SIMULACIÓN DE LOS MECANISMOS DE ADMINISTRACIÓN Y CONTROL DE UN SISTEMA OPERATIVO PARTE II

IDALIDES DEL CARMEN SIMANCAS GUARDO

JEBELL DAVID ZABALETA GARCÍA

TECNOLÓGICA DE BOLÍVAR INSTITUCIÓN UNIVERSITARIA

FACULTAD DE INGENIERÍA DE SISTEMAS

CARTAGENA D.T. Y C.

2000

SIMULACIÓN DE LOS MECANISMOS DE ADMINISTRACIÓN Y CONTROL DE UN SISTEMA OPERATIVO PARTE II

IDALIDES DEL CARMEN SIMANCAS GUARDO

JEBELL DAVID ZABALETA GARCIA

Tesis de Grado para optar al título de Ingeniero de Sistemas

Director
JUAN MARTÍNEZ LAMBRAÑO
Ingeniero de Sistemas
Magister en Ciencias Computacionales

TECNOLÓGICA DE BOLÍVAR INSTITUCIÓN UNIVERSITARIA

FACULTAD DE INGENIERÍA DE SISTEMAS

CARTAGENA D.T. Y C.

2000

Cartagena, 24 de abril del 2.000

Señores
DEPARTAMENTO DE INVESTIGACIONES
Atn. Vilma Ojeda
Corporación Universitaria Tecnológica de Bolívar
Ciudad.

Respetados señores:

Por medio de la presente me permito someter a estudio y aprobación la Tesis de Grado titulada **"SIMULACIÓN DE LOS MECANISMOS DE ADMINISTRACIÓN Y CONTROL DE UN SISTEMA OPERATIVO PARTE II"** realizada por los estudiantes Idalides del Carmen Simancas Guardo y Jebell David Zabaleta García, quienes la presentaran a ustedes para optar al título de Ingeniero de Sistemas.

Cordialmente,

_____
JUAN MARTINEZ LAMBRAÑO
DIRECTOR.

Cartagena, 24 de abril del 2.000

Señores

DEPARTAMENTO DE INVESTIGACIONES

Atn. Vilma Ojeda.

Tecnológica de Bolívar Institución Universitaria

Ciudad.

Respetados señores:

Por medio de la presente nos permitimos hacer entrega formal de la Tesis de Grado titulada **"SIMULACIÓN DE LOS MECANISMOS DE ADMINISTRACIÓN Y CONTROL DE UN SISTEMA OPERATIVO PARTE II",** como requisito para optar al título de Ingenieros de Sistemas.

Atentamente,

JEBELL ZABALETA GARCÍA **IDALIDES SIMANCAS GUARDO**

Cartagena, 24 de abril del 2.000

Ingeniero
GONZALO GARZÓN
Decano Facultad de Ingeniería de Sistemas
Tecnológica de Bolívar Institución Universitaria
Ciudad.

Respetado Ingeniero:

Por medio de la presente nos permitimos hacer entrega formal de la Tesis de Grado titulada **"SIMULACIÓN DE LOS MECANISMOS DE ADMINISTRACIÓN Y CONTROL DE UN SISTEMA OPERATIVO PARTE II",** para su aprobación.

Atentamente,

JEBELL ZABALETA GARCÍA **IDALIDES SIMANCAS GUARDO**

ARTÍCULO 105:

La Corporación Universitaria Tecnológica de Bolívar se reserva el derecho de propiedad intelectual de todos los Trabajos de Grados Aprobados y no pueden ser explotados comercialmente sin su autorización.

Nota de Aceptación

_____

_____

_____

_____

Presidente del Jurado

_____

Jurado

_____

Jurado

Cartagena D. T. y C., 24 de abril del 2.000

DEDICATORIA

A Dios, por darme la vida y su infinito e incondicional Amor. Por su bondad hacia toda mi familia. Por su ayuda para realizarme como mujer y brindarme las facultades necesarias para mi formación como Profesional. Por protegerme y guiarme en su camino y permitir que llegará a esta etapa tan importante en mi Vida.

A mis padres, **Antonio Simancas Torres** e **Idalides Guardo Puello**, por traerme al mundo, por sus constante esfuerzos para darme la mejor educación y ver realizados todos mis sueños, por enseñarme a luchar y no caer en todos los problemas que se me han presentado. Y sobre todo por su Amor y preocupación en cualquier momento.

A mis hermanos, **Inés Elena** y **Antonio José Simancas Guardo**, por su compañía, respaldo y paciencia durante la realización de este Trabajo.

A mi sobrino, **Diego Andrés**, mi pequeña luz, eres la alegría de toda mi familia, por venir al mundo justo en la etapa final de este Proyecto, la más dura y difícil, por tu sonrisa y por llenarme de tanta Felicidad.

A mi novio, **Edwin Alberto Villalba Tapia**, por aparecerte en un momento trascendental e inesperado de mi vida, por entregarme su paciencia, comprensión y sobre todo su Amor.

A mi compañero de Tesis, **Jebell David Zabaleta García**, por brindarme su amistad incondicional y confianza, por escucharme a cada instante y convertirse a lo largo de estos cinco años de carrera en el mejor compañero de Trabajos.

Idalides del Carmen Simancas Guardo

DEDICATORIA

A Dios por ser mi guía, maestro Principal y por darme la fuerza para levantarme en todos y cada uno de los tropezones durante mi carrera.

A mis padres, Abel Zabaleta y Elsa García, por haberme dado la vida, la fortaleza y sus buenos concejos para salir airoso en todas las instancias importantes en m vida.

A mis hermanos, Elkin Fabian, Oscar Abel y Hasmel Eduardo, por haberme soportando en mis momentos de dolor y amargura y por celebrar junto a mí en los momentos de gozo y alegría.

A mi novia, Carmen Cecilia, por haber llegado en un momento tan crucial en mi vida y haber caminado a mi lado durante todo este tiempo siempre brindándome su amistad y amor.

A Juan Martinez, mi director de Tesis, por ser más que eso, mi maestro y amigo y por no dudar en ningún momento de mis capacidades.

A Fabio Castro, Jorge Pinedo, Robinson Alvarez y Javier Pinedo, por ser mis verdaderos amigos.

A Gabriel Oliveros y Carlos Botero, por la palmada de aliento en el momento más indicado y por ser mis amigos y colegas en las buenas y en las malas.

A mi compamera de tesis, Idalides Simancas, por soportarme e imponer el orden durante el desarrollo de todos los trabajos en los que estuvimos juntos durante la carrera.

Jebell David Zabaleta García

AGRADECIMIENTOS

Los autores expresan sus agradecimientos a:

Juan Martínez Lambraño, Ingeniero de Sistemas y Director de la Tesis, por sus enseñanzas, motivación y valiosas orientaciones a lo largo del desarrollo de esta Investigación.

Carlos Ferriol, Licenciado en Ciencias Computacionales, por su colaboración profesional y humana y a la vez enseñarnos las Bases de la Programación en Delphi.

Jaime Arcila, Ingeniero Electricista, por su valiosa ayuda en la aclaración de conceptos y ejecución de programas de Algorítmica Paralela.

Gonzalo Garzón, Ingeniero de Sistemas y Decano de la Facultad de Ingeniería de Sistemas, por brindar su ayuda en tan oportunos momentos.

Compañeros, amigos y a todas aquellas personas que de una u otra forma colaboraron para que la culminación este Trabajo de grado se llevara a cabo exitosamente.

