Centro de Investigación Científica y de Educación Superior de Ensenada

SISTEMAS OPERATIVOS DE TIEMPO REAL Y SUS APLICACIONES POTENCIALES EN ROBOTICA

TESIS
MAESTRIA EN CIENCIAS

URANIA CESEÑA BORBON

Ensenada, B. Cfa., Mexico. Junio del 2000.

TESIS DEFENDIDA POR

Urania Ceseña Borbón

Y APROBADA POR EL SIGUIENTE COMITÉ

Dr. Rafael Kelly Martínez

Director del Comité

Dr. Jesus Favela

Miembro del Comité

Dr. Horacio Soto

Miembro del Comité

M.C. David Saucedo

Miembro del Comité

Dr. Jesus Favela

Jefe del Departamento de Ciencias de la Computación

Dr. Federico Graef Ziehl

Director de Estudios de Posgrado

22 de junio del 2000

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA

DIVISIÓN DE FÍSICA APLICADA
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

SISTEMAS OPERATIVOS DE TIEMPO REAL Y SUS APLICACIONES POTENCIALES EN ROBÓTICA

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de MAESTRO EN CIENCIAS presenta:

Urania Ceseña Borbón

Ensenada, Baja California, México. Junio de 2000.

RESUMEN de la Tesis de Urania Ceseña Borbón, presentada como requisito parcial para la obtención del grado de MAESTRO EN CIENCIAS en CIENCIAS DE LA COMPUTACION. Ensenada, Baja California, México. Junio de 2000.

SISTEMAS OPERATIVOS DE TIEMPO REAL Y SUS APLICACIONES POTENCIALES EN ROBÓTICA

Resumen aprobado por:

Dr. Rafael Kelly Martínez
Director de tesis

El objetivo de esta tesis consiste en estudiar los conceptos y características de los sistemas operativos de tiempo real así como también desarrollar una aplicación en robótica que utilice dichas características.

Para cumplir con este objetivo se ha empleado el sistema operativo de tiempo real QNX para desarrollar la aplicación titulada *Manejador de Procesos de Tiempo Real.* Esta aplicación es desarrollada en lenguaje C y utiliza bibliotecas de funciones del ambiente gráfico Photon de QNX para el desarrollo de una interfaz gráfica que facilita su uso.

La aplicación brinda al usuario la posibilidad de crear programas de tiempo real a partir de programas escritos en C estándar, asignar prioridad y periodo de ejecución a los programas e intercambiar variables entre ellos con el propósito de transferirse información.

La funcionalidad del *Manejador de procesos* fue probada al ser utilizado en el control por computadora de un robot manipulador de transmisión directa.

Palabras Clave: Tiempo real, proceso, periodo de ejecución, sistema operativo de tiempo real, robótica.

ABSTRACT of the Thesis presented by Urania Ceseña Borbón as a partial requirement for obtaining the degree of MASTER of SCIENCE in COMPUTER SCIENCES. Ensenada, Baja California, Mexico. June 2000.

REAL TIME OPERATING SYSTEMS AND THEIR POTENTIAL APPLICATIONS TO ROBOTICS

The goal of this thesis is to study the concepts and features of real-time operating systems, and to develop robotic software tools which take advantage of such features.

To accomplish this objective, a software tool called *"Real-time Process Manager"* was developed using the C language and the graphics interface of QNX Photon.

The Real-time Process Manager consists of a graphic user interface that enables the user to generate programs for real time applications based on programs written in standard C, assign priority and execution period to each program, and also interchange data among them.

The functionality of this *Process Manager* was tested in the computer control system of a direct drive robot arm.

Keywords: Real-time, process priority, execution period, real-time operating system, robotics.

DEDICATORIA

A ti Señor Dios

Por darme la oportunidad de compartir este trabajo contigo y la fortaleza para terminarlo.

A mis padres

Por estar siempre de mi lado y apoyarme en todo momento. Gracias Papa y Mama, por darme la oportunidad de ser parte de ustedes.

A mis hermanos

Omar, Anabel, Franco, Olimpia y Edecy por su cariño, amor y respeto.

A ti Ramón

Gracias por tu amor y por estar siempre a mi lado en los momentos más difíciles. Te quiero.

AGRADECIMIENTOS

A mi director de tesis Dr. Rafael Kelly por la oportunidad para llevar a cabo este trabajo de tesis.

A los miembros de mi comité de tesis, Dr. Horacio Soto, Dr. Jesus Favela y a M.C. David Saucedo por apoyar este trabajo de tesis.

De manera muy especial a Ricardo Campa por tus consejos y comentarios los cuales hicierón que este trabajo se convirtiera en una realidad. Gracias amigo.

A mis compañeros y amigos, los que estan lejos y los no tan lejos. Gracias por estar siempre conmigo, por sus consejos y su compañía.

Al Centro de Investigación Científica y de Educación Superior de Ensenada.

Al Consejo Nacional de Ciencia y Tecnología por otorgar la beca para realizar mis estudios de maestría.

CONTENIDO

Página

I INTRODUCCIÓN 1

I.1 Antecedentes 1

I.2 Motivación 4

I.3 Objetivos 5

II SISTEMAS OPERATIVOS DE TIEMPO REAL 6

II.1 Características deseables de los sistemas de tiempo real 6

II.2 Limitaciones de los sistemas operativos de tiempo real actuales 7

II.3 El estándar POSIX 9

II.4 Principales sistemas operativos de tiempo real 9

II.4.1 Comerciales 9

II.4.2 Públicos 11

II.4.3 Herramientas de desarrollo comerciales 12

III PROCESOS 13

III.1 Definición de proceso 13

III.2 Estados de los procesos 13

III.3 Prioridad de procesos 15

III.4 Comunicación entre procesos 15

III.5 Manejo del tiempo 17

III.6 Calendarización de procesos 18

Página

III.6.1 Esquema *FIFO* 19

III.6.2 Esquema *Round robin* 19

III.6.3 Esquema *Adaptivo* 20

IV DESARROLLO DEL SISTEMA **21**

IV.1 Definición del problema 21

IV.2 Requerimientos del sistema 22

IV.2.1 Características del usuario 22

IV.2.2 Requisitos de software 23

IV.2.3 Requisitos del hardware 23

IV.3 Diseño e implementación 23

IV.3.1 Funcionamiento general del sistema 24

IV.3.2 Elaboración de programas de usuario 25

IV.3.3 Generación de programas de tiempo real 29

IV.3.4 Estructura de los programas de tiempo real 31

IV.3.5 Memoria compartida 34

IV.3.6 Manejo de Procesos 38

IV.4 Descripción del *Manejador de Procesos* 41

V APLICACIÓN DEL SISTEMA A ROBÓTICA **48**

V.1 Diagrama general de la aplicación a robótica 50

V.2 Procedimiento para la realización de una sesión experimental 53

VI CONCLUSIONES Y TRABAJO FUTURO **58**

Página

VI.1 Conclusiones 58

VI.2 Trabajo Futuro 60

LITERATURA CITADA **61**

Anexo A MANUAL DEL USUARIO DEL *MANEJADOR DE PROCESOS* **63**

A.1 Instalación del *Manejador de Procesos* 63

A.2 Inicio de la sesión 65

A.3 Operación del *Manejador de Procesos* 66

A.3.1 Estructura de un programa del usuario 76

A.4 Ejemplo de una aplicación a robótica. 79

A.5 Precauciones 82

A.6 Procedimiento para la realización de una sesión experimental 84

Anexo B EJEMPLOS DE PROGRAMAS DE TIEMPO REAL **88**

B.1 Pasos para iniciar la sesión 91

Anexo C USO DE LA TARJETA DE ADQUISICIÓN DE DATOS **93**

C.1 Funciones para manejo de la tarjeta 93

C.2 Procedimiento para adaptar una nueva tarjeta 94

C.3 Algoritmos para prueba de intervalos de tiempo 95

LISTA DE FIGURAS

Figura		Página
1	Estructura jerárquica de un sistema de cómputo.	2
2	Estructura de un sistema de control en tiempo real.	3
3	Estados de los procesos y posibles transiciones entre ellos.	14
4	Diagrama general del sistema	24
5	Estructura general de un programa de usuario	25
6	Ejemplo de un programa a)estándar; b)con estructura para tiempo real.	27
7	Transformación del programa de usuario por el *Manejador de Procesos.*	29
8	Ejemplo de ejecución de un programa síncrono (periódico).	30
9	Estructura de un programa generado por el Manejador de Procesos	32
10	Diagrama de flujo general para procesos de tiempo real.	34
11	Ciclo de ejecución principal para programas tipo texto.	35
12	Ciclo de ejecución principal para programas tipo gráfico.	35
13	Ventana principal del *Manejador de Procesos*	42
14	Ventana para cambiar el directorio de trabajo.	42
15	Carpeta para selección de programas.	44
16	Carpeta para relación de variables.	45
17	Carpeta para mostrar relaciones entre variables.	46
18	Carpeta para ejecución de programas.	46
19	Ubicación de los sistemas de tiempo real.	48
20	Fotografía del robot manipulador	49
21	Diagrama general de la aplicación a robótica	51

Figura		Página
22	Relaciones de variables para la aplicación a robótica	55
23	Barra de opciones del ambiente Photon.	66
24	Icono para abrir una terminal en el Photon.	66
25	Inicio de sesión del *Manejador de Procesos*	67
26	Cambio de directorio de trabajo.	68
27	Selección de los programas.	69
28	Asignación de prioridad	71
29	Asignación de periodo.	72
30	Carpeta para relación de variables.	72
31	Relaciones de variables	73
32	Ventana de confirmación para eliminado de una relación.	74
33	Carpeta para ejecución de programas.	75
34	Estructura general de un programa de usuario	76
35	Ejemplo de programa a)estándar; b)con estructura para tiempo real. . .	78
36	Brazo mecánico del laboratorio	80
37	Relaciones de variables para la aplicación a robótica	86
38	Ejecución del programa `Hab_Apa.c`.	97
39	Ejecución de los programas `Habilitar.c` y `Apagar.c`.	98

LISTA DE TABLAS

Tabla		Página
I	Archivos de cabecera contenidos en `librerias_qnx.h`.	53
II	Asignación de parámetros a los programas de la aplicación a robótica. .	54
III	Archivos de cabecera contenidas en `librerias_qnx.h`.	83
IV	Asignación de parámetros a los programas de la aplicación a robótica. .	85

SISTEMAS OPERATIVOS DE TIEMPO REAL Y SUS APLICACIONES POTENCIALES EN ROBÓTICA

I INTRODUCCIÓN

I.1 Antecedentes

El software para computadora se puede clasificar en general en dos tipos: los programas de sistema, que controlan la operación de la computadora en sí, y los programas de aplicación, los cuales resuelven problemas para los usuarios.

El programa fundamental de todos los programas de sistema es el sistema operativo [Tanenbaum 93].

Los sistemas operativos nacieron por la necesidad de hacer más fácil el trabajo a los programadores en la complejidad del hardware. La forma en que esto ha evolucionado de manera gradual es colocando un nivel de software por arriba del hardware, con el fin de controlar todas las partes del sistema y presentar al usuario una interfaz o máquina virtual que facilite la comprensión del programa. Este nivel de software es el sistema operativo [Tanenbaum 93].

En la figura 1 se representa en forma general los niveles que componen un sistema de cómputo, observándose la posición del sistema operativo como intermediario entre el hardware y las aplicaciones de software.

Los sistemas operativos han evolucionado a través de los años. Con esta evolución han surgido nuevos sistemas operativos que cumplen con una serie de requerimientos o características. Entre los que podemos mencionar están los sistemas operativos para controlar varias computadoras (sistemas distribuidos), sistemas operativos que trabajan con varios procesadores (procesamiento simétrico), sistemas con requerimientos en

Figura 1: Estructura jerárquica de un sistema de cómputo.

el manejo del tiempo, conocidos como sistemas operativos de tiempo real, sistemas operativos por lotes, sistemas operativos de ventanas, etc. Los sistemas operativos de tiempo real son los que marcan la pauta para el desarrollo de este trabajo de tesis.

Un sistema operativo de tiempo real (Real-Time Operating System, RTOS) es un tipo de sistema operativo que ha sido diseñado para operar bajo restricciones de tiempo. Es decir, un RTOS es un sistema operativo capaz de garantizar los tiempos requeridos de los procesos bajo su control.

Aunque el término *tiempo real* es usado en muchas áreas diferentes, no siempre se le da una interpretación correcta. Muy frecuentemente algunas personas dicen que un sistema opera en tiempo real si es capaz de reaccionar rápidamente a un evento externo. De acuerdo a esta interpretación, un sistema es considerado de tiempo real si éste es rápido. El termino *rápido*, de cualquier manera, tiene un significado relativo y no captura la principal propiedad que caracteriza a este tipo de sistemas [Buttazo 97].

Por ejemplo, no tiene sentido desarrollar un sistema computacional de tiempo real para control aéreo sin considerar las limitaciones de tiempo de cada avión. En la Figura 2 se muestra un diagrama de bloques general de un sistema de control tiempo real.

Se le llama *sistema de tiempo real* al software que se desarrolla para ser ejecutado en

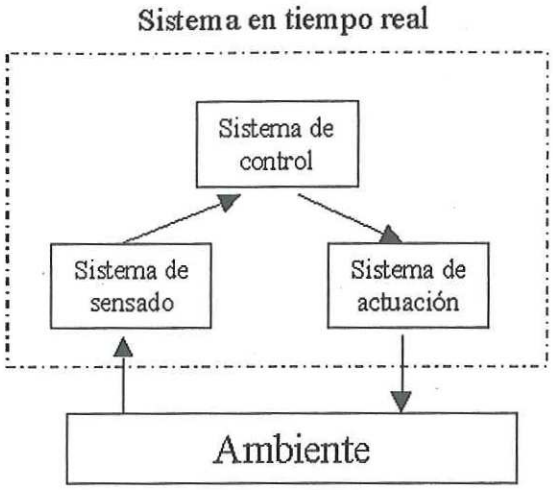

Figura 2: Estructura de un sistema de control en tiempo real.

un sistema operativo de tiempo real. Este tipo de software debe ser capaz de reaccionar en tiempos precisos a eventos externos. Como consecuencia, el comportamiento de estos sistemas depende no sólo de los resultados computacionales obtenidos sino también del tiempo en el cual son producidos esos resultados.

Una reacción que ocurra muy tarde puede resultar inadecuada y, en el peor de los casos, hasta peligrosa para los resultados que se esperan. A continuación se indican algunas áreas de aplicación de los sistemas de tiempo real:

- Control de plantas nucleares y químicas.
- Aplicaciones en el área automotriz.
- Control de ferrocarriles.
- Sistemas de control de vuelos.
- Sistemas de telecomunicaciones.
- Automatización industrial.
- Robótica.

- Sistemas militares.

Si el sistema operativo no contara con métodos específicos para manejar tareas en tiempo real, el sistema podría trabajar aparentemente bien por un periodo de tiempo pero posiblemente fallaría. El alto porcentaje de fallas que ocurren en las plantas nucleares o en sistemas de defensa son causados la mayoría de las veces por errores del software en el sistema de control [Butazzo 97].

I.2 Motivación

Actualmente el Laboratorio de Robótica del CICESE cuenta con sistemas para la programación de algoritmos de control de robots tales como:

- SPAC (Sistema Programable de Algoritmos de Control) [Reyes 97].
- SEAC (Sistema Experimental de Algoritmos de Control) [Jaramillo 99].

Estos sistemas brindan al usuario un ambiente gráfico semejante a *Windows* de Microsoft y la posibilidad de experimentar con diferentes tipos de controladores.

El manejo del tiempo de ejecución siempre ha sido un punto muy importante a considerar para el desarrollo de estos sistemas de software, debido a que existe la necesidad de controlar un dispositivo mecánico en intervalos de tiempo del orden de milisegundos. Para cumplir con esta necesidad los sistemas *SPAC* y *SEAC* se desarrollaron para ser ejecutados sobre el sistema operativo *MS-DOS* y mediante el uso de interrupciones. Aunque se cumple así con los requerimientos de tiempo, existe la limitante de que se debe ejecutar sólo un proceso a la vez, es decir, se necesita una PC dedicada exclusivamente al control del robot.

Es por ello que surgió la necesidad de buscar otros paradigmas de desarrollo de software, como por ejemplo la utilización de un sistema operativo de tiempo real que brindara otras alternativas diferentes a las ya conocidas.

I.3 Objetivos

Los objetivos que persigue esta tesis son:

- Entender los conceptos y características de los sistemas operativos de tiempo real.
- Desarrollar una aplicación en un sistema operativo de tiempo real con el fin de poner en juego sus características más relevantes, como son la asignación de periodos y prioridades de los procesos, así como el intercambio de información entre varios procesos.
- Probar la funcionalidad del sistema desarrollado, mediante una aplicación de robótica.

II SISTEMAS OPERATIVOS DE TIEMPO REAL

Un sistema operativo de tiempo real (RTOS, por sus siglas en inglés) es aquel que ha sido diseñado para operar bajo restricciones de tiempo. Es decir, un sistema operativo de tiempo real es aquel que es capaz de garantizar la ejecución de los procesos en los tiempos requeridos, bajo su control.

El procesamiento en tiempo real debe de manejarse con precisión, ya que no sólo depende de los resultados lógicos de un cálculo, sino también del instante en que éste se produzca.

Una de las características más importantes de los sistemas operativos en tiempo real es que deben reconocer y responder a eventos discretos en intervalos (periodos) de tiempo predecibles. La presición requerida del sistema dependerá de la aplicación en cuestión.

II.1 Características deseables de los sistemas de tiempo real

Las aplicaciones de control complejas que requieren de tiempos precisos en la ejecución de sus tareas necesitan de sistemas operativos predecibles. Lo predecible de un sistema operativo se logra introduciendo un cambio en los paradigmas de diseño de los sistemas de tiempo compartido tradicionales.

A continuación se presentan algunas propiedades básicas de los sistemas de tiempo real, los cuales deberán incluir [Buttazo 97]:

- *Puntualidad:* Los resultados tienen que ser precisos, no sólo en sus valores sino también en su duración. Como una consecuencia, los sistemas operativos de tiempo real deben proveer mecanismos específicos para el manejo de tareas y su duración.

- *Capacidad de sobrecarga de tareas:* Los sistemas de tiempo real no deberán colap-

sarse cuando presenten condiciones de sobrecarga (cuando varias tareas requieran los mismos recursos del sistema de cómputo). Deberán ser diseñados de tal manera que puedan responder a cualquier condición posible de funcionamiento.

- *Predictibilidad:* Para garantizar el mayor aprovechamiento de los recursos y que sean cumplidos los requerimientos en el tiempo preciso. El sistema deberá ser capaz de ser predecible en cualquier condición de calendarización de una tarea. Esta es una de las propiedades más importantes de los sistemas de tiempo real rígidos (Hard Real Time, HRT). Y esta predicción está basada en las características del núcleo del sistema operativo y en la información asociada con cada tarea. El primer componente que contribuye a que el sistema operativo sea predecible es la administración de todos los recursos del sistema de cómputo (procesador, memoria, dispositivos etc.).

II.2 Limitaciones de los sistemas operativos de tiempo real actuales

La mayoría de los sistemas operativos de tiempo real actuales son utilizados para aplicaciones de control y son versiones modificadas de los sistemas operativos de tiempo compartido. Es por ello que todavía conservan características de los sistemas operativos de tiempo compartido que son vistas como limitaciones. A continuación se describen algunas de éstas [Buttazzo 97]:

- *Procesamiento multitareas:* Es un método de programación que se utiliza actualmente para proveer de un conjunto de llamadas al sistema (funciones) para el manejo de procesos (crear, activar, terminar, retardar, suspender, etc.). Muchas de estas llamadas al sistema no consideran el tiempo que se toma para llevarlas a cabo, es por ello que introducen retardos en las tareas en ejecución y pueden provocar resultados no deseados.

- *Calendarización de tareas basada en prioridades:* Este mecanismo de calendarización es muy flexible ya que permite implementar varias estrategias para el manejo de procesos, sólo cambiando las reglas para la asignación de prioridades a las diferentes tareas. Sin embargo, cuando las tareas de una aplicación tienen requerimientos de tiempo, el problema se presenta para decidir qué prioridades asignarle a cada una de las tareas.

- *Habilidad de respuesta rápida para interrupciones externas:* Esta característica por lo general se obtiene dependiendo de la prioridad con la que se esté ejecutando el proceso. Pero aún así introduce un tiempo de retardo para el proceso que se esté ejecutando.

- *Mecanismos de comunicación y sincronización de procesos:* Los mecanismos básicos de comunicación son utilizados para sincronizar el uso de los recursos de un sistema de cómputo entre varias tareas o procesos sin tomar en cuenta el manejo de las prioridades de los procesos.

- *Núcleo pequeño y conmutación de contexto:* Esta característica reduce la saturación de líneas de código inútiles del sistema para de esta manera mejorar el tiempo promedio de respuesta al conjunto de tareas. Sin embargo un promedio pequeño en el tiempo de respuesta de un conjunto de tareas no garantiza que se puedan cumplir con eficiencia todas las tareas. Por otro lado, un núcleo pequeño implica funcionalidad limitada, lo cual afecta la predictibilidad del sistema.

- *Soporte de reloj de tiempo real como referencia del tiempo interno:* Esto es una característica esencial para cualquier núcleo de sistema operativo de tiempo real que maneja actividades críticas con respecto al tiempo de interacción con el ambiente. A pesar de esto, en la mayoría de los sistemas operativos comerciales éste es sólo un mecanismo para el manejo del tiempo. Es decir, en la mayoría de estos sistemas no hay funciones para establecer las restricciones de tiempo de las tareas y mucho menos la activación automática de los periodos de las tareas.

II.3 El estándar POSIX

Por muchos años las técnicas usadas para el diseño y la implementación de software con las características de tiempo real han permanecido constantes. Para lograr esto fue necesario establecer estándares; el más conocido con referencia a los sistemas operativos tipo UNIX es POSIX.

POSIX (Interfase de Sistemas Operativos Portables de UNIX) es un conjunto de reglas que han sido desarrollados por la comunidad de programadores de software bajo los estándares ISO/IEEE para que las aplicaciones sean portables a nivel de código fuente.

El estándar POSIX 1003.1b ha sido desarrollado para servicios de sistemas operativos en tiempo real e incluye definiciones de interfaces para manejo de tareas, entrada/salida asíncrona, semáforos, mensajes, manejo de memoria, señales, calendarización, relojes y temporizadores [Gallmeister 95].

II.4 Principales sistemas operativos de tiempo real

A continuación se describen los principales sistemas operativos de tiempo real comerciales y públicos. Se mencionan también algunas herramientas que permiten transformar un sistema operativo tradicional en uno de tiempo real.

II.4.1 Comerciales

- *QNX:* se basa en una arquitectura de micronúcleo que implementa cuatro servicios: calendarización de procesos, comunicación interprocesos, comunicación de red de bajo nivel y uso de interrupciones. Las tareas correspondientes al manejo de dispositivos y los sistemas de archivos son implementados como procesos de usuario. El núcleo del sistema operativo es muy pequeño (7 kilobytes de código) y rápido.

QNX cumple con el estándar POSIX. Esto permite a los desarrolladores familiarizados con UNIX adecuarse fácilmente con el sistema [QNX 98].

