucción final de su código, cuando esto sucede, el proceso pudiera regresar algunos datos a su proceso padre. En ocasiones el proceso pudiera terminar por una llamada del sistema (por ejemplo abort), o bien un proceso padre le indica a su proceso hijo que debe terminar, ya sea por que este haya excedido el uso de alguno de los recursos, o bien, la tarea que originalmente le había sido encomendada ya no es necesaria.

En la mayoría de los sistemas no se permite que un proceso hijo exista si su padre ha terminado, con lo que al morir un proceso padre mueren con el todos sus hijos, con lo que se da un fenómeno conocido como terminación en cascada, que es iniciado por el sistema operativo.

Como mencionamos anteriormente, durante su existencia los procesos pasan por una serie de estados y en general podemos hablar de tres estados: el estado de listo, el estado de ejecución y el estado de espera o bloqueado.(fig. 1.1)

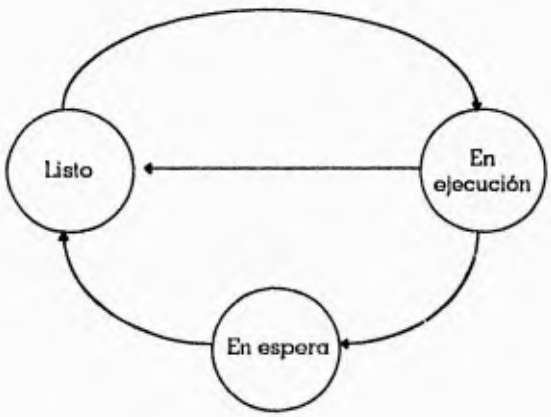


Figura 1.1 Diagrama de estados

El estado de listo. Se dice que un proceso esta listo, cuando éste espera a que se le asigne el procesador.

El estado de ejecución. Cuando un proceso tiene a su disposición el procesador se dice que se está ejecutando. De acuerdo al esquema de planificación, el proceso que se esta ejecutando puede dejar voluntariamente este estado ante la necesidad de una E/S o algún evento, pasando al estado de espera; o bien, si se establece un reloj de interrupción y el proceso excede el tiempo, el proceso será retirado del procesador y pasará al estado de listo.

El estado de espera. Se dice que un proceso se encuentra bloqueado si se encuentra en espera de algún evento, o de alguna E/S. Una vez que esta se concluya, el proceso se desbloqueará pasando al estado de listo.

1.3.3 LA FUNCIÓN DEL NÚCLEO DEL SISTEMA OPERATIVO EN LA ADMINISTRACIÓN DE PROCESOS

El núcleo o kernel del sistema operativo, es la parte del mismo que se mantiene en la memoria principal, así que todas las operaciones que implican los procesos, son controlados por esta parte del sistema. El núcleo también se encarga de procesar las interrupciones, y en general, la rápida respuesta a éstas mantendrá tiempos de respuesta aceptables a los usuarios interactivos. Dentro de las funciones principales del núcleo referentes a los procesos encontramos las siguientes:

- Creación y destrucción de procesos
- Cambio de estados de proceso
- Suspensión y reanudación de procesos
- Sincronización de procesos
- Comunicación entre procesos

1.3.4 PROCESOS CONCURRENTES

Los procesos como tales no existen aisladamente en un sistema, de tal forma que se ejecutan concurrentemente con otros procesos, y dependiendo de su interacción podemos clasificar a los procesos como [BUR90]:

- Independientes
- Cooperativos
- Competidores

Los *procesos independientes* son aquéllos que no se comunican o sincronizan con otros. Si algún error llegara a ocurrir en ellos, los procedimientos de recuperación de errores pueden ser iniciado por el proceso aislado del resto del sistema sin que afecte a los demás.

Los *procesos cooperativos* son aquellos que se ejecutan asincrónicamente, pero que en momentos deben comunicarse y sincronizar sus actividades entre ellos para poder realizar algunas operaciones en común. En consecuencia al ocurrir un error se involucran en la recuperación del mismo todos los procesos cooperativos.

Los *procesos competidores* necesitan sincronizarse y comunicarse para la obtención de recursos, en esencia son independientes, normalmente un error que ocurra en uno de ellos no afectará a los demás, sin embargo cuando el error ocurre en la obtención de un recurso los demás procesos quedarán involucrados.

1.3.5 SINCRONIZACIÓN DE PROCESOS

La sincronización de los procesos se hace necesaria cuando estos operan en común para lograr una tarea específica, involucrando para ello datos compartidos; para mantener una consistencia de los mismos es necesario que en ocasiones sólo un proceso a la vez accese y/o modifique estos datos.

Las secciones críticas de los procesos se referirán a aquella parte del código que involucre el manejo de los datos u otra clase de recursos en los que sea evidente un uso exclusivo de los mismos.

La sincronización de los procesos cooperativos se hace necesaria cuando se presenta el problema de productor/consumidor. Un proceso productor es aquel que produce cierta información que el proceso consumidor necesitará, para ello es necesario que el proceso consumidor sólo obtenga información cuando ésta se encuentre lista, de manera tal, que el consumidor tendrá que esperar a que la información se produzca.

Para sincronizar procesos y establecer secciones críticas se han implementado varias soluciones a nivel programación como el algoritmo de Dekker, o bien, soluciones a nivel instrucción hardware como la instrucción *testandset*, cuyo objetivo es probar el valor de una variable booleana, y dejarla en un valor en una sola instrucción, de tal forma, que una vez iniciada la instrucción no pueda ser interrumpida [DEI84]. Existen muchos intentos por resolver la

sincronización entre procesos implementados a nivel programación y serán responsabilidad del programador, sin embargo, algunas otras soluciones como los semáforos y la comunicación entre procesos son proporcionadas por el sistema operativo, y serán las que presentaremos aquí.

1.3.5.1 SEMÁFOROS

Los semáforos se usan para coordinar el acceso a regiones de memoria compartida. El sistema de semáforos provee de un método de sincronización poderoso de control de acceso a procesos cooperativos o competidores. Estos son usados para controlar el uso de recursos compartidos y sincronizar procesos con eventos externos.

Un semáforo S es una variable entera, que aparte de ser inicializada, puede ser accesada solamente por dos operaciones atómicas: *wait* y *signal*, estas operaciones se describen por varios autores como P y V, sus nombres provienen del idioma holandés *proberen* para probar y *verhogen* para incrementar. Las definiciones de estas operaciones son las siguientes:

wait (S) : Si S > 0
entonces S = S - 1
de otro modo (espera en S)

signal (S) : Si (uno o más procesos están en espera en S)
entonces (deja proseguir a uno de estos procesos)
de otro modo S = S + 1

Con este mecanismo cuando el valor del semáforo es cero, el recurso no esta disponible. En este caso el proceso puede esperar a que el valor del semáforo sea incrementado, o regresar sin el acceso al recurso.

Para sincronizar procesos con semáforos se hace lo siguiente: cuando un proceso va a entrar a una sección crítica ejecuta la operación *wait*, y cuando sale de ésta ejecuta la operación *signal*. Si fuera el caso de un problema de

productor/consumidor, el proceso productor ejecutaría la operación *signal* cuando los datos estuvieran listos, y el proceso consumidor ejecutaría la operación *wait.*

En muchos sistemas operativos las operaciones relacionadas con los semáforos están implementadas como llamadas al sistema, a manera de ejemplo presentaremos el manejo de los semáforos en UNIX.

LOS SEMÁFOROS EN UNIX

Un semáforo en UNIX es una estructura *(sem)* que contiene los siguientes cuatro campos:

- ***semval*** Es el valor del semáforo, representando el total de recursos de los cuales el semáforo es responsable.
- ***sempid*** Es el identificador del proceso (PID) del último proceso que tuvo acceso al semáforo
- ***smncnta*** Es el número de procesos que están esperando a que el valor del semáforo se vuelva mas grande que el valor actual.
- ***semzcnt*** Es el número de procesos esperando que el valor del semáforo sea igual a cero.

Los semáforos son usados por medio de un arreglo de ellos identificado como *semid;* cada semáforo es un arreglo identificado por un número, empezando con 0, y tiene una estructura *sem* asociada a él. Cada proceso crea una estructura local *(sops)* que es asociada al número de semáforo, y usa la estructura junto con la llamada *semop* para incrementar y decrementar el semáforo.

Las rutinas asociadas a los semáforos para operarlos son las siguientes:

Llamada al sistema	Descripción
semget(key,nsems,semflg)	crea un semáforo
semctl(semid,semnum,cmd,arg)	controla el semáforo
semop(semid,sops,nsops)	modifica un semáforo

El semáforo se manipula de dos formas con la llamada semop, incrementando o decrementando el valor del semáforo, siendo el sistema operativo el encargado de que sólo un proceso pueda manipular un semáforo en un momento dado.

A los procesos se les permite preguntar por el valor de un semáforo, y dependiendo del valor de éste, el proceso accesa o no al recurso, teniendo la posibilidad de suspender su ejecución en espera de que el valor del semáforo permita la operación. Esta habilidad del proceso de suspender su ejecución es conocida como "bloqueo de la operación del semáforo".

CREANDO UN SEMÁFORO

Antes de que un semáforo pueda ser operado o controlado se necesita una estructura única de identificación al arreglo de semáforos *(semid)*. Cuando se crea un conjunto de semáforos se inicializa una estructura *(sem)* por semáforo. El número de semáforos *(nsems)* es seleccionable por el usuario, asociándose una estructura de datos como única identificadora del semáforo, incluyendo la siguiente información:

- Permisos de operación
- Apuntador al primer semáforo del conjunto (arreglo)
- Número de semáforos en el conjunto
- Hora de la última operación del semáforo
- Hora del último cambio del semáforo.

1.3.5.2 LOS MONITORES

El método de semáforos es bueno, aunque para problemas de concurrencia más complejos, su uso suele dificultar la corrección y pruebas de un programa. Otra solución es el uso de monitores.

Un monitor es un módulo en el cual los datos que se comparten se encapsulan y agrupan, junto con una serie de procedimientos que accesarán a esos datos y otros recursos compartidos, el monitor también incluirá un buffer para el paso de datos. Para lograr la exclusión mutua de los recursos, el monitor sólo atenderá a un proceso a la vez, el proceso llamará al monitor con una petición de entrada, si este estuviera ocupado el proceso deberá esperar, y el monitor será el encargado de administrar esa espera. De esta forma los monitores facilitan una construcción de alto nivel para implementar el acceso mutuamente exclusivo a un recurso compartido. Sin embargo, no ofrecen ningún medio de sincronización de procesos y, por lo tanto, la construcción del monitor deberá complementarse permitiendo utilizar señales dentro de ellos.

Para procesos cooperativos, en el que exista un proceso consumidor y otro productor, el monitor funcionaría así:

Sin importar si el buffer esta lleno o vacío, el proceso productor transfiere datos al buffer llamando a algún procedimiento del monitor para recibir. Si el proceso consumidor intentara entrar al monitor en ese momento, se le haría esperar, cuando finalmente el proceso consumidor puede entrar al monitor, si existe información lee el buffer y lo vacía.

1.3.5.3 COMUNICACIÓN ENTRE PROCESOS

La comunicación entre procesos IPC (Interprocess Communication) es una forma general que brindan algunos sistemas operativos para sincronizar actividades entre procesos. La idea general de está comunicación es establecer un sistema de transmisión de mensajes entre los procesos, funcionando básicamente con las operaciones: *send (mensaje)* y *receive (mensaje)*. En el presente capítulo mencionaremos de una forma breve y general como funciona la

comunicación entre procesos, retomando el tópico más a detalle en el siguiente capítulo, por ser una característica importante de los sistemas operativos de tiempo real.

Existen varias formas lógicas de implementar la comunicación con las operaciones send/receive, dando como resultado:

- Comunicaciones directas o indirectas
- Comunicaciones asimétricas o simétricas
- Comunicación por copia o referencia del mensaje
- Comunicación de mensajes de tamaño fijo o variable

La *comunicación directa* involucra el identificador del proceso al que se quiere mandar o del que se desea recibir un mensaje, en contraste, la comunicación indirecta hace uso de buzones en los cuales serán depositados los mensajes para ser extraídos por el o los procesos que recibirán el mensaje.

La *comunicación asimétrica* se define como aquella en la que un proceso puede recibir mensajes de cualquier proceso, sin embargo cuando desee mandar un mensaje debe especificar el destinatario. Para la comunicación simétrica tenemos que en ambos casos de mandar o recibir mensajes debemos conocer el proceso con el cuál estableceremos la comunicación.

Otra característica es pasar los mensajes *por copia o referencia*, dependiendo del sistema es común que los mensajes sean pasados haciendo referencia de un área de memoria con lo que se ahorra tiempo de transferencia, sin embargo, cuando hablamos de un procesamiento de cómputo distribuido y los mensajes tienen que viajar a través de la red, por lo que es usual pasarlos como copia del mensaje original.

Los mensajes brindan al programador una infinita posibilidad de formas para sincronizar actividades entre sus procesos, esto se debe a la flexibilidad que nos ofrecen los mensajes; un mensaje puede incluir una secuencia de control y/o una serie de datos, que cada proceso interpretará a su manera. Dependiendo del sistema operativo los mensajes pueden ser de un tamaño establecido o bien ser dinámicos en lo que se refiere a su tamaño.

1.4 LA PLANIFICACIÓN DEL PROCESADOR

En los ambientes multitarea en los cuales el objetivo principal es ejecutar simultáneamente varios procesos, optimizando para ello el uso de los recursos de la máquina, es esencial la planificación de cada uno de los recursos a compartir, y dentro de estos recursos el principal a compartir es el procesador, es por ello que se debe establecer un criterio para su uso, obedeciendo a las necesidades de los procesos.

La asignación del procesador es un problema bastante complejo que se maneja en la teoría de los sistemas operativos. El procesador es el encargado de ejecutar tanto los procesos de los usuarios del sistema como los procesos mismos del sistema operativo, cada acción dentro de un sistema de cómputo es iniciado por el procesador, este debe responder a errores, peticiones de los procesos (llamadas al sistema), interrupciones de entrada/salida, etc. Una meta de los ambientes multitarea es mantener siempre un proceso ejecutándose en el procesador, mientras los demás procesos listos esperan en una cola. La planificación se da cuando el sistema operativo debe seleccionar entre los procesos que se encuentran en la cola el siguiente a ser ejecutado.

La planificación del procesador se divide en niveles, siendo de manera general 3:

- Planificación de alto nivel
- Planificación de nivel medio
- Planificación de bajo nivel

La *planificación de alto nivel* selecciona que procesos serán cargados en la memoria y competirán de forma activa por los recursos del sistema.

La *planificación de nivel intermedio* selecciona los procesos que competirán por el procesador, respondiendo a las fluctuaciones del sistema, dando como resultado que en ocasiones suspenda y active ciertos procesos. Cuando un proceso es suspendido normalmente pasará a la memoria virtual.

La *planificación de bajo nivel* cumple con la función de despachar los procesos al procesador cuando este queda disponible por algún evento que

suspenda la ejecución del proceso actual. Ejemplos de los eventos que suspenden la ejecución son:

- Interrupción de reloj
- Interrupción por E/S
- Llamada al sistema operativo
- Señales

Para lograr una mejor comprensión de los esquemas de planificación revisaremos algunos conceptos importantes.

Reloj de interrupciones. El sistema operativo puede activar un mecanismo de reloj de interrupción cuando es asignado a un proceso, de tal forma, que si antes de un tiempo establecido el proceso no abandona el procesador se activa la interrupción pasando al proceso a la cola de procesos listos.

El uso de prioridades. Estas son asignadas a los procesos, y el planificador dará prioridad de uso del procesador a aquellos procesos que tengan prioridades más altas.

Funciones de selección. Sirven para determinar del conjunto de procesos listos el siguiente a ejecutar, algunas de estas funciones pueden estar basadas en la prioridad, los requerimientos de recursos o las características de ejecución de los procesos.

Modos de decisión. Existen dos modelos que determinan el momento en que debe adoptarse una función de selección:

No apropiativos. En este caso el procesador una vez que ha sido asignado a un proceso no se le puede retirar, hasta que el proceso voluntariamente abandone el procesador, ya sea por que termine su ejecución o pase a un estado de espera por una petición de E/S.

Apropiativos. Al contrario de el caso anterior, el procesador se le puede retirar a un proceso una vez que le ha sido asignado.

1.4.1 ESQUEMAS DE PLANIFICACIÓN

Los esquemas de planificación son las políticas que se van a seguir, junto con un conjunto de reglas y criterios para determinar el proceso que obtendrá el procesador. Estos esquemas varían de uno a otro, obedeciendo a diferentes conceptos y necesidades.

Los esquemas de planificación mas comunes y generales son:

- Primero en llegar - primero en ser atendido (First Come First Served FCFS)
- Asignación en rueda(Round Robin)
- Proceso más corto primero(Short Process Next SPN)
- Tiempo restante más corto(Shortest Remaining Time SRT)
- Siguiente con relación de respuesta máxima(HRRN)
- Colas de retroalimentación (Feedback)

FCFS Primero en llegar - primero en ser atendido

Este esquema de planificación también es conocido como FIFO (First Input First Output), siendo un algoritmo no apropiativo, y el criterio que se aplica es muy simple, una vez que se libera el procesador el siguiente proceso a ser atendido es el que más tiempo tenga en la cola de procesos listos. Este esquema es bastante predecible, aunque no es justo en el sentido de que procesos largos hacen esperar a procesos cortos, y no obedece a prioridades, no es un esquema útil para usuarios interactivos, sin embargo en algunos sistemas suele utilizarse con una variación, en la cual se tienen varias colas de procesos listos obedeciendo a la prioridad de los mismos, atendiéndose a cada cola con el criterio FIFO.

Asignación en Rueda (Round Robin)

Este esquema es muy parecido al anterior, con la única diferencia que obedece a un algoritmo apropiativo. En este caso se establece un "cuanto" que será el tiempo máximo que el proceso pueda hacer uso del procesador, al concluirse este tiempo se activa una interrupción de reloj. Así, los procesos que antes que se de la interrupción de reloj no hayan abandonado al procesador, son retirados de éste y puestos al final de la cola de espera de trabajos listos. El principal problema en este esquema es determinar un cuanto aceptable que no cree una sobrecarga en el sistema.

SPN Proceso más corto primero

Este esquema obedece a una política no apropiativa en la cual el proceso con un tiempo de procesamiento esperado menor es seleccionado. La principal dificultad de este esquema es estimar el tiempo de procesamiento de un proceso.

SRT Tiempo restante más corto

Esta es la versión apropiativa del esquema anterior, pudiendo llegar a existir más sobrecarga que en el esquema anterior debido a que, cuando un nuevo proceso se añade a la cola de listos se compara el tiempo estimado de éste con el tiempo estimado restante del proceso actual en el procesador.

HRRN Siguiente con relación de respuesta máxima

Es un esquema no apropiativo en el que se atiende al trabajo que maximice la siguiente función:

$$F(p) = \frac{\text{tiempo de espera} + \text{tiempo de servicio}}{\text{tiempo de servicio}}$$

Con esta función se favorece a procesos que su ejecución ha sido mínima, sin embargo para aquellos procesos que requieren más tiempo de procesamiento, el valor de la función aumenta directamente al tiempo que han estado esperando.

Colas de retroalimentación

Es un esquema apropiativo, que cuenta con un número finito de colas de listos, los procesos nuevos entran a la cola superior que es atendida de acuerdo a un esquema FIFO, cuando termina el cuanto y el proceso no ha abandonado al procesador pasa a formarse al final de la siguiente cola inferior de espera, y así sucesivamente hasta llegar a la última cola inferior si es que antes no ha terminado el proceso, esta última cola funciona con un esquema de asignación en rueda. Ahora bien el siguiente proceso a ser atendido será el primero en la cola superior, atendiéndose una cola inferior cuando no existan procesos listos en las colas superiores a ella, y el tamaño del cuanto aumentará en las colas inferiores.