TABLA DE CONTENIDO

INTRODUCCIÓN

1. MARCO TEÓRICO

1.1 LOS PROCESOS DESDE LA PERSPECTIVA DEL SISTEMA OPERATIVO

1.1.1 Máquina de Estados Finitos

1.1.2 El Bloque Descriptor de Procesos (BDP)

1.2 PLANIFICACIÓN DE PROCESOS

1.2.1 Algoritmos de Planificación

1.2.1.1 Planificación Primero el Trabajo más Corto

1.2.1.2 Planificación Tipo Round Robin

1.2.1.3 Planificación por Prioridad

1.2.1.4 Planificación con Colas Múltiples con Prioridad

1.3 SIMULACIÓN DE UN AUTÓMATA QUE REPRESENTA LA DINÁMICA DE LOS PROCESOS DE UN SISTEMA OPERATIVO (UNIX)

1.4 SINCRONIZACIÓN ENTRE PROCESOS

1.4.1 Exclusión Mutua

1.4.1.1 Métodos para lograr la exclusión Mutua

1.4.2 Semáforos

1.4.3 Problemas Clásicos

1.4.3.1 Productores y Consumidores

1.4.3.2 El Barbero Dormilón

1.4.3.3 La Cena de los Filósofos

1.4.3.4 Lectores y Escritores

1.4.3.5 El Fumador de Cigarros

1.5 EL MODELO CLIENTE/SERVIDOR

1.5.1 Tipos de Servidores

1.5.1.1 Servidores Interactivos

1.5.1.2 Servidores Concurrentes

1.5.2 Esquema General de un Servidor y un Cliente

1.6. ALGORÍTMICA PARALELA Y CONCURRENCIA

1.6.1 Paso de Mensajes

1.6.2 PVM (Parallel Virtual Machine)

2. DESCRIPCIÓN DEL PROYECTO

2.1 ANTECEDENTES DEL PROBLEMA

2.2 EL PROBLEMA DE LA INVESTIGACIÓN

2.2.1 Identificación del Problema

2.2.2 Planteamiento del Problema

2.2.3 Análisis del Problema

2.2.4 Formulación del Problema

2.3 OBJETIVOS DE LA INVESTIGACIÓN

2.3.1 Objetivo General

2.3.2 Objetivos Específicos

3. ENTORNO DEL DISEÑO DEL SOFTWARE

3.1 INTRODUCCIÓN

3.2 POBLACION A QUIEN VA DIRIGIDO

3.3 PRESENTACIÓN

3.4 ÁREAS DEL CONTENIDO

3.5 COMPLEJIDAD

3.6 CONTROL INSTRUCCIONAL

3.7 AYUDAS

3.8 LOS LENGUAJES

4. DESCRIPCION DE ASPECTOS DE EJECUCIÓN DEL SOFTWARE

4.1 ASPECTOS GENERALES

4.2 ASPECTOS PARTICULARES

4.2.1 Planificación de Procesos

4.2.2 Simulación de un Autómata que representa la Dinámica de los Procesos de un Sistema Operativo (UNIX)

4.2.3 Sincronización entre Procesos

4.2.4 Modelo Cliente/Servidor

4.2.5 Algorítmica Paralela y Concurrencia

4.3 MENSAJES EN LAS SIMULACIONES

5. DESCRIPCIÓN DE ASPECTOS DEL DISEÑO DEL SOFTWARE

5.1 ASPECTOS GENERALES

5.2 ASPECTOS PARTICULARES

5.2.1 Planificación de Procesos

5.2.2 Simulación de un Autómata que representa la Dinámica de los Procesos de un Sistema Operativo (UNIX)

5.2.3 Sincronización entre Procesos

5.2.4 Modelo Cliente/Servidor

5.2.5 Algorítmica Paralela y Concurrencia

5.3 LIMITACIONES Y SUPUESTOS

5.3.1 Algoritmos de Planificación.

5.3.2 Simulación de un Autómata que representa la Dinámica de los procesos de un Sistema Operativo (UNIX).

5.3.3 Algoritmos de Sincronización entre Procesos.

5.3.3.1 Productores y Consumidores

5.3.3.2 El Barbero Dormilón

5.3.3.3 Lectores y Escritores

5.3.4 El Modelo Cliente/Servidor.

5.3.4.1 UDP

5.3.4.2 Simulaciones con Cliente/Servidor

5.4. ESTRUCTURAS DE DATOS RELEVANTES

5.4.1 Algoritmos de Planificación.

5.4.2 Algoritmos de Sincronización entre Procesos.

5.4.3 Modelo Cliente/Servidor.

5.5 PARADIGMAS DE PROGRAMACIÓN

5.5.1 Hilos

5.5.2 Sockets

6. RECOMENDACIONES

7. CONCLUSIONES

BIBLIOGRAFÍA

ANEXOS

MATERIAL ACOMPAÑANTE

LISTA DE TABLAS

Tabla 1. Limitaciones.

Tabla 2. Aleatoriedad para las E/S.

Tabla 3. Campos para la Estructura Proc.

Tabla 4. Campos para la Estructura Procesp.

LISTA DE FIGURAS

Figura 1. Máquina de Estados Finitos.

Figura 2. Bloque Descriptor de Procesos.

Figura 3. Diagrama de estados para los procesos de UNIX.

Figura 4. Protocolo Solicitud/Respuesta.

Figura 5. Esquema Cliente/Servidor Interactivo.

Figura 6. Esquema Cliente/Servidor Concurrente.

Figura 7. Comunicación Orientada a conexión.

Figura 8. Comunicación no orientada a conexión.

Figura 9. Demonio y librerías en PVM.

Figura 10. Entorno inicial de Delphi.

LISTA DE ANEXOS

Pág.

Anexo A. Manual de soporte del Software

GLOSARIO

FORK: esta instrucción hace que un proceso *Padre* produzca una copia idéntica, llamada proceso *Hijo*, con sus propias variables de memoria y asociaciones de recurso.

GRAFO: está formado por un conjunto de nodos (llamados también *vértices*) y un conjunto de líneas llamadas *aristas* (o *arcos*) que conectan los diferentes nodos.

KERNEL (Núcleo del Sistema Operativo): gestiona básicamente los procesos.

MULTIPROGRAMACIÓN: es un Sistema Operativo que proporciona gestión de memoria y gestión de archivos, además, soporta la ejecución concurrente de programas.

MULTITAREA: es cuando el Sistema Operativo realiza la ejecución concurrente de programas sobre un solo procesador sin soportar necesariamente formas elaboradas de gestión de memoria y gestión de archivos.

PROCESO: instancia de un programa en ejecución.

PROTOCOLO: juego de reglas que definen la forma en que deben efectuarse las comunicaciones en las redes, incluyendo el formato, la temporización, la secuencia, la revisión y la corrección de errores.

SYSTEM CALL (Llamadas al Sistema): son las instrucciones que proporcional el Sistema Operativo y que sirven de interfaz con los programas del Usuario.

RESUMEN

El área de sistemas operativos abarca una serie de temas que si son tratados en forma puramente teórica son de difícil comprensión; por esta razón surge la necesidad del estudio de ellos, buscando la forma de hacer de estos temas y de los modelos que los soportan, elementos tangibles de fácil asimilación durante su estudio.

Como parte determinante dentro del manejo de los Sistemas Operativos se puede destacar el estudio de la Comunicación de Procesos y la Sincronización entre Procesos como aspectos fundamentales, ya que los Procesos son mecanismos esenciales para definir y gestionar la ejecución concurrente de los programas bajo el control de un Sistema Operativo.

El presente proyecto tiene como finalidad la implementación de una herramienta de simulación de los mecanismos de administración y control de un sistema operativo donde el estudiante pueda verificar y afianzar los conocimientos teóricos adquiridos en el curso de sistemas operativos.

Hablar acerca de los Sistemas Operativos, es una tarea muy larga, pues se trata de investigar y estudiar una cantidad de libros que diariamente salen al mercado y que contienen todos sus conceptos básicos. En el más estricto sentido de la expresión el sistema operativo estaría definido como un conjunto de procedimientos o rutinas de software encargados de administrar los recursos del computador. Estos recursos pueden ser físicos y lógicos. Cuando el sistema operativo administra recursos, este debe llevar un control del estado del mismo, para así poder determinar a quien se lo asigna, cuando se lo asigna y por cuanto tiempo.

En un sistema Operativo se conocen muchos algoritmos de Planificación, su propósito es determinar el proceso que debe ejecutarse a continuación, teniendo en cuenta factores tales como el tiempo de respuesta, la eficacia y la equidad. Entre los algoritmos de planificación más conocidos están Primero el trabajo más corto, Round Robin, Prioridad y Colas múltiples con Prioridad.

En algunas ocasiones, los procesos interactuan para accesar en forma concurrente a recursos compartidos, tales como estructuras a datos o dispositivos de E/S, lo que trae como consecuencia condiciones de competencia en la que se requiere una excelente sincronización entre los procesos para lograr un buen resultado. Sin embargo, estas condiciones de competencia se puede evitar utilizando el concepto de región crítica las cuales proporcionan la exclusión mutua.

Los procesos de puede comunicar entre sí para asegurar la integridad de la operación total, mediante las primitivas de comunicación entre procesos y se usan para garantizar que dos procesos no se encuentran al mismo tiempo dentro de sus regiones críticas. En este trabajo, se ha propuesto los *Semáforos.* Así mismo, existe una variedad de problemas Clásicos que tienen un lugar destacado en la teoría y la práctica de los sistemas operativos.

El Modelo Cliente/Servidor es el concepto de interacción más común entre aplicaciones en una red que maneje la comunicación entre procesos. Todos los servicios estándares de alto nivel propuesto en Internet funcionan según este concepto. Su implementación con *sockets*, que son una generalización del mecanismo de acceso a archivos de UNIX proporciona un punto final de la comunicación y abre las puertas a una fácil manera de programación.

Él computo distribuido, es correr un proceso en un conjunto de maquinas conectadas por una red, de aquí el concepto de Programación Paralela y Concurrente, que conduce a un nuevo paradigma de programación basado en el lenguaje PVM (Parallel Virtual Machine).

PVM habilita una colección de computadoras heterogéneas para ser usadas como una coherente y flexible herramienta computacional distribuida, basadas en el Modelo de Paso de Mensajes. La idea en PVM es tener una colección de computadoras para que parezca que se tiene una gran maquina virtual. PVM, es de fácil implantación y uso. No requiere la instalación de privilegios especiales.

Por último, la mecánica de los procesos básicamente se puede modelar mediante una máquina de estados finitos, también en el sistema Operativo UNIX, se representa la dinámica de los procesos mediante un diagrama de transición de estados. Este diagrama a diferencia del convencional o tradiciona l esta compuesto por 9 estados.