- *VxWorks:* es un sistema operativo de tiempo real propietario, debido a que utiliza una tarjeta de adquisición de datos particular. Un anfitrión (host) es usado para el desarrollo de software y para correr partes que no sean de tiempo real en una aplicación. El núcleo del VxWorks corre tareas de tiempo real en la computadora objetivo (target). Para la intercomunicación de las máquinas se utiliza el protocolo TCP/IP.

  VxWorks no es compatible con UNIX pero provee algunas funciones de interfaces que están contenidas en el estándar POSIX [Aranda y Vignoni 97].

- *REAL/IX:* es un sistema operativo monolítico, es decir que no tiene una estructura establecida, debido a que está compuesto por un conjunto de procedimientos independientes entre sí, donde cada uno puede llamar a otro procedimiento cada vez que así se requiera.

  REAL/IX tiene varias características del sistema UNIX y su procesamiento es de tiempo real. El núcleo es muy poderoso y esto se debe al uso de semáforos que ayudan a proveer acceso exclusivo a los recursos utilizando las funciones de `sleep/wakeup` y la deshabilitación de las interrupciones.

  El uso de semáforos facilita la portabilidad a máquinas de multiprocesadores. Además REAL/IX cumple con el estándar POSIX, lo que lo hace portable en aplicaciones UNIX. Además de la calendarización en tiempo real, las capacidades de REAL/IX incluyen la preasignación de memoria y espacio de archivos, entradas/salidas síncronas y asíncronas, comunicación entre procesos, temporizadores y conexión de interrupciones [Aranda y Vignoni 98].

- *LynxOS:* este sistema operativo es una reimplementación del sistema operativo UNIX, pero con el manejo de tiempo real basado en prioridades. Puede ser utilizado en sistemas con multiprocesadores. Permite el manejo de ventanas gráficas

y bibliotecas de funciones para el desarrollo de aplicaciones gráficas. Se pueden desarrollar sistemas integrados (embedded) [Aranda y Vignoni 98].

- *VRTX:* es un sistema operativo de tiempo real optimizado para sistemas integrados. Es muy conocido por ser altamente confiable. Se desarrollan con él aplicaciones que soportan multitareas y multiprocesamiento. Utiliza una arquitectura modular que facilita la selección de aquellos componentes requeridos por la aplicación, es por ello que minimiza los requerimientos de memoria [Microtec Division 98].

II.4.2 Públicos

- *RTEMS:* (Real Time Executive for Military System) es un sistema operativo de tiempo real de que provee un ambiente para el desarrollo de aplicaciones militares integradas y contiene las siguientes características:

  Presenta un entorno homogéneo para monoprocesador y multiprocesador, calendarización por prioridades, calendarización monolítica, comunicación entre tareas y sincronización, herencia de prioridades, gestión de interrupciones y gestión dinámica de memoria [Aranda y Vignoni 98].

- *RT-Linux:* es una variante de Linux para manejar tareas donde el tiempo es crítico. Consiste en adherir una capa más al núcleo del Linux entre la capa de tiempo real y las interrupciones del hardware.

  La idea básica de Linux es correr bajo el control de un núcleo de tiempo real. Cuando hay una tarea de tiempo real a realizarse, el sistema operativo asigna todos los recursos a esta tarea hasta que finaliza. Pero cuando no existe ninguna tarea de tiempo real, el núcleo de tiempo real calendariza a Linux para ejecutar todas las demás tareas que no son de tiempo real, con esto podemos decir que depende de las prioridades, asignando las bajas para aquellas tareas que no son de tiempo real [Barabanov y Yodaiken 96].

- *Chimera:* es un sistema operativo de tiempo real mediante el cual se programa para multiprocesadores. Está basado en sistemas operativos para bus VME, el cual soporta múltiples procesadores. Chimera tiene todas las características del núcleo de tiempo real, permite además una calendarización estática y dinámica, manejo y detección de errores, comunicación entre diferentes procesadores y sincronización primitiva. Soporta la programación en C y C++ [Stewart y Khosla 97].

II.4.3 Herramientas de desarrollo comerciales

- *Hyperkernel:* es una herramienta gráfica para el desarrollo de aplicaciones de tiempo real donde el tiempo es rígido, es decir, las aplicaciones cumplen con los plazos de tiempo preestablecidos. Utiliza Windows NT como plataforma. Es posible desarrollar aplicaciones de control de dispositivos, servidores de red, etc. [Imagination Systems 98].

- *ControlShell:* es un sistema de programación de tiempo real y es utilizado para resolver problemas relacionados con el control de dispositivos de hardware. Utiliza la programación orientada a objetos de alto nivel en forma gráfica modular de Top-Down. Controla las señales de procesamiento, el control de retroceso y manejo de eventos [Real-Time Innovations 98].

- *RTX:* (Real Time eXtension) es una herramienta de desarrollo que agrega características de tiempo real al sistema operativo Windows NT. Tiene la ventaja de que soporta las interfases gráficas desarrolladas para el ambiente Windows. El RTX proporciona tanto componentes para el desarrollo de aplicaciones que requieren tiempos de respuesta rápidos así como también aquellas que no sean de tiempo real. Proporciona la facilidad para la comunicación entre procesos y sincronización entre los procesos en tiempo real y los que no lo son [VenturCom 99].

III PROCESOS

III.1 Definición de proceso

Una actividad que abarca una acción o una serie de acciones dedicadas a lograr cierto objetivo específico sobre una computadora es conocido en términos computacionales como *proceso* o *tarea*. Existen una infinidad de ejemplos de este tipo de actividades como son: la transferencia de información desde un dispositivo (disquete, cinta magnética, CD, etc.), la compilación de un programa fuente, etc.

Para llevar a cabo cualquier proceso es indispensable que se ejecuten una serie de instrucciones escritas en algún lenguaje, lo cual es conocido como programa; es decir, la ejecución de un programa tiene como objetivo llevar a cabo un proceso. Entonces la diferencia entre un programa y un proceso reside en su actividad, es decir, un programa tiene una actividad estática mientras que un proceso tiene una actividad dinámica.

Las computadoras modernas con ayuda del software (sistema operativo) hacen varias actividades "al mismo tiempo" (al menos eso hacen creer al usuario); a esto se le conoce como *procesamiento multitarea*. A la vez que ejecuta un programa del usuario, una computadora puede leer de un disco e imprimir en una terminal o impresora. En un sistema de multitareas (e.g. Windows, UNIX, QNX, etc.) la CPU también alterna de proceso en proceso, ejecutando cada uno de ellos por decenas o centenas de nanosegundos. En sentido estricto, la CPU ejecuta un sólo programa en cierto instante [Tanenbaum 93].

III.2 Estados de los procesos

Un proceso puede presentar tres estados en forma muy general, los cuales son:

- *Listo (Ready):* está listo para ejecución pero en espera de que se le asigne la CPU, debido a que se encuentra ocupada por otro proceso.

- *Ejecución (Run):* utiliza la CPU en un instante dado.
- *Bloqueado (Blocked):* un proceso es bloqueado o detenido debido a que no puede continuar con su ejecución, ya sea que se encuentra en espera de ciertos datos que no están disponibles todavía, o porque la CPU ha sido asignada a otro proceso.

Son posibles cuatro transiciones entre estos estados, como lo muestra la figura 3. La transición 1 ocurre cuando un proceso no puede continuar debido a que está esperando algún recurso del sistema que está ocupando algún otro proceso. En ciertos sistemas, el proceso debe ejecutar una llamada al sistema operativo para pasar al estado de bloqueo. Lo más frecuente es que un proceso intente leer información de un archivo, y éste no se encuentre disponible debido a que está siendo leído por otro, por lo que el proceso se bloquea en forma automática.

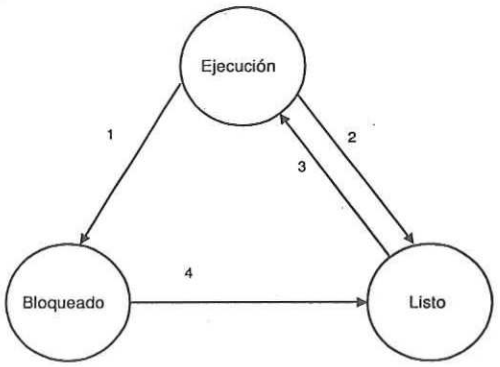

Figura 3: Estados de los procesos y posibles transiciones entre ellos.

Las transiciones 2 y 3 se deben al calendarizador de procesos (una parte del sistema operativo) sin que el proceso influya en ellas. La transición 2 ocurre cuando el calendarizador decide que el proceso en ejecución ya ha sido ejecutado el tiempo suficiente y que es tiempo de que otro proceso use la CPU. La transición 3 ocurre cuando los demás procesos han tenido su parte y es tiempo de que el primer proceso vuelva a ejecutarse.

La transición 4 aparece cuando ocurre el evento externo por el que espera el proceso bloqueado. Si no existe otro proceso en ejecución en ese momento se produce la

transición 3 en forma inmediata y el proceso comienza su ejecución. En caso contrario, tendría que esperar en estado de listo, hasta que la CPU esté disponible [Tanenbaum 93].

III.3 Prioridad de procesos

Siempre que dos o más procesos compartan concurrentemente los recursos de una sola CPU, es inevitable que ocurran conflictos con respecto al acceso a la CPU, debido que sólo se puede dar servicio a un programa en cualquier instante o momento; es por ello que surge la necesidad de asignarle a cada proceso una prioridad. Entonces la prioridad de un proceso depende de la importancia que tiene dicho proceso para ser ejecutado inmediatamente y que se le sean asignados los recursos que está requiriendo.

III.4 Comunicación entre procesos

Los procesos, según su aplicación, algunas veces requieren de comunicarse entre ellos, ya sea para coordinarse al querer tener el acceso a un mismo recurso o para el envío de información, es por ello común hablar de técnicas de *comunicación entre procesos* (interprocess communication, IPC). Las más importantes se mencionan a continuación [Gallmeister 95]:

Memoria compartida. En algunos sistemas operativos, los procesos que trabajan juntos comparten con frecuencia un espacio común para almacenamiento, en el que cada uno puede leer o escribir (memoria compartida). El espacio compartido puede estar en la memoria principal o ser un archivo compartido; la localización de la memoria compartida no modifica la naturaleza de la comunicación o los problemas que puedan surgir. Por ejemplo, cuando dos o más procesos leen o escriben datos compartidos el resultado final depende de quién ejecuta que y en qué momento. Esto se puede evitar prohibiendo que más de un proceso lea o escriba a la vez datos compartidos.

La memoria compartida es el método más rápido de IPC, donde la cooperación de los procesos se lleva a cabo asignando el mismo espacio de memoria y el mismo espacio de direcciones. Cuando un proceso escribe a una localidad de memoria en el área compartida, el dato está inmediatamente disponible para cualquier otro proceso que tenga acceso a esa misma área de memoria compartida.

Señales. Las señales se consideran una característica importante en la parte de multitareas en los ambientes UNIX/POSIX. Son utilizadas para diferentes propósitos, por ejemplo:

- Manejo de errores (un error es la incapacidad de manejar un resultado o un tipo de dato, como por ejemplo: la división entre cero, el acceso de apuntadores, etc.).
- Notificación de ocurrencia de eventos asíncronos entre procesos (E/S, expiración de cronómetros, etc).
- Emulación de multitareas.

Un proceso puede enviar una señal a otro proceso. Los procesos pueden indicar al sistema lo que deberá ocurrir al llegar una señal. Las opciones son ignorarla, capturarla o dejar que la señal detenga la ejecución del proceso. Si un proceso opta por capturar las señales enviadas a él, debe especificar un procedimiento para el manejo de las señales. Al llegar una señal, el control pasa de manera abrupta al administrador de señales. Cuando el administrador de señales termina, regresa el control al proceso que captura la señal, de forma análoga a las interrupciones de E/S del hardware. Un proceso sólo puede enviar señales a miembros de su propio grupo de procesos (padre, hermanos e hijos). Un proceso también puede enviar señales a todos los miembros de su grupo de procesos con una sola llamada al sistema.

Semáforos. En los sistemas multitareas (como es el caso de los sistemas UNIX/POSIX) que manejan diferentes procesos, donde algunos de éstos necesitan esperar a que un

evento suceda para seguir su ejecución, se implementa lo que se conoce como *semáforo.* Un semáforo es básicamente un contador, el cual registra los eventos y la cola de procesos que esperan por tal evento. La idea básica de un semáforo es prevenir que más de un proceso acceda a un mismo recurso del sistema.

III.5 Manejo del tiempo

A fin de poder implementar sistemas operativos que soporten el multiprocesamiento es indispensable que el hardware de la computadora cuente con un reloj, el cual es un dispositivo que genera una interrupción periódicamente, con lo cual se establece un intervalo de tiempo conocido como *quantum.*

El *quantum* es la unidad mínima de tiempo o el mínimo intervalo de tiempo manejado por el núcleo del sistema operativo [Tanenbaum 93].

Los *cronómetros* o *temporizadores* (timers) son contadores que se incrementan o decrementan en cada ciclo de reloj. Existen básicamente dos tipos:

- *Cronómetro de disparo:* es un contador de tiempo el cual básicamente está formado por dos variables que registran un tiempo de inicio y un tiempo final. Cuando el cronómetro se inicializa, copia el valor de la variable que registró el tiempo de inicio en un contador y después de cada pulso del reloj físico (hardware) el contador del cronómetro se incrementa. Cuando el contador llega al valor de la variable que registró el tiempo final, provoca una interrupción y se detiene. Por lo regular éstos son utilizados para tareas que se ejecuta una sóla vez.

- *Cronómetro de repetición o periódico:* es un contador de tiempo el cual está formado por una variable que registra el tiempo de inicio y otra variable que registra el periodo de repetición. En este tipo de cronómetro la cuenta inicial se decrementa y al llegar a cero provoca una interrupción, entonces el valor de la variable que registró el periodo se copia de manera automática en el contador y

todo el programa se repite en forma indefinida.

III.6 Calendarización de procesos

Cuando más de un proceso espera ser ejecutado por la CPU, el sistema operativo debe decidir cuál de ellos debe ejecutarse en primer término. Para realizar esta tarea se crean los algoritmos para la calendarización de procesos, llamados *planificadores o calendarizadores.*

Un calendarizador (scheduler) es un programa de sistema que controla el uso de los recursos, tales como el tiempo, la memoria o los dispositivos de entrada/salida, que quizás sean requeridos por algunos o todos los procesos. El objetivo de un calendarizador es asignar estos recursos a uno de los procesos que lo requiera mediante una selección, y que un proceso no se apodere por siempre de los recursos del sistema.

Un calendarizador realiza esta selección en el inicio de cada intervalo de tiempo (*quantum*) generado por el reloj de la computadora o cuando un proceso termina su ejecución.

Un buen algoritmo de calendarización debe contener las siguientes características [Tanenbaum 93]:

- *Equidad:* que se garantice que cada proceso obtenga una proporción justa de la CPU.
- *Eficiencia:* que se mantenga ocupada la CPU el 100% del tiempo.
- *Tiempo de respuesta:* que se minimice el tiempo de respuesta para los procesos interactivos.
- *Tiempo de regreso:* que se minimice el tiempo que deben esperar los procesos para obtener sus resultados.

- *Rendimiento:* que se minimice el número de tareas procesadas en cierta cantidad de tiempo.

Los sistemas operativos de tiempo real que se basan en POSIX, corren sus procesos con políticas de calendarización las cuales son las siguientes [Gallmeister 95]:

- *FIFO (First-In-First-Out)*: Calendarización basada en prioridades.
- *Round robin*: Calendarización basada en prioridades y con *quantum*.
- *Adaptivo*: Calendarización basada en prioridades y con *quantum*.

III.6.1 Esquema *FIFO*

Existen distintos niveles de prioridad y cada proceso tiene asignada una prioridad, de tal manera que si existe más de un proceso en cada nivel se van formando colas de procesos. Cuando el algoritmo de calendarización se activa, verifica cada nivel de prioridad (empezando con el nivel de más alta prioridad) revisando si hay una cola de procesos en ese nivel, si es así entonces toma el primero de la cola y lo envía a ejecutar, cuando el proceso termina la ejecución el algoritmo verifica si existe otro proceso en la cola antes de continuar con el siguiente nivel.

III.6.2 Esquema *Round robin*

Este algoritmo es mejor aún que el FIFO, debido a que permite asignar un *quantum* (ver *sección III.5*) a cada proceso, verificando además su prioridad. Supongamos que un proceso de baja prioridad se está ejecutando y llega uno de más alta prioridad entonces se asigna el procesador al de más alta prioridad. Por otro lado, supongamos que tenemos procesos en el mismo nivel de prioridad, lo que este algoritmo hace es que les asigna un *quantum* a cada uno de los procesos de la misma prioridad de tal manera que el primero de la cola de procesos se ejecuta y al terminar su *quantum* se pasa al

final de la cola y esto sucede con todos y cada uno de los procesos que se encuentran en la cola.

III.6.3 Esquema *Adaptivo*

Al igual que en el FIFO, maneja distintos niveles de prioridad y cada proceso tiene asignada una prioridad y un intervalo de tiempo de ejecución, si el nivel de prioridad tiene más de un proceso esperando por ser ejecutado se forma una cola de procesos. El algoritmo envía a ejecutar siempre al proceso de más alta prioridad. Cuando se presenta un caso en el cual se tienen procesos de la misma prioridad envía a ejecutar el primero de la cola en un periodo de tiempo; si al finalizar este periodo el proceso no termina su ejecución, la prioridad del proceso se reduce a una prioridad inferior para que pueda dar lugar a la ejecución de los otros proceso del mismo nivel que se encuentran esperando. Cuando al proceso le toca volver a ejecutarse y no termina su ejecución se le vuelve a asignar la prioridad original, de tal manera que la próxima vez estará localizado en el nivel de prioridad con el que inició la ejecución.

IV DESARROLLO DEL SISTEMA

Al sistema de software desarrollado en este trabajo de tesis se le ha denominado *Manejador de Procesos de Tiempo Real* (o simplemente *Manejador de Procesos*) y fue implementado sobre el sistema operativo QNX, versión 4.24, utilizando el compilador Watcom C (versión 10.6) para su codificación y las bibliotecas de funciones del ambiente gráfico Photon microGUI, versión 1.13, del QNX. Su función principal es la manipulación y coordinación de procesos de tiempo real y la transferencia de información entre ellos.

Otra de las características que tiene el *Manejador de procesos* es que transforma, mediante la incorporación de una prioridad y periodo de ejecución, un programa basado en lenguaje C estándar (*programa de usuario*) a un programa con características de tiempo real (*programa de tiempo real*).

En este capítulo se describen los pasos seguidos para el desarrollo de este sistema, utilizando la metodología del ciclo de vida clasico de la ingeniería de software [Pressman 93].

IV.1 Definición del problema

Desde un inicio el problema a resolver era el diseño e implementación de un sistema que permitiera administrar la ejecución de procesos de tiempo real con diferentes cadencias (periodos de ejecución) y prioridades, e intercambiar información entre ellos. El sistema a desarrollar debía tener, las siguientes características:

- Utilizar un sistema operativo de tiempo real para su implementación, debido a la imperiosa necesidad de controlar en forma precisa el tiempo.
- Presentar una interfaz gráfica con el fin de facilitar al usuario su utilización.

IV.2 Requerimientos del sistema

Cuando se desea desarrollar algo, ya sea hardware o software, es indispensable establecer o identificar los elementos necesarios para lograrlo. Para ello es necesario llevar a cabo un análisis del problema a resolver, y de esta forma, obtener los requerimientos indispensables para el buen funcionamiento y rendimiento del software. A continuación se mencionan los requerimientos que fueron identificados para este problema en particular:

Sistema operativo de tiempo real. Es importante que la aplicación sea ejecutada sobre un sistema operativo de tiempo real, ya que se pretende probar los beneficios que puedan brindar este tipo de sistemas operativos en el área de control automático.

Ambiente gráfico. El sistema debe presentar al usuario una interfaz amigable. El ambiente brindará al usuario la posibilidad de ejecutar programas de tiempo real, asignar periodos y prioridades a los programas, así como la transferencia de información entre los mismos.

IV.2.1 Características del usuario

Con el fin de que el sistema *Manejador de Procesos* sea de mayor utilidad y beneficio, se sugiere que el usuario:

- Tenga conocimientos básicos de lenguaje C estándar (ANSI).
- Maneje conceptos de sistemas de tiempo real, tales como prioridad y periodo de ejecución de procesos.
- Cuente con las rutinas de entrada y salida, si requiere utilizar un dispositivo externo (tal como una tarjeta de adquisición de datos).

IV.2.2 Requisitos de software

Para el desarrollo del sistema *Manejador de Procesos*, la PC de trabajo deberá tener instalado:

- QNX, versión 4.24: Sistema operativo de tiempo real que fue elegido por sus características para el manejo del tiempo real, su facilidad para utilizar tarjetas de adquisición de diferentes fabricantes y por que puede ser ejecutado sobre una PC con un sólo procesador, entre otras cosas.
- Photon microGUI, versión 1.13: Ambiente gráfico de ventanas de QNX que ofrece al programador las bibliotecas de funciones para el desarrollo de aplicaciones con interfaces gráficas.
- Watcom C, versión 10.6: Compilador utilizado para la programación en lenguaje C bajo QNX, el cual brinda al usuario la posibilidad de crear programas que se ejecutarán como procesos de tiempo real.

IV.2.3 Requisitos del hardware

Para garantizar el buen funcionamiento del sistema *Manejador de Procesos* se sugiere que la computadora cuente con al menos 48 MB de memoria RAM, velocidad de procesamiento de 166 MHz, monitor SVGA y procesador Pentium.

Si el sistema va a ser utilizado para el control de un dispositivo físico, se debe considerar también todo el hardware necesario para tal tarea (tarjeta de adquisición, conexiones, etc.).

IV.3 Diseño e implementación

El diseño del *Manejador de Procesos* se logró a partir de la información establecida en la secciones anteriores. A continuación se describen de manera general las características

del sistema desarrollado.

IV.3.1 Funcionamiento general del sistema

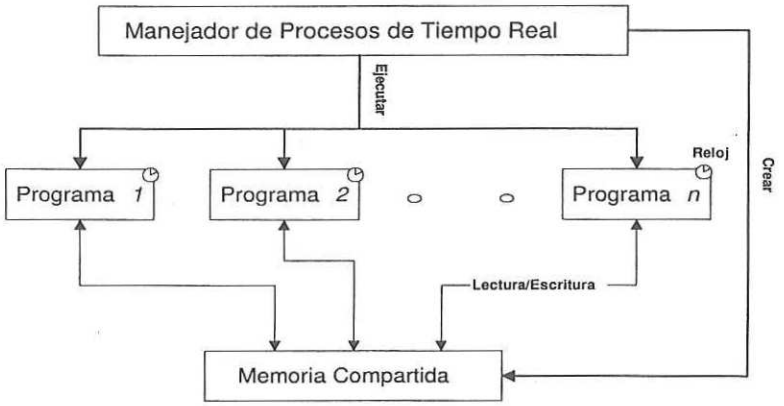

Figura 4: Diagrama general del sistema

El *Manejador de Procesos* debe realizar dos funciones básicas:

1. Utilizar la información contenida en un programa de usuario con la estructura típica en lenguaje C, para generar un nuevo programa que contará con las características para ser ejecutado como un proceso de tiempo real en QNX. A estos programas generados por el *Manejador de Procesos* se les denomina "programa de tiempo real" (ver sección IV.3.3).