En cada uno de los esquemas anteriormente presentados pueden existir variaciones que dependerán del sistema y la forma en que estos hayan sido implementados, algunos autores como Coffman presentan modelos matemáticos para el estudio del comportamiento de estos esquemas [COF73].

La teoría de los sistema operativos es muy amplia, aquí solo hemos presentado de forma general algunos de los conceptos que son útiles para el desarrollo de aplicaciones en tiempo real e influyen de una forma directa en el funcionamiento de un sistema en tiempo real, tal es el caso de la planificación del procesador como el método que el sistema operativo provea para la comunicación entre procesos. En el siguiente capítulo hablaremos más ampliamente de los sistemas operativos de tiempo real.

CAPITULO 2

LOS SISTEMAS OPERATIVOS PARA EL CONTROL EN TIEMPO REAL

Los sistemas operativos juegan un papel muy importante en la administración de los recursos del sistema, de tal forma, que los programadores puedan enfocarse a los problemas específicos de la aplicación a desarrollar en lugar de ocuparse en detalles inherentes a la arquitectura de la computadora. En los sistemas dedicados y de tiempo real, el sistema operativo y la aplicación están fuertemente acoplados, y no es claro como pudieran independizarse. Esto plantea el dilema de como proveer de un alto nivel de abstracción a los programadores y lograr sus requerimientos de desempeño, que son completamente dependientes de la implementación y el ambiente. Los actuales sistemas operativos de propósito general no ofrecen soluciones para las aplicaciones de tiempo crítico, y resultan inadecuados para las demandas de los programadores de tiempo real.

Los núcleos o kernels más exitosos para aplicaciones de tiempo real, son versiones optimizadas de los sistemas convencionales de tiempo compartido. VRTX, VxWorks y Lyns son ejemplos clásicos [BES91]. Las características que ofrecen incluyen un rápido cambio de contexto, un manejo eficiente de interrupciones, rápida adquisición de datos, soporte de relojes de tiempo real, definición por parte del usuario de temporizadores e interrupciones, así como esquemas de planificación basados en prioridades.

Una responsabilidad adicional de los sistemas operativos de tiempo real es la de administrar la información del mundo real y/o cualquier actividad de tiempo real. Esta información debe verse como un recurso compartido por múltiples procesos (incluyendo al sistema operativo), que desea ser accesada concurrentemente. Este acceso debe administrarse de alguna forma para asegurar su consistencia y autenticidad. Para satisfacer tiempos críticos requeridos, el grado de concurrencia debe controlarse por medio de una interacción de los protocolos de control de comunicación y los algoritmos usados en los esquemas de planificación del procesador.

Actualmente ningún sistema operativo se ha impuesto de forma significativa en el mercado como plataforma para las aplicaciones de tiempo real. Un esfuerzo por brindar las bases para la evaluación de los sistemas operativos de tiempo real ha sido proporcionada por POSIX.4 y sus extensiones, dentro de los estándares de el IEEE [IEE90].

2.1 EL ESTÁNDAR POSIX

POSIX ha establecido la normatividad de los sistemas operativos. En 1986, el IEEE decide crear el grupo POSIX con el objetivo de proponer las normas para un amplio conjunto de servicios de los sistemas operativos abiertos.

El primer estándar POSIX, el 1003.1, define la interface entre los programas de aplicaciones y el sistema operativo. Dicho estándar, que se basa en el modelo UNIX, ha sido adoptado como norma ISO 945-1:1990.

El modelo UNIX fue adoptado por POSIX.1 debido a que se trata del único sistema abierto completamente multitarea que existe para la mayoría de las plataformas. El ambiente de desarrollo UNIX es muy popular entre los equipos de desarrollo de software, sin embargo, el requerimiento básico de un sistema de tiempo real es su capacidad de responder a sucesos externos en un tiempo muy corto, y las implementaciones UNIX de AT&T y Berkeley tienen planificaciones del procesador no apropiativas, por lo que su comportamiento no es aceptable para los sistemas en tiempo real. Así, varios de los requerimientos impuestos por el tiempo real exigen modificaciones profundas al núcleo UNIX, no obstante la interface de programación UNIX sea en general satisfactoria, es necesario definir numerosas extensiones para soportar aplicaciones de tiempo real.

Para poder garantizar un tiempo de respuesta determinista, es indispensable un sistema operativo que permita al usuario asignar prioridades fijas a los procesos y un esquema apropiado de planificación. Todo esto se empieza a estudiar para crear las extensiones para el tiempo real dentro del grupo de POSIX desde 1987, denominando a este grupo POSIX.4. Las extensiones de este grupo incluyen la planificación del procesador basado en prioridades, memoria compartida, temporizadores de alta resolución, E/S asíncrono y síncrono, etc.

2.2 CARACTERÍSTICAS DE LOS SISTEMAS OPERATIVOS DE TIEMPO REAL

Dentro de las características de los sistemas operativos con facilidades para aplicaciones de tiempo real deben contemplarse los siguientes [BUR90]:

- Rápido intercambio de procesos
- Núcleo de tamaño pequeño
- Respuesta rápida a interrupciones externas
- Multitarea con un buen sistemas de comunicación entre procesos
- Planeación del procesador apropiativa basada en prioridades de los procesos
- Primitivas para definir pausas en los procesos por un período de tiempo
- Alarmas especiales y time-outs.

El éxito de una aplicación de tiempo real, depende de varios elementos trabajando conjuntamente de forma óptima, incluyendo no sólo la selección del equipo e interfaces, sino además el análisis y diseño de la aplicación, así como debe considerarse la programación misma; por consiguiente, es importante la selección de un buen sistema operativo que brinde al programador las facilidades suficientes para concentrarse en el diseño de la aplicación y se olvide de la interface con el hardware. Dependiendo de la aplicación que se desarrolle se puede hablar de aplicaciones de tiempo real duro, y aplicaciones de tiempo real blando, siendo las primeras muy estrictas en cuanto a los tiempos de respuesta del sistema, permitiéndose una mayor flexibilidad en el segundo caso de aplicaciones, no obstante, en ambos casos es necesaria una buena planificación del procesador que permita a los procesos trabajar de una forma aceptable.

2.2.1 LA PLANIFICACIÓN DEL PROCESADOR

Como ya se mencionó en el capítulo anterior, la planificación del procesador es esencial en un sistema operativo, en las aplicaciones de tiempo real es indispensable que esta planificación se base en los *deadlines* [BUR90], la idea es que los procesos sean ejecutados de acuerdo a tiempos límite definidos para

iniciar o terminar su ejecución, sin embargo, la mayoría de los sistemas operativos no ofrecen la capacidad de trabajar con *deadlines*.

La propuesta para el tipo de esquemas de planificación basados en *deadlines* será establecer el siguiente proceso a usar el procesador en base a la siguiente información de cada proceso [BUR90] :

- *Tiempo de listo*. El tiempo en el cual el proceso pasa al estado de listo.
- *Deadline de inicio*. El tiempo en el cual el proceso deberá iniciar su ejecución.
- *Deadline de terminación*. El tiempo en el cual el proceso debe terminarse. En este caso el planificador sólo considerará un deadline para establecer su criterio.
- *Tiempo estimado de procesamiento*. El tiempo que tardará al proceso en completar su ejecución.
- *Requerimientos de recursos*. Refiriéndose a aquellos recursos que sean distintos al procesador.
- *Prioridad*. Esta deberá ser representativa de la importancia relativa del proceso.

Los sistemas operativos actuales diseñados para soportar aplicaciones de tiempo real tienen como objetivo iniciar los procesos tan rápidamente como sea posible, esto significa que se trata de minimizar el tiempo que estos se encuentran en el estado de listos en espera del procesador. Para ello, la alternativa es hacer énfasis con un manejo de interrupciones rápido y un despachador de procesos eficiente.

Si consideramos que los tiempos de respuesta en un tiempo real duro son muy rígidos, y son delimitados por las características inherentes del mundo real, encontraremos ciertas ventajas en un esquema de planificación basado en *deadlines*, siempre y cuando estos se fijen de acuerdo con los requerimientos del mundo real y se establezcan en función de los eventos que ocurran; sin embargo, para aplicaciones de tiempo real blando donde no resultan tan críticos estos tiempos, la mejor opción es optar por un esquema de planificación basado en prioridades y con la propiedad de apropiatividad del procesador.

2.2.2 LATENCIAS DE INTERRUPCIÓN Y CAMBIO DE CONTEXTO

Un término comúnmente empleado para comparar rendimientos en los sistemas operativos son los tiempos de latencia y el cambio de contexto.

Los tiempos de latencia de una interrupción se refieren al tiempo máximo que toma al sistema operativo atender a la interrupción, por definición una interrupción tiene una prioridad muy alta, de tal forma que pueda apropiarse del procesador, sin embargo, en todo sistema existen frecuentemente caminos de procesamiento críticos no reentrables, los cuales deben ser concluidos antes de ejecutar la interrupción. La longitud de estos caminos, o sea, el número de instrucciones que deben ejecutarse antes que pueda ser atendida la interrupción, determinará el tiempo de latencia en el peor de los casos. Aunque estos tiempos resulten insignificantes, o parecieran muy pequeños, cuando hablamos de aplicaciones en tiempo real y en particular las de tiempo duro, llegan a ser muy importantes.

Otro concepto, no menos importante, es el cambio de contexto. En todo sistema multitarea, hay que cuidar que la conmutación de un proceso a otro ocurra en el menor tiempo posible, el tiempo de cambio de contexto está determinado por el tiempo que el sistema operativo tarda en guardar el contenido de los registros y el estado de la computadora del proceso que va a ser retirado del procesador, así como el tiempo que tarda en restablecer estas condiciones para el siguiente proceso.

2.2.3 COMUNICACIÓN ENTRE PROCESOS

La comunicación entre procesos IPC (Interprocess communication), esta basada en la transmisión de mensajes como se mencionó en el capítulo anterior. En los sistemas operativos de tiempo real se pretende dar las facilidades necesarias para que los procesos puedan sincronizar actividades, pasar datos o simplemente notificar eventos.

Para ello se requiere que funcionalmente la comunicación entre procesos nos brinde las siguientes características:

- Un proceso pueda recibir mensajes de cualquier proceso.

- Un proceso se pueda bloquear en espera de un mensaje.
- Un proceso se pueda bloquear (pasar al estado de espera) hasta que el proceso receptor reciba su mensaje, lo atienda y le envíe una respuesta.
- Un proceso transmisor pueda enviar un mensaje sin esperar respuesta del receptor.
- Los mensajes que recibe un proceso puedan ser puestos en una fila de espera de acuerdo a las prioridades del proceso que los emitió.
- Los mensajes puedan ser de tamaño dinámico.

Con esto, el programador se olvida de los detalles referentes a la implementación de la comunicación entre procesos y se da un poderoso conjunto de opciones con las cuales puede establecerse una comunicación completa entre los procesos, evitando con ello el uso de semáforos.

2.4 SELECCIÓN DEL SISTEMA OPERATIVO

La selección del sistema operativo no es una tarea sencilla, si bien es cierto, que es una parte esencial dentro de una aplicación de tiempo real, no menos importante será la selección del equipo de cómputo donde se ejecutará, y aun así, el éxito de la aplicación dependerá fuertemente del diseño e implementación de la misma.

Algunos sistemas operativos de tiempo real son aplicables a un rango de configuraciones de sistemas, mientras que otros están enfocados a una plataforma en particular e incluso a un microprocesador específico. Los sistemas operativos que se pueden usar para el desarrollo de este tipo de aplicaciones pueden ser sistemas operativos de propósito general que de alguna forma han reforzado sus características para ofrecer servicios para aplicaciones de tiempo real, o bien, sistemas operativos diseñados exclusivamente para este tipo de aplicaciones.

Dentro de los sistemas operativos de tiempo real actuales, ninguno ha invadido, o ha sido demasiado popular para imponerse en el mercado, y por ello muchos de estos sistemas operativos resultan costosos. En general, estos sistemas operativos ofrecen las mismas características, y para su evaluación debemos fijarnos en los tiempos de latencia, tiempos de cambio de contexto, tipos de planeación que ofrecen, facilidades de E/S síncronas y asíncronas, y las ventajas que ofrecen en la comunicación entre procesos. No hay que perder de vista que las características inherentes a los tiempos están fuertemente ligadas a la computadora en la cual se operará, por lo que otro aspecto importante en su evaluación es el aspecto económico.

Para la realización del presente proyecto, seleccionamos el sistema operativo QNX [QNX][HIL92][HIL93][HIL94][OAK94], debido principalmente a su característica de ser un sistema operativo para PC, su bajo costo, comparado con otras plataforma, la operabilidad que tiene en redes, y la facilidad de particionar aplicaciones de una forma muy transparente para ejecutarse en un ambiente distribuido en una red LAN. A continuación se presenta un resumen de sus principales características operativas.

2.5 EL SISTEMA OPERATIVO QNX

El sistema operativo QNX está diseñado para aplicaciones de tiempo real, siendo un sistema para PC's, por lo que la implementación de este tipo de aplicaciones reduce su costo considerablemente si se compara con otras plataformas.

Las principales facilidades que nos brinda este sistema operativo son: un ambiente multitarea, una planeación del procesador basada en las prioridades de los procesos, así como un rápido intercambio entre procesos. Otra de las características importantes de este sistema operativo es la facilidad que brinda para la comunicación entre procesos.

2.5.1 EL NÚCLEO DE QNX

El núcleo del sistema operativo de QNX cumple con el requisito de ser pequeño, considerándose un Microkernel ya que sólo ocupa 8 kbytes. El núcleo se dedica a la planificación del procesador, es el ruteador de los mensajes y el manejo de interrupciones. Los demás servicios del sistema operativo son manejados por los siguientes administradores (fig. 2.1):

- Administrador de procesos
- Administrador de Archivos
- Administrador de Dispositivos
- Administrador de la Red

El ruteo de los mensajes integra el soporte de los servicios de red, dando como resultado que el usuario no necesite tener cuidado si los receptores residen en el mismo nodo o se encuentran en nodos diferentes de la red. Los procesos servidores que proveen de varios servicios son implementados directamente con un paso de mensajes transparentes a través de la red, obteniendo así que QNX pueda ser un sistema operativo completamente distribuido en una red.

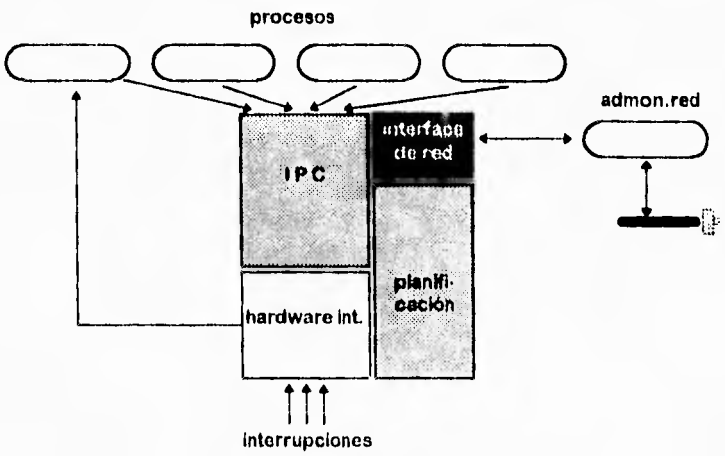


Figura 2.1 Microkernel de QNX

2.5.2 REDES EN QNX

El sistema operativo QNX nos brinda la facilidad de compartir archivos y dispositivos periféricos en una red de área local.

Cualquier proceso de cualquier máquina de la red puede hacer uso de cualquier recurso de otra máquina, no existiendo diferencias entre los recursos locales o remotos, así como no es necesario construir procedimientos especiales en las aplicaciones para hacer uso de los recursos, con lo cuál no es necesario construir código especial para accesar un archivo o dispositivo en cualquier nodo de la red.

Además este sistema operativo provee de un proceso de comunicaciones transparente para los usuarios de la red, dando como resultado que los procesos puedan comunicarse fácilmente tanto local o remotamente cuando hablemos de un procesamiento distribuido.

Con la arquitectura que nos brinda el microkernel de QNX y las facilidades para el ruteo de mensajes, QNX puede manejar una red de computadoras y presentarla a las aplicaciones lógicamente como una sola máquina, sin importar cuantas computadoras físicas sean soportadas por la red. Así, las aplicaciones desarrolladas pueden correr sin ningún cambio, sin importar que el número de computadoras de la red aumente, o bien, que la aplicación se ejecute de forma distribuida.

2.5.3 LA COMUNICACIÓN ENTRE PROCESOS

QNX brinda un conjunto sencillo de capacidades para la comunicación entre procesos, y su función es la transmisión de mensajes entre los procesos cooperativos. Los mensajes como tales son paquetes de bytes que pasan de un proceso a otro, sin que estos tengan un significado especial para el sistema, los datos ha transmitir tendrán significado únicamente para el proceso que lo envía y el que lo recibe.

En lo que se refiere a la transmisión de mensajes se tienen las siguientes instrucciones: *send*, *receive* y *reply*, afectando con estas operaciones los estados de los procesos (Tabla 2.1)

Si el proceso ejecuta:	El proceso pasa al estado:
Send(), y el mensaje que envía no ha sido recibido por el proceso.	bloqueado - SEND
Send(), y el mensaje ha sido recibido, pero el proceso no le ha respondido.	bloqueado - REPLY
Receive(), pero no se ha recibido ningún mensaje	bloqueado - RECEIVE

Tabla 2.1

A manera de resumen, cuando un proceso envía un mensaje a otro proceso por medio de la llamada Send, este se bloquea hasta que recibe un Reply del proceso al que mando el mensaje. Mientras que si un proceso ejecuta un Receive sin que exista un mensaje pendiente, este se bloqueará hasta que otro proceso le mande un mensaje.

Existe otra forma de comunicación entre procesos a través de proxies, esta es una forma de pasar mensajes pero sin bloquearse los procesos, o esperar una respuesta. Algunos ejemplos del uso de proxies son los siguientes:

- Un proceso quiere notificar a otro que un evento ha ocurrido, pero no puede esperar a que el proceso destinatario reciba y responda al mensaje.
- Un proceso quiere mandar algunos datos a otro proceso, pero no necesita una respuesta o algún indicador de que el proceso destinatario ha recibido el mensaje.
- Un manejador de interrupciones desea informarle a algún proceso que los datos están disponibles para ser procesados.

Para esta forma de comunicación, se crea un proceso tipo proxy con un mensaje único previamente establecido, de tal forma que el proceso que desea mandar el proxy lo hace ejecutando la función *Trigger (proxy)*, y el proceso receptor recibirá el mensaje, sin tener que responder a él.

QNX también soporta el uso de señales para la comunicación entre procesos. Las señales son la forma tradicional de comunicación entre procesos en los sistemas operativos tradicionales, como es el caso de Unix. Ejemplos clásicos de señales es el caso de *kill*, la cual puede ser generada desde un shell o bien desde un proceso, (las funciones de C *kill()* o *raise()*).

2.5.4 LA PLANIFICACIÓN DEL PROCESADOR EN QNX

Las primitivas de planeación del procesador que nos brinda QNX, están de acuerdo con la especificación de POSIX 1003.4 [QNX94], dando como resultado una planificación apropiativa, intercambiando de contexto de acuerdo a las prioridades con esquemas de planificación en rueda (round-robin), FIFO, y planificación adaptable.

2.5.4.1 LAS PRIORIDADES DE LOS PROCESOS

En QNX, cada proceso tiene asignada una prioridad, el esquema de planificación seleccionará al siguiente proceso que correrá observando la prioridad asignada a cada proceso que se encuentre en el estado READY, siendo aquél que tenga la mayor prioridad el que pase a ocupar el procesador. Las prioridades son asignadas a los procesos en un rango de 0 (el mas bajo) a 31 (el mas alto). La prioridad inicial es heredada del proceso padre, y es de 10 para todas las aplicaciones que se iniciaron desde el Shell. Sin embargo estas prioridades pueden ser dinámicas manejándolas el programador a su conveniencia.