Para el desarrollo de este software se llevó a cabo una exhaustiva investigación acerca de todos los temas relacionados con esta Fase (II), se analizaron y seleccionaron aquellos aspectos importantes que permitieran dar inicio a la Implementación de las Simulaciones.

Como plataformas computacionales se utilizaron WINDOWS y LINUX (versión 6.1) y los siguientes lenguajes de Programación:

1. Delphi (versión 4.0) ya que cuenta con las herramientas necesarias que facilitan el manejo de Hilos, Sockets y el Modelo Cliente/Servidor.
2. PVM (Parallel Virtual Machine).

Como resultado de todo este trabajo los estudiantes y demás personas interesadas en el estudio de los Sistemas Operativos, podrán entender de una manera más rápida y fácil, las abstracciones concernientes a estos temas, además, de facilitar la labor de enseñanza del Orientador de la asignatura.

INTRODUCCIÓN

Teniendo en cuenta que el hombre tiende a representar o imaginar cualquier concepto en forma abstracta, en ocasiones estos pueden estar equivocados por la complejidad del tema a tratar, es por tal motivo que el presente proyecto tiene por objeto proveer una sencilla herramienta de software, de fácil uso, en la cual se simulan algunos mecanismos de administración y control de un sistema operativo.

La importancia de este proyecto radica en que se amplía la visión de las personas interesadas en el estudio del diseño e implementación de los sistemas operativos, y en que el conocimiento teórico podrá ser afianzado de una manera rápida, ya que permite visualizar los diferentes algoritmos de planificación por los que se rigen los procesos, su comunicación y la solución de los problemas clásicos de sincronización entre los mismos.

Para el desarrollo de este software se llevó a cabo una exhaustiva investigación acerca de todos los temas relacionados con esta Fase (II), se analizaron y seleccionaron aquellos aspectos importantes que permitieran dar inicio a la Implementación de las Simulaciones. En un sentido general, este trabajo resume todos aquellos aspectos primordiales y determinantes en este tema.

Se tratan aspectos sencillos como *La comunicación y Sincronización entre procesos, el autómata que representa la dinámica de los procesos en UNIX y los conceptos básicos del Modelo Cliente/Servidor.* Además, de trabajar con un nuevo paradigma de programación como es la *Programación Concurrente* a través del lenguaje PVM.

Este documento que ahora presentamos, al igual que el manual del usuario y el software producto de la investigación pueden ser de gran ayuda a todos aquellos que deseen abordar el estudio de los temas correspondientes a esta fase en los sistemas operativos.

Cabe anotar que el conocimiento requerido para utilizar de manera óptima el software de la investigación es mínimo, en lo relativo a los sistemas operativos. Lo anterior se debe a que la aplicación puede ofrecer un soporte práctico a las clases teóricas y magistrales de esta materia.

1. MARCO TEÓRICO

1.1 LOS PROCESOS DESDE LAPERSPECTIVA DEL SISTEMA OPERATIVO

Un proceso es la entidad más pequeña individualmente planificable, formada por códigos y datos y caracterizada por atributos y un estado dinámico. El código se compone de las instrucciones de máquina y de las Llamadas de servicio al sistema. Los atributos asociados con un proceso son asignados por el programador del sistema o por el mismo sistema operativo.

1.1.1Máquina de estados Finitos. El sistema operativo contempla la ejecución de un proceso típico en el curso de su actividad en forma de progresión a través de una sucesión de estados. Figura 1

La máquina de estados finitos está compuesta por diferentes estados:

- ✓ Trabajos inactivos: se refiere a los procesos que aún no son conocidos, y por tanto no son contabilizados por el sistema operativo.

- ✓ Trabajos listos para ejecución: poseen todos los recursos necesarios para su ejecución, excepto el procesador.

- ✓ Trabajos en ejecución: poseen todos los recursos necesarios para su ejecución, incluyendo el procesador.

- ✓ Trabajos en solicitud de entrada/salida: el proceso queda suspendido en espera de que esta condición desaparezca y pueda continuar con su tarea.

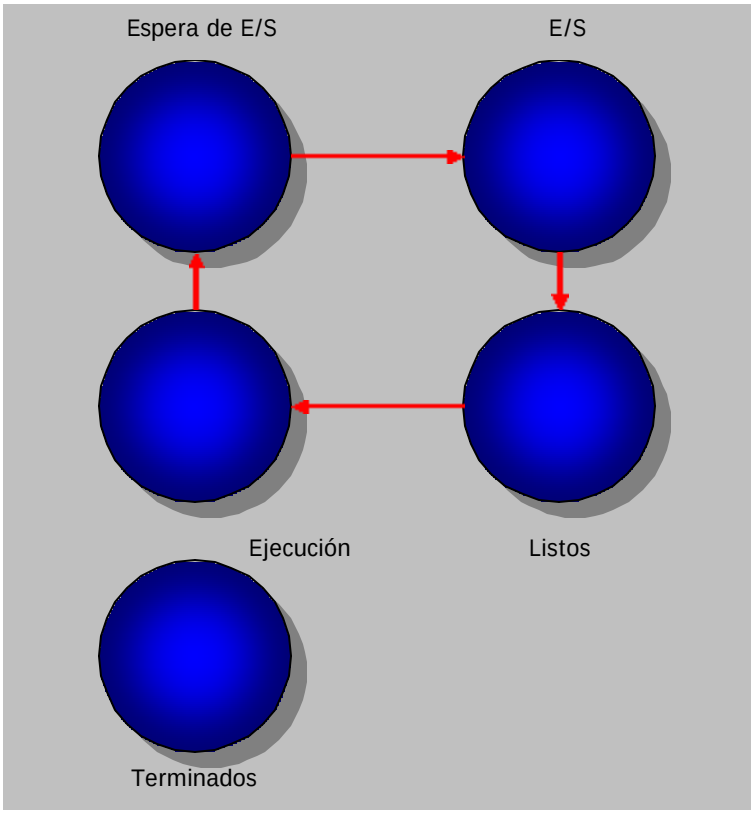

Figura 1. Máquina de estados Finitos.

La Dinámica es la siguiente: un proceso que es planificado entra a formar parte de la lista de espera, posteriormente se le asigna el procesador de acuerdo a su tamaño, o alguna otra política previamente definida. Si el quantum (hace referencia a un intervalo de tiempo que se le asigna a un proceso para su posterior tratamiento), se vence y el proceso aún no culmina, el sistema le desasigna procesador y lo pasa a estado de listo en espera de una nueva asignación.

Si el proceso en ejecución solicita una operación de E/S, se produce una interrupción con el objeto de atender esa solicitud y pasa a un estado de E/S, una vez termina la operación de E/S regresa al estado de listo. Esto se repite hasta que el proceso termina.

1.1.2 Bloque Descriptor de Procesos (BDP). Es una estructura de datos, figura 2, con los campos necesarios para registrar los diferentes aspectos del estado del proceso y de la utilización de recursos.

Cuando un proceso se crea, inicializa un registro BDP para que sirva como descripción en timpo de ejecución, hasta cuando termina, tiempo en el cual el BDP es liberado y entra a formar parte de la piscina de disponibilidades para futuros procesos del sistema.

ID	PC	Quantum	Listos	Ejecución	E/S	Tamaño	Registros del Procesador

Figura 2. Bloque Descriptor de Procesos.

Contiene entre otras la siguiente información:

- ✓ Identificación del proceso.
- ✓ Contador de Programa.
- ✓ Quantum.
- ✓ Estado del proceso.
- ✓ Registros y banderas del procesador.
- ✓ Información de planificación y estadísticas de uso
- ✓ Apuntadores a registros para la administración de memoria (registros y tablas).
- ✓ Estado de E/S (dispositivos asignados, operaciones pendientes).
- ✓ Prioridad.
- ✓ Información para la administración de archivos.
- ✓ Información de mantenimiento.

1.2 PLANIFICACIÓN DE PROCESOS

La planificación hace referencia a un conjunto de políticas y mecanismos incorporados al sistema operativo que rigen el orden en que se ejecutan los trabajos, y el algoritmo que utiliza se denomina ***algoritmo de pglanificación***.

Su objetivo es optimizar el rendimiento del sistema de acuerdo con los criterios considerados más importantes por los diseñadores del mismo. Para tal efecto se debe tener en cuenta los siguientes aspectos:

Equidad: es garantizar que cada proceso obtiene su proporción justa de la CPU.

Eficacia: es mantener ocupada la CPU el 100% del tiempo de trabajo del sistema.

Tiempo de respuesta: minimizar el tiempo de respuesta de la máquina para los procesos interactivos de un sistema multiusuario.

Tiempo de regreso: minimizar el tiempo de respuesta de la máquina para los procesos por lotes de un sistema multiusuario.

Tiempo de espera: es el tiempo que un proceso o trabajo tarda esperando la asignación de recursos debido a la competencia con otros en un sistema de multiprogramación.