2. Una vez creados los programas de tiempo real, el manejador debe iniciar y terminar su ejecución, y proporcionar los medios para que los programas puedan compartir información entre ellos.

Todo esto debería hacerse de manera transparente al usuario, quien únicamente estaría encargado de abrir los *programas del usuario*, establecer las relaciones entre ellos e iniciar su ejecución.

En la figura 4 se observa la forma en que se resolvió el problema de compartición de información entre los procesos. Antes de ejecutar los programas, el *Manejador de Procesos* crea una sección de memoria compartida que es accesada por todos los procesos, de manera que si alguno de ellos modifica el valor de una variable en memoria compartida, automáticamente ese cambio es visto por todos los demás procesos.

En las siguientes secciones se describe con mayor detalle la estructura de los programas de usuario y de tiempo real, la memoria compartida y la forma en que el *Manejador de Procesos* se encarga de administrar los procesos de tiempo real.

IV.3.2 Elaboración de programas de usuario

Seguir un orden en el desarrollo de un objetivo e implantar reglas es un punto importante cuando se desea establecer una estructura en el código fuente de los programas. El fin es unificar un criterio de programación. Para lograrlo, se propone un formato general del programa de usuario, el cual estará dividido principalmente por las siguientes secciones (ver la Figura 5):

```
Palabra_clave[float  var1,  var2, ...,  varn]

Variables y Funciones Opcionales

main(){
        .....
        .....
        Instrucciones
        .....
        .....
}
```

Figura 5: Estructura general de un programa de usuario

- `Palabra_clave[float var1, var2,...,varn]`: Primer sección que debe contener un programa de usuario donde:

  - `Palabra_clave` determina el tipo de programa de usuario, es decir, los programas del usuario podrán ser de tipo texto (`TEXT`) o tipo gráfico (`GRAPHICS`). La diferencia es que los programas tipo `GRAPHICS` pueden desplegar una ventana gráfica, usando herramientas del Photon.
  - `var1, var2, .., varn` representan los nombres de las variables de este programa que se registraran en la memoria compartida, las cuales deben ser tipo `float`.

    A continuación se muestra la sintaxis:

    * Proceso tipo Texto:

      `TEXT[float var1,var2,...,varn]`

    * Proceso tipo Gráfico:

      `GRAPHICS[float var1,var2,...,varn]`

  Nota importante: Para el buen funcionamiento del programa se sugiere que las variables declaradas en esta sección por ningún motivo se declaren en alguna otra parte del programa, debido a que esto puede originar una anomalía en el funcionamiento del sistema. Por otro lado las palabras claves `TEXT` y `GRAPHICS` siempre deberán estar escritas en mayúsculas.

- **Variables y Funciones Opcionales**: Esta sección es opcional, ya que dependerá de las necesidades del programa de usuario. El usuario podrá incluir ya sea variables o funciones necesarias para el funcionamiento del programa. Las variables que se declaren en esta sección no podrán ser reconocidas por los demás programas, es decir no serán compartidas.

- **main()**: Es la función principal del programa de usuario. Esta función será ejecutada en forma repetitiva, ya sea en forma periódica (síncrona) o no (asíncrona),

cuando el programa de usuario sea transformado a un programa de tiempo real.

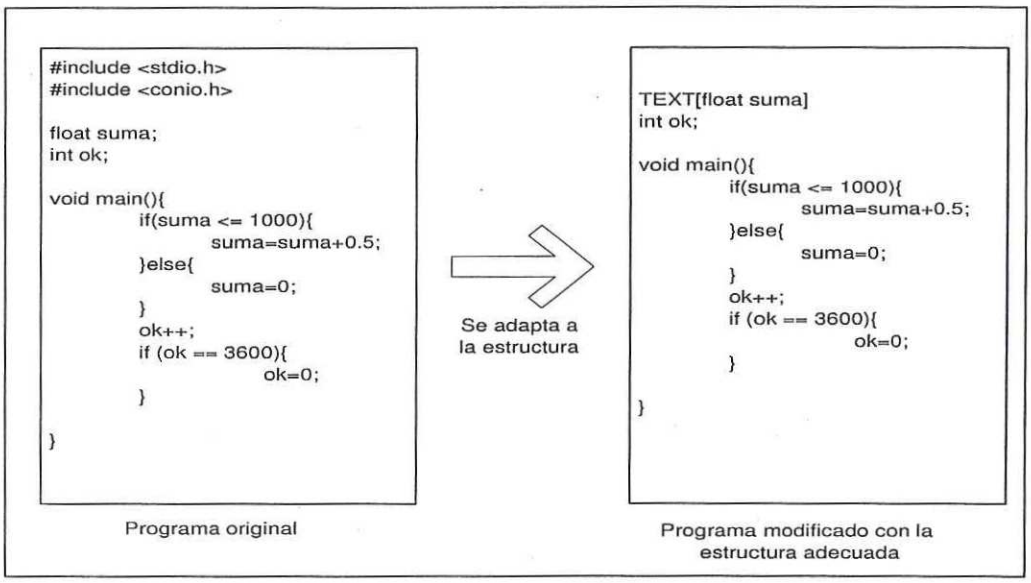

Figura 6: Ejemplo de un programa a)estándar; b)con estructura para tiempo real.

La figura 6 muestra un programa de usuario antes de tener la estructura y después de hacer los cambios correspondientes para que cumpliera con dicha estructura. Esto lo debe hacer el usuario antes de ejecutar el *Manejador de Procesos.* En el ejemplo mostrado se define un programa tipo texto (`TEXT`), con una sola variable compartida (`suma`). Es importante mencionar que las bibliotecas de funciones más comunes se introducen en el archivo `librerias_qnx.h` (ver *Manual del usuario*), por lo que se omiten en el programa modificado.

Recomendaciones para el usuario: A continuación se listan algunos aspectos que se sugieren para la elaboración de un *programa de usuario.*

- Compilación del programa de usuario. Antes de que un programa tenga la estructura del programa de usuario, deberá ser compilado para revisar que no tenga errores de ningún tipo (sintaxis o lógicos). Para compilar el programa fuente

`nombre.c` y obtener el programa ejecutable `nombre.x` (ver *Apendice B*), en QNX la instrucción es la siguiente:

```
cc -o programa.x nombre.c
```

- Archivos de cabecera. El programa de usuario *no* deberá contener funciones que se encuentren en bibliotecas a ser utilizadas sobre el sistema operativo *MS-DOS* o *Windows*. Una de estas bibliotecas es `dos.h` por lo que en el *programa de usuario* deberá quitarse la instrucción.

```
#include <dos.h>
```

- Uso de banderas para control del programa. Como se mencionó anteriormente el `main()` del programa de usuario se transforma en una rutina repetitiva en el programa de tiempo real. Es por ello que se sugiere el uso de banderas cuando se tiene la necesidad de que una instrucción se ejecute un número determinado de veces. Por ejemplo, no sería muy razonable que un archivo se abra y cierre continuamente en la rutina repetitiva; lo más común, es cuando se desea abrir el archivo al inicio de la ejecución del programa (sólo una vez) y después de una cierta cantidad de iteraciones (por ejemplo: 200 veces) se cierre el archivo, aunque la rutina repetitiva siga ejecutándose indefinidamente. A continuación se muestra la manera en que un programa así sería implementado.

```
TEXT[float ok]
int band=0;
main(){
   if(band==0){
       abrir_archivo();
   }
   band++;
   ...
   Instrucciones
   ...
   if(band == 200){
       cerrar_archivo();
```

```
    }
}
```

- Evitar el uso de estructuras de control. Por la misma razón indicada en el punto anterior, se debe evitar la utilización de estructuras de repetición `do-while()`, `while()`, `for()`. No se debe olvidar que una vez convertido en un programa de tiempo real (ver sección siguiente) el contenido de `main()` se ejecutará repetitivamente.

IV.3.3 Generación de programas de tiempo real

El código fuente del programa de usuario es transformado por el *Manejador de Procesos* (ver figura 7) con el propósito de crear un nuevo programa el cual será ejecutado como un programa de tiempo real. El procedimiento consiste en lo siguiente:

Figura 7: Transformación del programa de usuario por el *Manejador de Procesos.*

- El *Manejador de Procesos* verifica si el programa cuenta con la estructura del programa de usuario.
- Toma el código del programa de usuario y le agrega funciones para manejo de cronómetros, prioridades, memoria compartida, etc., que le dan características de tiempo real.

- El nuevo programa es almacenado en el mismo directorio agregándole al nombre del programa original el prefijo `qnx_`.

Existen tres elementos que determinan el tipo de programa de tiempo real generado por el *Manejador de Procesos*, los cuales son:

a) El identificador `TEXT` o `GRAPHICS`: Que indica si un programa es gráfico o de texto (ver sección anterior).

b) El periodo de ejecución: Es el intervalo de tiempo en que se inicia la ejecución del programa una y otra vez. Este periodo es asignado por el usuario a cada programa, desde el *Manejador de Procesos*.

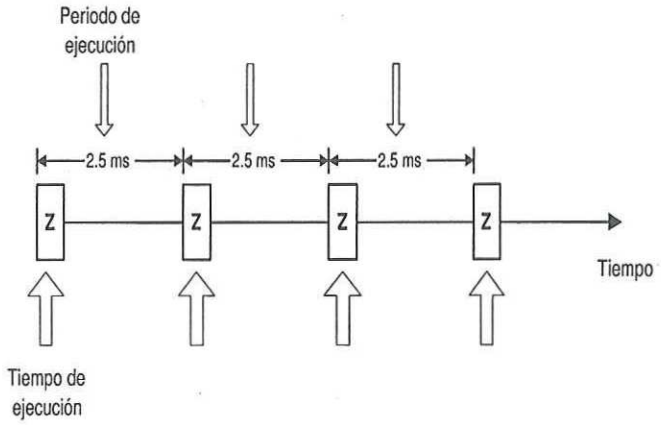

Figura 8: Ejemplo de ejecución de un programa síncrono (periódico).

Por ejemplo, supóngase que se tiene un programa denominado *Z*, el cual se requiere que sea ejecutado periódicamente cada 2.5 milisegundos. Si este programa es puesto en ejecución, entonces repetirá su ejecución cada 2.5 mseg (ver Figura 8) por tiempo indefinido hasta que el usuario suspenda su ejecución a través del *Manejador de Procesos*. Los programas de tiempo real se clasifican en *síncronos* y *asíncronos*, dependiendo de la existencia o no de un periodo de ejecución.

Programa síncrono: Es el programa que será ejecutado cada vez que se cumpla el periodo asignado. El valor del periodo estará en el intervalo de 0.1 hasta 999999 (milisegundos).

Programa asíncrono: Es el programa que será ejecutado cuando el procesador no sea utilizado por ningún otro proceso. En el *Manejador de Procesos*, el usuario asigna un periodo igual a cero a este tipo de programas.

c) La prioridad: Es el orden de importancia que se le otorga a un programa para su ejecución(ver capítulo anterior). En QNX se maneja un rango de 1 hasta 29 para los programas de usuario, siendo la importancia en orden ascendente.

Nota importante: En QNX existen en su forma general dos tipos de usuarios: el administrador del sistema que tiene acceso al directorio raíz `root` del QNX y el usuario común (éste es dado de alta por el administrador). Solo el administrador puede utilizar el rango indicado (1 - 29); para el usuario común la prioridad máxima es 19.

IV.3.4 Estructura de los programas de tiempo real

Un programa generado por el *Manejador de Procesos* está constituido por cuatro secciones (ver Figura 9) que a continuación se describen:

- **Sección de Archivos de cabecera.**

  Esta sección está formada por los archivos de cabecera necesarios para que el programa pueda ser ejecutado en tiempo real.

- **Sección de Variables.**

  Está formada por la declaración de las variables del programa de usuario, las nuevas variables que requiere el programa como son el tipo de programa (`TEXT` o

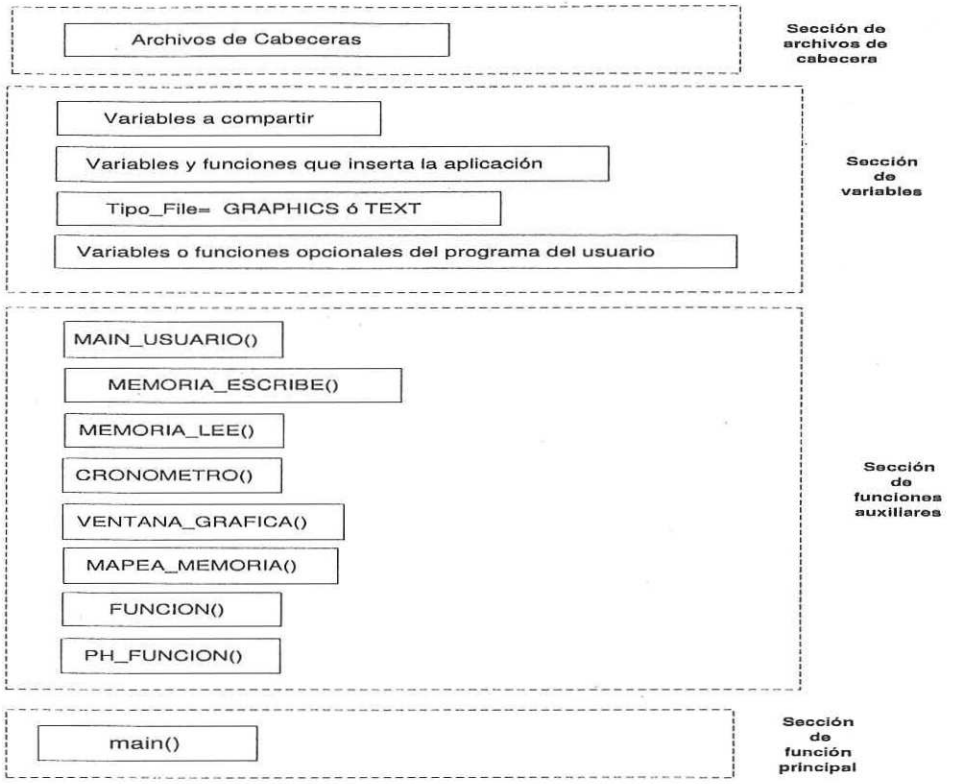

Figura 9: Estructura de un programa generado por el Manejador de Procesos

GRAPHICS), las variables para crear un cronómetro y/o una gráfica, las variables que identifican el periodo y la prioridad, entre otras.

Esta sección se divide en cuatro partes:

- *Variables a compartir:* En esta parte están todas las variables del programa del usuario que se quieren compartir (fueron leídas de la primer línea del programa de usuario).
- *Variables y funciones que inserta la aplicación:* Cuando un programa de usuario es convertido a un programa de tiempo real, este nuevo programa requiere variables para la creación de un cronómetro, de una ventana, para la memoria compartida, etc. y es aquí donde se introducen.
- *Tipo_file=* `GRAPHICS` *o* `TEXT`*:* El valor asignado a la variable `Tipo_file` de-

penderá del tipo de identificador que indique el usuario en su programa (`TEXT` o `GRAPHICS`).

- *Variables o funciones opcionales del programa del usuario:* Aquí es donde están todas aquellas variables del programa de usuario que no son compartidas con otros programas, así como también las funciones que el programa del usuario requiera. Estas son copiadas directamente del *programa de usuario.*

- **Sección de Funciones auxiliares.**

  Contiene las funciones del programa de usuario y las funciones para crear un cronómetro, accesar a la memoria compartida, crear una ventana gráfica y auxiliar a la correcta ejecución del programa. A continuación se describen:

  - *MAIN_USUARIO():* Corresponde a la función `main()` del programa de usuario. Esta es la función que se ejecutará en forma repetitiva al entrar en ejecución el *programa de tiempo real.*
  - *MEMORIA_ESCRIBE():* Esta función se encarga de la escritura de variables a la memoria compartida.
  - *MEMORIA_LEE():* Lee la memoria compartida y asigna los valores correspondientes a las variables dentro del programa.
  - *CRONOMETRO():* Crea un cronómetro usando las funciones de QNX.
  - *VENTANA_GRAFICA():* Despliega una ventana gráfica en la pantalla siempre y cuando el tipo de programa sea GRAPHICS.
  - *MAPEA_MEMORIA():* Esta función se utiliza para que el programa pueda hacer uso de la memoria compartida.
  - *FUNCION():* Verifica si existen variables compartidas de escritura o de lectura y realiza la acción correspondiente, posteriormente llama a MAIN_USUARIO(). FUNCION() se ejecuta de manera directa cuando el programa es tipo texto.

- *PH_FUNCION():* Cuando un programa es tipo gráfico se utiliza esta función, la cual incluye las instrucciones necesarias para generar una ventana gráfica para Photon. Posteriormente hace una llamada a FUNCION().

- **Sección de función principal:** Esta sección esta formada por la función `main()`, la cual es hecha, para que pueda ser utilizada igualmente por los dos tipos de programas, es decir, para uno tipo gráfico o para uno tipo texto. Por medio de sentencias condicionales se determina qué instrucciones serán utilizadas en cada caso. Esto se muestra en forma esquemática en las figuras 10 - 12.

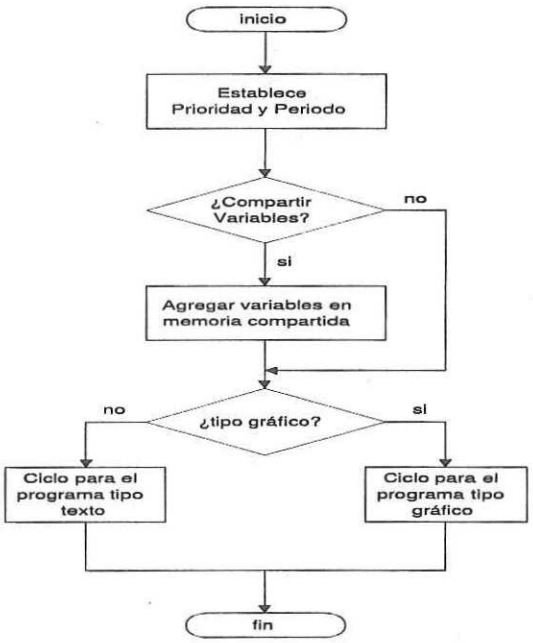

Figura 10: Diagrama de flujo general para procesos de tiempo real.

Independientemente del tipo de programa en `main()` se deberá verificar si existen variables compartidas, si se utiliza un cronómetro o una ventana gráfica, etc.

IV.3.5 Memoria compartida

La *memoria compartida* utilizada por el sistema es un arreglo de datos tipo estructura, la cual contiene información sobre todos los programas que se están ejecutando y las

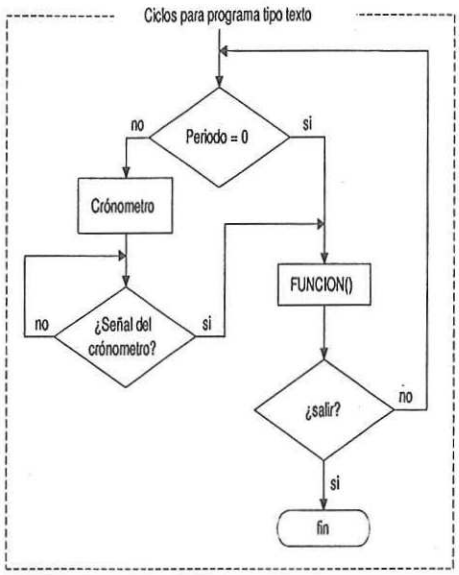

Figura 11: Ciclo de ejecución principal para programas tipo texto.

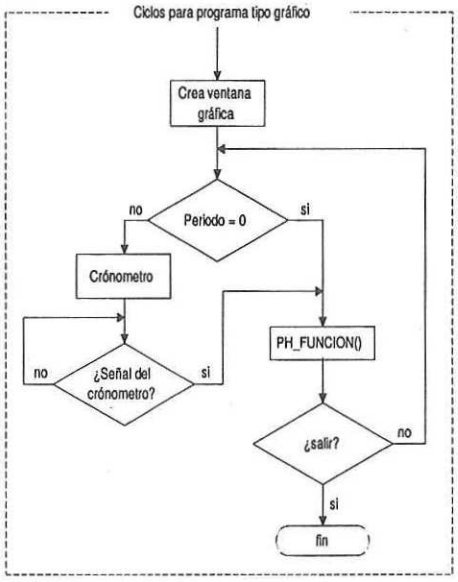

Figura 12: Ciclo de ejecución principal para programas tipo gráfico.

variables que comparten.

- Cada variable compartida almacena en memoria un elemento de la siguiente estructura:

```
struct x{
```

```
    int num_file;
    float valor;
    char Num_binario[60];
    char T;
}*shmemory;  //declaracion del objeto de memoria shmemory del tipo x.
```

donde:

- `num_file`: Identifica al programa que utiliza esta variable.
- `valor`: Contiene el valor de la variable.
- `Num_binario[60]`: Cadena utilizada como máscara para identificar todas las demás variables con las que se esta compartiendo valor.
- `T`: Bandera que puede tomar uno de los siguientes tres valores:

  'w' = variable compartida de escritura.

  'r' = variable compartida de lectura.

  '*' = variable no relacionada.

A continuación se describen cada una de las acciones que se llevan a cabo con la memoria compartida por parte del *Manejador de Procesos* y de los programas de tiempo real. Estas acciones son:

- Creación de la memoria compartida.
- Enlace de la memoria compartida.
- Lectura de datos de la memoria compartida.
- Escritura de datos a memoria compartida.

- **Creación de la memoria compartida:** Se utiliza para esto la función `shm_open()` de Watcom C para QNX. La sintaxis es la siguiente:

`int shm_open(const char *name, int oflag, mode_t mode);`donde:

`name` Es el nombre que se le asigna a la memoria compartida.

`oflag` Corresponde a la forma en la cual se crea o se hace acceso a la memoria compartida, es decir, se puede llevar a cabo un acceso de: solo lectura, lectura y escritura, para crear la memoria compartida.

`mode` Corresponde a un conjunto de bits que crean una mascára para el acceso a la memoria compartida.

Despues de crear el apuntador a la memoria compartida se debe establecer su tamaño, mediante la función `ltrunc()` que recibe entre sus argumentos el apuntador a la memoria y el tamaño deseado.

Nota: Para la versión de QNX utilizada, el tamaño mínimo de la memoria compartida es una página (4096 bytes) y si se excede se utilizan múltiplos de este tamaño.

El encargado de crear la memoria compartida, asignarle su tamaño y llenarla con sus valores iniciales es el *Manejador de Procesos.*

- **Enlace de la memoria compartida:** Cualquier programa que quiera accesar la memoria compartida debe crear un apuntador a ella (mediante `shm_open()`) y luego utilizar la función `mmap()` para enlazar (mapear) el apuntador a alguna dirección particular dentro de la memoria. La sintaxis de la función `mmap()` es la siguiente:

```
void *mmap(void *addr, size_t len, int prot, int flags, int fildes, off_t
off);
```

donde:

`addr` Es una dirección correspondiente a la memoria compartida, donde el único valor valido es cero.

`len` Es el tamaño de la memoria compartida que se desea mapear.

`prot` Corresponde a la forma en la cual se hace acceso a la memoria.

`flags` Es la forma en la cual se hace el acceso a la memoria compartida.

`fildes` Corresponde al descriptor de la memoria compartida proporcionada por la función `shm_open`.