2.5.4.2 LOS ESQUEMAS DE PLANIFICACIÓN

El esquema de planificación es heredado del proceso padre, y es asignada la planificación adaptable para las aplicaciones iniciadas desde el Shell, este esquema de planificación puede ser modificado por el programador en sus aplicaciones.

Como se mencionó anteriormente QNX ofrece tres esquemas de planificación: FIFO, planificación en rueda (round-robin) y planificación adaptable.

El esquema FIFO, nos ofrece la ventaja de garantizarnos exclusión mutua para dos procesos que tienen la misma prioridad, ya que ningún proceso puede apropiar a otro el procesador mientras se ejecuta. De esta forma, si se comparte un segmento de memoria, cualquiera de los procesos lo puede modificar sin necesidad de alguna forma de semáforos.

El esquema de planificación en rueda se rige por un cuanto de 100 milisegundos, tiempo que ha sido determinado por los diseñadores del sistema operativo para garantizarnos un funcionamiento óptimo de los procesos concurrentes.

Por último el esquema de planificación adaptable, funciona como sigue:

- Si un proceso consume su cuanto, su prioridad se reduce en 1 si un proceso de la misma prioridad se encuentra listo.
- Si un proceso ha esperado por 1 segundo, su prioridad se aumenta en 1.
- Si un proceso se bloquea, inmediatamente su prioridad vuelve a ser la original.

Este esquema es recomendado para ambientes en donde existen muchos procesos corriendo en background, y se comparte la computadora con varios usuarios interactivos.

Finalmente en muchas aplicaciones de QNX, la mayoría de las transacciones entre los procesos siguen un modelo cliente/servidor. Los procesos servidores proveen de algún servicio a los procesos clientes que envían mensajes a éstos pidiendo un servicio. Usualmente los clientes sobrepasan en número a los servidores, dando como resultado, que sea común que la prioridad del servidor sea mayor a cualquiera de las prioridades de los clientes [QUA91]. Para ello es usual usar un esquema de planificación en rueda. No obstante, cuando un proceso de baja prioridad es atendido por el servidor, se altera de una forma indirecta la prioridad del cliente, si la operación que atiende es rápida no afectará a los otros procesos este esquema, pero si el tiempo en que tarda el servidor es mayor, se da el caso que un proceso cliente con mayor prioridad al que se está atendiendo, tenga que esperar a que el servidor termine de atender al cliente actual.

QNX nos brinda las facilidades de establecer un esquema en el cual se resuelva el dilema anterior, para ello el servidor correrá con la misma prioridad que el proceso cliente que lo está solicitando, y si en algún momento se encuentra esperando un mensaje de un proceso de mayor prioridad, el servidor automáticamente ajustará su prioridad a la de este nuevo proceso, para ello es necesario que los mensajes sean atendidos en orden de prioridad en lugar de ordenarlos por tiempo, para ello se usará una función especial llamada *qnx_pflags*.

2.5.5 ESPECIFICACIONES DEL SISTEMA OPERATIVO QNX 4

Dentro de las principales especificaciones del sistema operativo QNX para tiempo real [QNX], tenemos que, en lo que se refiere al núcleo cuenta con un microkernel pequeño de aproximadamente 8 kbytes, cumpliendo con el estándar de planificación de procesos en tiempo real (POSIX 1003.4), para lo cual:

- Soporta 32 niveles de prioridades.
- Soporta 3 algoritmos de planificación: FIFO, round-robin y adaptable, siendo seleccionada por cada procesos el algoritmo.
- Maneja un cambio de contexto por prioridades y completamente apropiativo.
- Los servidores pueden manejar su prioridad de acuerdo al mensaje que reciben.
- Solamente maneja 16 llamadas al kernel, los demás servicios del sistema operativo son proporcionados por procesos opcionales.

- El microkernel sólo maneja la planificación del procesador y el ruteo de los mensajes.

Cumple con el estándar POSIX 1003.4 relativo a los relojes y timers, por lo que:

- Soporta múltiples timers por proceso
- Los timers pueden ser síncronos o asíncronos
- La resolución de los timers es de nanosegundos

Los tiempos referentes al comportamiento del sistema operativo se muestran a continuación [QNX]*:

	Tiempo
Cambio de contexto	6 microsegundos
Latencia de interrupción	7 microsegundos
E/S disco	2.8 Mbytes/segundo
E/S serial	115 Kbauds

* Tiempos medidos en un procesador 486DX2 a 66 Mhz con un disco Maxtor MXT-124OS.

Procesador ***	Latencia de interrupción**	Cambio de contexto
60 Mhz Pentium	4	5
66 Mhz 486	7	6
33 Mhz 486	8	12
33 Mhz 386	11	30
16 Mhz 386SX	32	90
8 Mhz 286	65	175

** Estas latencias indican el peor de los casos.
*** Tiempos en microsegundos

Hemos presentado las principales grandes ventajas que ofrecen los sistemas operativos específicos para aplicaciones en tiempo real, con ellos se logra que el diseño de las aplicaciones se enfoque a la funcionalidad de las mismas y su interacción con el mundo exterior. El éxito de esta clase de sistemas operativos para aplicaciones en tiempo real, radica en la potencialidad de las funciones y servicios que ofrecen, sin crear capas de software adicional.

CAPITULO 3

METODOLOGÍAS DE DISEÑO DE SOFTWARE Y SU APLICACIÓN EN SISTEMAS DE TIEMPO REAL

Un sistema en tiempo real debe integrar varios elementos como se menciona a lo largo del presente trabajo, por un lado tenemos los dispositivos de hardware que se comunicarán con el mundo real, las arquitecturas o dispositivos electrónicos que transmitirán las señales a la computadora.

En lo que se refiere al análisis y diseño de la aplicación en general, como lo describe Allworth [ALL87], el punto de inicio es la especificación de requerimientos, en donde es fundamental definir la frecuencia natural de los procesos que van a ser muestreados y controlados, así como las limitaciones del ancho de banda inherentes a los dispositivos que se usarán. Así mismo, deben determinarse las estrategias de control y las técnicas que se emplearán para ello, ya sea que se implementen por software o bien por dispositivos electrónicos. Debe considerarse el equipo de cómputo que va a ser usado, ya sea que se disponga de él o se adquiera, así como el sistema operativo que será usado.

La disposición de los sensores, los transductores, actuadores así como la localización de los dispositivos de alarma deben ser considerados para determinar los requerimientos de las señales, el muestreo, la digitalización de señales analógicas (cuando existan), y su transmisión a la computadora.

El analista y diseñador del software debe considerar las implicaciones y facilidades que le dará el equipo de cómputo, siendo parte importante de estas consideraciones el sistema operativo que será usado, así como el lenguaje de desarrollo.

Dentro de esta etapa de análisis del proyecto deben delimitarse las funciones, alcances y limitaciones del software, teniendo claros los puntos anteriormente expuestos.

3.1 ANÁLISIS DEL SOFTWARE DE LOS SISTEMAS EN TIEMPO REAL

Una vez que se han definido las condiciones generales del proyecto, en lo que se refiere a la implementación del software, deberá existir como todo proyecto de software un análisis y diseño del mismo. Cabe hacer la aclaración que en el análisis del proyecto a nivel general deben delimitarse algunos de los requerimientos del software, y estos serán tomados como punto de partida.

Ahora bien, para el análisis de esta clase de sistemas se involucra un conjunto de atributos dinámicos que deben considerarse en los requerimientos funcionales de la aplicación del sistema de tiempo real [PRE90]:

- Manejo de interrupciones y cambio de contexto
- Tiempo de respuesta
- Razón de transferencia de datos y tiempo invertido
- Asignación de recursos y manejo de prioridades
- Sincronización de tareas y comunicación entre tareas

Como podemos observar estos atributos están relacionados directamente con el sistema operativo que emplearemos, así como el equipo en el que se ejecutará el sistema. Algunos métodos de análisis que pueden auxiliarnos en esta clase de sistemas son el SREM con su extensión SYSREM como lo describe Sacha [SAC89], los detalles generales de este tipo de análisis pueden revisarse en [PRE90]. Se pueden usar las herramientas tradicionales para el análisis de requerimientos como lo propone Pressman[PRE90], tales como SACT, SREM, PSL/PSA y TAGS.

Un problema que es muy común que se presente en esta clase de sistemas es poder separar la parte del análisis de la parte de diseño, debido al acoplamiento que debe existir entre el mundo real y la aplicación, por lo que en muchas ocasiones es inevitable considerar elementos del diseño en la parte del análisis. En este capítulo nos enfocaremos a dar una panorámica de los elementos de análisis y diseño que pueden ser usados en el desarrollo del software de un sistema en tiempo real.

3.1 ANÁLISIS DEL SOFTWARE DE LOS SISTEMAS EN TIEMPO REAL

Una vez que se han definido las condiciones generales del proyecto, en lo que se refiere a la implementación del software, deberá existir como todo proyecto de software un análisis y diseño del mismo. Cabe hacer la aclaración que en el análisis del proyecto a nivel general deben delimitarse algunos de los requerimientos del software, y estos serán tomados como punto de partida.

Ahora bien, para el análisis de esta clase de sistemas se involucra un conjunto de atributos dinámicos que deben considerarse en los requerimientos funcionales de la aplicación del sistema de tiempo real [PRE90]:

- Manejo de interrupciones y cambio de contexto
- Tiempo de respuesta
- Razón de transferencia de datos y tiempo invertido
- Asignación de recursos y manejo de prioridades
- Sincronización de tareas y comunicación entre tareas

Como podemos observar estos atributos están relacionados directamente con el sistema operativo que emplearemos, así como el equipo en el que se ejecutará el sistema. Algunos métodos de análisis que pueden auxiliarnos en esta clase de sistemas son el SREM con su extensión SYSREM como lo describe Sacha [SAC89], los detalles generales de este tipo de análisis pueden revisarse en [PRE90]. Se pueden usar las herramientas tradicionales para el análisis de requerimientos como lo propone Pressman[PRE90], tales como SACT, SREM, PSL/PSA y TAGS.

Un problema que es muy común que se presente en esta clase de sistemas es poder separar la parte del análisis de la parte de diseño, debido al acoplamiento que debe existir entre el mundo real y la aplicación, por lo que en muchas ocasiones es inevitable considerar elementos del diseño en la parte del análisis. En este capítulo nos enfocaremos a dar una panorámica de los elementos de análisis y diseño que pueden ser usados en el desarrollo del software de un sistema en tiempo real.

3.2 MÉTODOS DE ANÁLISIS DE REQUERIMIENTOS

Por su naturaleza los programas secuenciales resultan inadecuados para esta clase de sistemas, por lo que se recurre a dividir la aplicación en módulos (procesos) que se ejecuten simultáneamente en la computadora (programación concurrente). Desafortunadamente no existen métodos formales para el análisis y diseño de programas concurrentes, por lo que algunos autores como lo describen Subelj y Trobec [SUB89] han sugerido un análisis enfocado al flujo de datos con ciertas extensiones, otros autores como lo muestra Pressman en sus trabajos [PRE90] sugieren el uso de herramientas matemáticas en la etapa de análisis, basándose en la teoría de colas para equilibrar la concurrencia de los procesos, siendo necesario conocer:

- El número de llamadas a un proceso
- El tiempo de procesamiento para un proceso específico
- El tiempo total empleado por el sistema

Kalisnaky y Ready [KAL89] nos indican que a pesar de los esfuerzos por hacer extensiones del análisis estructurado para aplicaciones en tiempo real, para sistematizar la transformación de las especificaciones de requerimientos en el diseño específico se debe plantear:

- Un modelo funcional
- Un modelo de comportamiento
- La especificación de la interface de hardware y software
- La interface hombre máquina

Con lo cual se pasa a un diseño gráfico en donde deben hacerse distinción entre los elementos secuenciales de software de los componentes concurrentes, así como definir los detalles de la comunicación entre procesos y su sincronización.

3.3 MÉTODOS DE DISEÑO DE SOFTWARE

Para esta etapa del desarrollo del software existen varias tendencias, y encontramos diversos enfoques, algunos de ellos son variaciones de las metodologías tradicionales como las metodologías de flujo de datos, de estructuras de datos y orientadas a objetos. Algunas otras involucran el concepto de estados, o bien un sistema de paso de mensajes, así como existen estudios sobre redes de Petri. Estas metodologías tratan de ser coherentes con las necesidades de las aplicaciones de tiempo real, y hay que recordar que esta clase de aplicaciones se corren en sistemas multitarea, por lo que es necesario dividir las aplicaciones en procesos, y en ocasiones los métodos no nos brindan una solución clara de como hacer esta partición.

3.3.1 EL MÉTODO DARTS

El método DARTS (Design Approach for Real-Time Systems), fue desarrollado por Hassan Gomma, este es un método de diseño que utiliza la notación del flujo de datos, para ello se crean los diagramas de flujo de datos, se definen los diccionarios de datos y se establecen las interfaces entre las principales funciones del sistema [PRE90]. El siguiente paso después de definir el diagrama de flujo de datos, será descomponer este en tareas o subsistemas que puedan ser ejecutados como procesos en un ambiente concurrente. Para la descomposición en procesos nos podemos guiar por lo siguiente:

- Todos aquellas funciones que interactuén con dispositivos de E/S deberán ejecutarse en un proceso separado y dedicado a ello.
- Las funciones de tiempo crítico deben ser ejecutadas con una prioridad alta, para ello deberán constituir procesos independientes.
- Las funciones con demasiados cálculos deben ejecutarse como procesos de baja prioridad, siempre y cuando los resultados de estos cálculos no representen un tiempo crítico.

- **Aquellos módulos o funciones que transfieran entre si un gran número de datos, con demasiado tráfico, deben ser vistas como un único proceso, para evitar sobrecargar al sistema por la comunicación de mensajes entre procesos.**

Dentro de este diseño hay que considerar los módulos de comunicación de mensajes entre procesos, ya sea para sincronizarlas o simplemente transferir datos.

3.3.2 EL MÉTODO TAGS

El método TAGS (Technology for the Automated Generation of Systems) fue desarrollado por Teledain Brown Engineering, Inc. En este método las especificaciones del software se descomponen en dos niveles de problemas. En el primero se encuentran los componentes independientes de el desarrollo del sistema así como se establecen las dependencia del flujo de datos entre los componentes . En el segundo se encuentran los componentes de la estructura de control que son descritos en un lenguaje gráfico IORL (Input/Output Requirement Language). En este lenguaje se describen diagramas de bloques esquemáticos, identificando los componentes del sistema y las interfaces de datos entre ellas. [SAC89].

3.3.3 MÉTODOS ORIENTADOS A OBJETOS

Un diseño orientado a objetos puede ser usado para este tipo de sistemas, pero hay que tomar en consideración la descomposición en procesos, esta se puede hacer como programas concurrentes desarrollados con esta metodología, o bien, los procesos pueden ser objetos vivos dentro del sistema. Y habrá que definir los métodos pertinentes para la comunicación entre procesos, una buena perspectiva de este tipo de metodología nos la da Booch [BOO94]. Ellison nos presenta una metodología para la asignación de procesos [ELL94], que se basa en:

- **Estrategias de consolidación de instancias. Todas las instancias de un mismo objeto deberán ser manejadas por un mismo proceso, en donde un proceso puede manejar varios objetos.**

- **Estrategias de dispersión de instancias. Diferentes instancias del mismo objeto pueden ser distribuidas en diferentes objetos.**

- **Estrategias de dispersión de estados. Las acciones de los procesos de una misma máquina de estados pueden distribuirse en diferentes tareas.**

3.3.4 REDES PETRI

Estas se basan en la idea de que un sistema dedicado actúa en un ambiente en el cual existe un número de objetos asíncronos y relativos. En el modelado de la red se consideran tanto el propósito del sistema, como su ambiente. La unidad primaria de especificación son los procesos, los cuales son vistos como una representación abstracta de sus objetos computacionales y del ambiente [SAC89]. Se basan en una representación gráfica en la que se usan una serie de estructuras circulares que son llamados lugares, los cuales representan datos o procesos, y una serie de rectángulos para representar transiciones u operaciones. Los procesos y transiciones son etiquetadas con sus respectivas funciones de transición y son conectados por arcos bidireccionales. Ejemplos de estas redes los encontramos en [LAP93].

3.3.5 MÁQUINAS DE ESTADOS FINITOS

Los autómatas de estados finitos o máquinas de estados finitos, son modelos matemáticos usados en el diseño de compiladores, sistemas de comunicaciones, etc. Estos modelos pueden ser aprovechados para auxiliar nuestro diseño, partiendo del principio de que un sistema puede ser representado por un número único de estados, en donde los estados del sistema variarán dependiendo del tiempo en que ocurran una serie de eventos específicos. Se pueden utilizar para este modelado las máquinas de Moore y Mealy como lo propone Laplante [LAP93].

3.3.6 EL DISEÑO CLIENTE/SERVIDOR COMO HERRAMIENTA AUXILIAR

Actualmente las aplicaciones en redes han involucrado el concepto de aplicaciones cliente/servidor, existiendo para ello varias herramientas CASE muy poderosas que incluyen el manejo de bases de datos distribuidas. El concepto cliente/servidor nos puede auxiliar en la descomposición en procesos de las aplicaciones en tiempo real, viendo al servidor como el administrador de un recurso y al cliente como el programa de aplicación, en donde, los servidores aceptan peticiones de los procesos clientes y se encargan que estas sean efectuadas.

Rescatando esta idea, en las aplicaciones en tiempo real, este diseño es muy útil sobre todo para lo que se refiere al monitoreo de las señales del mundo exterior. Con un diseño c/s tendremos un control centralizado de los dispositivos de entrada/salida de la computadora, evitando con ello interbloqueo de los procesos al querer accesar estos dispositivos. Un texto ilustrativo del uso de este diseño lo podemos encontrar en las notas de Quantum Software Systems[QUA91], y un enfoque general de esta metodología lo tenemos en el libro de Vaughn [VAN94].

Independientemente de la metodología que se use en este tipo de aplicaciones en tiempo real, se debe considerar la descomposición en procesos de la aplicación. Desafortunadamente no hay una metodología formal estandarizada, en la que se involucren todas las etapas del desarrollo de aplicaciones en tiempo real, a pesar de que algunos autores han hecho esfuerzos por dar a conocer sus avances en este sentido. El caso mas usual es combinar algunos conceptos que nos puedan ser útiles en la descomposición de nuestra aplicación en procesos, y posteriormente usar alguna de las metodologías tradicionales sobre las que hay una gran experiencia y aportaciones en ellas.

Para nuestro sistema en particular se sugiere iniciar con el análisis del proyecto general delimitando las partes concernientes o que se vean involucradas en el funcionamiento de la aplicación, tal es el caso del tipo de comunicación que

se establece entre los procesos físicos reales y la computadora. Posteriormente, después del análisis de la arquitectura de cómputo que se utilizará en el sistema, y considerando las facilidades brindadas por el sistema operativo, se propone utilizar una metodología de análisis y diseño estructurado, cuidando que la modularización que obtengamos se base en la guía presentada en este capítulo referente al método DARTS. A continuación para la construcción de los módulos nos basaremos en el modelo cliente/servidor, estableciendo los procesos servidores necesarios para lograr un buen uso de los recursos compartidos, tal sería el caso de los puertos por los que se establezca la comunicación con el mundo real.

Hay que recordar que en un sistema en el que existe un flujo de datos constante entre los módulos se identifican esquemas productor/consumidor, la idea será establecer el tipo de mensajes y la forma en que estos serán pasados para sincronizar actividades de esta clase de procesos. Esta parte de la definición de mensajes debe ser detallada y cuidadosa, tratando de establecer un estándar en los mensajes a transmitir, de manera que facilite la implementación de la aplicación y no sobrecargue al sistema.