Rendimiento: es maximizar el número de tareas procesadas por unidad de tiempo.

1.2.1 Algoritmos de Planificación

Antes de describir cada uno de los algoritmos de planificación a tratar en este trabajo, es indispensable aclarar que en todos ellos, cuando ocurre un ***System Call*** (Llamada del sistema), deben abandonar cualquier tarea que estén realizando y atender dicha llamada, guardando el estado actual del sistema para luego volver a retomarlo.

1.2.1.1 Primero El Trabajo Más Corto. La dinámica del algoritmo consiste en que el planificador verifica los trabajos que se encuentran en la lista de entrada y elige el de menor tiempo para su ejecución, una vez culminada la asignación para tal trabajo, sigue eligiendo los trabajos a ejecutar considerando el mismo criterio.

1.2.1.2 Planificación tipo Round Robin (Reparto de Tiempo). Este tipo de planificación se fundamenta en el concepto de ***Quantum*** que hace referencia a un intervalo de tiempo que se le asigna a un proceso para su posterior tratamiento; luego de lo anterior el planificador toma la lista de procesos ejecutables y empieza a alternar procesos dependiendo de eventos como: finalización del Quantum, bloqueo de E/S, System Call, entre otras.

1.2.1.3 Planificación por Prioridad. La idea fundamental es que cada proceso del sistema tiene asignada una prioridad desde el momento de su creación, la cual puede ser constante o variable durante la ejecución. El planificador siempre elige al proceso de mayor prioridad; se debe tener en cuenta que los System Call son los procesos de más alta prioridad en un sistema operativo.

1.2.1.4 Planificación con Colas Múltiples con Prioridad. Se caracteriza porque establece clases de prioridades donde en cada una de ellas se encuentran uno o más procesos ejecutables, los cuales se planifican dentro de su clase con cualquier planificador viable para la situación.

1.3 SIMULACIÓN DE UN AUTÓMATA QUE REPRESENTA LA DINÁMICA DE LOS PROCESOS DE UN SISTEMA OPERATIVO (UNIX)

La dinámica de los procesos en UNIX se puede representar mediante un diagrama de transición de estados, que consiste en un ***grafo dirigido,*** cuyos ***nodos*** representan los estados que pueden alcanzar los procesos y cuyas ***ramas*** representan los eventos que hacen que un proceso cambie de un estado a otro.

El diagrama de estados para los procesos de UNIX se observa en la figura 3 y los estados que en él se reflejan son:

1. El proceso se está ejecutando en modo usuario.
2. El proceso se está ejecutando en modo ***kernel (Modo Supervisor)***.
3. El proceso no se está ejecutando, pero está listo para ejecutarse tan pronto como el ***kernel*** lo ordene.
4. El proceso está durmiendo cargado en memoria.
5. El proceso está listo para ejecutarse, pero el ***swapper*** (proceso cero) debe cargar el proceso en memoria antes de que el ***kernel*** pueda ordenar que pase a ejecutarse.
6. El proceso está durmiendo y el ***swapper*** ha descargado el proceso hacia una memoria secundaria (área de ***swap*** del disco) para crear espacio en la memoria principal donde poder cargar otros procesos.
7. El proceso está volviendo del modo ***kernel*** al modo usuario, pero el ***kernel*** se apropia del proceso y hace un cambio de contexto, pasando otro proceso a ejecutarse en modo usuario.
8. El proceso acaba de ser creado y esta en un estado de transición; el proceso existe, pero ni esta preparado para ejecutarse (estado 3), ni durmiendo (estado 4). Este estado es el inicial para todos los procesos, excepto el proceso cero.
9. El proceso ejecuta la llamada ***exit*** y pasa al estado ***zombi.*** El proceso ya no existe, pero deja para su proceso padre un registro que contiene el código de salida y algunos datos estadísticos tales como los tiempos de ejecución. El estado de ***zombi*** es el estado final de un proceso.

El proceso cero es especial, es creado cuando arranca el sistema, y después de hacer una llamada a ***fork*** se convierte en el proceso ***swapper*** (encargado del ***swaping*** o gestión de la memoria virtual)

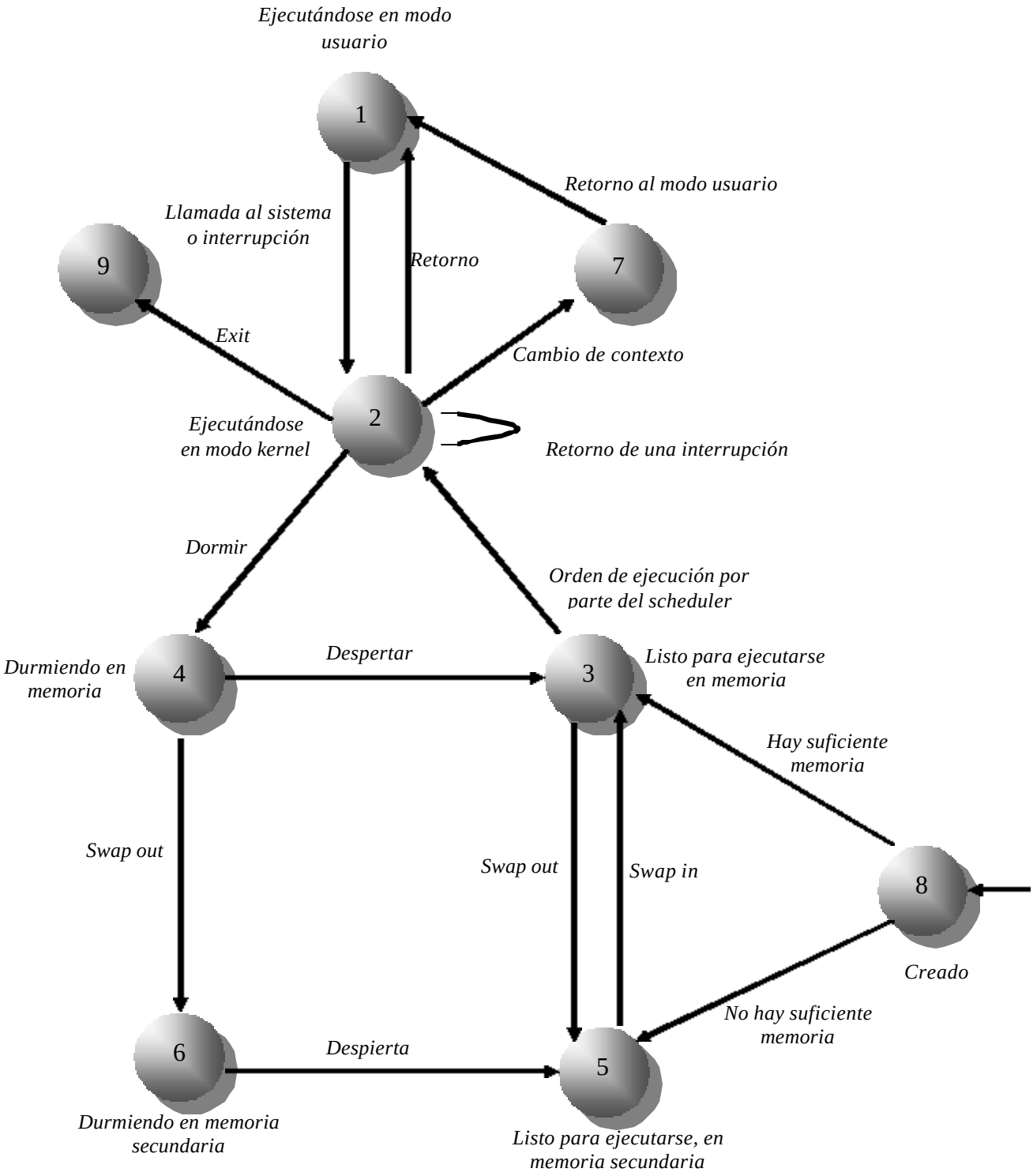

Figura 3. Diagrama de estados para los procesos de UNIX

1.4 SINCRONIZACIÓN ENTRE PROCESOS

Es un conjunto de protocolos y mecanismos utilizados para preservar la integridad y consistencia del sistema cuando varios procesos concurrentes comparten recursos que son reutilizables en serie.

1.4.1 Exclusión Mutua. Este método se utiliza cuando hay una sección crítica, que es una secuencia de instrucciones con un comienzo y un final claramente delimitados, que generalmente representa la actualización de una o más variables compartidas. El principio de exclusión mutua consiste en que el proceso que hace uso de la región critica inhibe al resto de los procesos que compiten por el acceso a ella. Una vez liberada se otorga su uso a otro proceso.

Se puede definir una semántica para la solución al problema de Exclusión mutua, así:

- ✓ Asegurar la exclusión mutua entre los procesos cuando se accede al recurso compartido.
- ✓ No hacer suposiciones acerca de las velocidades de procesamiento de cada proceso.
- ✓ Garantizar que el aborto o terminación de un proceso fuera de la región crítica no afecte la capacidad del resto de procesos competidores.
- ✓ No puede ser que un proceso espere indefinidamente para acceder a la región crítica.