`off` Es apartir de donde se llevará a cabo el mapeo dentro de la memoria compartida.

En el caso del sistema desarrollado, cada uno de los programas de tiempo real mapea toda la memoria (el arreglo de estruturas) por lo que cada programa debe encargarse de seleccionar el elemento correspondiente y procesarlo adecuadamente.

La sintaxis de la función `mmap()` es la siguiente:

- **Escritura de datos en memoria compartida:** Esta operación es realizada en la función `MEMORIA_COMPARTIDA()` para cada programa de tiempo real.

  Debido a la forma en que se está usando la memoria compartida, es necesario utilizar estructuras de selección para identificar cuales variables son de escritura (según el campo `T` de la estructura) y luego hacer la asignación del valor correspondiente a todas las variables que comparten ese valor.

- **Lectura de datos a memoria compartida:** Primero se debe identificar la posición dentro del arreglo de estructuras de la variable de lectura correspondiente. Una vez identificada simplemente se asigna su valor a una variable dentro del programa correspondiente. Esto se realiza dentro de la función `MEMORIA_LEE`.

IV.3.6 Manejo de Procesos

Además de crear los programas de tiempo real y la memoria compartida, el *Manejador de Procesos* se encarga también de iniciar y terminar la ejecución de los procesos.

- **Argumentos de la función `main()`:** La información necesaria para el funcionamiento de cada *programa de tiempo real* es pasada al mismo a través de los

argumentos de la función `main()`. Los argumentos (arreglo `argv[]`) de la función son:

1 `Qnx_periodo`: Esta variable es tipo flotante y corresponde al periodo con el que se ejecutará el programa en milisegundos.

2 `Qnx_prioridad`: Es tipo entero y corresponde a la prioridad que le fue asignada al programa.

3 `Qnx_inc_fp`: Es tipo entero y corresponde al total de variables que se encuentran almacenadas en memoria compartida.

4 `Qnx_maskara`: Cadena utilizada como máscara para identificar dentro de la memoria compartida las variables correspondientes a este programa.

5 `Qnx_num_file`: Es tipo entero y repesenta al orden en que fueron seleccionados por el usuario desde el *Manejador de Procesos.*

6 `Qnx_comparte_memoria`: Es tipo caracter e indica si este programa contiene variables compartidas relacionadas con otros programa.

7 `valorX`: Utilizado sólo por los programas gráficos. Indica la posición horizontal en que será desplegada la ventana.

8 `valorY`: Utilizado sólo por los programas gráficos. Indica la posición vertical en que será desplegada la ventana.

- *Ejecución de programas de tiempo real*: Para iniciar la ejecución de un programa y enviar argumentos a su función `main()`, el *Manejador de Procesos* utiliza la función `spawnl()`. Mediante esta función se envían los argumentos mencionados anteriormente a cada uno de los programas de tiempo real, al momento de crearlos. La sintaxis de la funcion `spawnl()` es la siguiente:

```
int spawnl(int mode, char *path, char *file, char *arg0, ..., argn,
NULL);
```

donde:

`mode` Determina la forma en la cual el programa sera puesto en ejecución.

`path` Corresponde al nombre del programa ejecutable.

`file` Es el nombre del programa.

`arg0,...,argn` Corresponden a los argumentos que se enviarán al programa cuando se ponga en ejecución.

- *Terminación de la ejecución*: Cuando un programa es puesto en ejecución por el *Manejador de Procesos* se crea un identificador de dicho programa, el cual es utilizado por la función `kill()` para terminar con su ejecución. La sintaxis de la función `kill()` es la siguiente:

  ```
  int kill(pid_t pid, int sig);
  ```

  donde:

  `pid` Corresponde al identificador del proceso que se desea terminar.

  `sig` Es el tipo de señal que se desea enviar al proceso.

- `Cronómetros`: Son utilizados por los procesos síncronos. En QNX se utiliza la estructura `struct itimerspec` para configurar el reloj físico a que interrumpa cada cierto intervalo de tiempo en segundos.

  Cada vez que transcurre el intervalo indicado, el cronómetro produce una señal que es recibida por cada uno de los procesos, que anteriormente debieron haber sido configurados para procesar tal señal.

  La función utilizada para configurar un cronómetro en QNX es `timer_settime()`, cuya sintaxis es:

  ```
  int timer_settime(timer_t timerid, int flags, struct itimerspec *value,
  struct itimerspec *ovalue);
  ```

  donde:

  `timerid` Es el identificador del cronómetro.

`flags` Corresponde al tipo de cronómetro.

`value` Corresponde al rango de repetición del cronómetro.

`ovalue` Se le asigna NULL para indicar un cronómetro de repetición.

IV.4 Descripción del *Manejador de Procesos*

El sistema *Manejador de Procesos* se desarrolló para ser ejecutado sobre el sistema operativo de tiempo real QNX. El sistema podrá ser utilizado en cualquier área donde se requiera llevar el control de un dispositivo externo en tiempo real y además trabajar con varios programas que se transfieran información entre ellos. El usuario puede asignar periodo de ejecución y prioridad a los programas que seleccione.

El sistema *Manejador de Procesos* brinda al usuario una interfaz gráfica de fácil manejo. Mediante esta interfaz, el usuario tiene manera de crear programas de tiempo real a partir de programas de usuario codificados en lenguaje C, ponerlos en ejecución y detenerlos cuando él lo desee, además de poder asignarles un periodo de ejecución y una prioridad.

El sistema *Manejador de Procesos* funciona de manera similar al ambiente *Windows* de Microsoft. Al igual que en *Windows* todas las funciones pueden ser seleccionadas mediante el uso del ratón.

La interfaz gráfica del sistema *Manejador de Proceso* se divide en tres secciones (ver Figura 13):

1. **Barra de título:** contiene el nombre del sistema "Manejador de Procesos" y los iconos de minimizar "-" y cerrar "×" la ventana.

2. **Barra de menús:** contiene dos menús desplegables:

   - `Menú Opciones`: contiene tres submenús:

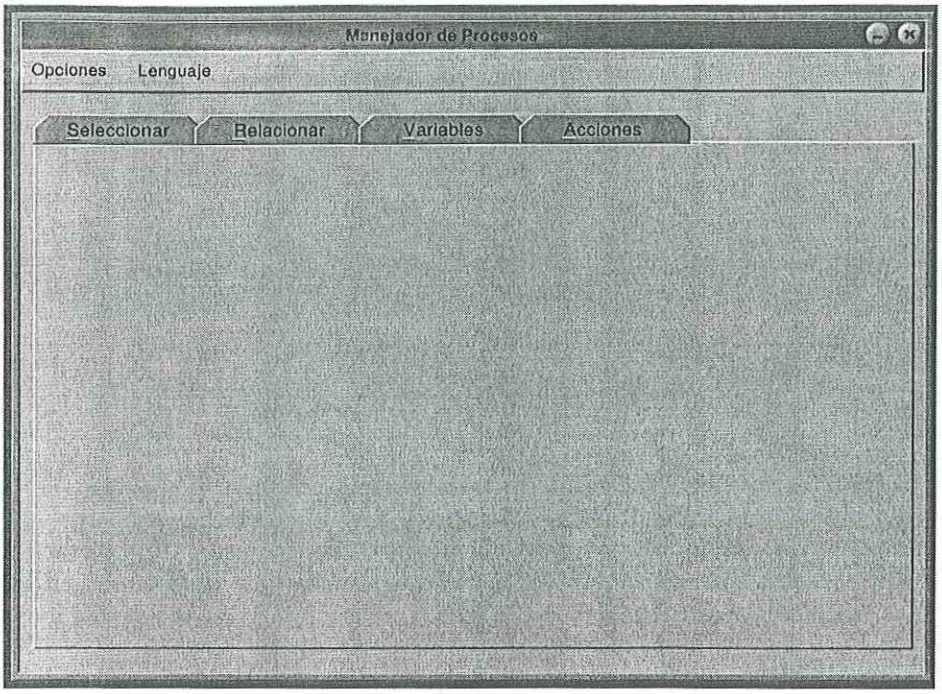

Figura 13: Ventana principal del *Manejador de Procesos*

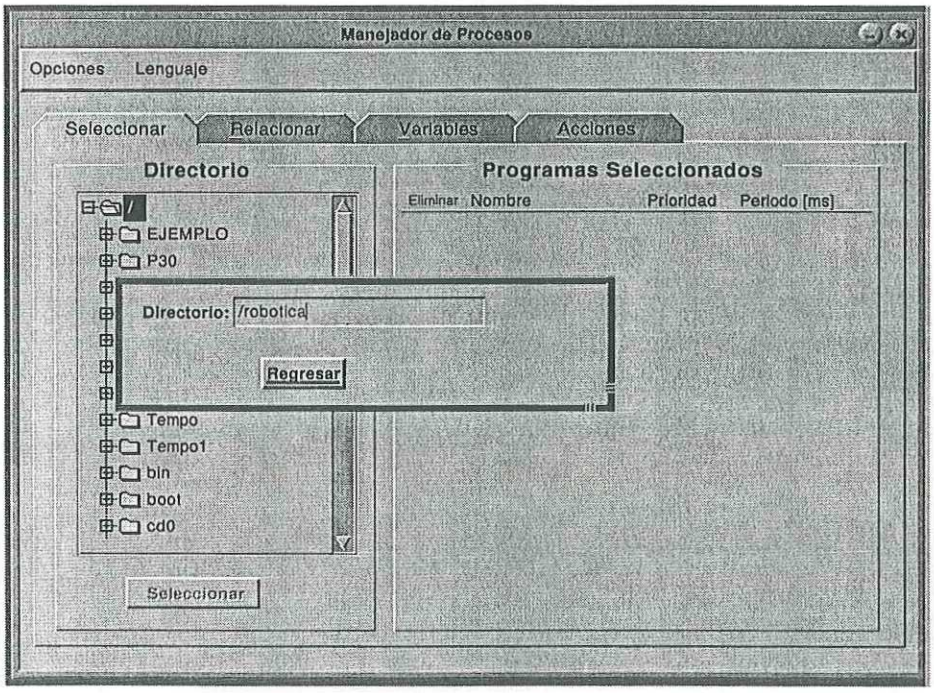

Figura 14: Ventana para cambiar el directorio de trabajo.

a) `Ruta`: permite al usuario cambiar el directorio de trabajo (ver Figura 14).

b) `Inicializar`: para reinicializar una sesión.

c) `Salir`: termina la ejecución del *Manejador de Procesos* y de todos los programas.

- `Menú Lenguaje`: contiene a su vez los submenús `Inglés` y `Español` los cuales se utilizan para cambiar todas las etiquetas que utiliza el sistema *Manejador de Procesos* al idioma seleccionado. Por omisión todas las etiquetas están en español.

3. Sección de carpetas: es la sección principal del *Manejador de Procesos*, debido a que en esta sección es donde se llevan a cabo las operaciones más importantes del sistema. Las carpetas son:

- Seleccionar
- Relacionar
- Variables
- Acciones

El funcionamiento de cada una da las carpetas se describe a continuación:

Carpeta *Seleccionar.* Esta carpeta está compuesta por dos partes (ver Figura 15):

- Directorio: permite al usuario seleccionar los programas a ser ejecutados. Tales programas deberán tener la estructura del programa de usuario (ver sección `IV.3.2`).
- Programas Seleccionados: muestra la lista de los programas seleccionados por el usuario. La lista está compuesta a su vez por cuatro columnas:

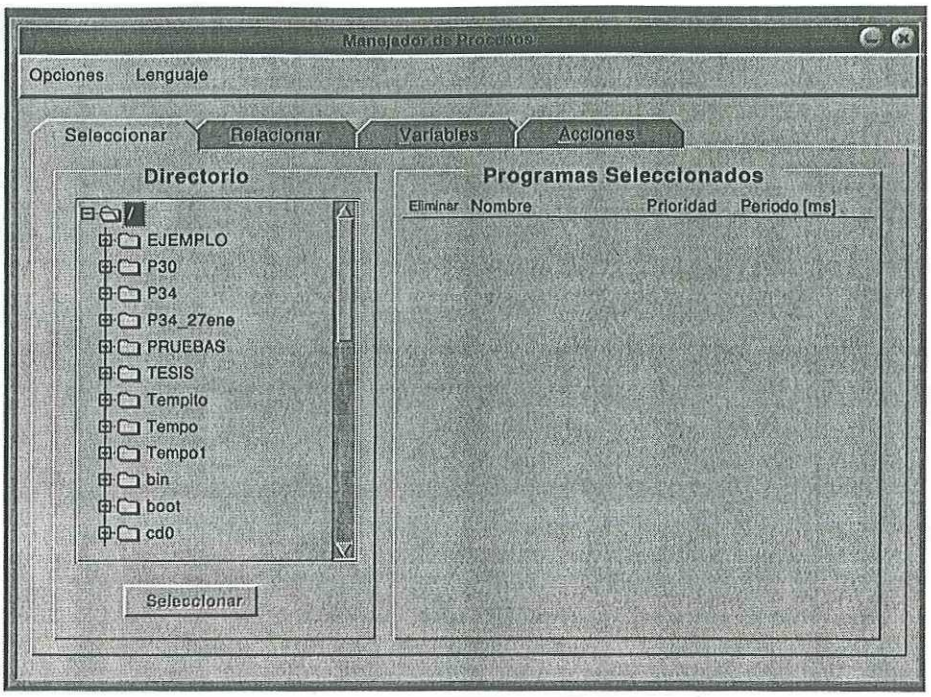

Figura 15: Carpeta para selección de programas.

- Eliminar: a cada programa que se encuentra en la lista le corresponde un rombo, que al ser seleccionado permite eliminar dicho programa de la lista.
- Nombre: contiene los nombres de los programas seleccionados.
- Prioridad: permite asignar una prioridad a cada programa que aparezca en la lista.
- Periodo: permite asignar un periodo de ejecución (en milisegundos) a cada programa que aparezca en la lista.

Cuando el usuario selecciona un programa en la sección *Directorio*, por omisión el *Manejador de Procesos* le asigna una prioridad 10 y un periodo de 100 mseg. Estos valores están sujetos a cambios dependiendo de las necesidades del usuario.

Carpeta *Relacionar.* La función principal de esta carpeta es permitir hacer las relaciones entre las variables de los programas seleccionados. Estar relaciones repre-

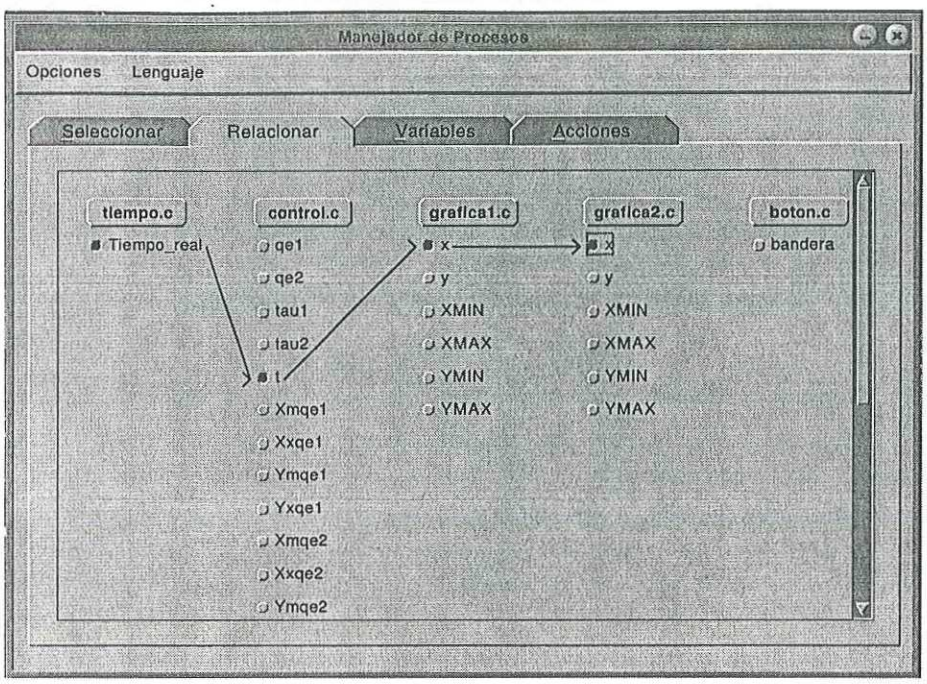

Figura 16: Carpeta para relación de variables.

sentan el pase de información entre las variables de los programas una vez que sean puestos en ejecución por el *Manejador de Procesos* (ver Figura 16).

Carpeta *Variables.* La información que muestra esta carpeta corresponde a las relaciones de variables que se llevaron a cabo en la carpeta *Relacionar.* Las variables relacionadas quedan enmarcadas dentro de un rectángulo donde la variable de escritura aparece en la parte superior del rectángulo y las variables de lectura aparecen dentro del rectángulo (ver Figura 17).

Carpeta *Acciones.* Esta carpeta provee al usuario los medios para poner en ejecución los programas que fueron seleccionados. Además le da la posibilidad de detener cada uno de los programa que fueron puestos en ejecución con anterioridad. La carpeta *Acciones* presenta al usuario la siguiente información (ver Figura 18):

- Nombre: nombres de los programas seleccionados.

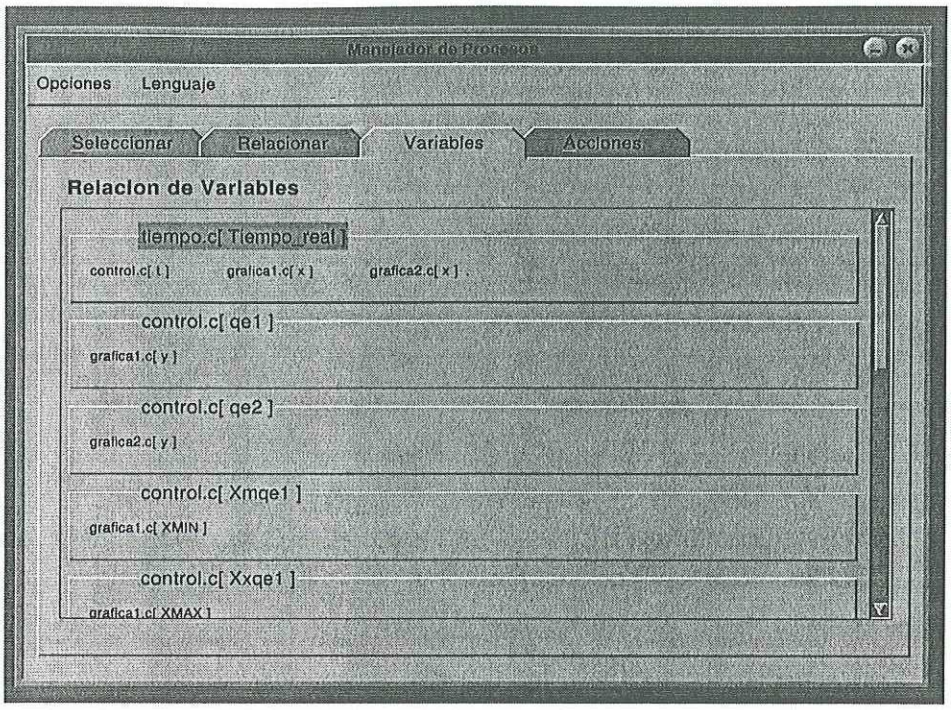

Figura 17: Carpeta para mostrar relaciones entre variables.

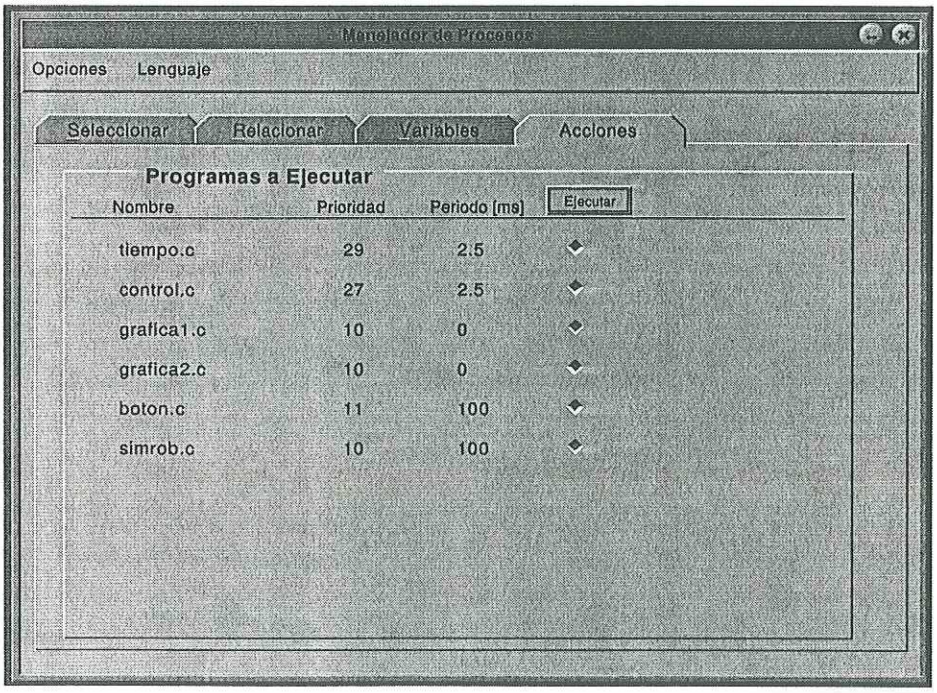

Figura 18: Carpeta para ejecución de programas.

- Prioridad: prioridades que se les asignará a cada uno de los programas en tiempo de ejecución.
- Periodo: periodo de cada uno de los programas.
- Ejecutar: esta columna es encabezada por un el botón que al ser presionado pone en ejecución los programas que aparecen en la lista. Existe además un rombo por cada programa de la lista, el cual al ser presionado detiene la ejecución de ese programa.

V APLICACIÓN DEL SISTEMA A ROBÓTICA

En el capítulo III se explicaron las características más importantes de los sistemas operativos de tiempo real. Se destacó que un aspecto indispensable de un sistema operativo de tiempo real es su capacidad para llevar un control preciso de los tiempos en que se ejecuta una tarea. Los sistemas operativos de tiempo real surgieron de la intersección de dos grandes áreas del conocimiento como son la ingeniería de sistemas computacionales y la ingeniería de control (ver Figura 19) [Arzen 99].

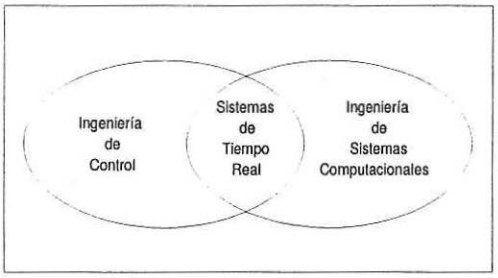

Figura 19: Ubicación de los sistemas de tiempo real.

Es comprensible entonces que el uso de los sistemas operativos de tiempo real se dé en el área de robótica. La robótica se enfoca al estudio de robots, donde un robot es un sistema electromecánico capaz de realizar alguna tarea de forma automática y que puede ser reprogramado para ejecutar otras acciones.

Una dificultad importante en robótica es la velocidad con que deben ejecutarse distintos algoritmos y la necesidad de intercambiar información en tiempo real entre los mismos. Por este motivo el uso de varias unidades de procesamiento interactuando para satisfacer los requerimientos de cálculo en tiempo real encuentra una aplicación inmediata en robótica [Ramos et al., 97].