En esta parte la tendencia para la definición de mensajes es la siguiente:

- Los procesos pueden recibir o mandar mensajes a un proceso particular para sincronizarse, en este caso el formato del mensaje sólo involucra a los dos procesos.

- Los procesos productor/consumidor trabajarán de la siguiente forma, el proceso productor notificará al proceso consumidor la existencia de nuevos datos, y continuará su operación, en el momento que el proceso consumidor pueda solicitar la información mandará un mensaje, en este caso el proceso consumidor esperará a que el productor le conteste el mensaje. Para este esquema deben definirse los mensajes de notificación, de petición y de respuesta.

- Los procesos pueden recibir mensajes de varios procesos de forma asíncrona, por lo que el proceso estará en espera de un mensaje que puede ser de cualquier proceso, de esta forma hay que revisar todos los procesos involucrados para tal efecto y establecer un formato único y común a todos los procesos para el envío de esta clase de mensajes.

- **Los procesos servidores por definición solo atienden peticiones, por lo que esta clase de procesos se limita a recibir y atender un mensaje que les llegue y contestarlo. Un proceso servidor jamás enviara un mensaje, sólo responde a ellos. Por lo cuál, deben definirse los mensajes que recibe y los mensajes con los que responde.**

De esta forma, ante la carencia de una metodología que satisfaga todas nuestras necesidades, se combinaron varias de ellas para el desarrollo del presente trabajo. En los siguientes capítulos nos referiremos al proyecto en general, así como los detalles del análisis y diseño de la aplicación de monitoreo y control desarrollada.

CAPITULO 4

ARQUITECTURA DEL SISTEMA DE MONITOREO Y CONTROL DE PROCESOS EN TIEMPO REAL

El objetivo del proyecto es poder controlar de forma distribuida diferentes procesos, teniendo un monitoreo central por medio de una computadora que nos muestre el estado de éstos en forma gráfica y numérica. El proyecto consta de varias partes, siendo el objetivo del presente trabajo enfocarse a la parte correspondiente a la aplicación residente en la computadora. En este capítulo presentamos una perspectiva general del proyecto global.

4.1 LA ARQUITECTURA

Para este proyecto se propone una arquitectura abierta y flexible, en donde los procesos físicos serán controlados por microcontroladores de forma distribuida, con arquitecturas y programas de control propias para cada proceso, creando con ello una independencia con el resto del sistema. Los microcontroladores tienen la posibilidad de comunicarse a través del puerto serial a la computadora, de manera que puedan ser monitoreados en todo momento, así como la computadora puede modificar el estado de los actuadores del sistema. Con esta arquitectura se pretende además del control distribuido en los microcontroladores, la posibilidad de tener un control central a través de la computadora. Como aplicación consideraremos un sistema de control automático para locales, en donde se tengan diferentes sensores y actuadores, distribuyéndolos de forma tal, que cada proceso consista de una serie de sensores con sus respectivos actuadores, así como los algoritmos de control propios para cada proceso.

4.2 LOS PROCESOS

Nuestro sistema consiste de 4 módulos básicos que sensan diferentes variables tales como ruido, temperatura y presión, así como detectarán objetos móviles en un área determinada, el número de procesos puede ser incrementado,

CAPITULO 4

ARQUITECTURA DEL SISTEMA DE MONITOREO Y CONTROL DE PROCESOS EN TIEMPO REAL

El objetivo del proyecto es poder controlar de forma distribuida diferentes procesos, teniendo un monitoreo central por medio de una computadora que nos muestre el estado de éstos en forma gráfica y numérica. El proyecto consta de varias partes, siendo el objetivo del presente trabajo enfocarse a la parte correspondiente a la aplicación residente en la computadora. En este capítulo presentamos una perspectiva general del proyecto global.

4.1 LA ARQUITECTURA

Para este proyecto se propone una arquitectura abierta y flexible, en donde los procesos físicos serán controlados por microcontroladores de forma distribuida, con arquitecturas y programas de control propias para cada proceso, creando con ello una independencia con el resto del sistema. Los microcontroladores tienen la posibilidad de comunicarse a través del puerto serial a la computadora, de manera que puedan ser monitoreados en todo momento, así como la computadora puede modificar el estado de los actuadores del sistema. Con esta arquitectura se pretende además del control distribuido en los microcontroladores, la posibilidad de tener un control central a través de la computadora. Como aplicación consideraremos un sistema de control automático para locales, en donde se tengan diferentes sensores y actuadores, distribuyéndolos de forma tal, que cada proceso consista de una serie de sensores con sus respectivos actuadores, así como los algoritmos de control propios para cada proceso.

4.2 LOS PROCESOS

Nuestro sistema consiste de 4 módulos básicos que sensan diferentes variables tales como ruido, temperatura y presión, así como detectarán objetos móviles en un área determinada, el número de procesos puede ser incrementado,

o bien sustituir algún proceso por otro diferente. Por el momento sólo manejaremos 4 procesos, cuyas características se describen a continuación.

- El proceso 1 tiene como objetivo detectar un objeto móvil dentro de un área determinada, usando sensores de proximidad, de tal forma que al detectar movimiento, estos activarán a un motor de pasos que moverá una cámara de vídeo para seguir la trayectoria del objeto móvil.

- El proceso 2 manejará sensores ópticos y tendrá como actuadores válvulas solenoides que controlarán el acceso a ciertas áreas, la idea es cerrar puertas cuando se obstruya alguno de los sensores. La disposición de los dispositivos será por pares, un sensor óptico trabajará con una válvula solenoide, y cuando el sensor óptico sea obstruido la válvula solenoide cerrará la puerta respectiva.

- El proceso 3 tiene como objetivo sensar la temperatura y la presión del lugar, teniendo ciertos umbrales para activar los actuadores respectivos y mantener un ambiente adecuado en el local. Se cuentan con 4 sensores: 2 de temperatura y 2 de presión; a los que corresponden 4 actuadores que controlan: un calentador, un ventilador, un compresor de aire y una válvula solenoide encargada de controlar el paso del aire que sale del compresor.

- El proceso 4 sensará niveles de ruido, cuya función es detectar la presencia de una persona en un área determinada, teniendo como actuadores triacs que activarán la iluminación del lugar; para ello se cuentan con 4 sensores y 4 actuadores, estos trabajan por parejas, es decir 1 sensor con 1 actuador.

Estos procesos independientes entre sí, serán controlados cada uno por un microcontrolador diferente, con una arquitectura particular para cada caso. El proceso 1 será controlado por un algoritmo difuso, mientras los procesos restantes realizarán estrategias de control on/off que se implementarán en las arquitecturas correspondientes [ALV95].

4.3 EL CONTROL DE LOS PROCESOS

El control de los procesos se hará de una forma distribuida en los diferentes microcontroladores que se usarán para cada proceso, la finalidad de ello, es modular la aplicación, permitiendo trabajar aisladamente cada proceso, y dando la flexibilidad de modificar totalmente la estrategia de control, o bien cambiar de arquitectura, sin que esto afecte a la aplicación de software que se encargará del monitoreo y control central.

Las técnicas de control emplearán algoritmos de control difusos y control on/off. Los algoritmos de control difusos están compuestos por una serie de reglas de inferencia, las cuales, se establecen de acuerdo al conocimiento del problema y la experiencia previa para resolverlo. En general este tipo de sistemas de control difuso resultan estables, y esta estabilidad es proporcionada por cada una de las reglas de inferencia; cada regla brinda una estabilidad parcial al sistema, dando como resultado que todas las reglas de un sistema de control difuso deberán ser estables para lograr la estabilidad del sistema [ALV95].

4.4 ARQUITECTURAS DE LOS CONTROLADORES

Cada controlador está constituido por un chip de características propias, en este caso microcontroladores que se encargaran de los algoritmos de control. Los microcontroladores a utilizar son el Motorola MC68HC11 y el Intel 8031 [PRI95] [DIA95].

Para el proceso 1, la arquitectura está conformada por dos microcontroladores MC68HC11 que trabajan en paralelo [PRI95], atendiendo cada uno a una variable de entrada provenientes de los sensores de posición, uno para la posición X, y otro para lo posición Y, estos microcontroladores se encargan del proceso de fuzzificación, entregando los datos correspondientes del proceso al microcontrolador 8031, que se encarga del proceso de defuzzificación (fig. 4.1).

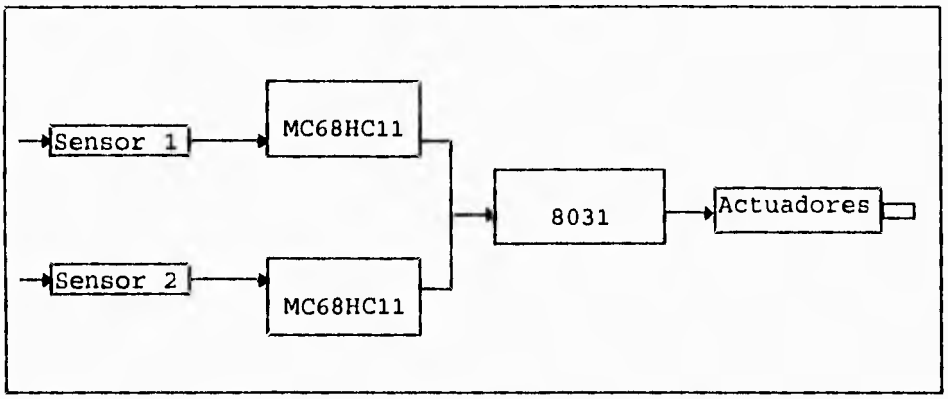


Figura 4.1 Arquitectura del proceso 1

Para el proceso 2 se utiliza un microcontrolador 8031, cuyo algoritmo será un control on/off, de tal forma que solo procesa señales binarias provenientes de los sensores, y así mismo la salida serán señales binaria (fig.4.2).

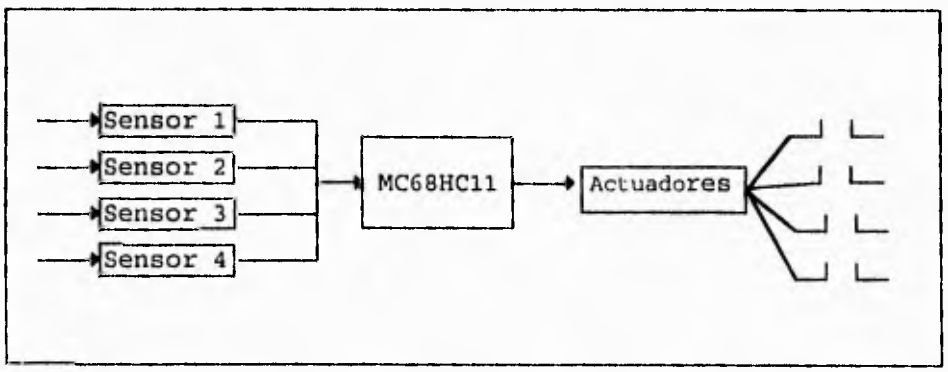


Figura 4.2 Arquitectura del proceso 2

Para los procesos 3 y 4, al igual que el proceso anterior se manejaran algoritmos de control on/off, para ello se utilizarán microcontroladores MC68HC11, ya que este cuenta con un convertidor a/d integrado en el mismo chip, permitiendo procesar las señales analógicas provenientes de los sensores, teniendo a la salida un puerto que activará algún tipo de actuador mediante una etapa de potencia [DIA95] (fig. 4.3).

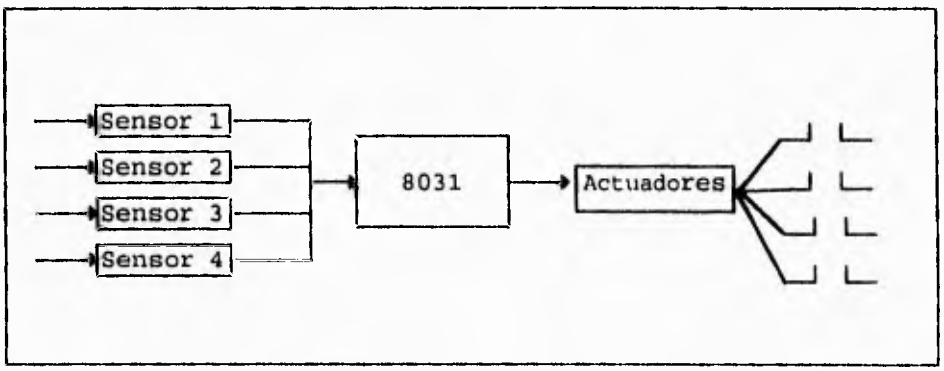


Figura 4.3 Arquitectura de los procesos 3 y 4

4.5 EL CONTROL CENTRAL

El control central estará implementado en una computadora personal que trabajará con un sistema operativo de tiempo real, de tal forma que el software desarrollado se dedique a monitorear los procesos, y modificar el estado de los controladores y/o actuadores del sistema, obedeciendo a una estrategia de control que trabaje de forma automática o atendiendo a peticiones del usuario.

En la computadora tendremos de forma gráfica y numérica el estado de los procesos, teniendo la posibilidad de interactuar con los procesos por medio de

comandos que el usuario seleccionará. Los comandos nos permitirán activar o desactivar los algoritmos de control de cada uno de los controladores, o bien indicarles el cambio de estado de alguno de los actuadores que manejan. En esta primera etapa del proyecto, se propone un control on/off con una estrategia muy simple para el control automático, sin embargo, las posibilidades de control son infinitas, ya que la arquitectura del sistema proporciona los valores de los sensores en todo momento y la posibilidad de interactuar con los actuadores del sistema.

4.6 LAS COMUNICACIONES

Para la comunicación de los controladores de los diferentes procesos con la computadora se propone utilizar una comunicación serial, con las siguientes características: una velocidad de transmisión de 9600 bauds, con un paquete de datos de 8 bits, manejando un bit de inicio y paro. Todos los procesos estarán conectados al puerto serial de la computadora, de tal forma que la computadora manejará ciertos códigos para comunicarse con los microcontroladores, ya sea para pedirles información, o bien para darles alguna instrucción específica.

La computadora se dedicará a sensar el estado de los procesos con tiempos de muestreo determinados por la frecuencia natural de cada proceso, o bien por la frecuencia de muestreo crítica. Para pasar la información del estado de los procesos no se utilizará ninguna forma de encriptado de datos, sin embargo para una aplicación de tipo industrial que maneje productos comerciales se pueden usar protocolos de comunicación que incluyan tareas mas sofisticadas.

Para evitar colisiones de mensajes, la computadora será la responsable de controlar el tráfico de éstos, para ello se deberán observar las siguientes reglas:

- Los procesos no pueden mandar ninguna información sin que les sea solicitada por la computadora.
- La computadora mandará un código de acción por el puerto serial de tal forma que todos los microcontroladores lo reciban.
- Los microcontroladores al momento de recibir el código de acción, verificarán si les corresponde, en caso de no ser así, continuarán con su labor.

- La computadora manejará códigos de acción de dos tipos: el primero solicitará la información del estado actual del proceso, y el segundo tipo, se referirá a comandos que se les den a los microcontroladores para activar o desactivar su algoritmo de control, o bien, modificar el estado de sus respectivos actuadores.

- Al mandar un código de acción la computadora para solicitar información esperará un tiempo razonable a que los datos le sean pasados, si no recibiera ningún dato, asume que el controlador está desconectado o bien existe algún error en él, de manera tal que continuará trabajando con los siguientes procesos.

En lo que se refiere a la recepción de información por puerto serial en la computadora, se debe multiplexar las señales provenientes de los controladores como lo muestra la figura 4.4. El encargado de multiplexar la señal será la computadora, utilizando para ello el puerto paralelo direccionando el circuito multiplexor conectado a él. De esta forma, cada vez que la computadora mande un comando a alguno de los controladores, deberá antes dejar habilitado el canal de comunicación respectivo para recibir la información que el controlador le envíe.

Hemos presentado los aspectos generales del proyecto global, en el que funcionará nuestra aplicación de software que es el objetivo principal de este trabajo, para detalles específicos de la arquitectura presentada en este capítulo se pueden consultar [PRI95] [DIA95], y en lo que se refiere a los algoritmos usados en las arquitecturas presentadas [ALV95].

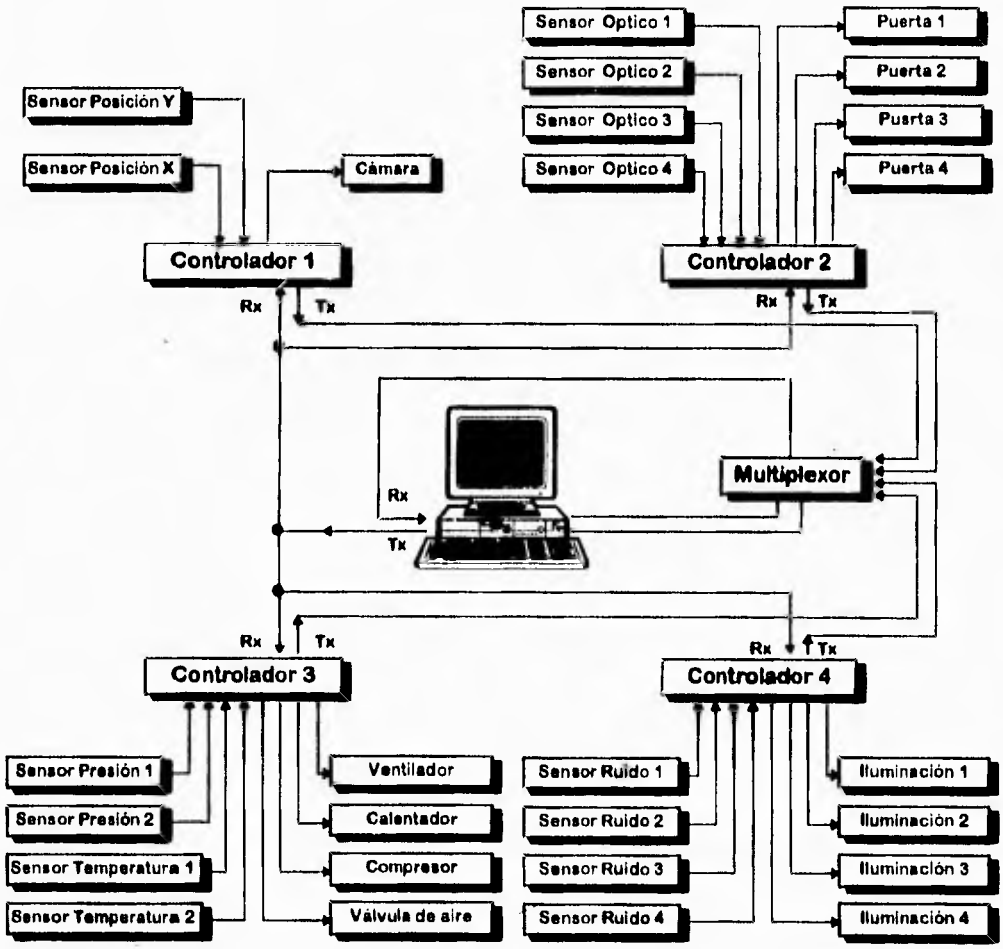


Figura 4.4 Arquitectura del sistema

CAPITULO 5

DISEÑO E IMPLEMENTACIÓN DE LA APLICACIÓN DE MONITOREO Y CONTROL EN TIEMPO REAL

Las aplicaciones en tiempo real como se ha mencionado deben integrar varias partes, es por ello que una vez que se han definido las partes que interactuarán con el software, tales como dispositivos de comunicación, la parte electrónica inherente a los procesos que se controlarán, se inicia la etapa de análisis, diseño e implementación de la aplicación de software.

5.1 ANÁLISIS DE LA APLICACIÓN DE SOFTWARE

Para describir nuestra aplicación de software utilizaremos las siglas SMCTR (Sistema de Monitoreo y Control en Tiempo Real), el cuál es el tema central del presente trabajo.