Básicamente un proceso sigue el siguiente protocolo para la exclusión mutua:

- ✓ Protocolo de negociación del recurso
- ✓ Uso exclusivo de la región critica
- ✓ Liberación de la región critica.

1.4.1.1 Métodos para lograr la Exclusión mutua

- ✓ **Variables de Cerradura:** es una solución sencilla de software para lograr la exclusión mutua en la cual se utiliza una variable como bandera de la región crítica, esta variable puede tomar dos valores dependiendo si hay o no un proceso dentro de la región; si el valor de la variable es el correspondiente a ocupado ningún otro proceso podrá ingresar mientras no se haya modificado el valor por el proceso saliente.

- ✓ **Alternancia estricta:** la solución de software que implementa esta técnica son segmentos de código que mantienen variables, las cuales determinan el turno en que los procesos tendrán acceso a la región crítica, ningún proceso puede entrar en ella dos veces seguidas; el tiempo correspondiente a la espera de un proceso por su turno, se denomina ***Espera ocupada.***

  Esta técnica sólo debe usarse cuando existe una esperanza razonable de que la espera será corta, además es poco aconsejable puesto que desperdicia mucho tiempo de CPU.

- ✓ **Solución de Peterson:** este método se fundamenta en dos procedimientos en los cuales se coordina la entrada y salida de la región crítica, esta solución solamente se implementa para dos proceso, puesto que para más, se deberían realizar muchas confirmaciones de software.

1.4.2 Semáforos. Un semáforo es una variable que puede tener valores enteros los cuales pueden ser manipulados mediante dos operaciones denominadas ***Dowm*** y ***Up***, originalmente llamadas por Dijkstra (Quien planteó este mecanismo en 1965) ***P*** y ***V;*** el código de estas operaciones son los siguientes:

```
Down (s):
        While (s≤0) { /*se sigue probando hasta cuando s>0*/ }
          s = s-1;
Up (s):
      s = s+1;
```

1.4.3 Problemas Clásicos

1.4.3.1 Productores y Consumidores. Formulación del problema: *"Idear un protocolo de sincronización que permita a productores y consumidores operar concurrentemente a sus*

respectivas velogcidades de servicio de tal modo que los datos producidos se consuman en el orden exacto en que son producidos".

1.4.3.2 El Barbero Dormilón. Formulación del problema: *"Una peluquería tiene un barbero, una silla de peluquero y n sillas para que se sienten los clientes en espera, si es que los hay. Si no hay clientes presentes, el barbero se sienta en su silla de peluquero y se duerme. Cuando llega un cliente, éste debe despertar al barbero dormilón. Si llegan más clientes mientras el barbero corta el cabello de un cliente ellos se sientan (si hay sillas desocupadas) o salen de la peluquería (sí todas las sillas están ocupadas). El problema consiste en programar al barbero y los clientes sin entrar en condiciones de competencia".*

1.4.3.3 La Cena de los Filósofos. Formulación del problema: *"Cinco filósofos se sientan a la mesa. Cada uno tiene un plato de espaguetis. Este es tan escurridizo, que un filósofo necesita dos tenedores para comerlo. Entre cada dos platos hay un tenedor.*

La vida de un filósofo consta de periodos alternados de comer y pensar (esto es una abstracción, pero las demás actividades son irrelevantes en este caso). Cuando un filósofo tiene hambre intenta obtener u tenedor para su mano izquierda y otro para su mano derecha, alzando uno a la vez y en cualquier orden. Si logra obtener los dos tenedores, come un rato, después deja los tenedores y continúa pensando. El problema consiste en programar a cada filósofo para que lleve a cabo sus actividades sin que se presente algún bloqueo".

1.4.3.4 Lectores y Escritores. Formulación del problema: *"Imaginemos una enorme base de datos, como por ejemplo un sistema de reservaciones de una línea aérea, con muchos procesos en competencia, que intentan leer y escribir en ella. Se puede aceptar que varios procesos lean la base de datos al mismo tiempo pero si uno de los procesos esta escribiendo (es decir, modificando) la base de datos, ninguno de los demás procesos debería tener acceso a ésta, ni siquiera los lectores. El problema consiste en programar a los lectores y escritores".*

1.4.3.5 El Fumador de Cigarros. Formulación del problema: *"Tres fumadores empedernido están dentro de un cuarto con un vendedor de materia prima para cigarro. Para fabricar y utilizar un cigarro cada fumador necesita tres ingredientes: tabaco, papel y cerillos; el vendedor tiene una amplia existencia de todo esto. Un fumador tiene su propio tabaco, el segundo tiene su propio papel y el tercero sus propios cerillos. La acción comienza cuando el vendedor coloca dos de los ingredientes en una mesa, con el fin de permitir a uno de los fumadores que cometa un atentado contra la salud. Cuando dicho fumador termina, el o ella despierta al vendedor, quien coloca entonces otros dos ingrediente (en forma aleatoria), con lo que bloquea a otro fumador. El problema consiste en programar a los fumadores y al vendedor".*

1.5 EL MODELO CLIENTE/SERVIDOR

La idea detrás de este modelo, es la estructuración del sistema operativo como un grupo de procesos en cooperación llamados **Servidores**, que son procesos que se están ejecutando en un nodo (computador o periférico conectado a la red) y que gestiona el acceso a un determinado recurso a los usuarios, llamados **Clientes**. Las máquinas de los clientes y servidores ejecutan, por lo general, el mismo micronúcleo y ambos se ejecutan como procesos de usuario. Una máquina puede ejecutar un único proceso o varios clientes, varios servidores o combinaciones de ambos.

Se basa en un protocolo ***solicitud/respuesta*** en el cual el cliente envía un mensaje de solicitud al servidor para pedir cierto servicio, el servidor hace el trabajo y regresa los datos solicitados o un código de error para indicar la razón por la cual un trabajo no se pudo llevar a cabo, como se muestra en la figura 4:

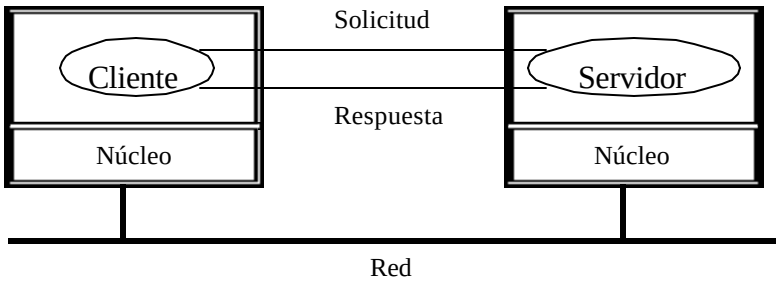

Figura 4. Protocolo *solicitud/respuesta*

Las ventajas de la figura **4** son:

- ✓ *Sencillez:* El cliente envía un mensaje y obtiene una respuesta. No se tiene que establecer una conexión sino hasta que ésta se utilice. El mensaje de respuesta sirve como agradecimiento a la solicitud.

- ✓ *Eficiencia:* La pila del protocolo es más corta y por tanto más eficiente.

1.5.1 Tipos de Servidores. Un servidor está continuamente esperando peticiones de servicio de los clientes. Cuando se produce una petición el servidor despierta y atiende al cliente. Cuando el servicio concluye el servidor vuelve al estado de espera. De acuerdo con la forma de prestar el servicio podemos considerar servidores de dos tipos:

1.5.1.1 Servidores Interactivos. El servidor no sólo recoge la petición de servicio, sino que él mismo se encarga de atenderla. Esta forma de trabajo presenta un inconveniente: si el servidor es lento en atender a los clientes y hay una demanda del servicio muy elevada, se van a originar tiempos de espera elevados, esto sucede si el servidor maneja colas de espera ya que si no las maneja simplemente no se puede recibir la petición mientras el servidor se encuentre ocupado.

1.5.1.2 Servidores Concurrentes. El servidor recoge cada una de las peticiones de servicio y crea otros procesos para que se encarguen de atenderlas. Este tipo de servidores

sólo es aplicable en sistemas multiprocesos. la ventaja que tiene este tipo de servicio es que el servidor puede recoger peticiones a muy alta velocidad porque está descargando la tarea de atención al cliente. En aquellas aplicaciones donde los tiempos de servicio son variables y/o la demanda de servicio es muy elevada es recomendable implementar este tipo de servidores.

1.5.2 Esquema General de un Servidor y un Cliente. Las acciones que debe llevar a cabo el programa servidor son las siguientes:

1. Abrir el canal de comunicaciones e informar a la red tanto de la dirección a la que responderá como de su disposición para aceptar peticiones de servicio.

2. Esperar que un cliente le pida servicio en la dirección que él tiene declarada.

3. Cuando recibe una petición de servicio, si es un servidor interactivo, figura 5, atenderá al cliente. Si el servidor es concurrente, figura 6, creará un proceso mediante **fork** para que le dé servicio al cliente.