Para probar la funcionalidad del *Manejador de Procesos* se utilizaron los siguientes elementos:

Brazo mecánico. Es un robot manipulador de dos grados de libertad, formado por dos motores rotacionales de transmisión directa, uno para el hombro y el otro para el codo, unidos por dos eslabones (ver la figura 20). Los motores son operados mediante una señal de voltaje analógico. Cada uno de ellos consta de codificadores ópticos incrementales.

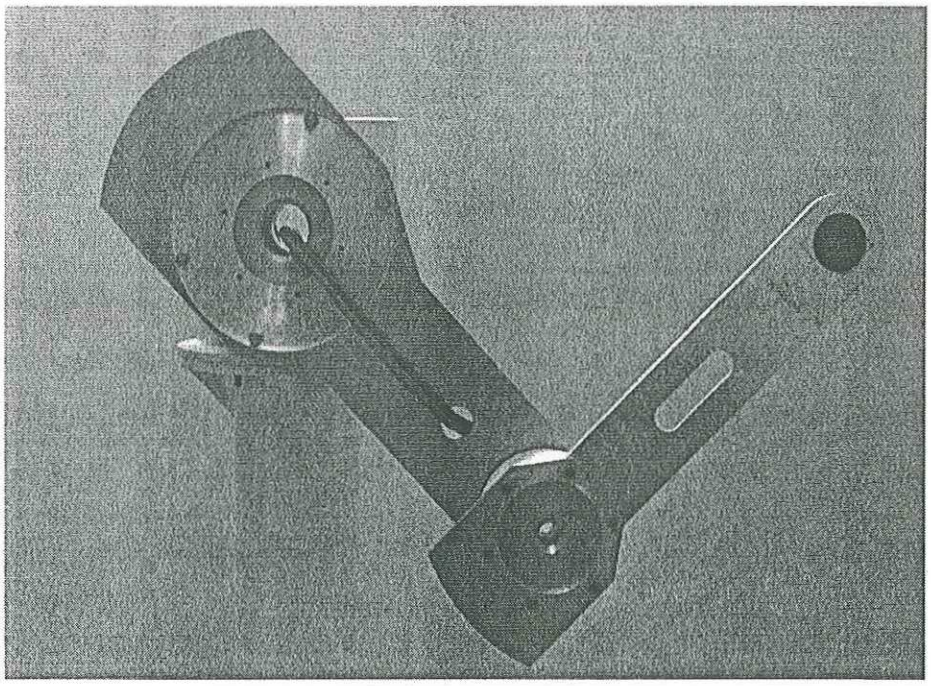

Figura 20: Fotografía del robot manipulador

Tarjeta de adquisición de datos. Se utilizó la tarjeta MFIO-3A para establecer la interfaz de comunicación entre el sistema de software y el robot. Tiene tres canales de 16-bit D/A, entradas para tres codificadores ópticos, 24 bits de programación digital de E/S y soporta el bus estándar de 16-bit ISA. El fabricante de la tarjeta es Precision MicroDynamics.

Controlador. Es el algoritmo numérico que permite operar un sistema físico en forma automática, al comparar la información obtenida del mismo contra la de una tarea

deseada. Es conocido también como algoritmo de control.

Programas auxiliares. Se crearon dos programas que grafican valores en XY (`grafica1.c` y `grafica2.c`), un programa que lleva la cuenta del tiempo real (`tiempo.c`), otro programa que contiene los drivers para la utilización de la tarjeta de adquisición de datos (`mfiolib.h`) y que además contiene el algoritmo de control (`control.c`) y por último otro programa que crea un botón (ventana gráfica) que al ser presionado envía al brazo mecánico a la posición original o casa (`boton.c`).

Los programas en tiempo de ejecución llevan a cabo las siguientes tareas: transferir información entre los programas, ejecutarse con un periodo y una prioridad, controlar el brazo mecánico y graficar las variables indicadas por el usuario (posiciones, errores, etc.) y al presionar el botón con la etiqueta "Ir a Casa" (este botón lo crea el programa `boton.c`) el brazo regresa a la posición de origen (casa).

V.1 Diagrama general de la aplicación a robótica

La figura 21 muestra de manera general como es utilizado el sistema *Manejador de Procesos* en una aplicación en el área de robótica, la cual consiste en utilizar el sistema como una herramienta de experimentación de algoritmos de control para la manipulación de robots. A continuación se describe cada uno de los bloques de la figura 21.

- `tiempo.c`: este programa tiene la tarea de llevar la cuenta del tiempo real transcurrido desde que inicia la ejecución del algoritmo. Se le asigna además, la prioridad más alta (29) debido a que el tiempo transcurrido es de suma importancia para el algoritmo de control, así como para los programas que grafican. Se le asigna un periodo de ejecución de 2.5 milisegundos (lo mismo que al algoritmo de control).

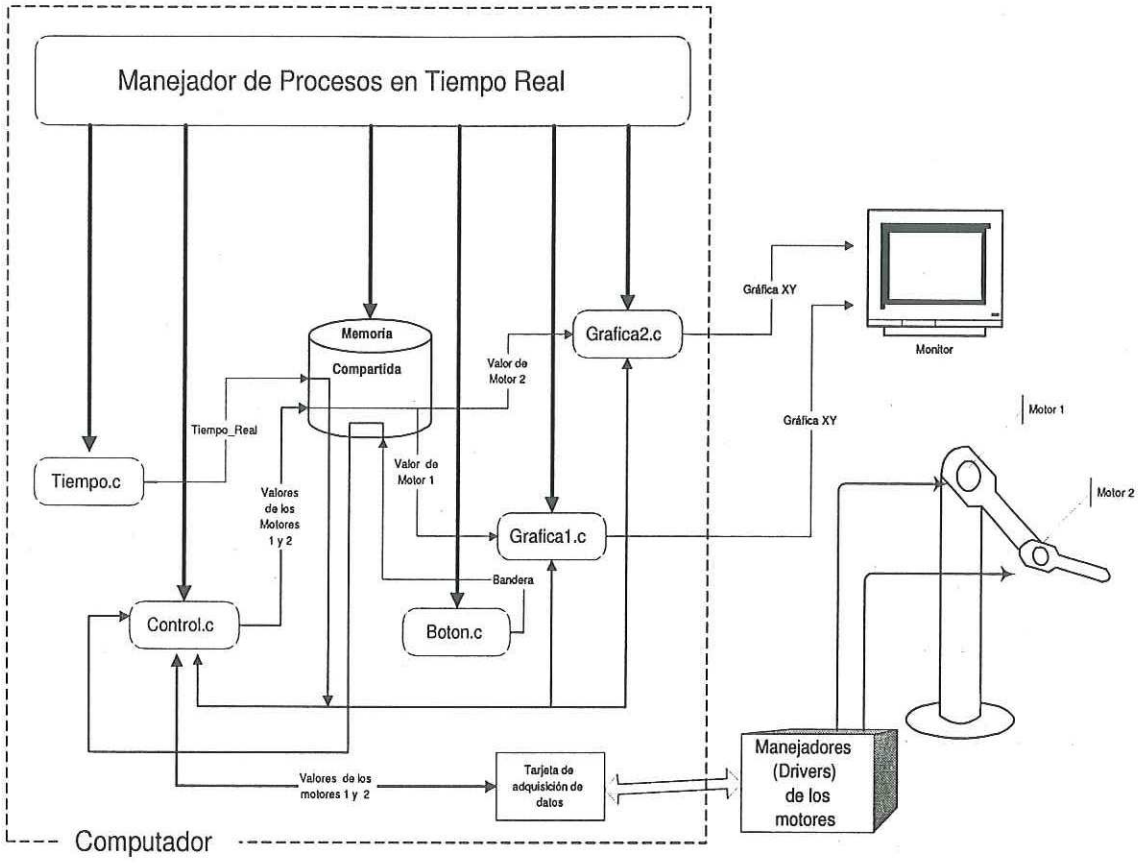

Figura 21: Diagrama general de la aplicación a robótica

- `control.c`: este es el programa que contiene el algoritmo de control, el cual permite la ejecución y manipulación del robot. En general lleva a cabo las siguientes acciones:

  1. Lee la posición del motor de cada una de las articulaciones del robot.
  2. Estima la velocidad de cada articulación.
  3. Ejecuta un algoritmo de control para determinar los pares necesarios para cada uno de los motores.
  4. Aplica los pares calculados a los motores a través de la tarjeta de adquisición.

Una vez realizada la tarea deseada, se debe regresar el robot a su posición original y ésto se lleva a cabo cambiando el algoritmo de control a uno que lleva al robot a su posición de casa (home). Para lograr esto es necesario utilizar una variable que cambie su valor para indicar el cambio de algoritmo de control. Este programa se ejecuta con un periodo de 2.5 milisegundos.

- `grafica1.c`: el objetivo de este programa es desplegar en la pantalla una gráfica con las variables que lee de la memoria compartida, como son las coordenadas X-Y y los valores máximos y mínimos de las escalas de la gráfica. Se le asigna la prioridad 10, que es el valor asignado por omisión por el QNX.

- `grafica2.c`: Este programa es una copia del programa `grafica1.c`, con diferente nombre. Tiene que ser así ya que el *Manejador de Procesos* no puede seleccionar dos programas que se llamen igual. Tanto `grafica1.c` y `grafica2.c` son programas asíncronos (sin periodo de ejecución).

- `boton.c`: su función consiste en desplegar en pantalla un botón con la etiqueta "Ir a Casa", que al ser presionado cambia el valor de una bandera que luego será leída por `control.c` a través de la memoria compartida. La prioridad asignada a este programa es 11 y se ejecuta con un periodo igual a 100 mseg.

- `mfiolib.h`: es un archivo de cabecera que contiene las funciones para el uso de la tarjeta de adquisición de datos MFIO-3A.

- `librerias_qnx.h`: es un archivo de cabecera que contiene funciones que utilizan los programas de tiempo real generados por el *Manejador de Procesos*. En este archivo se incluyen los archivos indicados en la tabla I. Si el usuario utiliza alguna cabecera que no se encuentre en la tabla, deberá incluirla también. Este archivo se encuentra ubicado en el mismo directorio que el *Manejador de Procesos* (ver sección de *Manual del usuario*).

Tabla I: Archivos de cabecera contenidos en `librerias_qnx.h`.

Archivo	Descripción
`Ap.h`	Contiene funciones gráficas del Photon.
`conio.h`	Maneja entrada y salida estándar.
`errno.h`	Contiene mensajes de error.
`fcntl.h`	Contiene variables definidas por el QNX.
`math.h`	Contiene funciones matemáticas.
`process.h`	Utilizada para la ejecución de procesos.
`Pt.h`	Contiene funciones gráficas del Photon.
`signal.h`	Utilizada para el cronómetro.
`stdio.h`	Entrada y salida estándar.
`stdlib.h`	Entrada y salida estándar.
`string.h`	Manejo de cadenas.
`sys/kernel.h`	Utilizada por el cronómetro.
`sys/mman.h`	Utilizada para la creación de la memoria compartida.
`sys/proxy.h`	Utilizada para la creación del cronómetro.
`time.h`	Utilizada para el cronómetro.
`unistd.h`	Utilizada para la creación del cronómetro y la memoria compartida.

V.2 Procedimiento para la realización de una sesión experimental

En esta sección se indican los pasos a seguir para llevar a cabo experimentos con el robot, usando el *Manejador de Procesos de Tiempo Real.*

Paso 1. Posicionarse en el directorio `/Manejador`, una vez que el usuario esté dentro de QNX (ver *Manual del usuario*).

`cd Manejador` < *Enter* >

Paso 2. Ejecutar el *Manejador de Procesos* desde la linea de comandos (o bien, desde el ambiente *Photon*).

`./mptr.x` < *Enter* >

Paso 3. Abrir la *carpeta Seleccionar* y desde ahí, se abren los programas y se les asignan prioridades y periodos, según la tabla II:

Tabla II: Asignación de parámetros a los programas de la aplicación a robótica.

Programa	Prioridad	Periodo (mseg)
`tiempo.c`	29	2.5
`control.c`	27	2.5
`grafica1.c`	10	0
`grafica2.c`	10	0
`boton.c`	11	100

Paso 4. Abrir la *carpeta Relacionar* y establecer las siguientes relaciones, que se muestran en la figura 22.

- *Relación 1:* Esta relación tiene como fin transferir el valor de la variable `Tiempo_real` del programa `tiempo.c` a las variables `t`, `x` y `x` de los programas `control.c`, `grafica1.c`, y `grafica2.c` respectivamente. La variable `Tiempo_real` acumula el tiempo transcurrido desde que se inicia el programa `tiempo.c` (los incrementos dependen del periodo con el cual se ejecute el programa, en este caso 2.5 milisegundos).
- *Relación 2:* Transfiere el valor de la variable a graficar en `grafica1.c`. En la figura 22 se muestra como ejemplo `qe1` que representa el error de posición del motor 1.

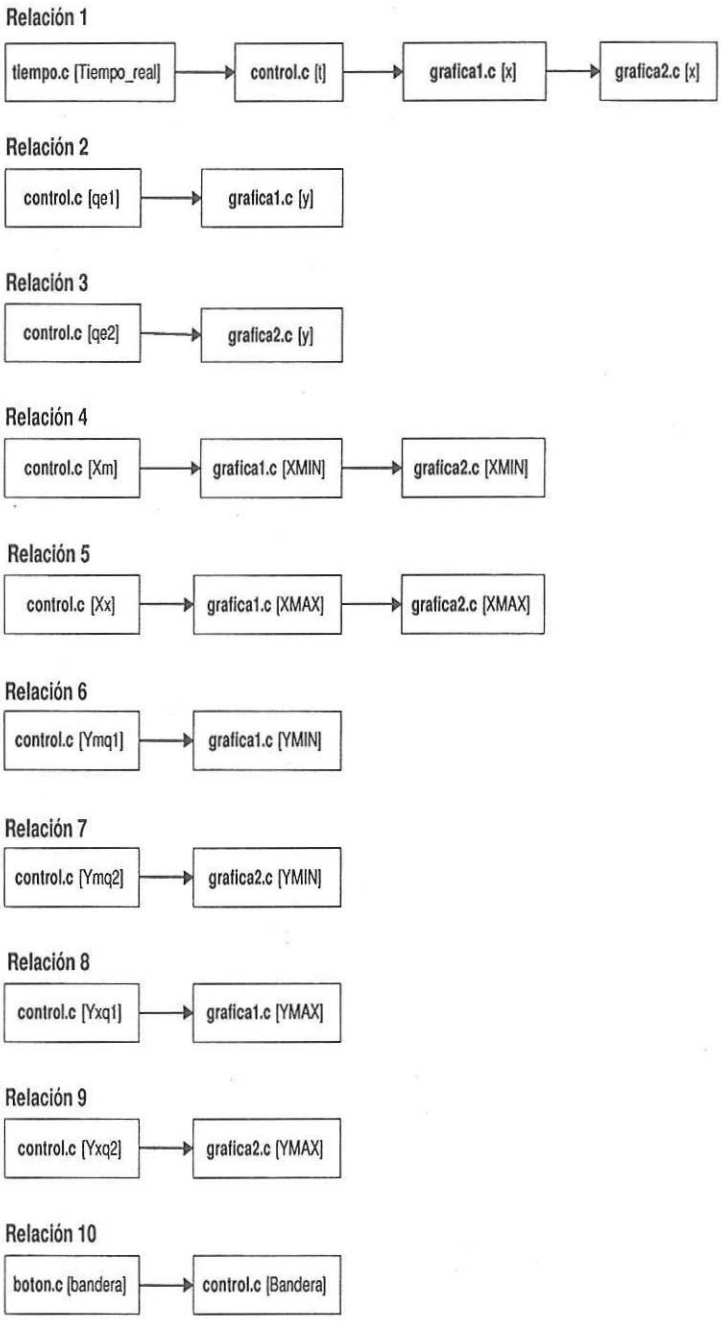

Figura 22: Relaciones de variables para la aplicación a robótica

– *Relación 3:* Transfiere el valor de la variable a graficar en `grafica2.c`. En la figura 22 se muestra como ejemplo `qe2` (error de posición del motor 2).

- *Relaciones 4, 5, 6, 7, 8 y 9:* Estas relaciones representan los valores mínimo y máximo de los ejes que se muestran en las gráficas generadas por `grafica1.c` y `grafica2.c`. En el ejemplo se utilizan las escalas etiquetadas para `qe1` y `qe2` que se definen dentro de `control.c`.
- *Relación 10:* Se lleva a cabo para transferir el valor de la variable `bandera` del programa `boton.c` a `control.c` (ver Figura 22). Esta bandera es activada al presionar el botón e inicia el algoritmo de "ida a casa".

Paso 5. Abrir la *carpeta Variables* para verificar que todas las relaciones estén bien. Si hubo algún error al realizar las relaciones también aquí se puede corregir (ver *Manual del usuario*).

Paso 6. Antes de ejecutar los procesos se debe revisar que todas las conexiones de hardware al robot estén bien, y se deben energizar los drivers de los motores.

Paso 7. Abrir la *carpeta Acciones* y revisar si la asignación de periodos y prioridades es correcta. En caso de ser así, presionar el botón *"Ejecutar"*, para iniciar la ejecución del algoritmo de control. Una vez que se presiona este botón entra en ejecución el algoritmo de control, y el robot se empieza a mover de acuerdo a la tarea especificada por el usuario. Mientras tanto en la pantalla se muestran las gráficas generadas por `grafica1.c` y `grafica2.c`.

Paso 8. En el momento que el usuario lo considere conveniente (cuando se haya concluido la tarea del robot) se debe presionar el botón *"Ir a casa"* para llevar el robot a su posición inicial.

Paso 9. Una vez en *"casa"* se puede volver a iniciar la ejecución de los programas, pero por ningún motivo debe olvidarse el llevar el robot a *casa* antes de reiniciar. Para esto se debe terminar con la ejecución de los programas (presionando el rombo que le corresponde a cada programa en la *carpeta Acciones*) y luego volver al *paso 7*.

Paso 10. Salir del *Manejador de Procesos.* Para llevar a cabo esta acción existen dos formas:

- Presionando el botón rojo con la etiqueta "Detener".
- Posicionarse en el menú *Opciones* y seleccionar *Salir.*

VI CONCLUSIONES Y TRABAJO FUTURO

VI.1 Conclusiones

Esta tesis tuvo como objetivo fundamental el diseño de una aplicación de software que permitiera usar los conceptos de procesamiento en tiempo real, y que finalmente se pudieran valorar sus prestaciones en el área de robótica.

Para lograr este objetivo se consideraron dos requerimientos básicos:

- La aplicación de software debería ser desarrollada en un sistema operativo de tiempo real.
- La validación experimental de la aplicación mediante el control de un robot mecánico.

El primer paso para lograr estos objetivos fue el estudio de la teoría sobre sistemas operativos de tiempo real. También se consideró la adquisición de un par de estos, optándose finalmente por QNX, que sirvió de plataforma para la realización de este trabajo.

La aplicación de software desarrollada recibe el nombre de *Manejador de Procesos de Tiempo Real* y posee las siguientes características:

- Permite la ejecución de varios programas como procesos de tiempo real, los cuales son independientes entre sí y se ejecutan "simultáneamente".
- Maneja programas tipo gráfico (que presentan una ventana gráfica) y tipo texto (que no muestran ningún gráfico).
- Los programas a ser ejecutados pueden ser: procesos síncronos (el programa es ejecutado en forma periódica) o procesos asíncronos (el programa es ejecutado cada vez que el procesador es liberado de la atención a los procesos síncronos).

Por omisión el *Manejador de Procesos de Tiempo Real* les asigna un periodo de 100 mseg a todos los programas.

- Los programas pueden compartir información entre ellos.
- Le presenta al usuario un ambiente amigable de fácil uso.
- El programa del usuario debe estar escrito en el lenguaje de programación ANSI C y cumplir con una estructura (ver sección IV.3.2).
- Se le puede asignar a cada programa de usuario una prioridad dependiendo de la importancia del programa, ésto lo determinará el usuario. Por omisión el *Manejador de Procesos de Tiempo Real* les asigna una prioridad de 10.

Para evaluar la funcionalidad del *Manejador de Procesos de Tiempo Real* en el área de robótica, se decidió probarlo en un brazo mecánico de laboratorio. Se diseñaron un grupo de programas para la ejecución del algoritmo de control y la graficación de resultados. Se hicieron pruebas primero en simulación y luego con el robot real, obteniéndose resultados aceptables.

Para comprobar la efectividad del comportamiento y los resultados obtenidos del experimento utilizando al sistema *Manejador de Procesos de Tiempo Real*, se llevaron a cabo comparaciones contra dos herramientas de software disponibles (SEAC [Jaramillo 99] y SPAC [Reyes 97]) obteniéndose resultados muy similares.

Las ventajas más importantes del *Manejador de Procesos de Tiempo Real* son las siguientes:

- Es un sistema de software que es considerado de propósito general.
- Brinda apoyo como una herramienta de experimentación de algoritmos de control.

VI.2 Trabajo Futuro

Aunque el *Manejador de Procesos* se ha probado que funciona bien, se considera que se puede optimizar. Por ejemplo, la forma como se utiliza la memoria compartida para la transferencia de información entre los procesos no es la más eficiente.

- Optimizar el uso de la memoria compartida.
- Implementar en el Manejador de Procesos otros tipos de calendarizadores.
- Que se pueda interactuar con otras computadoras.

LITERATURA CITADA

Aranda J. y Vignoni R., 1997, *"Sistemas Operativos de Tiempo Real"*. Notas del curso. IV Jornadas Iberoamericanas de Automática. Antigua, Guatemala. Septiembre 1997.

Arzen K., 1999, *"Real-Time Systems"*, Lecture 1 (Engineering Course). http://control.lth.se/~kurstr/material99.html.

Barabanov M. y Yodaiken V., 1996, *"Introducing Real-Time Linux"*. http://www.rtlinux.cs.nmt.edu.

Butazzo G., 1997, *"Hard Real-Time Computing Systems"*. Kluwer Academic Publishers.

Gallmeister B., 1995, *"POSIX.4 Programming for the Real World"*. O'Reilly & Associates, Inc.

Imagination Systems, 1998, *"Hyperkernel"*. http://www.imagination.com.

Jaramillo F., 1999, *"Plataforma para el ensayo de algoritmos de control para un robot manipulador"*. Tesis de maestría, CICESE.

Microtec Division, 1998, *"VRTX Real-Time Operating System"*. http://www.mentorg.com/microtec/brochures/vrtx.html.

Pressman R., 1993, *"Ingeniería del software"*. McGraw Hill.

QNX 1997, *"QNX Operating System"*. QNX Software System Ltd.

Real-Time Innovations, 1998, *"The Component-Based Programming System for Complex Electromechanical Systems ControlShell"*. Real-Time Innovations, Inc.

Reyes F., 1997, *"Control de un robot de transmisión directa de dos grados de libertad"*. Tesis doctoral, CICESE.

Ramos R., Costas C., Kang C.S., Son D.S. y Anigstein M., 1997, *"Procesamiento distribuido en tiempo real. Aplicación a un controlador robótico"*. Memorias de la VII Reunión de trabajo en Procesamiento de la Información y Control (RPIC). San Juan, Argentina. Septiembre 1997.