El análisis del sistema lo podemos definir como la organización de la información colectada durante la fase de investigación de la factibilidad del sistema. En general, el snálisis depende del sistema que se este realizando, ya que abarca los objetivos propios y características del mismo. En la etapa de análisis se pretende lograr tener un modelo lógico que involucre las funciones del sistema, tomando en cuenta los requerimientos del mismo.

Para nuestro SMCTR se utilizaron varias herramientas, optándose por una Metodología de Análisis y Diseño Estructurado de Sistemas SSADM (Structured System Analysis and Design Methodology), como auxiliar en el análisis y para estructurar el diseño, por su simplicidad de uso y popularidad. Así mismo, se utilizó el concepto cliente/servidor para establecer el funcionamiento de los procesos concurrentes de la aplicación, y se establece una estrategia propia para la comunicación de mensajes entre los procesos.

El objetivo que se pretende alcanzar con la Metodología de Análisis y Diseño Estructurado, es lograr una modularidad del sistema, que permita su desarrollo y mantenimiento, dando flexibilidad para cambios o mejoras posteriores.

La modularidad la obtendremos asignando tareas específicas a procesos que trabajen en un ambiente multitarea, sobre la plataforma de software y las facilidades que nos brinda el sistema operativo de tiempo real, como se discutió en los primeros capítulos, cuyo objetivo será poder atender a los controladores de los procesos físicos que tenemos en nuestro sistema. Una vez realizada la modularidad, se define a partir del funcionamiento de cada procesos su modo de operación, ya sea como un proceso cliente o como un proceso servidor.

A lo largo de este capítulo se presenta el análisis, así como el diseño de la aplicación de nuestro SMCTR.

5.1.1 OBJETIVOS

Los objetivos para la aplicación de software son los siguientes:

- Implementar un sistema que monitoree el estado de los controladores, verificando que el rango de las variables muestreadas por los sensores específicos de cada controlador se encuentren dentro de los límites especificados.

- Controlar de forma central y automática, el funcionamiento de todo el sistema a través de un control que pueda involucrar la interacción directa con los controladores del sistema, modificando el estado de los actuadores al igual que activar o desactivar los algoritmos de control correspondientes. Para esta etapa del proyecto el control será on/off, activando o desactivando los algoritmos de control de los controladores. Sin embargo, para etapas posteriores se pueden implementar algoritmos de control más complejos ya que

la arquitectura permite modificar el estado de los actuadores del sistema a través de la computadora, y por su funcionalidad se dispone del valor de los sensores en todo momento.

- Establecer una interface gráfica con el usuario, que permita visualizar de forma inmediata las condiciones de todos los procesos físicos involucrados, que reporte de forma numérica el estado de los sensores del sistema, así como desplegar alarmas visuales cuando se detecte que las variables monitoreadas se salen del rango.

- Permitir al usuario mandar comandos establecidos a los controladores de los procesos físicos, alterando con ello el modo de operación de los mismos, así como cambiar el estado de los actuadores del sistema.

5.1.2 ANÁLISIS DE REQUERIMIENTOS

En la actualidad existen varias plataformas de desarrollo para aplicaciones en tiempo real que permiten el control de procesos industriales, atendiendo a las necesidades de este tipo de aplicaciones. La principal plataforma de hardware para este tipo de herramientas son estaciones de trabajo con un alto desempeño en la graficación y el procesamiento, sin embargo, estos equipos por naturaleza son costosos, así como el software de los mismos. Se requiere de una plataforma más económica ya que las necesidades de procesamiento y los tiempos de respuesta de nuestro proyecto no son estrictos, por lo que estaríamos hablando de un tiempo real blando, para ello se necesita una configuración mínima que cubra con las necesidades dictadas por la multitarea inherente a estas aplicaciones, así como un sistema operativo que brinde las facilidades de comunicación entre procesos, rápido cambio de contexto, un manejo eficiente de interrupciones, como se discutió en el Capítulo 2.

La selección del sistema operativo se hizo en base al cumplimiento de las necesidades discutidas en los primeros capítulos, conjuntamente buscando una

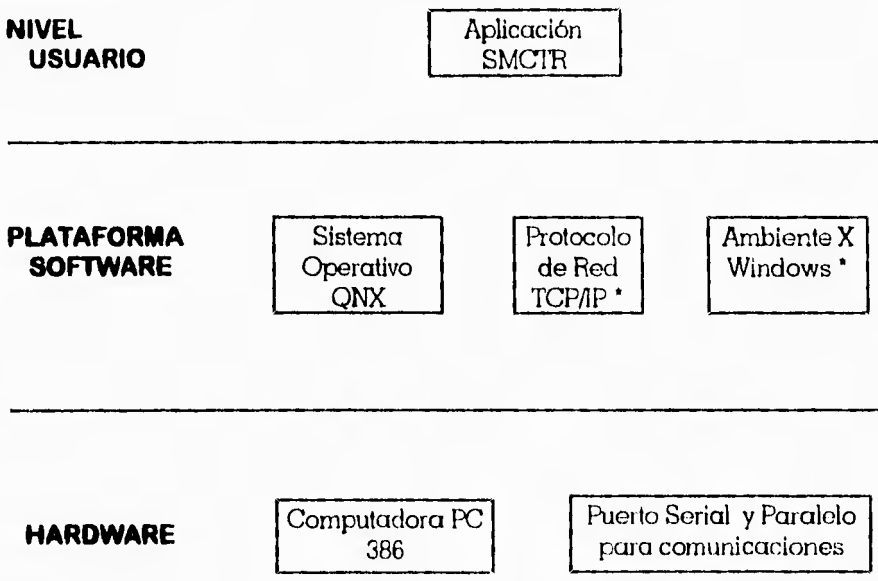
opción que pudiera correrse en una plataforma económica, para ello se optó por el Sistema Operativo QNX 4.21, cuyas principales características ya fueron expuestas en el Capítulo 2. Este sistema operativo es muy parecido a UNIX, con las funciones necesarias para el manejo de aplicaciones en Tiempo Real, con la gran ventaja es que puede correrse en computadoras PC.

Para la plataforma de QNX, existen módulos para GUIs (Graphics User Interface) como X-Windows, así como el protocolo de red TCP/IP. Sin embargo en la etapa actual de nuestro proyecto no se manejan estos módulos.

Como plataforma de hardware se utilizó una computadora PC cuya configuración es la siguiente:

Procesador 80386
Memoria RAM 4 MB
25 Mhz de velocidad de procesamiento
Disco duro de 80 MB
Monitor VGA
1 puerto serial dedicado a las comunicaciones
1 puerto paralelo dedicado a las comunicaciones

Como plataforma de desarrollo el Sistema Operativo QNX cuenta con su propio compilador de *lenguaje C*, que incluye las librerías necesarias para las funciones en tiempo real que maneja este sistema operativo. Esquemáticamente la arquitectura de software de nuestro SMCTR se muestra en la figura 5.1.



*** Módulos no usados en esta etapa del proyecto**

Figura 5.1 Arquitectura de Software

5.1.3 DEFINICIÓN LÓGICA

En la figura 5.2 podemos apreciar un diagrama que describe en forma esquemática los bloques del SMCTR. Como se puede observar nuestro SMCTR contiene 4 funciones básicas.

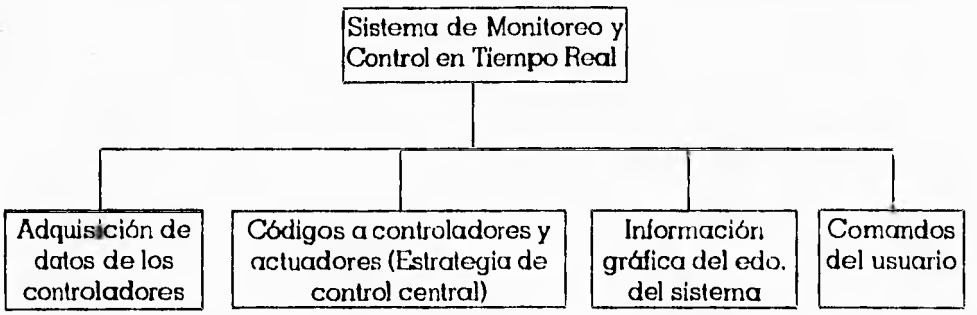


Figura 5.2 Funciones del SMCTR

1. Adquisición de datos (Monitoreo). **El sistema se encarga de solicitar los datos de los controladores de los procesos físicos, monitoreando con ello el estado actual del sistema, tanto de los sensores, como de los actuadores involucrados.**

2. Envío de códigos a los controladores y actuadores (Estrategia de control central). **El sistema enviará códigos a los microcontroladores para que tomen acciones específicas, basándose en una estrategia de control central especificada. Por medio de estos códigos se activa o desactiva el algoritmo de control de cada controlador, o bien se modifica el estado de los actuadores del sistema.**

3. Información gráfica del estado del sistema (Interface con el usuario). **El sistema muestra de forma gráfica y numérica el estado del sistema, incluyendo todos los sensores y actuadores involucrados en el mismo.**

4. Comandos del usuario. **El usuario podrá tener control de los controladores del sistema, activando o desactivando los algoritmos de control de los controladores, así como alterar el estado de los actuadores del sistema.**

El funcionamiento del SMCTR se muestra el esquema de la figura 5.3.

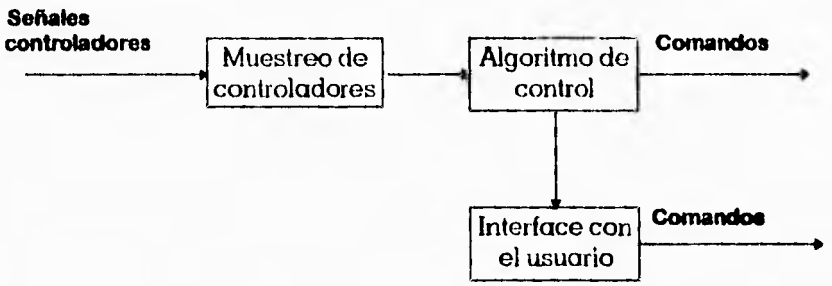


Figura 5.3 Funcionamiento del SMCTR

5.1.4 DIAGRAMAS DE FLUJO DE TRANSACCIONES

Dentro de todo sistema de cómputo, lo más importante del mismo son los datos y la información que pueda generarse con estos. Para entender el flujo de transacciones que tenemos en nuestro sistema, ejemplificaremos para un sólo controlador de un proceso físico, considerando que los demás controladores se comportan de una forma similar.

Debido a las características del sistema podemos identificar básicamente dos tipos de transacciones como lo muestra la figura 5.4, la primera se origina por el monitoreo del estado del sistema, siendo una transacción manejada totalmente por el sistema de forma automática y que funciona en todo momento, para ello intervienen la PC y el controlador del proceso físico en cuestión. El segundo tipo de

transacciones son el resultado de la interacción del usuario con los procesos físicos del sistema a través de la PC, para ello el usuario manda al controlador códigos de acción específicos.

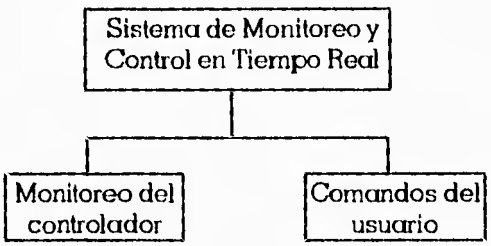


Figura 5.4 Transacciones del SMCTR

Transacción: Monitoreo de las condiciones del sistema

El sistema debe tener noción del estado del sistema en todo momento, y de acuerdo a la estrategia de control establecida, determinar los códigos de acción que mande a los controladores de los procesos físicos. Así mismo, deberá activar alarmas visuales cuando las condiciones del sistema así lo determinen.

En primera instancia, la PC solicita información al controlador del proceso físico, este muestreo deberá coincidir con la frecuencia natural del controlador. El controlador contesta a la llamada de la PC mandándole los datos correspondientes al estado actual de sus sensores y actuadores. La PC compara los datos muestreados con los de la muestra anterior, en caso de ser diferentes se aplica el algoritmo de control de acuerdo con la estrategia de control central establecida y determina si es necesario activar alguna alarma y/o mandar algún código de

acción a los controladores del sistema (Figura 5.5). En la figura se hace referencia a uno solo de los controladores de los procesos físicos, ya que la interacción con el resto de ellos es exactamente idéntica.

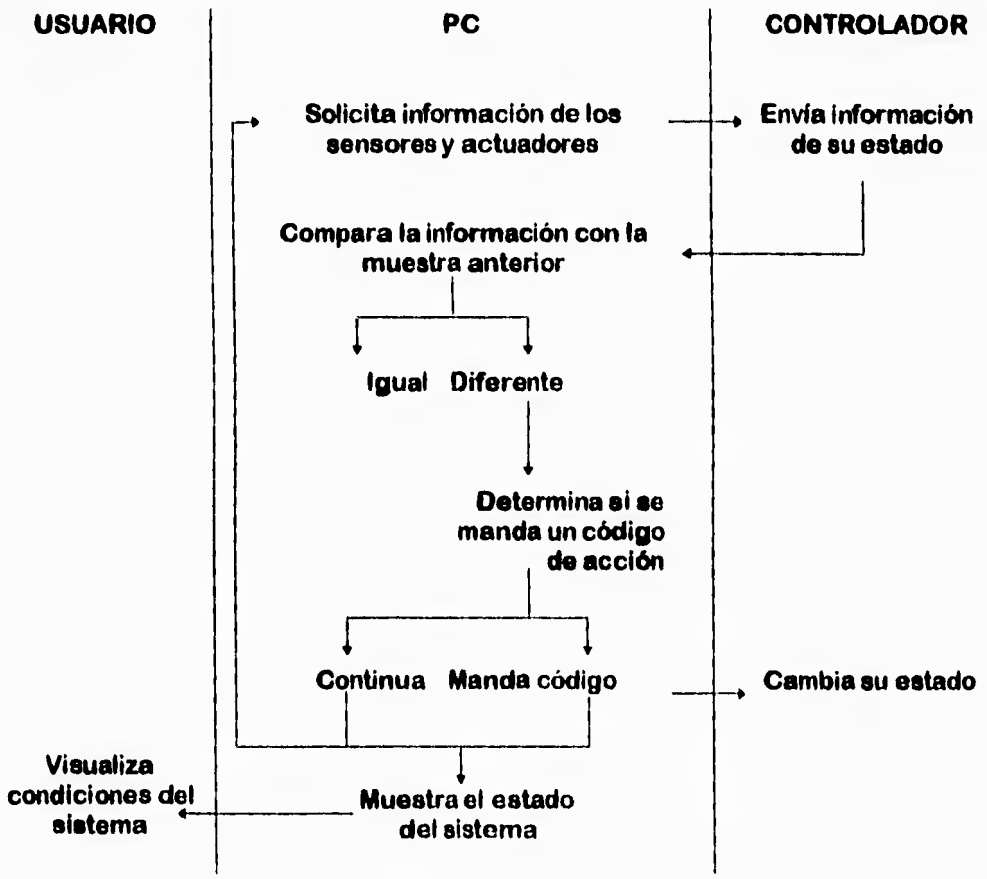


Figura 5.5 Transacción: Monitoreo de las condiciones del sistema

Transacción: Comandos especificados por el usuario.

El usuario puede especificar que se tomen acciones sobre el estado de los controladores, activando o desactivando los algoritmos de control distribuidos en ellos; o bien, alterando el estado de los actuadores del sistema. Para ello, el usuario selecciona un comando de un menú establecido. La transacción de esta solicitud se hace como se muestra en la figura 5.6.

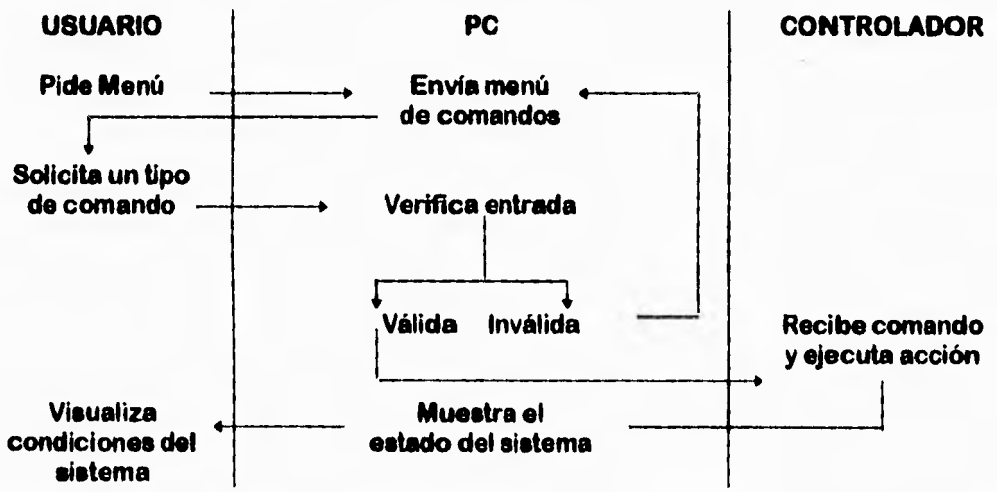


Figura 5.6 Transacción: Comandos especificados por el usuario

5.2 DISEÑO E IMPLEMENTACIÓN DE LA APLICACIÓN DE SOFTWARE

El diseño puede verse como el proceso de aplicar ciertas técnicas o criterios, cuyo objetivo será definir un proceso o sistema para permitir su realización física. En el diseño de software existen diversas metodologías y como mencionamos anteriormente nos auxiliamos de un diseño estructurado en la elaboración de nuestro SMCTR, por lo cual iniciaremos con los diagramas de flujo.

5.2.1 DIAGRAMAS DE FLUJO

Para la elaboración de los diagramas de flujo de datos, es necesario iniciar en un nivel genérico e ir detallando posteriormente los módulos en subniveles superiores, tantos como lo determine la complejidad de los mismos.

Primer Nivel

En el primer nivel tenemos la operación general del sistema (figura 5.6), en donde se muestra la interacción entre el usuario, el sistema y los controladores de los procesos físicos. En la figura aparece un sólo controlador, ya que la interacción con el resto de los controladores es la misma. Como se mencionó en el capítulo anterior, el sistema interactuará con los controladores del sistema a través de un algoritmo de control central, por lo que enviará códigos de acción a estos cuando así se requiera. Al usuario se le permite intervenir sobre el estado de los controladores y actuadores a través de comandos establecidos.

El SMCTR se encarga de monitorear de acuerdo a la frecuencia natural de los controladores el estado de los mismos, aplicando para ello un algoritmo de control y mandando como resultado un código de acción al controlador, el usuario tiene la posibilidad de ejecutar comandos, y se le presenta en todo momento una panorámica del estado actual del sistema.

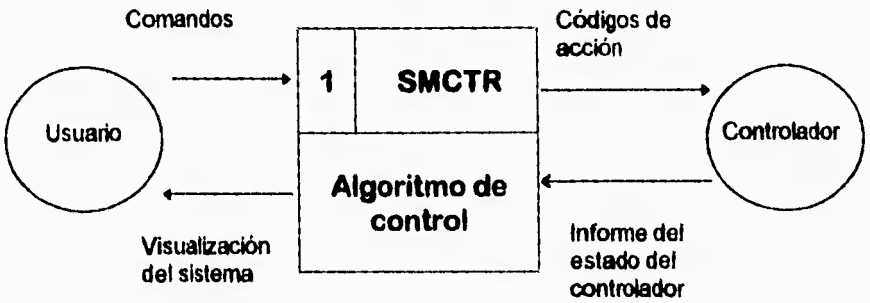


Figura 5.6 Diagrama de Flujo de Primer Nivel

Segundo Nivel

Los 4 módulos principales del SMCTR son:

1. **Interface gráfica del usuario**
2. **Controlador**
3. **Monitoreo**
4. **Comunicaciones**

La interface gráfica, es la encargada de establecer contacto con el usuario, a través de ella, el usuario solicitará la ejecución de ciertos comandos para los controladores de los procesos físicos, con estos comandos se activa o desactiva el algoritmo de control en cada uno de los controladores, o bien, se altera el estado de los actuadores. La interface gráfica despliega la información del estado actual del sistema en todo momento.