4. Volver al punto 2 para esperar nuevas peticiones de servicio.

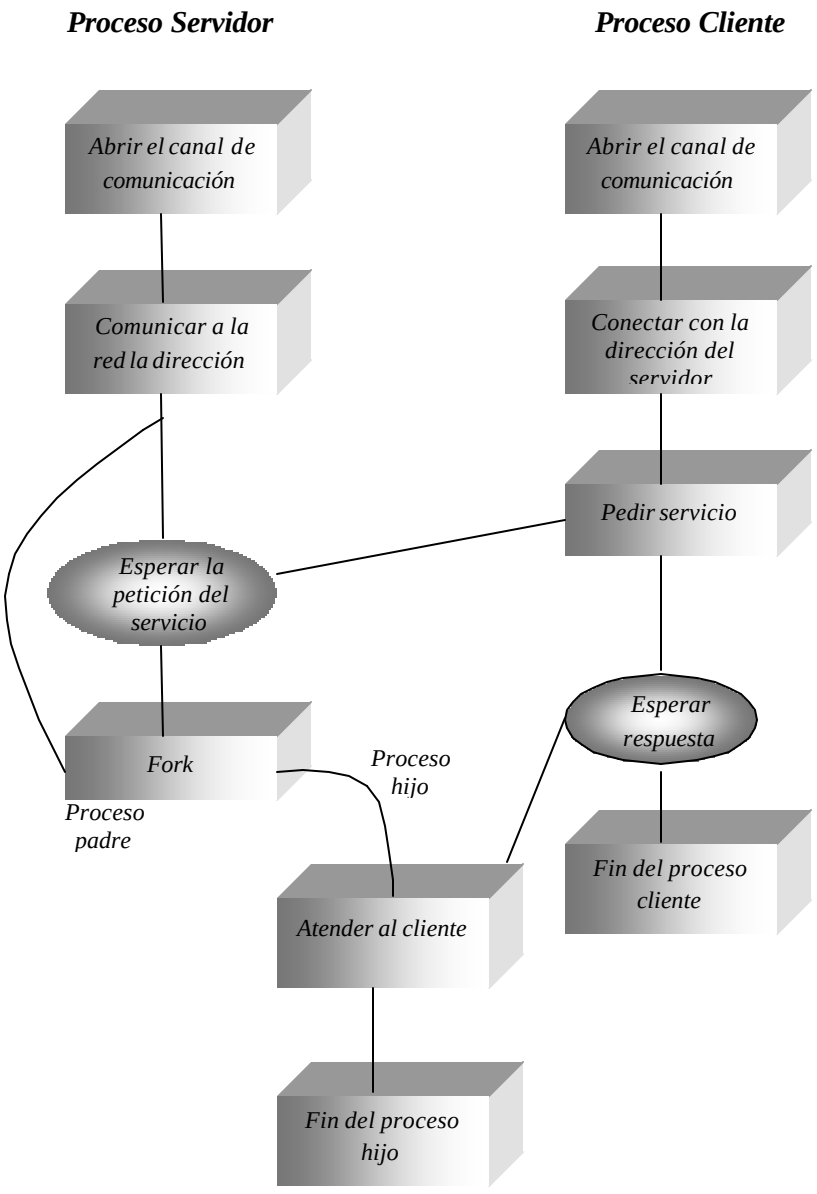

Figura 5. Esquema Cliente/Servidor Interactivo

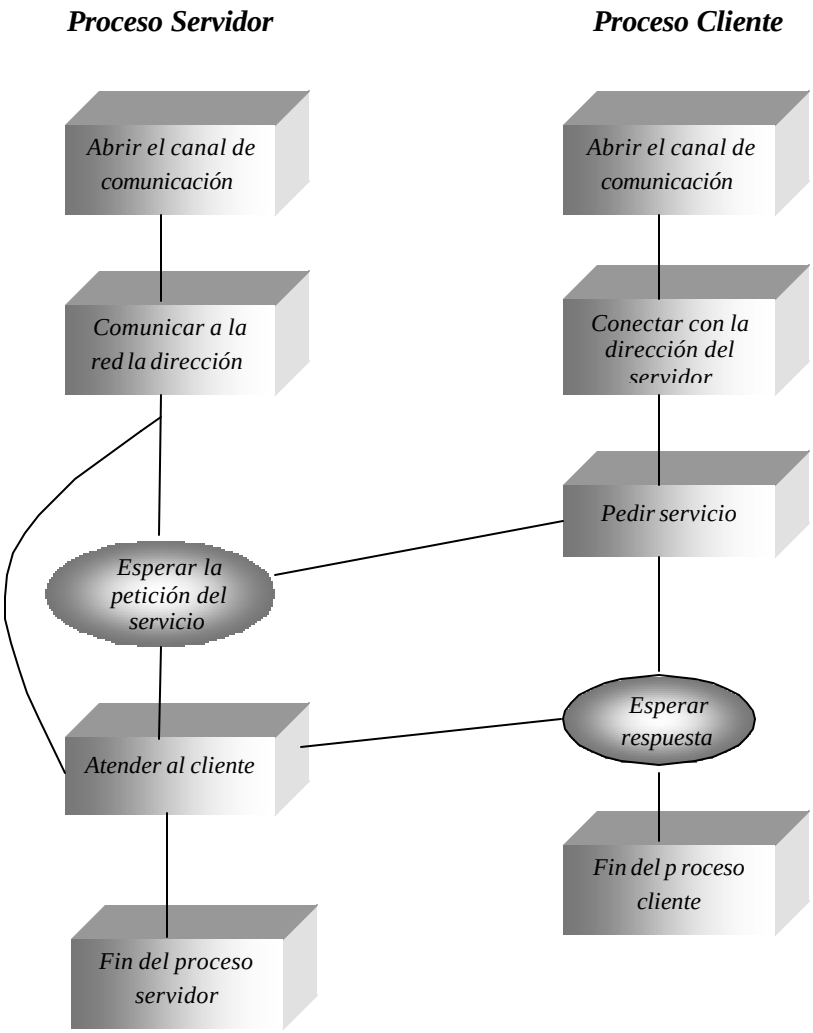

Figura 6. Esquema Cliente/Servidor Concurrente

El programa cliente por su parte llevaría a cabo las siguientes acciones:

1. Abrir el canal de comunicaciones y conectarse a la dirección de red atendida por el servidor. Naturalmente esta dirección debe ser conocida por el cliente y responderá al esquema de generación de direcciones de la familia de **sockets** que se esté empleando.

2. Enviar al servidor un mensaje de petición de servicios y esperar hasta recibir la respuesta.

3. Cerrar el canal de comunicaciones y terminar la ejecución.

Esquematizaremos a manera de ejemplo la secuencia de llamadas de un proceso servidor y del cliente en un sistema UNIX para un sistema de comunicación orientado a conexión y uno no orientado a conexión. Figuras 7 y 8**.**