Stewart y Khosla, 1997, *"Chimera Real-Time Operating System"*. http://www.cs.cmu.edu/ vaschelp/Software/Chimera.

Tanenbaum A., 1993, *"Sistemas Operativos Modernos"*. Prentice Hall Hispanoamericana.

VenturCom, 1999, *"RTX 4.3 User's Guide"*. VenturCom, Inc.

A MANUAL DEL USUARIO DEL *MANEJADOR DE PROCESOS*

El *Manejador de Procesos* es un administrador de procesos de tiempo real bajo el sistema operativo QNX. El sistema brinda al usuario la posibilidad de crear programas de tiempo real a partir de programas escritos en lenguaje C estándar, asignar prioridad y periodo de ejecución a los programas e intercambiar variables entre los diferentes programas con el propósito de transferir información entre ellos.

El presente manual consiste de las siguientes secciones:

- Instalación del *Manejador de Procesos.*
- Operación del *Manejador de Procesos.*
- Estructura general de los programas del usuario.
- Ejemplo de una aplicación a robótica.

A.1 Instalación del *Manejador de Procesos*

Requerimientos. Existen requerimientos mínimos de hardware y software que deberán ser cubiertos antes de poder ejecutar el *Manejador de Procesos.* Esto se hace con el propósito de lograr el funcionamiento adecuado del sistema.

Hardware:

- Pentium, 166 MHz, similar o superior.
- 48 Mb en RAM o mayor.
- Monitor SVGA.

Software:

- Sistema Operativo QNX versión 4.24 (discos Operating System version 4.24 Boot Disk, Install Disk 1, Disk 2, Disk 3 y Disk 4).
- Compilador Watcom C versión 10.6 (disco Watcom C License, CD Watcom C/C++ version 10.6).
- Herramienta de desarrollo "Photon Development" versión 1.13 (discos Photon Development version 1.13 Disk 1, Disk 2, Disk 3 y Disk 4).
- Herramienta de desarrollo "Photon microGUI" versión 1.13 (discos Photon microGUI version 1.13 Disk 1, Disk 2, Disk 3, Disk 4, Disk 5, Disk 6 y Disk 7).

Instalación del *Manejador de Procesos.* Una vez que el usuario encienda la computadora y esté en el sistema operativo QNX, desde la línea de comandos (#) deberá proporcionar cada una de las siguientes instrucciones.

1. `mkdir Manejador` < *Enter* >

   Con la instrucción `mkdir` se crea un directorio. Entonces aquí se está creando el directorio `Manejador`.

2. `cd Manejador` < *Enter* >

   La instrucción `cd` se utiliza para cambio de directorio de trabajo. Con esta instrucción el usuario se moverá al directorio `Manejador`.

3. `Dosfsys &` < *Enter* >

   Esta instrucción se utiliza para reconocer la información que se lea de la unidad `a:` con formato de MS-DOS.

4. `cp /dos/a/mp.gz` < *Enter* >

Con la instrucción cp se pueden copiar archivos. Lo que se hace en esta instrucción es copiar del disco `/dos/a` (unidad `a:` floppy $3\frac{1}{2}$) el archivo `mp.gz`.

5. `gunzip -d mp.gz` < *Enter* >

   La instrucción `gunzip -d` se utiliza para descomprimir archivos con extensión `.gz`. Con esta instrucción se genera un archivo sin extensión llamado `manejador` el cual a su vez agrupa otros archivos.

6. `tar -xvf manejador` < *Enter* >

   La instrucción `tar -xvf` desagrupa los archivos que se encuentran etiquetados por un nombre de archivo.

Con estas serie de instrucciones se da por terminada la instalación del sistema *Manejador de Procesos.*

A.2 Inicio de la sesión

Una vez que el usuario encienda la computadora y esté en el sistema operativo QNX desde la línea de comandos (#) deberá proporcionar los siguientes comandos:

1. `cd Manejador` < *Enter* >

   Con esta instrucción el usuario se posiciona en el directorio `Manejador`.

2. `ph` < *Enter* >

   La instrucción `ph` se utiliza para entrar al ambiente gráfico del QNX conocido como Photon. Para ejecutar el sistema *Manejador de Procesos* es necesario entrar al ambiente gráfico del sistema operativo QNX llamado Photon.

3. Se abre una terminal. Una vez que el usuario se encuentre ubicado dentro del ambiente gráfico Photon (ver figura 23), es necesario abrir una ventana de terminal (shell) y para lograrlo deberá presionar el icono de la figura 24.

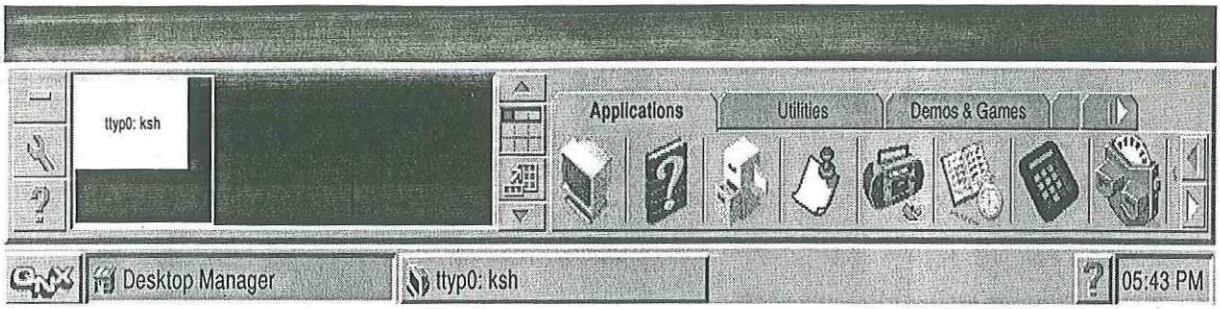

Figura 23: Barra de opciones del ambiente Photon.

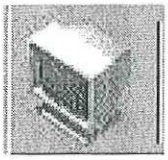

Figura 24: Icono para abrir una terminal en el Photon.

4. `./mptr.x` < *Enter* >

   Con esta instrucción se ejecuta el **Manejador de Procesos.**

A.3 Operación del *Manejador de Procesos*

Se le llama *Manejador de Procesos* al ambiente gráfico que tiene la labor de facilitar la coordinación y comunicación de varios procesos en tiempo real, donde cada uno de los procesos puede ser ejecutado a una cierta cadencia y se le puede asignar una prioridad. Al iniciar la sesión del *Manejador de Procesos* se presenta la interfase mostrada en la figura 25.

Esta contiene:

- Barra de título: Está formada por el nombre del programa y dos botones.
- Barra de menús: Contiene los menús *Opciones* y *Lenguaje.*
- Cejillas de Carpetas: En total son cuatro cejillas y cada una representa una

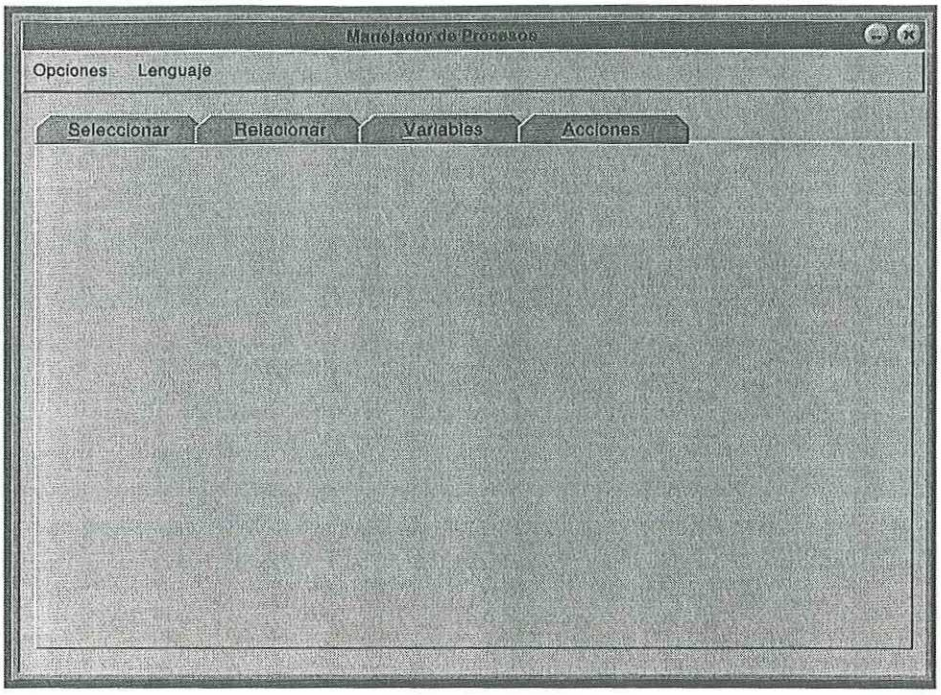

Figura 25: Inicio de sesión del *Manejador de Procesos*

carpeta.

La interfaz del sistema es muy parecido al ambiente *Windows* de Microsoft. Al igual que en *Windows* todas las opciones pueden ser accesadas mediante el uso del ratón y también mediante una combinación de teclas en el caso de las carpetas.

A continuación se describen cada uno de los módulos en que se divide el *Manejador de Procesos.*

Barra título. La conforma una barra que contiene el nombre del programa, es color verde cuando la ventana está activa y de color gris cuando está inactiva. Contiene además dos pequeños botones del lado derecho de la ventana. El icono "-" representa minimizar la ventana y el "×" salirse del programa principal.

Barra de menús.

- *Opciones:* Al seleccionar este menú se despliega una lista de tres opciones, las cuales son Ruta, Inicializar y Salir.

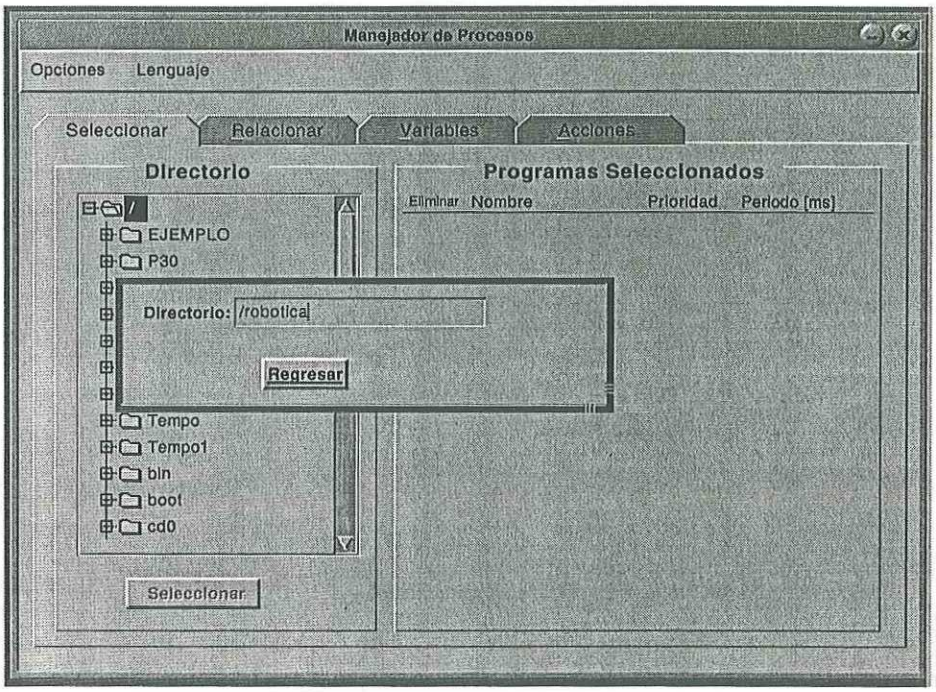

Figura 26: Cambio de directorio de trabajo.

  - *Ruta:* Cuando se selecciona esta opción aparece una cuadro de diálogo (ver figura 26) que consiste en una caja de texto y un botón. La caja de texto contiene el subdirectorio raíz, pero éste puede ser cambiado por el usuario. El cuadro de diálogo contiene además un botón para cerrar la ventana.
  - *Inicializar:* Inicia la sesión. Se recomienda que esta opción se utilice sólo si el usuario desea eliminar los programas que tiene seleccionados y que ya no desee trabajar con ellos.
  - *Salir:* Esta opción termina con la ejecución del sistema.

- *Lenguaje:* Cuando se selecciona este menú se despliega una lista con dos opciones: `Ingles` y `Español`, las cuales corresponden al lenguaje en que se mostrarán todas

las etiquetas y desplegado de texto del *Manejador de Procesos.* El lenguaje por omisión es Español.

3. Cejillas de Carpetas: Cada una de las cejillas Seleccionar, Relacionar, Variables y Acciones representa una carpeta diferente.

- Carpeta Seleccionar: Cuando se selecciona esta carpeta ya sea con el ratón o la combinación de teclas Alt-S, la ventana principal cambia como se muestra en la figura 27. Se observan las secciones *Directorio* y *Programas Seleccionados.*

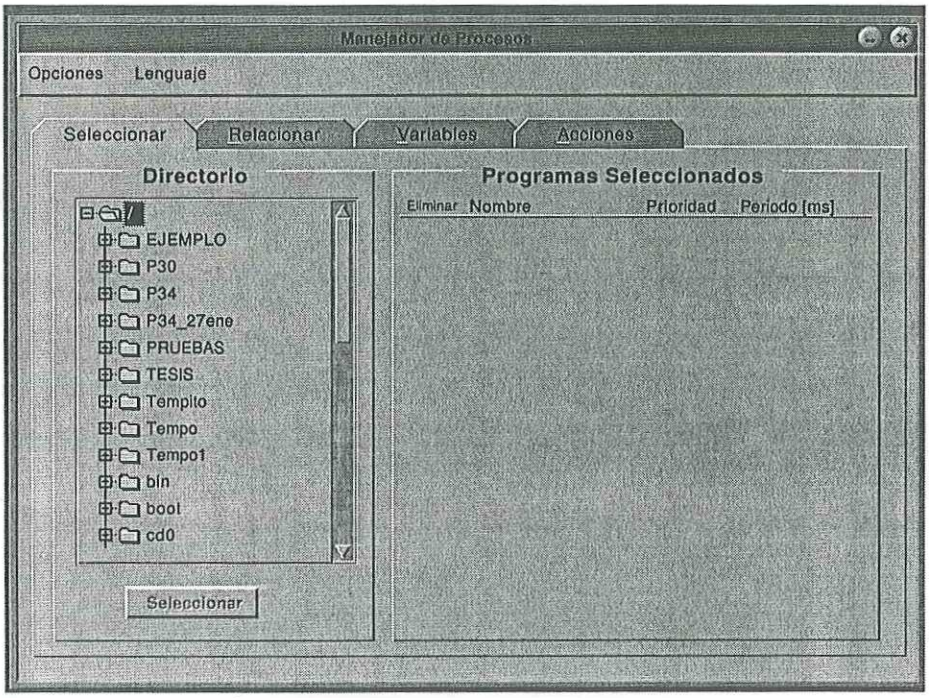

Figura 27: Selección de los programas.

1. *Directorio*: Por omisión muestra el directorio raíz. El usuario deberá buscar el programa deseado en su correspondiente ruta de directorio. Para seleccionarlo basta con posicionarse con el ratón en el icono que representa el programa, el cual se resaltará con un cuadro azul y presionar el botón *Seleccionar* el cual se encuentra ubicado al pie del directorio. Los posibles pro-

gramas a seleccionar deberán tener la estructura que en la siguiente sección se define. Además del uso del ratón se puede utilizar la opción *Ruta* del menú *Opciones* para la localización del directorio en el cual se encuentra el programa a seleccionar (ver el apartado anterior).

2. *Programas Seleccionados*: Presenta la lista de programas que el usuario ha seleccionado. La lista contiene cuatro columnas:
   * Eliminar: Esta columna muestra un rombo por cada programa seleccionado. Cuando el usuario presiona un rombo aparece un cuadro de diálogo solicitando la confirmación para eliminar de la lista el programa del rombo seleccionado.
   * Nombre: Contiene los nombres de los programas seleccionados por el usuario.
   * Prioridad: Contiene las prioridades que les corresponden a los programas seleccionados.
   * Periodo: Contiene los periodos de ejecución de cada uno de los programas seleccionados.

Cuando el usuario selecciona un programa en la sección directorio, éste aparece en la lista con prioridad 10 y periodo de 100 milisegundos, pero estos datos pueden ser fácilmente modificados por el usuario.

Las prioridades válidas están en el intervalo de números enteros del 0 al 29 y su importancia es ascendente, es decir, a mayor valor mayor prioridad. El intervalo para establecer el periodo de ejecución de un programa está entre 0.1 y 999999 milisegundos para programas síncronos. Para programas asíncronos se debe asignar un periodo cero.

Para cambiar los datos de las columnas periodo y prioridad el procedimiento es el siguiente:

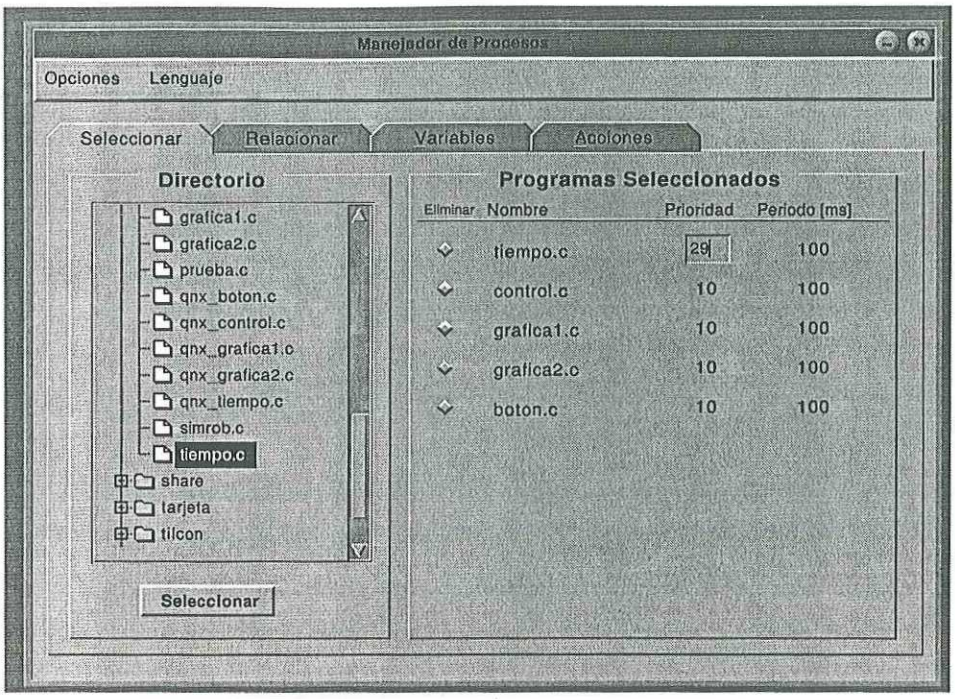

Figura 28: Asignación de prioridad

- Posicionar el cursor del ratón en la columna y fila correspondiente al periodo o a la prioridad que se desea cambiar.
- Cuando aparece el cursor del ratón en forma de mano, deberá presionar el botón izquierdo del ratón. Entonces aparece una caja de texto como se muestra en las figuras 28 y 29, según sea el caso.
- Escribir el nuevo valor y presionar $< Enter >$.

- Carpeta Relacionar: Cuando se selecciona esta carpeta ya sea utilizando el ratón o la combinación de teclas Alt-R, la información que se muestra corresponde a los nombres de los programas seleccionados en la carpeta Seleccionar y sus variables correspondientes. Cada nombre de programa representa un botón y cada variable un punto (ver figura 30).

  El procedimiento para realizar una relación es el siguiente:

  1. Presionar el botón del nombre de programa al que pertenece la variable que

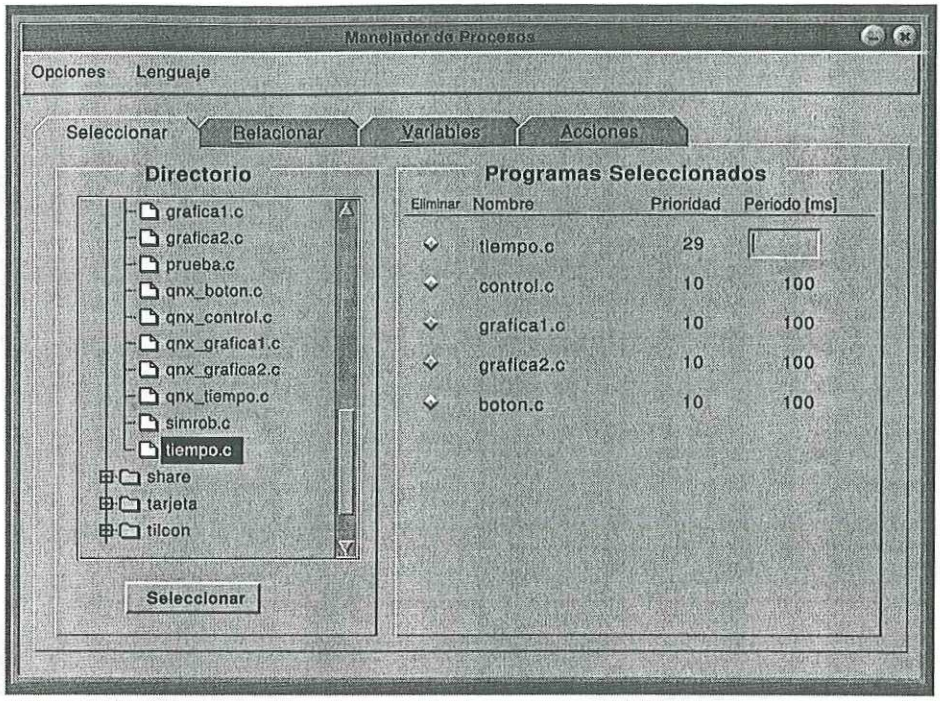

Figura 29: Asignación de periodo.

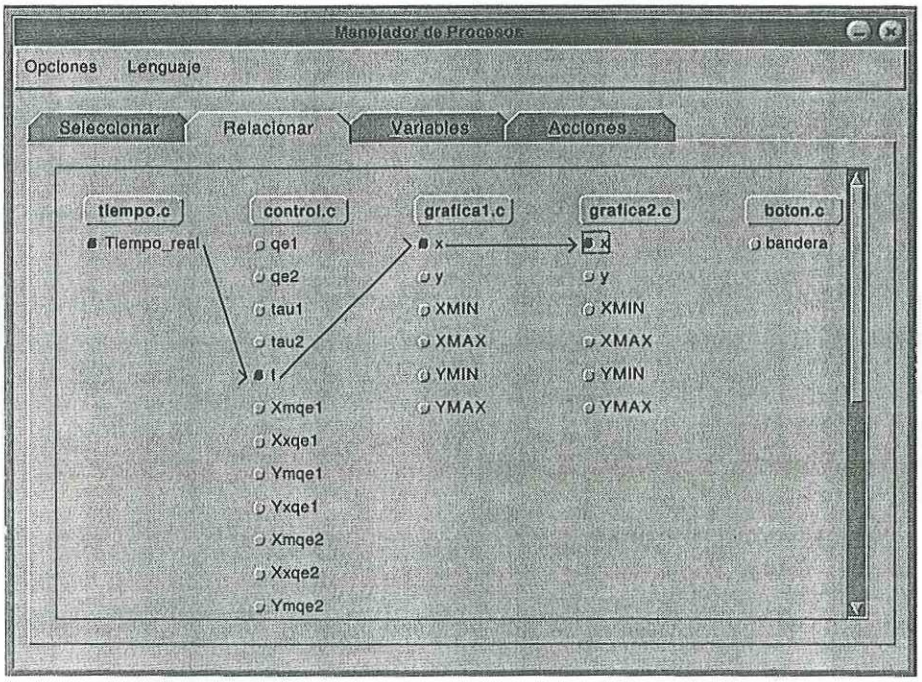

Figura 30: Carpeta para relación de variables.

se desee sea de escritura (la primer variable de la relación).