El controlador, es el encargado del control central del sistema. De acuerdo con la estrategia establecida, este activa las alarmas visuales para el usuario, así como enviará comandos de acción a los controladores del sistema, activando o

desactivando el algoritmo de control de cada controlador, o alterando el estado de los actuadores del sistema.

El monitoreo, solicita la información del estado de los sensores y actuadores de los respectivos controladores, de acuerdo a las frecuencias de muestreo establecidas (que corresponden a la frecuencia con la que operan los controladores). Así mismo, es el encargado de notificar de los cambios en los sensores y/o actuadores a la interface gráfica y al controlador.

La parte de comunicaciones establece y realiza la transacción de códigos a los controladores del sistema por medio del puerto serial. Así mismo, es el encargado de recolectar los datos generados con los códigos mandados. A este módulo llegarán las peticiones tanto del módulo de monitoreo como del controlador, para establecer contacto con los procesos físicos del sistema. (figura 5.7).

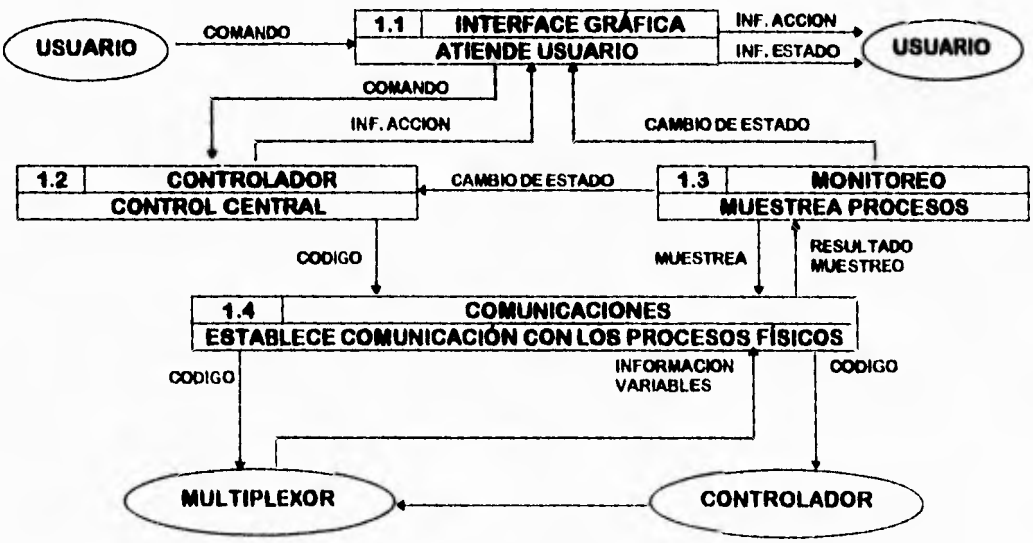


Figura 5.7 Diagrama de Flujo de Segundo Nivel

Tercer Nivel

Interface gráfica del usuario

La interface gráfica del usuario contiene los siguientes módulos:

1. **Indicador de Eventos**
2. **Manejador de peticiones del usuario**
3. **Indicador del estado del sistema**

El indicador de eventos, notifica del cambio de estado en alguno de los sensores y/o actuadores, así como de las acciones que hayan sido tomadas por el controlador. También indica el resultado de la ejecución de las peticiones del usuario.

El manejador de peticiones del usuario, valida los comandos del usuario y se encarga de emitir su ejecución a través del módulo controlador.

El indicador del estado del sistema, muestra de forma gráfica y numérica el estado actual de las condiciones del sistema.(Figura 5.8)

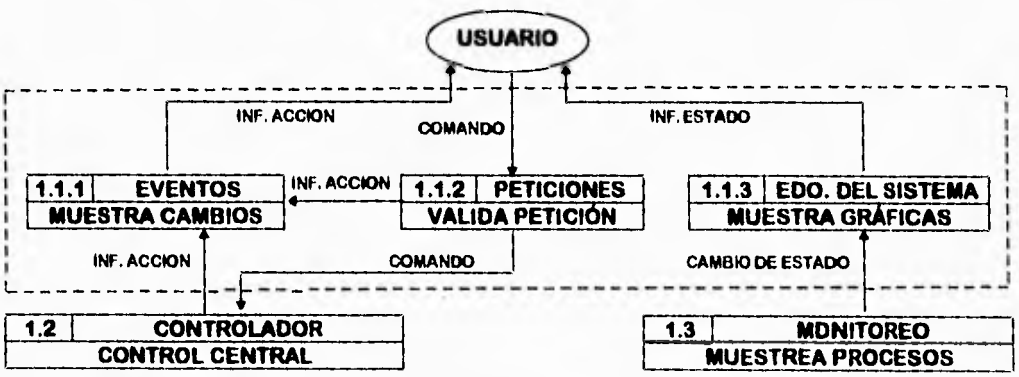


Figura 5.8 Diagrama de Flujo de Tercer Nivel : Interface gráfica con el usuario

El controlador

El controlador maneja los siguientes módulos:

1. **Algoritmo de control central**
2. **Validación de códigos**

El algoritmo de control central, se encarga de seleccionar los códigos que enviará a los controladores del sistema, para que activen o desactiven el algoritmo de control correspondiente o alteren del estado de los actuadores que manejan. Todo lo anterior obedece a la estrategia de control implementada.

La validación de códigos, verifica si es posible que los actuadores y/o controladores ejecuten dicho código considerando la estrategia implementada.

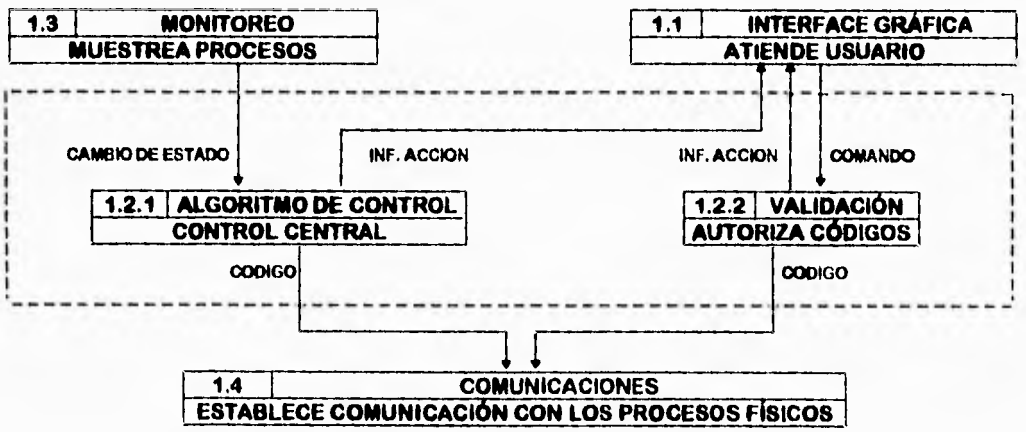


Figura 5.9 Diagrama de Flujo de Tercer Nivel: El controlador

El monitoreo

El monitoreo dedica un módulo a cada proceso físico, aquí explicaremos la descomposición de cada uno de estos módulos, refiriéndonos sólo a uno, ya que la del resto es exactamente igual.

1. Timer
2. Receptor de información
3. Indicador de cambio de estado

El timer es el encargado de notificar que ha transcurrido el tiempo del período de muestreo para solicitar la siguiente muestra.

El receptor de información, es el encargado de solicitar al módulo de comunicaciones se le envíe la información correspondiente del respectivo proceso físico.

El indicador de cambio de estado, compara la información actual con la de la muestra anterior, en caso de existir variación manda la notificación pertinente a los módulos de control y de interface gráfica. (figura 5.10)

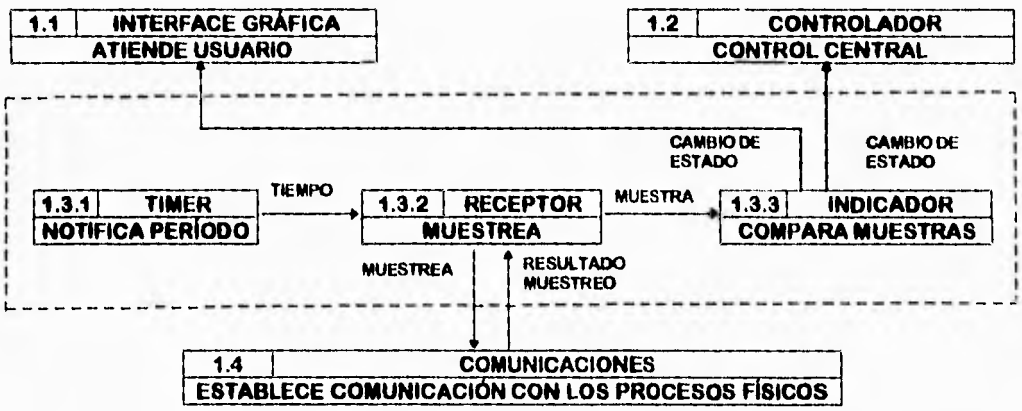


Figura 5.10 Diagrama de Flujo de Tercer Nivel: El monitoreo

Las comunicaciones

Este módulo lo podemos descomponer en los siguientes:

1. **Receptor de petición**
2. **Transmisor y receptor del puerto serial**
3. **Transmisor del puerto paralelo**
4. **Notificador de petición**

El receptor de petición, es el encargado de formar el paquete de bytes a transmitir por medio del puerto serial.

El transmisor y receptor del puerto serial, se encarga del envío y recepción de mensajes con los controladores.

El transmisor del puerto paralelo, se encarga de direccionar el multiplexor conectado al puerto paralelo, para la recepción de los mensajes de los controladores a través del puerto serie (figura 4.4).

Finalmente el notificador de la petición, informa al módulo generador de la petición de muestreo si la operación fue exitosa o existió falla, mandándole los datos recolectados (fig. 5.11).

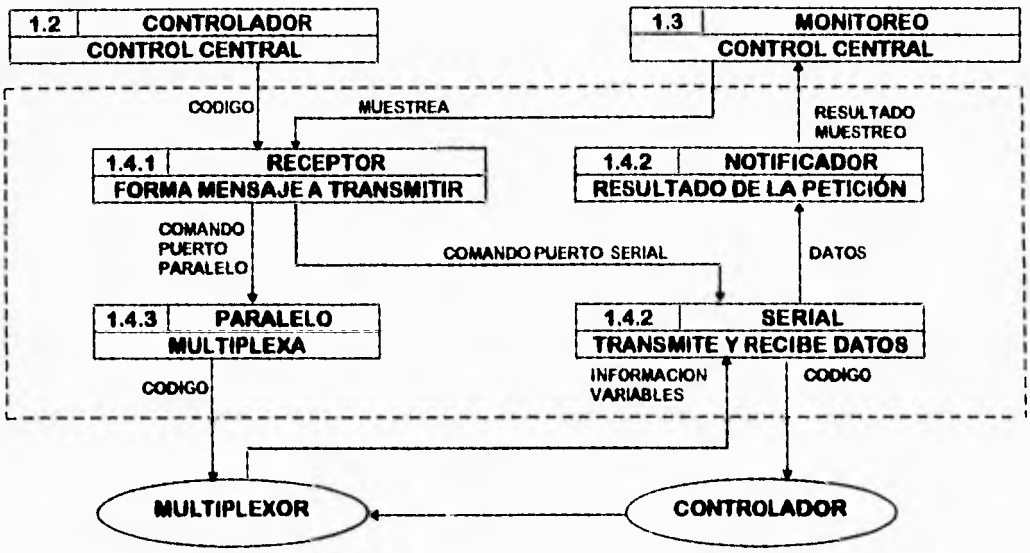


Figura 5.11 Diagrama de Flujo de Tercer Nivel: Las comunicaciones

5.2.2 DESCOMPOSICIÓN EN PROCESOS

Después del diseño preliminar del sistema se eligió la siguiente descomposición de procesos concurrentes en nuestro sistema, cada uno con sus respectivos submódulos que obedecen a la descripción de las páginas anteriores:

1. **Interface Gráfica**
2. **Monitoreo Proceso 1**
3. **Monitoreo Proceso 2**
4. **Monitoreo Proceso 3**
5. **Monitoreo Proceso 4**
6. **Controlador**
7. **Servidor de Comunicaciones**

Se optó por la descomposición del 2do nivel para formar los procesos concurrentes, cada uno de estos procesos contará con sus módulos generados por la descomposición del 3er nivel, pero estos serán programados de forma estructurada, a través de funciones y procedimientos; mientras que los módulos del 2do nivel convertidos en procesos concurrentes necesitarán comunicarse a través del IPC (Interprocess communication) suministrado por el sistema operativo.

A continuación mostramos las funciones de cada uno de estos procesos, así como describimos el proceso de comunicación entre procesos empleados, tanto a nivel interno con la PC como a nivel externo con los controladores de los procesos físicos. Para la implementación de estos procesos nos auxiliamos de un esquema cliente/servidor (Capítulo 3).

5.2.2.1 EL SERVIDOR DE COMUNICACIONES

El servidor se encarga de comunicarse por medio del puerto serial con los controladores, así como es el encargado de direccionar el mulitplexor conectado al puerto paralelo, con la finalidad de recolectar la información de un controlador específico. Éste es un servidor dedicado a nuestra aplicación específica, tiene un mecanismo de mensajes tal, que los procesos solicitan información del estado actual de un controlador en particular. El servidor se encarga de mandar un código a los controladores y espera a que responda el controlador involucrado, una vez que ha recolectado los datos, se los envía al proceso cliente que los ha solicitado.

El tipo de servicios que proporciona el servidor es de mandar códigos a los controladores de los procesos físicos, dependiendo del tipo de código, recolectará información del microcontrolador correspondiente, o bien el código de respuesta de ejecución de alguna acción.

El servidor trabaja con un mecanismo de mensajes orientado a transacciones, es decir, cada petición que recibe el servidor se considera como única, se atiende y se desecha. Esto implica que en el momento que el cliente llama al proceso servidor debe proporcionársele toda la información necesaria para atender a su petición en una sola llamada, y si el cliente muere el servidor no se

afecta. La forma en que trabaja este tipo de servidores se ilustra con el siguiente fragmento de código en C:

```
for (;;) {

    pid = Receive (0,&msg,sizeof msg);   /* Recibe mensajes de cualquier proceso */

    switch(msg.tipo)
    {
     case 1:
      .......
     break;
     case 2:
     ........
     break;
     default:
      ......
    }
     Reply(pid,&msg,replySize); /* Responde al mensaje */
}
```

De esta forma el servidor esperará un mensaje, y en cuanto lo tiene responde a él, de acuerdo al tipo de mensaje. La justificación de un servidor para nuestra aplicación se basa en evitar que dos o mas procesos deseen mandar un mensaje por el puerto serial al mismo tiempo, de esta forma solo el servidor atenderá a un proceso a la vez, y en estas condiciones, se evitarán colisiones de mensajes, ya que el servidor tiene control absoluto sobre el puerto serial.

Los mensajes que recibe el servidor son manejados a través del sistema operativo por prioridad, y la prioridad con la que corre el servidor se ajusta a la prioridad del proceso que este atendiendo.

5.2.2.2 LOS PROCESOS CLIENTES

Los procesos clientes del servidor de comunicaciones serán el resto de los procesos involucrados en nuestro SMCTR, exceptuando el proceso encargado de la interface gráfica. Los clientes por medio del sistema de comunicación de procesos suministrado por el sistema operativo, solicitan al servidor dos tipos de servicios:

- Los datos relacionados con un proceso físico en particular

- La ejecución de un código de acción

Para este tipo de mensajes los procesos utilizarán la función Send, con la cuál el proceso se bloquea en espera de la respuesta del servidor.

5.2.2.3 LOS PROCESOS ENCARGADOS DEL MONITOREO

El funcionamiento para los cuatro procesos encargados del monitoreo es idéntico. El proceso activa un timer que se encarga de notificarle a través del sistema de comunicación entre procesos que el tiempo especificado ha transcurrido. Este tiempo se ajusta al período de muestreo requerido por el sistema.

El proceso se encarga de checar constantemente la nueva muestra con la anterior, si encuentra variación notifica al proceso encargado de la interface gráfica el cambio ocurrido, también checa si las variables se encuentran dentro de un rango admisible, notificando al controlador los cambios generados en la muestra y su evaluación sobre el rango de los mismos.

El proceso así solo recibe dos tipos de mensajes, uno proveniente del timer, y el otro proveniente ya sea del proceso encargado de la interface gráfica o del controlador, estos mensajes son recibidos por prioridades, y el proceso se ajusta a la prioridad del proceso que este atendiendo.

5.2.2.4 EL PROCESO CONTROLADOR

El proceso controlador se encarga de requerir al servidor de comunicaciones, la ejecución de códigos de acción sobre los controladores físicos del sistema. Para ello, el proceso controlador cuenta con un algoritmo de control central que obedece a una estrategia establecida, y con el que se determinan los códigos de acción a solicitar. Así mismo, es el proceso controlador quien se encargará de activar las alarmas al usuario, cuando detecte que las variables sensadas se salen de los rangos establecidos.

El proceso controlador también se encarga de autorizar las peticiones de comandos del usuario, obedeciendo a la estrategia de control establecida. En este caso, el proceso encargado de la interface gráfica con el usuario, solicita la ejecución de cierto comando para activar o desactivar el algoritmo de control de uno de los controladores, o bien, alterar el estado de los actuadores.

Para evitar una sobrecarga en nuestro SMCTR se opta por sólo ejecutar el algoritmo de control central cuando así sea necesario, es decir, cuando existan modificaciones en las variables de entrada del mismo, que son el estado de los sensores de todo el sistema.

5.2.3 ASIGNACIÓN DE PRIORIDADES A LOS PROCESOS

La asignación de las prioridades a los procesos del sistema debe hacerse de forma tal, que el funcionamiento del sistema obedezca a las necesidades del mismo, así como cubrir los objetivos para los cuales el sistema fue diseñado. Revisando esto encontramos que las dos propósitos fundamentales de la aplicación son: el controlar los procesos externos a través del algoritmo de control previamente definido, y el segundo es la interface con el usuario, cuyo propósito es informar al usuario en todo momento del estado del sistema (la función de monitoreo), así como atender a sus comandos.

Cabe hacer notar que los comandos del usuario como se describió en las páginas anteriores tienen que ser validados, y esta validación se hará a través del proceso controlador.

De lo anterior hemos agrupado a los proceso en dos: los encargados del control y monitoreo, y los encargados de la interface usuario. (Tabla 1)

Para esta primer división de procesos hemos determinado que dados los objetivos del SMCTR los procesos encargados de la interface gráfica con el usuario (GUI), deben ser los de menor prioridad, asignándoles así mayor prioridad a los proceso que están relacionados con el control.

PROCESOS ENCARGADOS DEL CONTROL Y MONITOREO	PROCESOS ENCARGADOS DE LA GUI
• Monitoreo Proceso 1 • Monitoreo Proceso 2 • Monitoreo Proceso 3 • Monitoreo Proceso 4 • Controlador • Servidor de comunicaciones	• Interface gráfica

Tabla 1 División de los procesos por función

De la lista de procesos encargados del control, los dividimos en dos grupos los que tienen una prioridad constante, y los que tienen prioridad dinámica. (Tabla 2)

Los procesos de prioridad dinámica cambian su prioridad de acuerdo al proceso que estén manejando, de igual forma que los mensajes les son puestos en fila de espera atendiendo a esta prioridad.