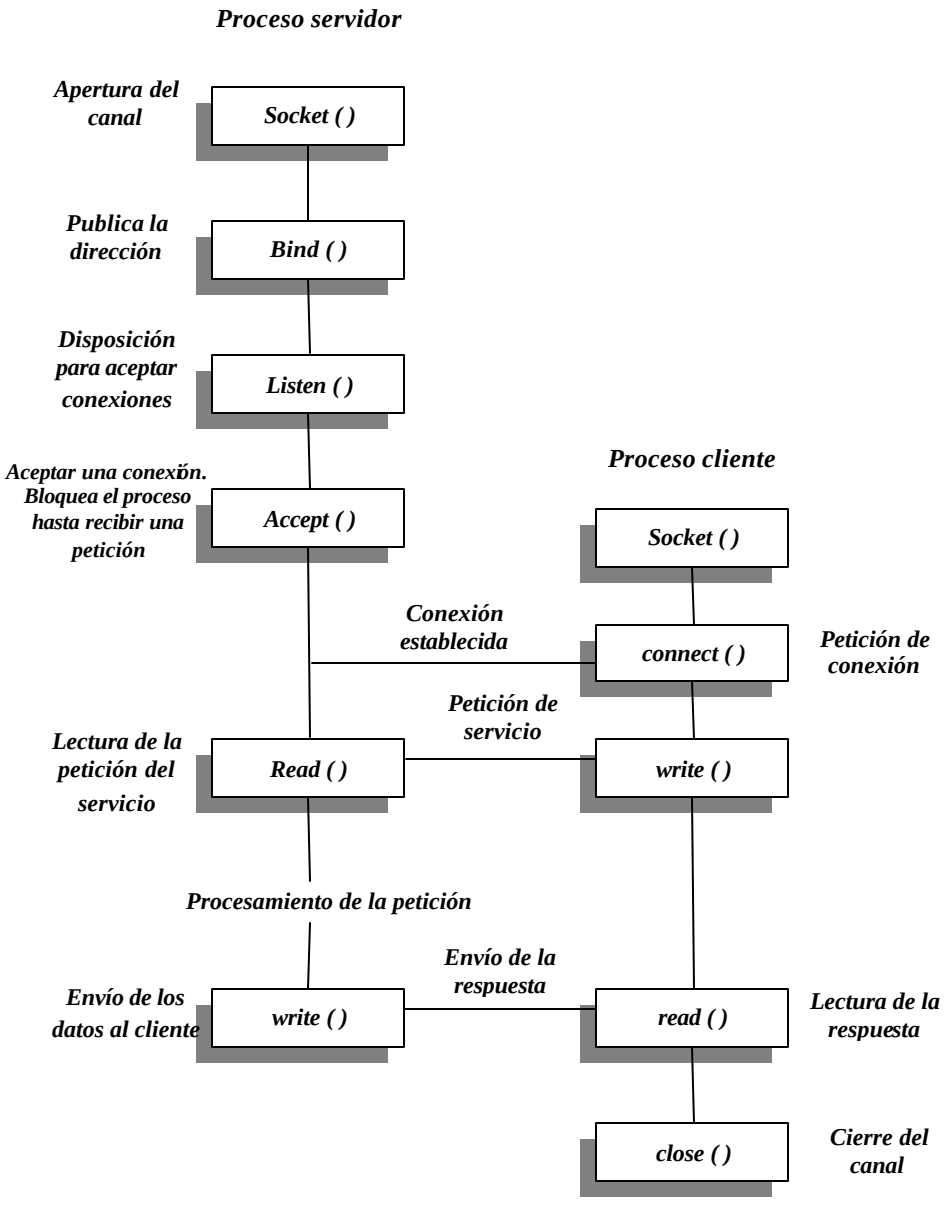

Figura 7. Comunicación Orientada a conexión.

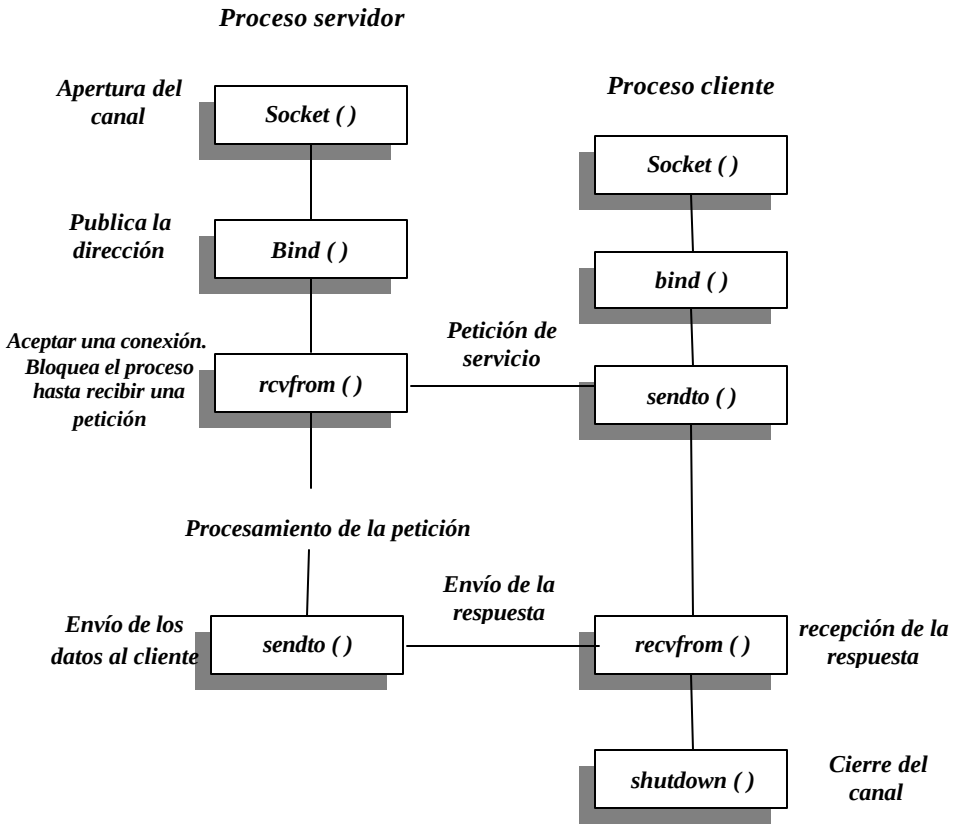

Figura 8. Comunicación No orientada a conexión.

En el diseño de este capítulo se utilizaron los sockets, que se encuentran explicados en el punto 5.5 de Paradigmas de Programación.

1.6 ALGORÍTMICA PARALELA Y CONCURRENCIA

1.6.1 Paso de Mensajes. El paso de mensaje entre procesos asume que existen dos o más procesos en "ejecución paralela" los cuales pueden comunicarse mediante el uso de primitivas de envío y recepción de mensajes ***send/receive,*** las cuales pueden ser provistas por el sistema operativo o bien por una herramienta en particular. Los mensajes son escritos y leídos desde un buffer o bien un canal de comunicación entre los procesos y dependiendo la arquitectura computacional o el ambiente donde ellos se ejecutan, se basarían de un conjunto de protocolos de comunicación para lograr el objetivo de la intercomunicación.

La comunicación entre procesos por medio del paso de mensajes puede hacerse en forma totalmente sincrona, es decir, un proceso no puede continuar su ejecución despues de un send si no hay otro proceso con un receive para los datos enviados, o bien, un proceso se bloquea en una primitiva receive hasta tanto no hayan datos por recibir.

Algunas herramientas de programación que utilizan el paso de mensaje son:

- ✓ Parallel Virtual Machine (PVM) y Message Passing Interface (MPI): para la simulación e implementación de soluciones paralelas.
- ✓ Promela: para la especificación de protocolos.

1.6.2 PVM (Parallel Virtual Machine). Es una herramienta que permite crear aplicaciones paralelas y distribuidas basado en el Modelo de ***Paso de Mensajes***, funciona sobre una colección heterogénea de computadoras corriendo UNIX en una o más redes.

Esta compuesto por dos partes principales, figura 9:

- ✓ Un "demonio": Es un programa que corre en una máquina en background, esperando de otros programas órdenes sobre tareas a realizar. Es llamado pvmd3 y tiene como función coordinar las comunicaciones entre procesos usando PVM, y guardar información sobre los procesos distribuidos. Solo puede haber un demonio por usuario y por máquina y debe correrse en todo el sistema distribuido a usar.

- ✓ Las librerías de rutinas de interfaces (libpvm3.a, libfpvm3.a & libgpvm3.a)

1. ***libpvm3.a:*** Librería en lenguaje C de rutina de interface: Son simples llamadas a funciones que el programador puede incluir en la aplicación paralela. Proporciona capacidad de:

   - ❖ Iniciar y terminar procesos.
   - ❖ Enviar y recibir mensajes.
   - ❖ Sincronizar procesos.
   - ❖ Conocer y cambiar dinámicamente la configuración de la máquina virtual paralela.

Las rutinas de la librería no se conectan directamente con otros procesos, si no que mandan y reciben la configuración de la máquina virtual paralela.

2. ***libgpvm3.a:*** Se requiere para usar grupos dinámicos.

3. ***libfpvm3.a:*** Para programas en Fortran.

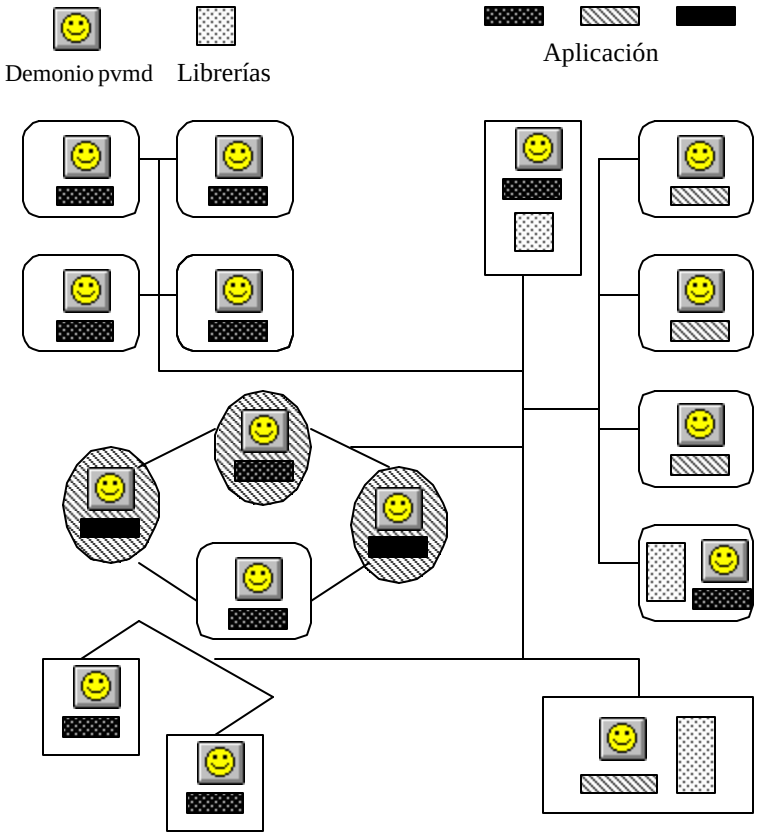

Figura 9. Demonio y librerías en PVM.