2. Presionar el punto de la variable de escritura (misma columna del botón del nombre de programa seleccionado).
3. Presionar el punto de la variable de lectura.
4. Si se desea otra variable de lectura vaya al punto 3.

Un ejemplo sencillo se muestra en la figura 30 donde vemos que la variable de escritura es `Tiempo_real` del programa `tiempo.c` y las de lectura serán: `t` del programa `control.c`, `x` del programa `grafica1.c` y `x` del programa `grafica2.c`.

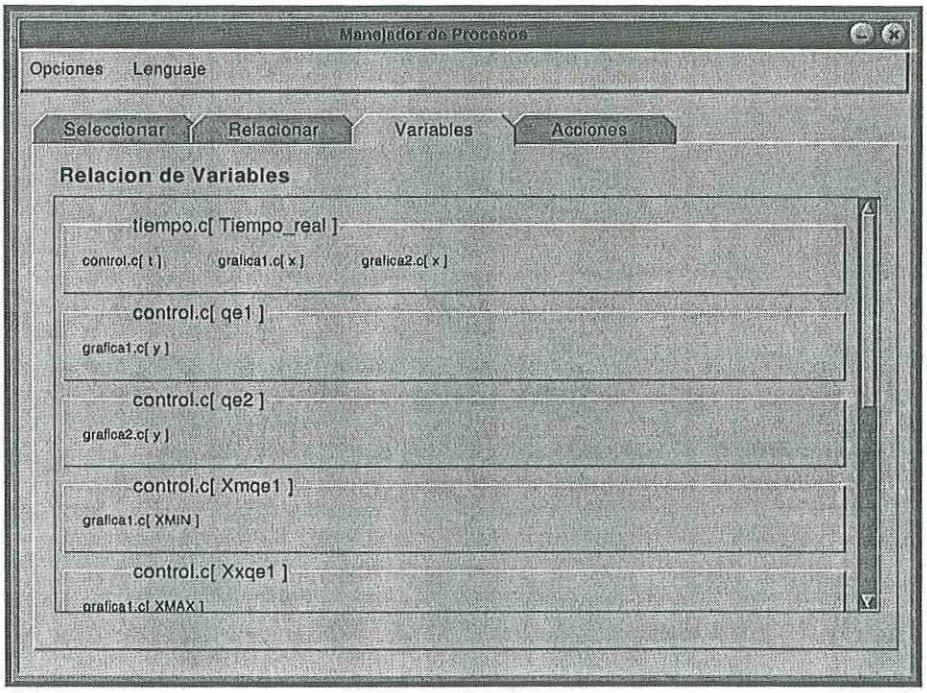

Figura 31: Relaciones de variables

- Carpeta Variables: Cuando se selecciona esta carpeta con el ratón o con la combinación de teclas Alt-V. La información que es muestra corresponde a las relaciones que se llevarón a cabo en la carpeta Relacionar. Las variables relacionadas quedan enmarcadas dentro de un rectángulo. La variable de escritura aparece en la parte

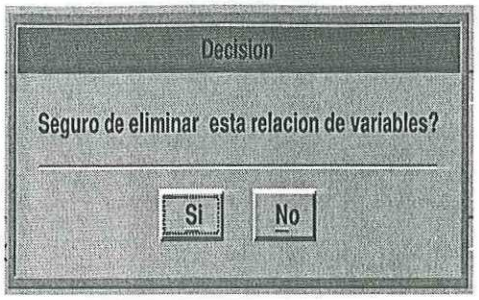

Figura 32: Ventana de confirmación para eliminado de una relación.

superior del rectángulo y las variables de lectura aparecen dentro del rectángulo como se muestra en la figura 31.

En esta carpeta el usuario puede eliminar una relación si así lo desea. El procedimiento es sencillo:

- Posicionarse sobre la variable de escritura de la relación a eliminar, una vez que el cursor del ratón, está posicionado cambia la forma de flecha a una pequeña mano.
- Cuando aparece la mano presionar el botón izquierdo del ratón. Aparece entonces una ventana de verificación como la que se muestra en la figura 32.
- Si el usuario elige la opción afirmativa entonces aparecerá un recuadro gris sobre la variable, lo que indica que esa relación fue eliminada.

- Carpeta Acciones: Cuando se selecciona esta carpeta mediante la combinación de teclas Alt-A o con el ratón se presenta una nueva ventana. La información que se muestra (vea la figura 33) aparece en cuatro columnas:
  - *Nombre*: Esta columna corresponde a los nombres de los programas seleccionados.
  - *Prioridad*: Corresponde a las prioridades de los programas.
  - *Periodo*: Esta columna representa los periodos de ejecución asignados a los programas (en milisegundos).

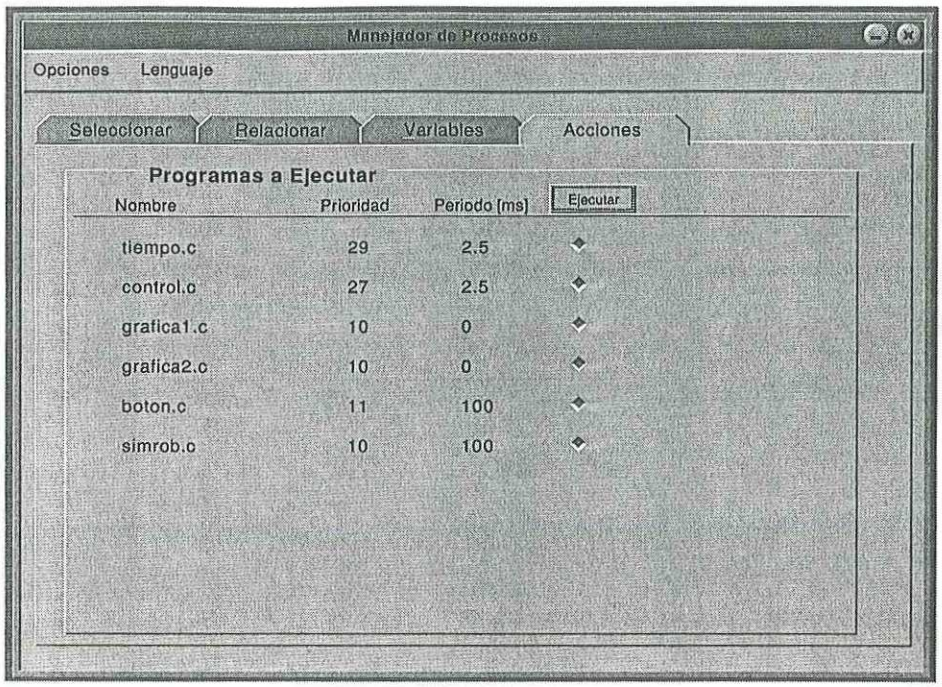

Figura 33: Carpeta para ejecución de programas.

- *Ejecutar*: Esta columna está compuesta por un botón con la etiqueta *Ejecutar* y con rombos:
  - Botón Ejecutar: Su función es mandar ejecutar a los programas que están en la lista.
  - Rombos: A cada programa le corresponde un rombo y su función es detener la ejecución del programa cuando sea seleccionado mediante el ratón.

Es importante mencionar que tanto la carpeta de *Relacionar* como la de *Variables* tienen la característica de mostrar barras de desplazamiento (scroll bars) cuando la presentación de la información sobrepasa el área de la carpeta. En este caso de esta forma el usuario deberá desplazarse mediante las barras a través de la interfaz para tener una mejor visualización.

A.3.1 Estructura de un programa del usuario

Se propone un formato general de programa de usuario, el cual estará dividido principalmente por las siguientes secciones (ver la Figura 34):

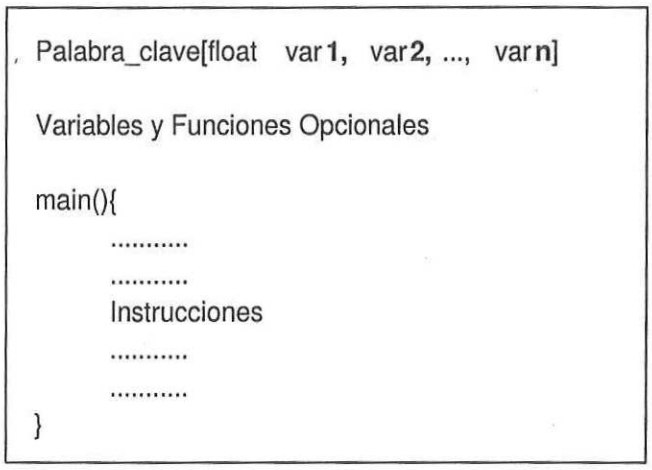

Figura 34: Estructura general de un programa de usuario

- `Palabra_clave[float var1, var2,...,varn]`: Primer sección que debe contener un programa de usuario donde:

  - `Palabra_clave` determina el tipo de programa de usuario, es decir, los programas del usuario podrán ser de tipo texto (`TEXT`) o tipo gráfico (`GRAPHICS`). La diferencia es que los programas tipo `GRAPHICS` pueden desplegar una ventana gráfica, usando herramientas del Photon.

  - `var1, var2, .., varn` representan los nombres de las variables que se registrarán en la memoria compartida las cuales deben ser tipo `float`.

    Estas variables podrán ser de tipo escritura o de tipo lectura, según la utilización que hagan de la memoria compartida.

    A continuación se muestra la sintaxis:

    * Proceso tipo Texto:

`TEXT[float var1,var2,...,varn]`

* Proceso tipo Gráfico:

`GRAPHICS[float var1,var2,...,varn]`

Nota importante: Para el buen funcionamiento del programa se sugiere que las variables declaradas en esta sección por ningún motivo se declaren en alguna otra parte del programa, debido a que esto puede originar una anomalía en el funcionamiento del sistema. Por otro lado las palabras claves `TEXT` y `GRAPHICS` siempre deberán estar escritas en mayúsculas.

- **Variables y Funciones Opcionales**: Esta sección es opcional, ya que dependerá de las necesidades del programa del usuario. El usuario podrá incluir ya sea variables o funciones necesarias para el funcionamiento del programa. Las variables que se declaren en esta sección no podrán ser reconocidas por los demás programas, es decir no serán compartidas.

- **main()**: Es la función principal del programa de usuario la cual en el programa de tiempo real generado por el *Manejador de Procesos*, correspondera a una función que será ejecutada repetitivamente ya sea en forma periódica (síncrona) o no (asíncrona).

La figura 35 muestra un programa de usuario antes de tener la estructura y después de hacer los cambios correspondientes para que cumpliera con dicha estructura. Esto lo debe hacer el usuario antes de ejecutar el *Manejador de Procesos*. En el ejemplo se muestra un programa tipo texto (`TEXT`) con una variable compartida (`suma`). Es importante mencionar que las bibliotecas de funciones se introducen en el archivo `librerias_qnx.h`.

Recomendaciones para el usuario: A continuación se listan algunos aspectos que se sugieren para la elaboración de un *programa de usuario*.

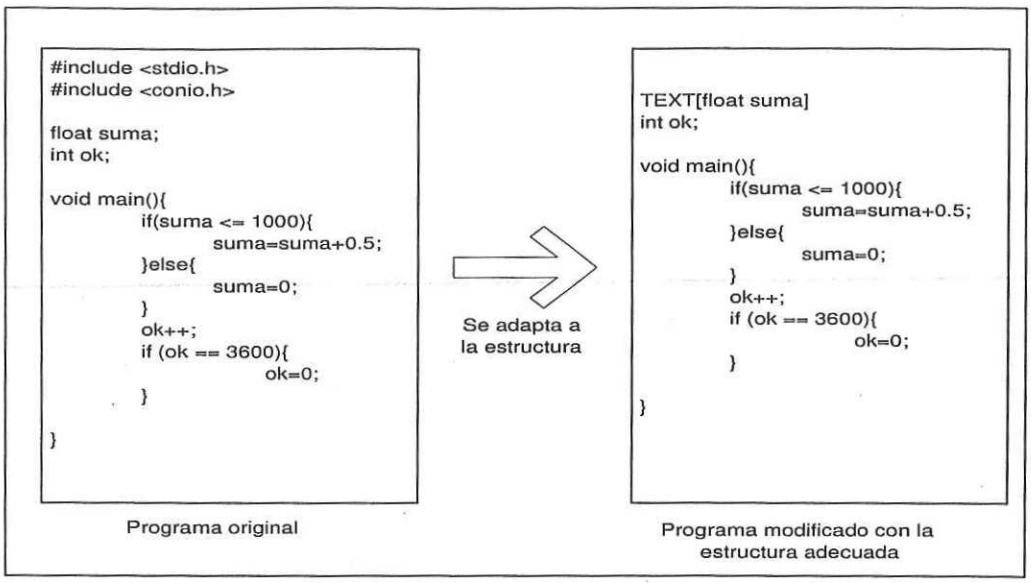

Figura 35: Ejemplo de programa a)estándar; b)con estructura para tiempo real.

- Compilación del programa de usuario. Antes de que un programa tenga la estructura del programa de usuario, deberá ser compilado para revisar que no tenga errores de ningún tipo (sintaxis o lógicos). Para compilar el programa fuente `nombre.c` y obtener el programa ejecutable `nombre.x` (ver *Apendice B*), en QNX la instrucción es la siguiente:

```
cc -o programa.x nombre.c
```

- Archivos de cabecera. El programa de usuario *no* deberá contener funciones que se encuentren en bibliotecas a ser utilizadas sobre el sistema operativo *MS-DOS* o *Windows*. Una de estas bibliotecas es `dos.h` por lo que en el *programa de usuario* deberá quitarse la instrucción.

```
#include <dos.h>
```

- Uso de banderas para control del programa. Como se mencionó anteriormente el `main()` del programa de usuario se transforma en una rutina repetitiva en el programa de tiempo real. Es por ello que se sugiere el uso de banderas cuando

se tiene la necesidad de que una instrucción se ejecute un número determinado de veces. Por ejemplo, no sería muy razonable que un archivo se abra y cierre continuamente en la rutina repetitiva; lo más común, es cuando se desea abrir el archivo al inicio de la ejecución del programa (sólo una vez) y después de una cierta cantidad de iteraciones (por ejemplo: 200 veces) se cierre el archivo, aunque la rutina repetitiva siga ejecutándose indefinidamente. A continuación se muestra la manera en que un programa así sería implementado.

```
TEXT[float ok]
int band=0;
main(){
   if(band==0){
       abrir_archivo();
   }
   band++;
   ...
   Instrucciones
   ...
   if(band == 200){
       cerrar_archivo();
   }
}
```

- Evitar el uso de estructuras de control. Por la misma razón indicada en el punto anterior, se debe evitar la utilización de estructuras de repetición `do-while()`, `while()`, `for()`. No se debe olvidar que una vez convertido en un programa de tiempo real el contenido de `main()` se ejecutará repetitivamente.

A.4 Ejemplo de una aplicación a robótica.

En esta sección se describe la forma de implementar una aplicación a robótica del *Manejador de Procesos*. Para llevar a cabo la aplicación se hará uso de una tarjeta de adquisición de datos y un brazo mecánico de dos articulaciones. Para el caso particular de esta tesis se utilizó lo siguiente:

Requerimientos de hardware

- Tarjeta de adquisición de datos MFIO-3A del fabricante Precision MicroDynamics.
- Brazo mecánico de dos articulaciones del CICESE (ver figura 36).

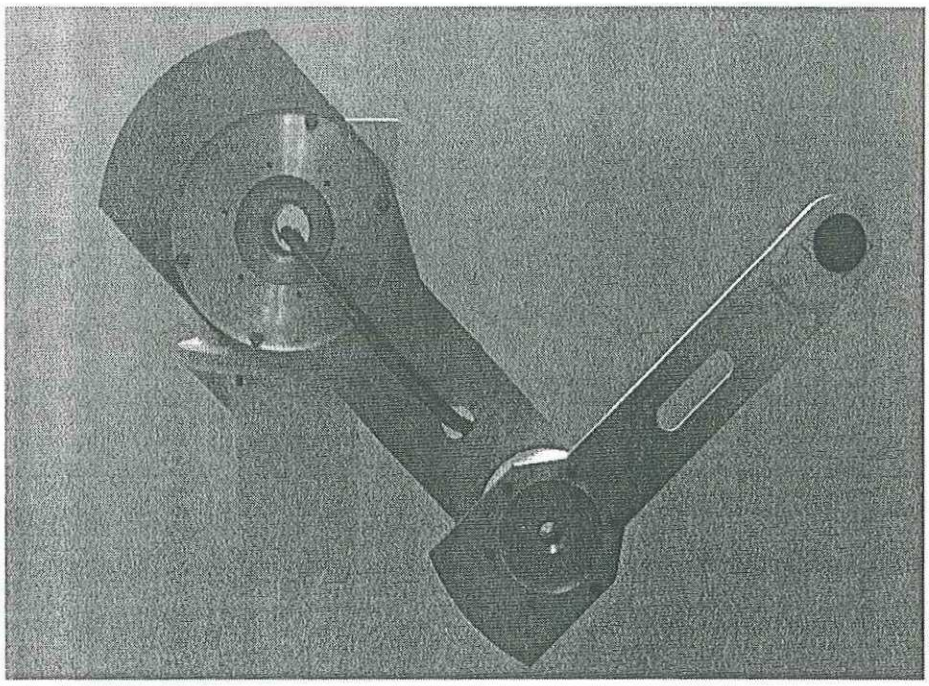

Figura 36: Brazo mecánico del laboratorio

Requerimientos de software. Si el usuario desea utilizar en su programa algún archivo de cabecera deberá incluirla en el archivo llamado `librerias_qnx.h` ubicado en el subdirectorio Manejador.

Para llevar a cabo la aplicación para robótica se considerarón los programas de la siguiente lista. Para tener una idea del objetivo de cada uno de estos programas a continuación se da una breve descripción.

- `tiempo.c`: este programa tiene la tarea de llevar la cuenta del tiempo real transcurrido desde que inicia la ejecución del algoritmo. Se le asigna la prioridad más

alta (29) debido a que el tiempo transcurrido es de suma importancia para el algoritmo de control así como para los programas que grafican. Se le asigna un periodo de ejecución de 2.5 milisegundos (lo mismo que al algoritmo de control).

- `control.c`: este es el programa que contiene el algoritmo de control, el cual permite la ejecución y manipulación del robot. En general lleva a cabo las siguientes acciones:

  1. Lee la posición del motor de cada una de las articulaciones del robot.
  2. Estima la velocidad de cada articulación.
  3. Ejecuta un algoritmo de control para determinar los pares necesarios para cada uno de los motores.
  4. Aplica los pares calculados a los motores a través de la tarjeta de adquisición.

  Una vez realizada la tarea deseada, se debe regresar el robot a su posición original y ésto se lleva a cabo cambiando el algoritmo de control a uno que lleva al robot a su posición de casa (home). Para lograr esto es necesario utilizar una variable que cambie su valor para indicar el cambio de algoritmo de control. Este programa se ejecuta con un periodo de 2.5 milisegundos.

- `grafica1.c`: el objetivo de este programa es desplegar en la pantalla una gráfica con las variables que lee de la memoria compartida, como son las coordenadas X-Y y los valores máximos y mínimos de las escalas de la gráfica. Se le asigna la prioridad 10, que es el valor asignado por omisión por el QNX.

- `grafica2.c`: Este programa es una copia del programa `grafica1.c`, con diferente nombre. Tiene que ser así ya que el *Manejador de Procesos* no puede seleccionar dos programas que se llamen igual. Tanto `grafica1.c` y `grafica2.c` son programas asíncronos (sin periodo de ejecución).

- `boton.c`: su función consiste en desplegar en pantalla un botón con la etiqueta "Ir a Casa", que al ser presionado cambia el valor de una bandera que luego será leída por `control.c` a través de la memoria compartida. La prioridad asignada a este programa es 11 y se ejecuta con un periodo igual a 100 mseg.

- `mfiolib.h`: es un archivo de cabecera que contiene las funciones (drivers) para el uso de la tarjeta de adquisición de datos MFIO-3A.

- `librerias_qnx.h`: es un archivo de cabecera que contiene funciones que utilizan los programas de tiempo real generados por el *Manejador de Procesos*. En este archivo se incluyen las funciones indicadas en la tabla III. Si el usuario utiliza alguna cabecera que no se encuentre en la tabla, deberá incluirla también. Este archivo se encuentra ubicado en el mismo directorio que el *Manejador de Procesos*.

A.5 Precauciones

Cuando se ejecuta un programa de tiempo real es importante tener en cuenta que pueden surgir algunas situaciones no deseadas. A continuación se describen las más comúnes:

- Programas con misma prioridad. En el QNX existe un administrador de procesos, el cual entre otras cosas se encarga de verificar las prioridades de los mismos (cuando se presenta el caso de que dos procesos tengan la misma prioridad, el administrador asigna un intervalo de tiempo o quantum a cada programa). En caso de que el proceso en ejecución no termine en el tiempo que le fue asignado, el administrador le asigna el procesador al otro proceso de la misma prioridad también por un intervalo de tiempo. Si esté tampoco termina en el tiempo asignado, el administrador asigna el procesador al primer proceso y se vuelve a repetir el mismo caso.

Tabla III: Archivos de cabecera contenidas en `librerias_qnx.h`.

Archivo	Descripción
`Ap.h`	Contiene funciones gráficas del Photon.
`conio.h`	Maneja entrada y salida estándar.
`errno.h`	Contiene mensajes de error.
`fcntl.h`	Contiene variables definidas por el QNX.
`math.h`	Contiene funciones matemáticas.
`process.h`	Utilizada para la ejecución de procesos.
`Pt.h`	Contiene funciones gráficas del Photon.
`signal.h`	Utilizada para el cronómetro.
`stdio.h`	Entrada y salida estándar.
`stdlib.h`	Entrada y salida estándar.
`string.h`	Manejo de cadenas.
`sys/kernel.h`	Utilizada por el cronómetro.
`sys/mman.h`	Utilizada para la creación de la memoria compartida.
`sys/proxy.h`	Utilizada para la creación del cronómetro.
`time.h`	Utilizada para el cronómetro.
`unistd.h`	Utilizada para la creación del cronómetro y la memoria compartida.

- Programas síncronos con tiempo de ejecución mayor que periodo de ejecución. Es importante que el usuario no asigne un periodo de ejecución menor que el tiempo que se tarda en ejecutarse el programa, debido a que provocaría errores en el sistema y posiblemente resultados no deseados e incorrectos.

- Programa asíncrono con prioridad mayor que programas síncronos. Como un programa síncrono realiza una actividad cada cierto periodo (por ejemplo cada

2.5 mseg) y uno asíncrono cada vez que el procesador le sea asignado, entonces, los resultados del programa síncrono no serán los deseados debido a que entrará cada vez que se le sea permitido y no cuando el lo requiera ya que quien ocupará el procesador sería la mayor parte del tiempo el programa asíncrono.