PRIORIDAD DINÁMICA	PRIORIDAD FIJA
• Monitoreo Proceso 1 • Monitoreo Proceso 2 • Monitoreo Proceso 3 • Monitoreo Proceso 4 • Servidor de comunicaciones	• Controlador

Tabla 2 División de los procesos por prioridad fija o dinámica

De la división anterior se procede a dar mayor prioridad al proceso controlador, si bien los procesos de monitoreo suministran la información a nuestra aplicación, también es cierto que una vez detectada una condición

anormal se debe tomar la acción de control pertinente, por lo que el controlador tiene prioridad sobre los procesos encargados del muestreo.

Dentro del grupo de procesos de prioridad dinámica excluímos al proceso servidor de comunicaciones ya que su prioridad dependerá única y exclusivamente del proceso al que esta atendiendo, por lo cual con una buena asignación de prioridades el proceso servidor trabajará de forma adecuada, atendiendo a las peticiones de sus procesos clientes.

Finalmente nos quedan los procesos encargados del muestreo, los cuales por su estructura son mas complejos en su funcionamiento. Para ello procedemos a revisar los estados posibles de nuestro proceso:

- Atendiendo la señal proveniente del timer.
- Atendiendo una solicitud de información por parte de la interface gráfica.
- Atendiendo una solicitud de información por parte del controlador.

En el primer caso el proceso timer lo crea y controla el proceso encargado del monitoreo del proceso físico particular, de tal forma que el proceso Monitoreo esta respondiendo a un evento que el mismo generó, en los dos casos restantes el proceso Monitoreo, atiende a procesos independientes a su estructura sobre los cuales no tiene control ni dominio.

Regresando a la estructura del estado en el que el proceso se encuentra atendiendo a la señal proveniente del timer, el proceso notifica a través de otros procesos proxy si ocurre un cambio en los datos que muestrea, avisando para ello a los proceso de interface gráfica y al proceso controlador. De tal forma que este proceso en particular se encarga de manejar los siguientes procesos tipo proxy:

- Timer
- Aviso a interface gráfica
- Aviso al controlador

En el caso del proceso timer este deberá tener una prioridad menor a la del proceso controlador y mayor a la de la interface gráfica. Y del conjunto de procesos timers tendrá una mayor prioridad aquel cuya frecuencia sea mayor.

De esta forma tenemos la siguiente tabla de asignación de prioridades a los procesos (Tabla 3).

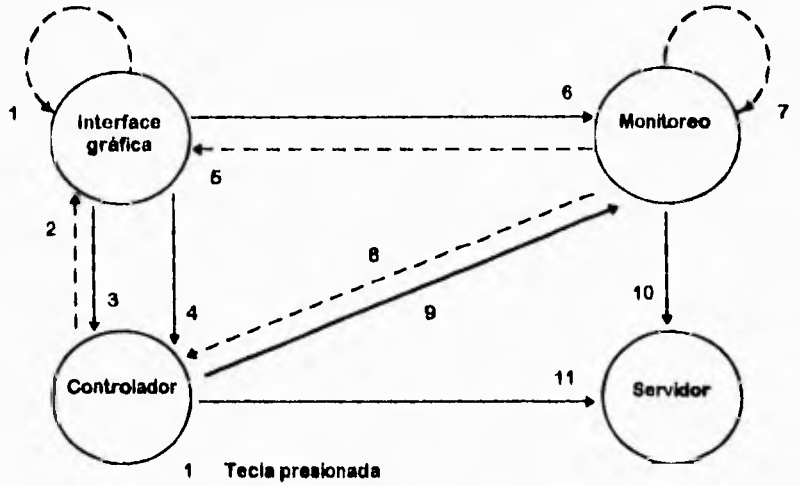
Nombre del proceso	Prioridad
Controlador	Mayor
Timer del Proceso 1	↓
Timer del Proceso 2	
Timer del Proceso 3	
Timer del Proceso 4	
Interface gráfica	Menor
Servidor de comunicaciones	Dinámica
Monitor Proceso 1	Dinámica
Monitor Proceso 2	Dinámica
Monitor Proceso 3	Dinámica
Monitor Proceso 4	Dinámica

Tabla 3 Asignación de prioridades

5.2.4 PLANIFICACIÓN DEL PROCESADOR

La estrategia para la planificación del procesador obedece a las necesidades de los sistemas en tiempo real, para lo cual se optó por el esquema de planificación adaptable descrito en el capítulo 2, el cuál nos brinda con un cuanto de 100 milisegundos una respuesta aceptable para nuestro sistema, considerando la asignación de prioridades previa.

5.2.5 DESCRIPCIÓN DE LOS MENSAJES ENTRE LOS PROCESOS



1 Tecla presionada
2 Acciones tomadas
3 Solicita información sobre acciones tomadas
4 Solicita la ejecución de una petición del usuario
5 Cambio de estado
6 Solicita información sobre cambio de estado
7 Timer
8 Cambio de estado
9 Solicita información cambio de estado
10 Solicita muestreo
11 Solicita acción sobre controladores o actuadores

Mensajes tipo proxy

Mensajes que bloquean al proceso en espera de una respuesta del receptor

Figura 5.12 Mensajes entre procesos

De acuerdo al diagrama de la figura 5.12 se describen a continuación las funciones de los mensajes entre los procesos.

1. Interface gráfica.

Recibe los siguientes mensajes:

- **Información de tecla presionada por el usuario.**
- **Notificación de cambio de estado.**
- **Notificación de eventos de acciones tomadas**

Manda los siguientes mensajes:

- **Solicita información del estado de un proceso monitor**
- **Solicita información de una acción tomada**
- **Solicita la ejecución de una petición del usuario**

Recibe mensajes de los procesos:

- **Monitoreo proceso 1**
- **Monitoreo proceso 2**
- **Monitoreo proceso 3**
- **Monitoreo proceso 4**
- **Controlador**

Manda mensajes a los procesos:

- **Monitoreo proceso 1**
- **Monitoreo proceso 2**
- **Monitoreo proceso 3**
- **Monitoreo proceso 4**
- **Controlador**

Manejo de los mensajes:

El módulo de interface gráfica recibe sólo mensajes tipo proxy, de tal forma, que los procesos que le envían mensajes no esperan a que este proceso les responda. En cuanto tenga oportunidad nuestro proceso de interface gráfica

mandará un mensaje solicitando la información respectiva a cada uno de los otros procesos con los que se comunica, de esta forma, el proceso de la interface gráfica espera a que le manden la información. Y cuando el usuario selecciona una opción para modificar el estado de los controladores de los procesos físicos, solicita al proceso controlador la ejecución de dicha petición.

2. Monitoreo proceso 1,2,3 y 4.

Estos cuatro procesos son independientes ya que cada uno corresponde y se encarga de un proceso físico particular, sin embargo su estructura de funcionamiento es completamente similar.

Recibe los siguientes mensajes:

- Notificación del timer
- Petición de datos del estado del proceso físico que muestrea

Manda los siguientes mensajes:

- Solicita se muestree el proceso físico
- Informa cambio en el estado del proceso físico que muestrea

Recibe mensajes de los procesos:

- Interface gráfica
- Controlador

Manda mensajes a los procesos:

- Interface gráfica
- Servidor de comunicaciones

- **Controlador**

Manejo de los mensajes:

El proceso recibe un mensaje tipo proxy del timer, indicando se ha cumplido el siguiente período de tiempo para la muestra, emitiendo un mensaje al proceso servidor requiriendo el muestreo del proceso físico. Cuando detecta un cambio de estado en las variables que muestrea, notifica a través de proxies a la interface gráfica y al controlador; y finalmente responde a los mensajes de solicitud de información generados por los dos procesos anteriores.

3. Controlador

Recibe los siguientes mensajes:

- **Notificación de cambio en las variables de un proceso físico**
- **Solicitua de la ejecución de una petición del usuario**
- **Solicitud de información sobre acciones tomadas**

Manda los siguientes mensajes:

- **Solicita información del estado de un proceso físico**
- **Notifica de una acción tomada por el algoritmo de control**
- **Solicita el envío de comandos a los controladores y/o actuadores del sistema.**

Recibe mensajes de los procesos:

- **Interface gráfica**
- **Monitoreo proceso 1**
- **Monitoreo proceso 2**
- **Monitoreo proceso 3**

- **Monitoreo proceso 4**

Manda mensajes a los procesos:

- **Interface gráfica**
- **Monitoreo proceso 1**
- **Monitoreo proceso 2**
- **Monitoreo proceso 3**
- **Monitoreo proceso 4**
- **Servidor de comunicaciones**

Manejo de los mensajes:

El proceso recibe notificación de un cambio de estado, ó la solicitud de la ejecución de una petición del usuario, así como de información de una acción tomada por el controlador. En el primer caso, solicita al proceso monitor correspondiente la información de las variables muestreadas, aplica el algoritmo de control central, y solicita al servidor de comunicaciones el envío de algún código a los controladores del sistema, notificando al proceso interface gráfica que se tomó una acción. Para el segundo caso, verifica si es posible ejecutar la petición del usuario, mandando al servidor de comunicaciones los códigos correspondientes. Para el último de los casos, el controlador responde informando la acción tomada.

4. Servidor de comunicaciones

Recibe los siguientes mensajes:

- **Solicita se mande un código a los controladores**
- **Solicita se envíe algún código a los actuadores**

No manda ningún tipo de mensaje, solamente responde a los mensajes que le llegan.

Recibe mensajes de los siguientes procesos:

- **Monitoreo Proceso 1**
- **Monitoreo Proceso 2**
- **Monitoreo Proceso 3**
- **Monitoreo Proceso 4**
- **Controlador**

Manejo de los mensajes:

El servidor de comunicaciones sólo atiende a peticiones de los procesos, por lo que en el momento que le llega un mensaje lo procesa y responde a él.

5.2.6 FORMATO DE LOS MENSAJES ENTRE LOS PROCESOS

A continuación se describen los formatos de los mensajes entre los procesos. Para la elaboración de estos se debe elegir un formato único para todos los mensajes que recibe un proceso, así como definir el formato de los mensajes con los que responde.

Mensajes al proceso de interface gráfica

Los mensajes que son mandados al proceso encargado de la interface gráfica son de tipo proxy (con excepción del mensaje tipo INFO_CONT), y contienen un sólo dato, cuyo tamaño es de un byte en el que se especifica el tipo de mensaje que se le manda.

Tipo
char

Dentro del tipo, se incluye el identificador de la clase de mensaje enviado, utilizando las siguientes claves:

Clave	Identificador	Descripción
07	TECLADO	Tecla presionada del teclado
09	PROCESO1	Cambio en los valores de la muestra proceso 1
10	PROCESO2	Cambio en los valores de la muestra proceso 2
11	PROCESO3	Cambio en los valores de la muestra proceso 3
12	PROCESO4	Cambio en los valores de la muestra proceso 4
16	NOTIFICA_ACCION	Notifica se tomó una acción por el controlador

Mensajes al servidor de comunicaciones

Este tipo de mensajes son enviados al servidor a través de la instrucción Send, y el proceso generador del mensaje espera una respuesta a su mensaje.

El formato de los mensajes enviados al servidor de comunicaciones consta de 2 campos, cada uno con tamaño de 1 byte.

Tipo	Código
char	char

El campo de tipo, tiene los siguientes claves:

Clave	Identificador	Descripción
13	INFO	Solicita información
14	COM_CONT	Manda un comando a un controlador

Mensajes de respuesta del servidor de comunicaciones

Para los mensajes tipo INFO, el proceso responde con el siguiente formato de mensaje

Datos	No. de datos	Error
char [5]	char	char

El primer campo lo constituye un arreglo de 5 bytes, donde contiene los datos muestreados, el segundo campo indica el número de datos que fueron recolectados en la muestra, y el último campo indica si existió un error de acuerdo con las siguientes claves:

Clave	Identificador	Descripción
00	CORRECTO	Los datos fueron leídos correctamente
01	NO_RESP	El controlador no responde
02	INCOMPLETO	No se obtuvo el total de datos solicitados

Para los mensajes tipo COM_CONT, el servidor de comunicaciones responde con el siguiente formato de mensaje:

Código	Error
char	char

El campo que contiene el código regresa el valor pasado por el controlador en respuesta al comando efectuado. Y el campo error puede tomar los siguientes valores:

Clave	Identificador	Descripción
00	CORRECTO	Los datos fueron leídos correctamente
1	NO_RESP	El controlador no responde

Mensajes a los procesos monitoreo

Los mensajes que reciben los procesos monitoreo tienen el siguiente formato:

Tipo
char

Manejando los siguiente códigos para el campo tipo:

Clave	Identificador	Descripción
01	TIMER_PROXY	Indica que debe ser tomada la siguiente muestra
02	INFO_CONT	Solicita información el proceso controlador
03	INFO_GUI	Solicita información el proceso encargado de la interface gráfica

Mensajes de respuesta de los procesos monitoreo

Estos procesos manejan para sus mensajes de respuesta a la solicitud de información por parte del proceso controlador, y del proceso encargado de la interface gráfica, el siguiente formato:

Datos	No. de datos
char [5]	char

Donde se manda un arreglo con los valores muestreados (máximo 5 valores), y el número de datos que se le están mandando como respuesta.

Mensajes al controlador

Los mensajes que recibe el proceso controlador tienen la siguiente estructura:

Tipo	Código
char	char

En donde se utilizan los siguientes códigos para el campo tipo:

Clave	Identificador	Descripción
05	PETICION_USUARIO	Indica que el usuario quiere ejecutar un comando
06	SOLICITA_ACCION	Solicita información de la última acción tomada por el controlador
09	PROCESO1	Cambio de estado en el proceso físico 1
10	PROCESO2	Cambio de estado en el proceso físico 2
11	PROCESO3	Cambio de estado en el proceso físico 3
12	PROCESO4	Cambio de estado en el proceso físico 4

Y para el segundo campo se utiliza los siguientes códigos:

Clave	Identificador	Descripción
3	DP1	Desactiva algoritmo de control del proceso físico 1
4	DP2	Desactiva algoritmo de control del proceso físico 2
7	DP3	Desactiva algoritmo de control del proceso físico 3
8	DP4	Desactiva algoritmo de control del proceso físico 4
9	AP1	Activa algoritmo de control del proceso físico 1
10	AP2	Activa algoritmo de control del proceso físico 2
11	AP3	Activa algoritmo de control del proceso físico 3
12	AP4	Activa algoritmo de control del proceso físico 4
23	C15	Posiciona cámara a 15°
24	C30	Posiciona cámara a 30°
25	C45	Posiciona cámara a 45°

Clave	Identificador	Descripción
26	C60	Posiciona cámara a 60°
27	C75	Posiciona cámara a 75°
28	C90	Posiciona cámara a 90°
29	C105	Posiciona cámara a 105°
30	C120	Posiciona cámara a 120°
31	C135	Posiciona cámara a 135°
32	C150	Posiciona cámara a 150°
33	C165	Posiciona cámara a 165°
34	A21ON	Activa actuador 1 del proceso físico 2
35	A22ON	Activa actuador 2 del proceso físico 2
36	A23ON	Activa actuador 3 del proceso físico 2
37	A24ON	Activa actuador 4 del proceso físico 2
38	A31ON	Activa actuador 1 del proceso físico 3
39	A32ON	Activa actuador 2 del proceso físico 3
40	A33ON	Activa actuador 3 del proceso físico 3
41	A34ON	Activa actuador 4 del proceso físico 3
42	A41ON	Activa actuador 1 del proceso físico 4
43	A42ON	Activa actuador 2 del proceso físico 4
44	A43ON	Activa actuador 3 del proceso físico 4
45	A44ON	Activa actuador 4 del proceso físico 4
46	A21OFF	Desactiva actuador 1 del proceso físico 2
47	A22OFF	Desactiva actuador 2 del proceso físico 2
48	A23OFF	Desactiva actuador 3 del proceso físico 2
49	A24OFF	Desactiva actuador 4 del proceso físico 2
50	A31OFF	Desactiva actuador 1 del proceso físico 3
51	A33OFF	Desactiva actuador 2 del proceso físico 3
52	A32OFF	Desactiva actuador 3 del proceso físico 3
53	A34OFF	Desactiva actuador 4 del proceso físico 3
54	A41OFF	Desactiva actuador 1 del proceso físico 4
55	A42OFF	Desactiva actuador 2 del proceso físico 4
56	A43OFF	Desactiva actuador 3 del proceso físico 4
57	A44OFF	Desactiva actuador 4 del proceso físico 4

Mensajes de respuesta del controlador

El controlador responde a la petición de solicitud de la última acción tomada con el formato que se muestra a continuación:

Código
char

Para lo cual se utilizan los mismos códigos de la tabla anterior, correspondiendo a las acciones tomadas por el algoritmo de control, o bien a la ejecución de alguna petición del usuario. A estos códigos, se le añaden la notificación de casos de alarma de las variables monitoreadas.

Clave	Identificador	Descripción
15	ALARMA1	Avisa alarma en el proceso físico 1
16	ALARMA2	Avisa alarma en el proceso físico 2
17	ALARMA3	Avisa alarma en el proceso físico 3
18	ALARMA4	Avisa alarma en el proceso físico 4
19	NORMAL1	Volvió a operación normal el proceso físico 1
20	NORMAL2	Volvió a operación normal el proceso físico 2
21	NORMAL3	Volvió a operación normal el proceso físico 3
22	NORMAL4	Volvió a operación normal el proceso físico 4

Mensajes de inicialización

Además de los mensajes antes mencionados, existe un tipo de mensaje que los procesos utilizan para inicializar su operación, y en este caso corresponde para notificar unos procesos a otros los identificadores de los procesos proxies que emplearan para la transmisión de este tipo de mensajes. Su composición se muestra a continuación:

Tipo	Proceso	Proxy
char	char	pid_t

El campo tipo contiene los siguientes códigos:

Clave	Identificador	Descripción
18	INFO_PROXY	Notifica el proceso proxy que será usado.
19	INFO_PROXY_REMOVE	Notifica que no se usará más el proceso proxy.

El campo proceso contiene la siguiente información:

Clave	Identificador	Descripción
08	CONTROLADOR	El mensaje proviene del proceso controlador
20	GUI	El mensaje proviene del proceso encargado de la interface gráfica

5.2.7 DESCRIPCIÓN DE LOS MENSAJES ENTRE EL SERVIDOR DE COMUNICACIONES Y LOS CONTROLADORES DE LOS PROCESOS FÍSICOS

Como se describió en el capítulo anterior en el que se hacía referencia a la arquitectura del proyecto, el tipo de mensajes que la PC mandará a los controladores son dos: el primero solicita información sobre el estado actual de los sensores y actuadores del sistema, el segundo tipo corresponde a códigos de acción que ejecutarán los controladores. Los códigos empleados para dicha comunicación son los siguientes:

Controlador que recibe el mensaje	Código del mensaje	Descripción del mensaje
Controlador 1	01	Pide información
	03	Desactiva algoritmo de control
	09	Activa algoritmo de control
	23	Posiciona cámara en 15°

Controlador que recibe el mensaje	Código del mensaje	Descripción del mensaje
Controlador 1	24	Posiciona cámara en 30°
	25	Posiciona cámara en 45°
	26	Posiciona cámara en 60°
	27	Posiciona cámara en 75°
	28	Posiciona cámara en 90°
	29	Posiciona cámara en 105°
	30	Posiciona cámara en 120°
	31	Posiciona cámara en 135°
	32	Posiciona cámara en 150°
	33	Posiciona cámara en 165°
Controlador 2	02	Pide información
	04	Desactiva algoritmo de control
	10	Activa algoritmo de control
	34	Activa actuador 1
	35	Activa actuador 2
	36	Activa actuador 3
	37	Activa actuador 4
	46	Desactiva actuador 1
	47	Desactiva actuador 2
	48	Desactiva actuador 3
	49	Desactiva actuador 4
Controlador 3	05	Pide información
	07	Desactiva algoritmo de control
	11	Activa algoritmo de control
	38	Activa actuador 1
	39	Activa actuador 2
	40	Activa actuador 3
	41	Activa actuador 4
	50	Desactiva actuador 1
	51	Desactiva actuador 2
	52	Desactiva actuador 3
	53	Desactiva actuador 4
Controlador 4	06	Pide información
	08	Desactiva algoritmo de control
	12	Activa algoritmo de control
	42	Activa actuador 1
	43	Activa actuador 2

Controlador que recibe el mensaje	Código del mensaje	Descripción del mensaje
Controlador 4	44	Activa actuador 3
	45	Activa actuador 4
	54	Desactiva actuador 1
	55	Desactiva actuador 2
	56	Desactiva actuador 3
	57	Desactiva actuador 4

Los mensajes que recibe la PC de los diferentes controladores se muestran a continuación. Estos datos de información tienen el protocolo de la comunicación serial usada descrito en el capítulo anterior.