A.6 Procedimiento para la realización de una sesión experimental

En esta sección se indican los pasos a seguir para llevar a cabo experimentos con el robot, usando el *Manejador de Procesos de Tiempo Real.*

Paso 1. Posicionarse en el directorio `/Manejador`, una vez que el usuario esté dentro de QNX (ver *Manual del usuario*).

`cd Manejador` < *Enter* >

Paso 2. Ejecutar el *Manejador de Procesos* desde la linea de comandos (o bien, desde el ambiente *Photon*).

`./mptr.x` < *Enter* >

Paso 3. Abrir la *carpeta Seleccionar* y desde ahí, se abren los programas y se les asignan prioridades y periodos, según la tabla IV:

Paso 4. Abrir la *carpeta Relacionar* y establecer las siguientes relaciones, que se muestran en la figura 37.

- *Relación 1:* Esta relación tiene como fin transferir el valor de la variable `Tiempo_real` del programa `tiempo.c` a las variables `t`, `x` y `x` de los programas `control.c`, `grafica1.c`, y `grafica2.c` respectivamente. La variable `Tiempo_real` acumula el tiempo transcurrido desde que se inicia el programa

Tabla IV: Asignación de parámetros a los programas de la aplicación a robótica.

Programa	Prioridad	Periodo (mseg)
`tiempo.c`	29	2.5
`control.c`	27	2.5
`grafica1.c`	10	0
`grafica2.c`	10	0
`boton.c`	11	100

`tiempo.c` (los incrementos dependen del periodo con el cual se ejecute el programa, en este caso 2.5 milisegundos).

- *Relación 2:* Transfiere el valor de la variable a graficar en `grafica1.c`. En la figura 37 se muestra como ejemplo `qe1` que representa el error de posición del motor 1.
- *Relación 3:* Transfiere el valor de la variable a graficar en `grafica2.c`. En la figura 37 se muestra como ejemplo `qe2` (error de posición del motor 2).
- *Relaciones 4, 5, 6, 7, 8 y 9:* Estas relaciones representan los valores mínimo y máximo de los ejes que se muestran en las gráficas generadas por `grafica1.c` y `grafica2.c`. En el ejemplo se utilizan las escalas etiquetadas para `qe1` y `qe2` que se definen dentro de `control.c`.
- *Relación 10:* Se lleva a cabo para transferir el valor de la variable `bandera` del programa `boton.c` a `control.c` (ver Figura 37). Esta bandera es activada al presionar el botón e inicia el algoritmo de "ida a casa".

Paso 5. Abrir la *carpeta Variables* para verificar que todas las relaciones estén bien. Si hubo algún error al realizar las relaciones también aquí se puede corregir (ver *Manual del usuario*).

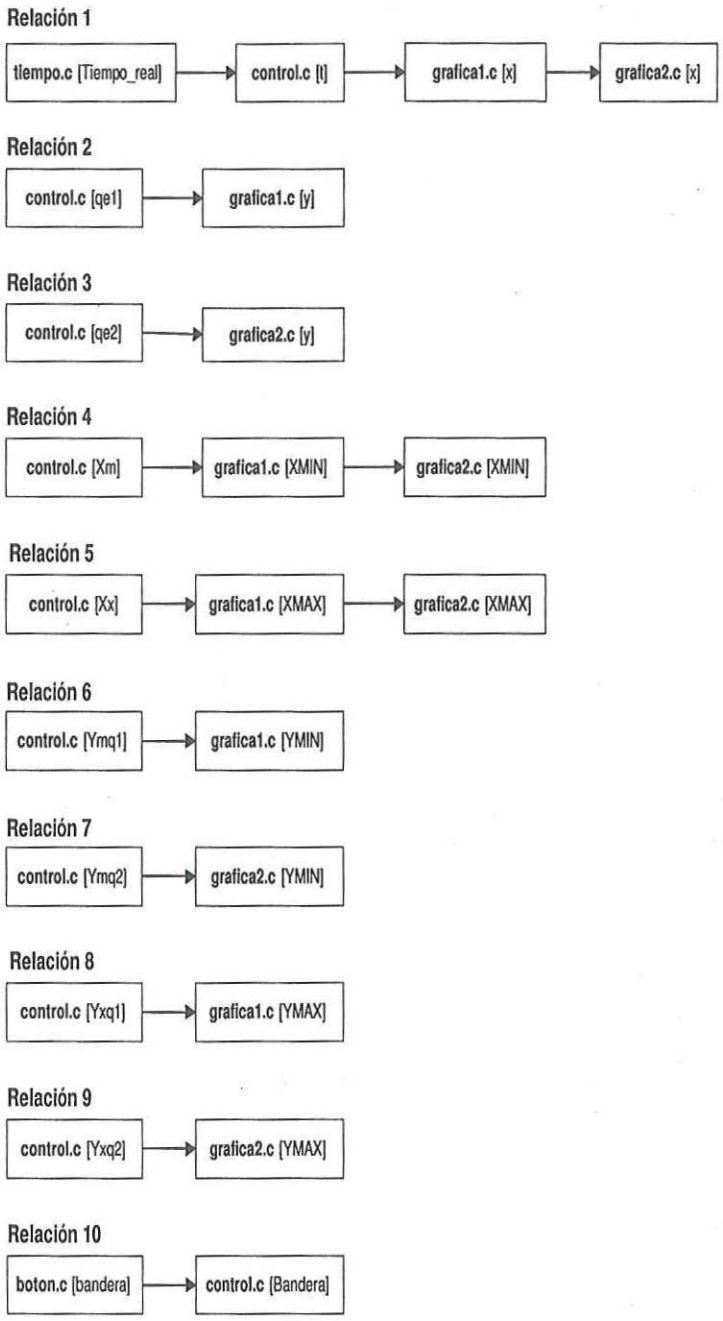

Figura 37: Relaciones de variables para la aplicación a robótica

Paso 6. Antes de ejecutar los procesos se debe revisar que todas las conexiones de hardware al robot estén bien, y se deben energizar los drivers de los motores.

Paso 7. Abrir la *carpeta Acciones* y revisar si la asignación de periodos y prioridades es correcta. En caso de ser así, presionar el botón *"Ejecutar"*, para iniciar la ejecución del algoritmo de control. Una vez que se presiona este botón entra en ejecución el algoritmo de control, y el robot se empieza a mover de acuerdo a la tarea especificada por el usuario. Mientras tanto en la pantalla se muestran las gráficas generadas por `grafica1.c` y `grafica2.c`.

Paso 8. En el momento que el usuario lo considere conveniente (cuando se haya concluido la tarea del robot) se debe presionar el botón *"Ir a casa"* para llevar el robot a su posición inicial.

Paso 9. Una vez en *"casa"* se puede volver a iniciar la ejecución de los programas, pero por ningún motivo debe olvidarse el llevar el robot a *casa* antes de reiniciar. Para esto se debe terminar con la ejecución de los programas (presionando el rombo que le corresponde a cada programa en la *carpeta Acciones*) y luego volver al *paso 7*.

Paso 10. Salir del *Manejador de Procesos*. Para llevar a cabo esta acción existen dos formas:

- Presionando el botón rojo con la etiqueta "Detener".
- Posicionarse en el menú *Opciones* y seleccionar *Salir*.

B Ejemplos de programas de tiempo real

Se crearon los programas `write.c` y `read.c` con el proposito de mostrar de una manera más clara la forma en que los programas bajo QNX pueden compartir valores, así como también como se asigna la prioridad y el periodo de ejecución a un programa.

Programa síncrono `write.c`: El objetivo de este programa es crear un objeto de memoria compartida con un dato tipo entero. Se ilustra además la asignación de prioridad y el uso de un cronómetro.

```
// Programa que crea un objeto de memoria compartida y escribe un
//valor tipo entero en la memoria. En las variables periodo y prioridad
//se asignan los valores correspondientes.
//Se crea tambien un cronometro mediante el cual se establece que el
//programa se ejecutara a la cadencia establecida por periodo

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <sys/mman.h>          //Para crear memoria compartida
#include <fcntl.h>
#include <time.h>
#include <signal.h>            //Para terminar ejecucion de un proceso
#include <sys/proxy.h>         //Para crear proxy
#include <sys/kernel.h>        //Para crear cronometro


#define MILLON 1000000

main(){
  int periodo=5;               //5 milisegundos
  int prioridad = 15;          //Prioridad que se le asignara
  int  fd, i=0;
  char c;
  int *memo;                   //Tipo de dato de la memoria compartida
  pid_t proxy;                 //Variables para crear un Cronometro
  timer_t tid;
  struct itimerspec timer;     //Estructuras usadas por el cronometro
  struct sigevent event;
```

```
                              //Esta funcion establece la prioridad
                              //al programa
setprio(0,prioridad);

                              //Con estas instrucciones se esta
                              //creando el cronometro
proxy=qnx_proxy_attach(0,0,0,-1);

event.sigev_signo=-proxy;  //liga un proxy a este proceso

                              //liga un proxy al timer
tid= timer_create(CLOCK_REALTIME, &event);

timer.it_value.tv_sec=1L;
timer.it_value.tv_nsec=0L;
timer.it_interval.tv_sec=0L;

                              //Se le indica al cronometro el
                              //periodo de ejecucin del programa
timer.it_interval.tv_nsec=periodo*MILLON;

                              //Inicializa el cronometro
timer_settime( tid,0,&timer,NULL);

                              //Instrucciones para crear la memoria
   //compartida. Donde X es el nombre de
                              //esta memoria
fd=shm_open("/X",O_RDWR|O_CREAT,0777);

if(fd == -1){
     printf("Error en crear memoria\n"); exit(1);
}

if(ltrunc(fd, 100, SEEK_SET) == -1){
     printf("Error en truncar");exit(1);
}

memo=mmap(0,100,PROT_READ|PROT_WRITE,MAP_SHARED, fd, 0);
if(memo == (void *) -1){
     printf("Error mmap()\n"); exit(1);
}
```

```
                              //Ciclo principal del programa
  for(;;){
                              //Entrara a la estructura if() cada
                              //vez que se cumpla el periodo
                              //Si se recibe la senal del proxy
        if(Receive(proxy, NULL, 0) == proxy){
             Reply(proxy,NULL,0);//Responde al proxy
             i++;             //Incrementa variable
             memo[0]=i;       //Se escribe valor en memoria
             printf("v-> %d", memo[0],i); //Imprime valor
             flushall();
             if(i >= 20000)  i=0;
             if (kbhit()){
                 printf("adiosito..."); break;
             }
        }
  }

                             //Instrucciones para cerrar la memoria
  close(fd);
  shm_unlink("/X");      //Elimina el objeto de memoria
}
```

Programa asíncrono `read.c`: El objetivo de este programa es leer de una memoria creada por el programa `write.c` un tipo de dato entero.

```
// Programa que lee de un objeto de memoria compartida un valor
//tipo entero y lo despliega en pantalla.
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>    //Se utiliza para la memoria compartida
#include <fcntl.h>

main(){
  int  fd, i=0;
  char c;
  int *memo;             //Tipo de dato de la memoria compartida
```

```
                        //Para  acceso a la memoria compartida
                        //donde X es el nombre del objeto de memoria
  fd=shm_open("/X",O_RDWR|O_CREAT,0777);

  if(fd == -1){
     printf("E. crear  SM\n");
     exit(1);
  }
                        //Mapeo de la memoria compartida
  memo=mmap(0,100,PROT_READ|PROT_WRITE,MAP_SHARED, fd, 0);

  if(memo == (void *) -1){
     printf("E. map\n");
     exit(1);
  }
                        //Ciclo principal del programa
  for(;;){
      printf("valor -> %d \t",memo[0]);
      if (kbhit()){
         printf("adiosito...");
         break;
      }
  }
}
```

B.1 Pasos para iniciar la sesión

Paso 1. Para crear el programa `write.c` se requiere:.

- Abrir un terminal (shell) (ver *Manual del usuario*).
- Desde la linea de comando (#), el usuario deberá introducir lo siguiente:

  `vedit write.c <Enter>`.

  Donde `vedit` es el editor de texto de QNX.
- Escribir el programa.
- Salir del editor.

Paso 2. Compilar programa `write.c` para crear el ejecutable (`write.x`).

```
cc -o write.x write.c <Enter>
```

Paso 3. Crear y compilar el programa `read.c` (seguir el paso 1 y 2).

Paso 4. Para poner en ejecución cada uno de los programas, el usuario deberá verificar que estén abiertas dos ventanas tipo terminal (shell).

Paso 5. En la línea de comando de una de las terminales, el usuario debera ejecutar el programa `write.x` mediante la siguiente instrucción:

```
./write.x <Enter>
```

Paso 6. Seleccionar la otra ventana terminal y ejecutar el programa `read.x` (ver paso 5).

Paso 7. Para terminar un programa, el usuario deberá seleccionar la terminal donde se está ejecutando el programa y presionar una tecla (es el mismo procedimiento para los dos casos).

C USO DE LA TARJETA DE ADQUISICIÓN DE DATOS

Para la aplicación a robótica del *"Manejador de Procesos de Tiempo Real"*, se probaron dos tarjetas de adquisición diferentes: la ServoToGo de la compañía del mismo nombre, y la MFIO-3A de Precision MicroDynamics. Debido al mejor desempeño y disponibilidad de ésta última se decidió utilizarla en forma definitiva.

La MFIO-3A es una tarjeta de adquisición de datos multifuncional de entrada/salida, con aplicaciones en control de movimiento, que se conecta a la computadora personal a través del bus ISA. Cuenta con 3 canales D/A de 16 bits, 3 entradas para codificadores incrementales, 24 bits programables de entrada/salida digital, lectura y escritura de datos sincronizadas, un reloj programable y un reloj centinela.

C.1 Funciones para manejo de la tarjeta

Una tarjeta de adquisición es un dispositivo que permite la comunicación entre un sistema de software y el mundo real. La tarjeta se instala en la PC y permite el paso de información (datos digitales o analógicos) entre la computadora y el exterior.

El acceso a la tarjeta adquisidora se logra a través de funciones de entrada/salida, generalmente escritas en lenguaje C, que son proporcionadas por el fabricante, y por lo tanto son propias de cada tarjeta.

Sin embargo, con el proposito de establecer una generalidad en el uso de tarjetas de adquisición de datos y de estas funciones, se han utilizado un conjunto de cinco funciones básicas en lenguaje C, para el manejo del robot y, en general, de servomotores de transmisión directa. El nombre de estas funciones y una breve descripción son dados a continuación:

- `void INICIALIZAR_TARJETA()`

Esta función deberá inicializar la tarjeta de aquisición de datos, es decir, configurar un puerto como salida, inicializar los motores (apagados), los encoders y los DACs.

- `void HABILITAR_SERVO(short motor)`

  Habilita el servo indicado por la variable `motor`.

- `void APAGAR_SERVO(short motor)`

  Apaga el servo indicando por la variable `motor`.

- `float LEER_POSICION_GRADOS(short motor, long PPR)`

  Regresa la posición (en grados) del motor `motor` y `PPR` indica el numero de pulsos del encoder por revolución del motor.

- `short ESCRIBIR_PAR_NM ( float Par_Nm, short motor, float Lim_sat, float Conv_Nm_V )`

  Escribe en el DAC del `motor` el valor correspondiente de voltaje de acuerdo al par solicitado (`Par_Nm`) y al factor de conversión del motor (`Conv_Nm_V`). También se considera el límite de saturación (`Lim_sat`) que de ocurrir pone a "1" el valor regresado por la función y en caso contrario es "0".

C.2 Procedimiento para adaptar una nueva tarjeta

1. Instalar tarjeta de adquisición. El usuario deberá seguir el procedimiento de instalación correspondiente, indicado por el fabricante.

2. Verificar compatibililidad de funciones entrada/salida de la tarjeta en QNX. El usuario deberá verificar que las funciones que el fabricante ofrece para el uso de la tarjeta sean compatibles con la versión de Watcom C para QNX. Esto se lleva a cabo con el propósito de evitar errores en el proceso de compilación.

Es importante decir que, en general, las funciones entrada/salida básicas usadas en el lenguaje C estándar (ANSI) son compatibles entre diferentes versiones del compilador. Sin embargo, en algunas ocasiones se utilizan macros o redefinición de funciones, por lo que hay que identificar las diferencias y ajustar a la versión de Watcom C utilizada.

Nota: Si se desea un sistema portable entre QNX y Windows debe tenerse en cuenta que es más fácil adaptar las funciones de Windows a QNX que lo contrario.

3. Adaptar funciones para manejo de servomotores. Una vez que se ha revisado la compatibilidad de las *funciones* de entrada/salida para la tarjeta en cuestión, se deben crear las cinco funciones correspondientes para el manejo de los servomotores (ver sección anterior). Estas funciones deben ser agrupadas en un archivo de cabecera. Para el caso de la tarjeta MFIO-3A el archivo de cabecera correspondiente es `mfiolibq.h`.

4. Incluir el archivo de funciones en `librerias_qnx.h`. El usuario deberá incluir el archivo creado en el paso anterior como archivo de cabecera en `librerias_qnx.h`. En este archivo se encuentran todas las funciones y archivos de cabecera utilizados por el *Manejador de Procesos* (ver sección V.1).

C.3 Algoritmos para prueba de intervalos de tiempo

Con el propósito de verificar los tiempos de ejecución de los programas, se llevaron a cabo dos tipos de pruebas:

1. Utilizando funciones del compilador Watcom C que capturan el tiempo del sistema. Se utilizaron las funciones `qnx_getclock()` y `clock_gettime()` con el propósito de medir el periodo de ejecución de los procesos y comprobar si se cumplían los plazos establecidos. El problema al utilizar estas funciones es que utilizan el reloj del sistema que es actualizado cada 55 milisegundos, por lo que no se puede utilizar en procesos que tengan periodos de ejecución menores a ese valor.

2. Utilizando un dispositivo externo al sistema de cómputo. El utilizar un osciloscopio para la verificación de los tiempos de ejecución de cada proceso resultó la mejor opción debido a que:

1. Brinda la posibilidad de verificar tiempos muy pequeños.
2. Es totalmente externo a lo que deseamos medir y no introduce ningún retardo.

Se implementarán dos tipos de pruebas. En una de ellas se utiliza un solo programa (`Hab_Apa.c`) y en la otra se ejecutan dos programas (`Habilitar.c` y `Apagar.c`) simultáneamente.

`Hab_Apa.c`: Este programa es ejecutado en forma periódica y tiene la tarea de habilitar el motor que se le indique si este se encuentra apagado o apagarlo si se encuentra habilitado. El comportamiento de este programa podemos observarlo en la figura 38, en el que se ha supuesto un periodo de ejecución de 0.5 milisegundos.

Por simplicidad, a continuación se muestra únicamente la función `main()` de `Hab_Apa.c`.

```
main(){
   ESTABLECER_PRIORIDAD(prioridad); //Establece prioridad del proceso
   ESTABLECER_PERIODO(periodo);     //Crea cronometro
```

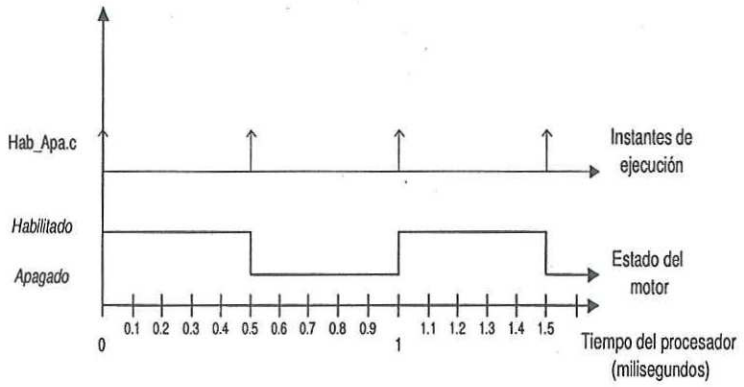

Figura 38: Ejecución del programa Hab_Apa.c.

```
    INICIALIZAR_TARJETA();        //inicializa el uso de la tarjeta

                                  //Ciclo de ejecucion
    for(;;){
      if(Receive(proxy, NULL, 0) == proxy){ //Si se recibe senal
         Reply(proxy,NULL,0);               //del timer
         if(band == 0){
              HABILITAR_SERVO(1);  //Habilita el motor 1
              band=1;
         }else{
              APAGAR_SERVO(1);     //apagar el motor 1
              band=0;
         }
      }
    }
}
```

Habilitar.c y Apagar.c: Estos programas tienen el mismo periodo (0.5 milisegundos en el ejemplo de la figura 39) y se ejecutan en forma "simultánea". El programa Habilitar.c tiene la función de habilitar el motor 1 y el programa Apagar.c tiene la tarea de apagar dicho motor. Cabe mencionar que primero es puesto en ejecución el programa Habilitar.c y luego el programa Apagar.c y tienen la misma prioridad (10 por default), es por ello que el motor estará apagado la

mayor parte del tiempo como se puede observar en la figura 39. La duración de encendido del motor depende de un ciclo `for` que se agregó a `Habilitar.c`

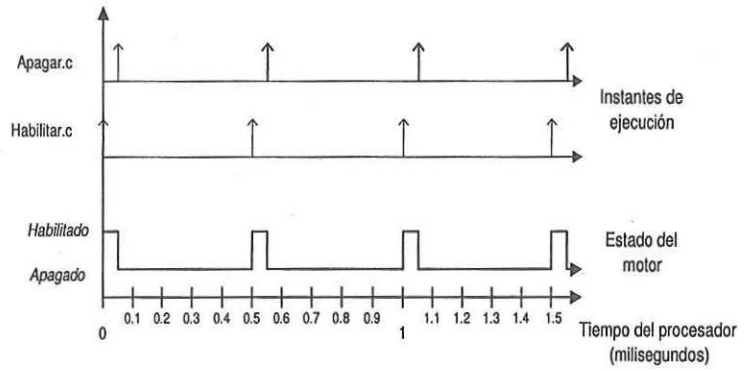

Figura 39: Ejecución de los programas `Habilitar.c` y `Apagar.c`.

El codigo fuente del programa `Habilitar.c` es el siguiente:

```
main(){

   ESTABLECER_PRIORIDAD(prioridad);    //Establece la prioridad
                                       //del proceso

   ESTABLECER_PERIODO(periodo);        //Crea cronometro

   INICIALIZAR_TARJETA();              //inicializa uso
                                       //de la tarjeta

                                       //Ciclo de ejecucion
   for(;;){
     if(Receive(proxy, NULL, 0) == proxy){//Si se recibe senal
        Reply(proxy,NULL,0);                //del timer
          HABILITAR_SERVO(1);  //Habilita el motor 1

          for(i=0;i<1000;i++)  // Retardo, para poder
                               // observar en el osciloscopio
     }
   }
}
```

El codigo fuente del programa Apagar.c es el siguiente:

```
main(){

   ESTABLECER_PRIORIDAD(prioridad);    //Establece la prioridad
                                       //del proceso

   ESTABLECER_PERIODO(periodo);        //Crea cronometro

                                       //Ciclo de ejecucion
   for(;;){
     if(Receive(proxy, NULL, 0) == proxy){//Si se recibe senal
        Reply(proxy,NULL,0);                //del timer
          APAGAR_SERVO(1);             //Apaga el motor 1
     }
   }
}
```