Controlador del Proceso 1:

Dato sensor pos. X	Dato sensor pos. Y	Posición cámara
1 byte	1 byte	1 byte

El mensaje esta contenido en 3 bytes de información, en los cuales se utiliza un byte por dato.

Controlador del Proceso 2:

Actuadores				Sensores			
A4	A3	A2	A1	S4	S3	S2	S1
1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit

El mensaje está contenido en un byte de información y cada uno de sus campos indica con un 1 o 0, si el sensor y/o actuador correspondiente esta activado o no.

Presión 1	Temp. 1	Presión 2	Temp. 2	A4 A3 A2 A1
1 byte	1 byte	1 byte	1 byte	1 byte

El mensaje esta contenido en 5 bytes, donde los datos de presión y temperatura corresponden a 1 byte cada uno y el byte restante contiene los valores on/off de los actuadores.

Controlador del Proceso 4

Niv. ruido 1	Niv. ruido 2	Niv. ruido 3	Niv. ruido 4	A1 A2 A3 A4
1 byte	1 byte	1 byte	1 byte	1 byte

El mensaje esta contenido en 5 bytes, al igual que el mensaje anterior, y los datos correspondientes a los niveles de ruido se encuentra en 1 byte cada uno, el último byte del mensaje contiene los valores on/off de los actuadores.

5.2.8 DESCRIPCIÓN DEL PAQUETE DE DATOS A TRANSMITIR POR EL PUERTO PARALELO

Por el puerto paralelo se transmitirá un paquete de 2 bits que direccionará el multiplexor conectado a este puerto:

MUX	
bit 1	bit 0

Multiplexando la señal Tx proveniente de los controladores de acuerdo a la siguiente tabla:

bit 1	bit 0	Descripción
0	0	Tx controlador 1
0	1	Tx controlador 2
1	0	Tx controlador 3
1	1	Tx controlador 4

5.2.9 DICCIONARIO DE DATOS

El diccionario de datos contiene la información referente a las estructuras de datos empleadas en un sistema, existiendo para ello varias nomenclaturas, la que optamos en nuestro caso contiene la siguiente información:

Para los datos compuestos (estructuras de datos)

- **Estructura: Nombre de la estructura**
- **Campos: campos componentes de la estructura**
- **Procesos: Proceso(s) que utiliza(n) la estructura**
- **Archivo: Archivo en el cual se declara la estructura**
- **Miniespecificación: Descripción de la información contenida**

Para los datos elementales (campos de las estructuras)

- **Nombre: nombre del campo**
- **Tipo: tipo del campo**
- **Estructura: estructura a la que pertenece**
- **Procesos: Proceso(s) que utiliza(n) dicho campo.**
- **Archivo: Archivo en el cual se declara el campo.**

En el diccionario de datos los procesos son identificados con la siguiente numeración (Tablas 4 y 5):

1. Interface gráfica
2. Monitoreo proceso 1
3. Monitoreo proceso 2
4. Monitoreo proceso 3
5. Monitoreo proceso 4
6. Controlador
7. Servidor de comunicaciones

Con esto concluimos el presente capítulo, el diseño aquí presentado pudiera parecer complejo para la etapa actual del proyecto, pero en el mismo se prevee el crecimiento del sistema, de forma tal, que cuando las estrategias de control sean más complejas se disponga del módulo correspondiente para ello, sin que su modificación afecte al resto del sistema, de igual forma en el momento en que se decidiera cambiar el tipo de comunicación con los procesos físicos del sistema, e implementar una comunicación por puerto paralelo, sólo se hará la modificación al módulo correspondiente.

Estructura	Campos	Procesos en los que se utiliza							Archivo al que pertenece	Miniespecificación
		1	2	3	4	5	6	7		
Control_struct	tipo	X					X		Mensajes.h	Mensaje de solicitud de petición del usuario
	codigo									
Info_proxy_msj	tipo	X	X	X	X	X	X		Mensajes.h	Mensaje que informa del pid del proxy que manejaran dos procesos
	proceso									
	proxy									
Reply_controlador	código	X					X		Mensajes.h	Mensaje respuesta del controlador
Reply_Info_gui	datos	X	X	X	X	X			Mensajes.h	Mensaje con los datos muestreados
	N									
Reply_server_comando	codigo						X	X	Mensajes.h	Mensaje de respuesta del servidor ante un comando
	error									
Reply_server_info	datos		X	X	X	X		X	Mensajes.h	Mensaje de respuesta del servidor ante una petición de información
	N									
	error									
Send_senal_struct	tipo	X	X	X	X	X	X		Mensajes.h	Mensaje para informar un evento
Server_struct	tipo		X	X	X	X	X	X	Mensajes.h	Mensaje para solicitud de servicios del servidor
	codigo									
Menu_struct	menu	X							Menus.h	Sirve para la operación de los menús
	padre									
	tamaño									
	pos									
Menu_item	hijo	X							Menus.h	Define una opción del menú
	titulo									
	codigo									

Tabla 4 Estructuras de datos compuestos

Nombre del campo	Tipo del campo	Estructua a la que pertenece	Proceso en el que se utiliza							Archivo
			1	2	3	4	5	6	7	
codigo	char	Menu_item	X							Menus.h
		Reply_server_comando						X	X	Mensajes.h
		Server_struct		X	X	X	X	X	X	Mensajes.h
		Reply_info_controlador	X					X		Mensajes.h
datos	char[]	Control_struct	X					X		Mensajes.h
		Reply_info_gui	X	X	X	X	X			Mensajes.h
		Reply_server_info		X	X	X	X		X	Mensajes.h
error	char	Reply_server_comando						X	X	Mensajes.h
		Reply_server_info		X	X	X	X		X	Mensajes.h
hijo	char	Menu_item	X							Menus.h
menu	menu_item []	Menu_struct	X							Menus.h
N	char	Reply_info_gui	X	X	X	X	X			Mensajes.h
		Reply_server_info		X	X	X	X		X	Mensajes.h
padre	char	Menu_struct	X							Menus.h
pos	char	Menu_struct	X							Menus.h
proceso	char	info_proxy_msj	X	X	X	X	X	X		Mensajes.h
proxy	pid_t	info_proxy_msj	X	X	X	X	X	X		Mensajes.h
tamaño	char	Menu_struct	X							Menus.h
tipo	char	info_proxy_msj	X	X	X	X	X	X		Mensajes.h
		Control_struct	X					X		Mensajes.h
		Send_senal_struct	X	X	X	X	X	X		Mensajes.h
		Server_struct		X	X	X	X	X	X	Mensajes.h
titulo	char[]	Menu_item	X							Menus.h

Tabla 5 Campos de las estructuras de datos

CAPITULO 6

RESULTADOS Y CONCLUSIONES

Para la evaluación del sistema, se debe medir su operación para el peor de los casos. Como todo sistema en tiempo real, este debe responder a los cambios del mundo exterior en un tiempo aceptable.

El problema que siempre se ha tenido en esta clase de sistemas, es determinar el tiempo en que esta respuesta será aceptable. Para nuestro SMCTR se han establecido dos tiempos de respuesta, correspondientes a las funciones de control y monitoreo respectivamente. En el caso de la función de control, el tiempo de respuesta lo definimos como: el intervalo de tiempo que le toma al sistema dadas las condiciones de entrada (todas las variables muestreadas del sistema), determinar el comando que se mandarán a los controladores. Para el segundo caso, el tiempo de respuesta correspondiente a la función de monitoreo se determina por: el intervalo de tiempo que le toma al sistema emitir el código de solicitud de información a un controlador específico, recibir la información, y desplegar el resultado en la pantalla.

Para la función de monitoreo se ha determinado que el tiempo máximo de respuesta deberá ajustarse a la frecuencia natural a la que muestrean los microcontroladores sus respectivos sensores, es decir, el reflejo de los cambios en las variables muestreadas deberá mostrarse en pantalla antes de que se tenga el siguiente valor de dicha variable. Dichas frecuencias varían de acuerdo al proceso y la rapidez de operación del microcontrolador para ejecutar el algoritmo de control local con el que opera, así como el tiempo que tarda la transmisión de información. Para resolver el dilema anterior Alvarez ha propuesto una frecuencia de operación tanto para los microcontroladores como para el monitoreo central con un período de 0.5 seg. [ALV95]. No obstante, la aplicación puede soportar frecuencias mas altas de muestreo, que estarán acotadas por las velocidades de operación de los dispositivos empleados para la comunicación y la frecuencia máxima de operación de los microcontroladores.

Para el caso del muestreo, teniendo en cuenta las velocidades actuales de transmisión serial (9,600 bauds) nos toma 1,042 μseg. el transmitir un comando a los controladores, y toma en promedio unos 6,000 μseg. al

controlador contestar al comando, añadiendo el tiempo de transmisión de la información solicitada por el comando (máximo se transmiten 5 bytes), nos da un tiempo de transmisión y reconocimiento de comandos de 12,250 μseg. aproximadamente. Estos tiempos son importantes cuando se desea aumentar la frecuencia de muestreo, y/o el número de procesos que monitoreará el sistema, con la carga actual del sistema (4 procesos) y considerando que sólo se tiene una línea de transmisión, los procesos no podrán ser monitoreados a una frecuencia mayor de 20 hertz (50 milisegundos de espacio entre muestras). Al aumentar la frecuencia de muestreo del sistema, se debe cuidar que el proceso de muestreo no sobrecargue al microcontrolador, o altere de forma significativa su ciclo de trabajo, ya que estos microcontroladores han asignado como prioridad máxima el atender a las peticiones del control central [ALV95].

El otro tiempo de respuesta correspondiente a la función de control del sistema, se acota como en el caso anterior, por el período de muestreo. Es decir, el algoritmo debe haber generado el código correspondiente antes de que se tenga la siguiente muestra. El tiempo del período de muestreo con el que debemos sincronizarnos es el que corresponde a la mayor frecuencia de muestreo. Para esta etapa del proyecto en el que tenemos una frecuencia de muestreo de 2 hertz para todos los procesos, se hizo una simulación de una carga crítica del sistema para determinar el máximo tiempo de procesamiento que el algoritmo de control podría utilizar sin alterar o sobrecargar al sistema.

Los procesos físicos que se controlan en esta etapa del proyecto, por su naturaleza tienen una dinámica lenta, por lo que para poder evaluar la sobrecarga fue necesario implementar una simulación para evaluar si el sistema la soportaba. La sobrecarga para nuestro SMCTR la definimos como el caso supuesto en el que cada muestra colectada fuera diferente a la anterior, activándose para ello el algoritmo de control, y el proceso de graficación correspondiente para mostrar el cambio de estado en las variables del sistema. Para esta simulación se generaron números aleatorios para cada uno de los valores de las variables muestreadas, observando con ello que no se satura el sistema en la parte correspondiente a la graficación. Sin embargo, hay que recordar que la estrategia de control usada en esta etapa del proyecto no requiere de mucho tiempo de procesamiento, por lo que se empleo un ciclo de operaciones matemáticas para medir el tiempo de cómputo del que se dispondría en mejoras posteriores e implementaciones del algoritmo de control. Se llegó finalmente a un tiempo máximo de procesamiento de 150 milisegundos ante las condiciones actuales del proyecto, lo que equivale a unas 245,750

divisiones de números enteros aproximadamente, dicho tiempo se estimo en base a la simulación de sobrecarga y considerando un uso interactivo del usuario en el envío de comandos a los controladores.

De igual forma, los comandos del usuario, deben atenderse y mandar las señales correspondientes a los controladores y actuadores en un tiempo no mayor a un período de muestreo, sin embargo, la ejecución de los códigos enviados tanto a controladores y actuadores, estará en función de las características de respuesta propias tanto de los microcontroladores como de los circuitos que operan los actuadores.

Esta es la primera etapa dentro del desarrollo del proyecto del que forma parte el sistema aquí expuesto, hasta este punto hemos encontrado un funcionamiento aceptable, que coincide con los objetivos planteados para su desarrollo. Con el diseño modular en procesos que se obtuvo, cabe la posibilidad de dar mantenimiento y seguir haciendo crecer a la aplicación dentro de las limitaciones inherentes principalmente a los tiempos de procesamiento, sin que esto implique rediseñar o volver a programar la aplicación. Cabe hacer notar que dentro de las necesidades que cubre el sistema operativo seleccionado, y los servicios que nos ofrece, la más evidente en la realización del presente sistema son las facilidades que ofrece para la comunicación entre procesos, con esto no queremos decir que todas las demás funciones del sistema operativo sobren, al contrario, son vitales para el buen desempeño del sistema. El sistema de comunicación entre procesos hace que el diseño se enfoque en la función del sistema, y nos provee de una herramienta muy poderosa para la sincronización de procesos, debido a la posibilidad de bloquear a los procesos en espera de la respuesta a su mensaje. Con esta forma de comunicación, se logra además, que cada proceso sea transparentes en su forma de operación y desempeño al resto de los procesos que se ejecutan concurrentemente con él. Lo anterior simplifica enormemente, los esfuerzos requeridos para dar mantenimiento al sistema o modificarlo.

Debido a las facilidades del sistema operativo QNX, nuestro sistema puede ejecutarse de forma distribuida en una red de computadoras, sin que ello cree la necesidad de modificar el código, por lo que, si el proyecto siguiera creciendo y un sólo procesador no bastara obtener tiempos de respuesta aceptable, el diseño aquí presentado sigue siendo válido.

Después de la implementación del sistema encontramos que los sistemas en tiempo real requieren de un gran esfuerzo para el análisis y diseño, esto en gran parte se debe a la gran cantidad de elementos que se requieren conocer para un funcionamiento óptimo de la aplicación. Estos conocimientos involucran no sólo conocimiento y dominio en el área computacional, también se requiere de conocimientos en comunicaciones y electrónica, es por ello que para el desarrollo de estos sistemas en tiempo real, se requiere de un equipo de personas expertas en diferentes áreas y de una gran coordinación entre ellas.

Debido al fuerte acoplamiento de todos los elementos que trabajan conjuntamente, este tipo de sistemas son siempre susceptibles a mejoras, y en la parte del diseño de la aplicación de software se sugiere ir implementando ciertos procesos para medir su funcionamiento, de ahí que podemos hablar de una etapa de diseño e implementación complementarias, que en ocasiones puede implicar volver a la etapa del análisis. De igual forma, las mejoras posteriores necesitan la evaluación de la operación del sistema actual, y en ocasiones habrá que volver a la etapa de análisis y diseño para implementar las mejoras, teniendo un sistema trabajando es más fácil hacer estudios de colas para nivelar la concurrencia, o bien modificar las prioridades de los procesos para un mejor desempeño del sistema.

La tendencia hacia este tipo de sistemas en tiempo real es cada vez mayor, y se requiere de ingenieros que tengan visión y conocimientos de computación, electrónica y comunicaciones a nivel general, con especialización en un área en particular. Las herramientas existen y los avances tecnológicos nos brindan la posibilidad de cada vez más acercarnos a ese sueño de controlar todo con tan solo oprimir una tecla, hoy a llegado el momento de retomar esos sueños y luchar por hacerlos realidad.

BIBLIOGRAFÍA

[ALV95] Alvarez, Jesús. "Diseño de un sistema de monitoreo y control distribuido en tiempo real". Tesis de Maestría. División de Estudios de Posgrado de la Facultad de Ingeniería. UNAM. 1995.

[ALL87] Allworth, "Introduction to real time software design". New York. Springer. 1987.

[BES91] Bestavros, Azer. "Time-constrained Reactive Automata". Tesis de doctorado. Universidad de Harvard. 1991.

[BOO94] Booch, Grady. "Object-oriented analysis and design with applications". 2nd. ed. The Benjamin. California, 1994.

[BUR90] Burns, Alan. "Real-Time Systems and their programming languages". Addison-Wesley, 1990.

[COF73] Coffman, Edward Grady. "Operating systems theory". New Jersey. Prentice-Hall, 1973.

[DEI84] Deitel, Harvey M. "An Introduction to Operating Systems". Massachusetts. Addison-Wesley, 1984.

[DIA95] Díaz, José E. y Martínez, Victor M. "Control para autoclave, usando un algoritmo difuso autosintonisable" Tesis de Licenciatura. Facultad de Ingeniería, UNAM. 1995

[ELL94] Ellison, Karen. "Developing real time embedded sortware in a market-driven company". Ed. Wiley & Sons, Inc. USA. 1994.

[GAL95] Gallmeister, Bill. "Programming for the real world POSIX.4". E.U. O'Reilly & Associates, Inc. 1995.

[HIL92] Hilebrand, Dan. "An Architectural Overview of QNX". QNX Software Systems Ltd. Abril 1992.

[HIL93] Hildebrand, Dan. "A Scalable Microkernel POSIX OS for Realtime Systems" . QNX Software Systems Ltd. Abril 1993.

[HIL94] Hildebrand, Dan. "QNX: Microkernel Technology for Open Systems Handheld Computing". QNX Software Systems, Ltd. Mayo 1994.

[IEE90] IEEE. "POSIX 1003.1-1990: Standard portable operating system interface for computer environments", 1990. Reference number ISO/IEC 9945-1: 1990. Institute of Electrical and Electronic Engineers.NY.

[KAL89] Kalisky y Ready. "Distinctions betweem requirements specification and design of real-time systems". IEEE Software Engineering for Real Time Systems, 1989.

[KEL92] Keller, Marilyn. "Software specification and design: a disciplined approach for real-time systems". New York. Wiley. 1992.

[LAP93] Laplante, Phillip. "Real Time Systems Design and Analysis, an engineer's handbook". IEEE Press, N.Y. 1993.

[OAK94] Oakley, Rob. "QNX Microkernel Technolog: A Scalable Approach to Realtime Distributed and Embedded Systems", Junio 1994.

[ORG72] Organick, Elliott. "The Multics system; an examination on its structure". Cambridge, MIT Press. 1972.

[PRE90] Pressman, Roger. "Ingeniería de Software, Un enfoque práctico". Mc Graw Hill. México 1990.

[PRI95] Priego, Juan C. y Salazar, Erick A. "Localización y direccionabilidad de un vehículo usando un controlador difuso implementado con microcontroladores". Tesis de Licenciatura. Facultad de Ingeniería, UNAM. 1995

[QNX] Página WWW: http://www.qnx.com/about_qnx.htlm

[QNX94] "QNX 4 Operating System. System Architecture". Manual del usuario. QNX Software Systems Ltd. Canada. 1994.

[QUA91] "The Client/Server model". Notas de aplicación Quantum Software Systems Ltd. 1991.

[QUA93] Quarteman, John. "Unix, Posix and Open Systems: the open standards puzzle". Addison'Wesley.1993.

[SAC89] Sacha, K.M. "Embedded systems software specification and design methods",IEEE Software Engineering for Real Time Systems, 1989.

[SIP76] Sippl, Charles J. "Data communications dictionary". Von Nostrand Reinhold. New York. 1976.

[SUB89] Subjelj y Trobec, "An example of concurrent program design", IEEE Software Engineering for Real Time Systems, 1989.

[VAU94] Vaughn, Larry. "Client/Server Systems Design & Implementation". Mc Graw Hills. E.U. 1994

[YOU82] Young, Stephen. "Lenguajes en Tiempo Real: diseño e implementación". Madrid. Paraninfo. 1982.