2. DESCRIPCIÓN DEL PROYECTO

2.1 ANTECEDENTES DEL PROBLEMA

Dentro del campo de las investigaciones realizadas referentes al estudio de los Sistemas Operativos, en los temas de esta segunda fase, es importante destacar el Proyecto OSIRIS, que ha sido desarrollado durante los años 1995 y 1996 por el grupo de investigación HIDRA del Departamento de Ingeniería de Sistemas y Computación de la Universidad de los Andes (Santafé de Bogotá) con la financiación de COLCIENCIAS (Instituto Colombiano para el desarrollo de la Ciencia y la Tecnología).

Este proyecto tiene como objetivo general realizar investigación conducente a la construcción de un laboratorio de herramientas de software que apoyen el desarrollo de sistemas distribuidos sobre redes de computadoras en la plataforma LINUX. Las herramientas componentes de este laboratorio están asociadas a modelos y metodología de uso que guían el trabajo del desarrollador de sistemas distribuidos.

El laboratorio OSIRIS está compuesto primeramente de Tutoriales que ilustran el uso y la potencialidad de las herramientas LINUX para "*Comunicación entre procesos*" (pipes, colas de mensajes, sockets, etc.). Comprende, además, herramientas de alto nivel que ofrecen al programador de aplicaciones distribuidas transparencia a la red y que responden a 2 paradigmas: RPC y paso de mensajes.

Las herramientas consideradas son Pipes, Fifos, señales, colas de mensajes y sockets. El tutorial consta de textos explicativos y programas en el lenguaje C (ANSI) que ilustran el uso de cada una de estas herramientas en la plataforma LINUX. Para mostrar la sincronización de múltiples procesos clientes y servidores se ilustran ejemplos de programas adicionales que combinan las herramientas.

2.2 EL PROBLEMA DE LA INVESTIGACIÓN

2.2.1 Identificación del Problema. El área de los Sistemas Operativos abarca una serie de temas que si son tratados en forma puramente teórica son de difícil comprensión, puesto que no existe forma de visualizar y seguir paso a paso cada uno de los mecanismos empleados por el sistema operativo para administrar eficientemente los recursos ofrecidos por el hardware.

2.2.2 Planteamiento del Problema. Debido a la naturaleza abstracta de los temas que abarca el área de los sistemas operativos, las personas interesadas en su estudio perciben cada una de sus gestiones y tratan de representarlas a su juicio.

Lo anterior muestra la carencia de una herramienta de apoyo para afianzar el conocimiento en los conceptos de Diseño e Implementación de los sistemas operativos.

2.2.3 Análisis del Problema. Teniendo presente la falta de una herramienta de apoyo en el estudio de los sistemas operativos, se estudio esta área y se decidió buscar el medio para mostrar en forma práctica los siguientes temas:

- ✓ Planificación de Procesos
- ✓ Simulación de un autómata que representa la dinámica de los procesos en un sistema operativo (UNIX)
- ✓ Sincronización entre procesos
- ✓ El modelo Cliente/Servidor
- ✓ Algorítmica paralela y concurrencia

2.2.4 Formulación del Problema. El problema consiste en la falta de una herramienta de apoyo para afianzar el conocimiento en los conceptos de diseño e implementación de los sistemas operativos.

2.3 OBJETIVOS DE LA INVESTIGACIÓN

2.3.1 Objetivo General. Implementación de una herramienta de simulación de los mecanismos de administración y control de un sistema operativo donde el estudiante pueda verificar y afianzar los conocimientos teóricos adquiridos en el curso de sistemas operativos

2.3.2 Objetivos Específicos

- ✓ Reconocer los diferentes tipos de planificación de procesos, partiendo de conceptos teóricos y teniendo en cuenta para cada uno de ellos sus ventajas y desventajas.

- ✓ Mostrar la dinámica de los procesos en el sistema operativo UNIX.

- ✓ Describir la sincronización entre procesos a través de aplicaciones que simulen los diversos métodos para lograr el acceso concurrente a un recurso compartido definido como región critica.

- ✓ Presentar soluciones a una variedad de Problemas Clásicos de la Sincronización entre Procesos

- ✓ Ejemplificar el funcionamiento del modelo cliente/servidor.

- ✓ Apropiación del conocimiento en la programación concurrente a través del lenguaje PVM.

- ✓ Ejemplificar el funcionamiento de la programación concurrente utilizando el lenguaje PVM.

3. ENTORNO DEL DISEÑO DEL SOFTWARE

3.1 INTRODUCCIÓN

Con la elaboración de esta herramienta, se busca darle un giro magistral y completo a las clases teóricas que se dictan actualmente en la Facultad de Sistemas. Este cambio hará que el usuario final, el estudiante y demás personas interesadas en estos temas, estén en la capacidad de asimilar de una manera más clara, todos aquellos conceptos que por su grado de complejidad no han sido comprendidos bien.

El software está diseñado para que por medio de los gráficos, tablas y las sencillas definiciones que presenta la ayuda, se pueda navegar interactivamente por el inmenso mundo de los mecanismos de administración y control de un sistema operativo.

A través de las ventanas que componen este gran programa, se tiene la oportunidad de observar gráficamente algunas operaciones internas que realizan los sistemas operativos. Además, este proyecto puede ser ampliado en un futuro, agregando nuevas simulaciones que permitan incrementar la eficacia y el rendimiento del software.

3.2 POBLACIÓN A QUIEN VA DIRIGIDO

A los estudiantes de la asignatura y las demás personas pertenecientes a una comunidad académica que posean los conocimientos básicos y que están interesadas en el estudio del diseño e implementación de los sistemas operativos.

3.3 PRESENTACIÓN

La "Simulación de los Mecanismos de Administración y Control de un Sistema Operativo" es un programa que se utiliza para afianzar los conceptos teóricos acerca del Diseño e Implementación de los Sistemas Operativos, buscando la forma de hacer de estos temas y de los modelos que los soportan elementos tangibles de fácil asimilación durante su estudio.

El programa desde su inicio es de fácil manejo porque todos los capítulos están compuestos por ventanas que contienen las simulaciones y botones de navegación que proporcionan un rápido aprendizaje del software.

Esta herramienta cuenta con cinco módulos operativos:

- ✓ Módulo de Planificación de Procesos
- ✓ Módulo del Autómata en UNIX
- ✓ Módulo de Sincronización entre procesos

- ✓ Módulo del Modelo Cliente/Servidor

- ✓ Módulo de Algorítmica Paralela y Concurrencia

El módulo de Cliente/Servidor tiene la ventaja de ser ejecutado de dos maneras diferentes: en máquinas distintas dentro de una red o en un solo equipo que posea tarjeta de red. Para visualizar las simulaciones del último módulo es necesario tener Linux, versión 6.0.

3.4 ÁREAS DE CONTENIDO

Este software es un avance conceptual, que nació después de cursar las asignaturas: "Sistemas Operativos" y "Sistemas Operativos Modernos" al observar que los temas allí tratados son puramente teóricos, aveces de difícil comprensión. Por esta razón, esta herramienta conduce a reforzar el proceso enseñanza/aprendizaje y servirá de apoyo tanto al docente como al estudiante interesados en el estudio del diseño e implementación de los sistemas operativos, en temas tan importantes como son la *Comunicación* y *Sincronización* entre Procesos.

3.5 COMPLEJIDAD

El software consta de una interfaz gráfica formada por una serie de botones que le proporcionan al usuario final tendrá la oportunidad de escoger la simulación que desea ejecutar. Sin embargo, es indispensable que las personas que hagan uso de esta herramienta tengan los conocimientos básicos en el área de los sistemas operativos y sus diferentes mecanismos de administración y control.

3.6 CONTROL INSTRUCCIONAL

Esta herramienta por tener una interfaz gráfica puede ser ejecutada por los usuarios finales a través del Mouse o por medio del teclado.

Esta compuesta por una ventana inicial que sirve de punto de partida para mostrar cada módulo, los cuales se desarrollan en ventanas diferentes e independientes. Al desplegar cualquiera de ellas, se observa una serie de controles (botones, cajas de chequeo o radio botones) que sirven para dar inicio a las simulaciones, mostrar definiciones y gráficos, que ayudan en el aprendizaje de los sistemas operativos.

Algunas simulaciones pueden estar compuestas por *Marcos de Páginas,* los cuales son utilizados para mostrar diversos aspectos de ese tema. Otras es necesario ejecutarlas al mismo tiempo con dos ventanas abiertas para fingir una red en un equipo.

Cada simulación no se ejecutan en tiempo real, y puede ser detenida en cualquier momento, aunque los datos obtenidos hasta ese instante no pueden ser recuperados.

3.7 AYUDAS

En este software, cada ventana cuenta con un controlador (botón) que conduce al usuario a las llamadas "*Ayudas*", que están compuestas por una breve descripción de conceptos y definiciones básicas utilizadas en las Simulaciones. Además, posee una serie de *Mensajes* que en un momento determinado indican que está ocurriendo en la ejecución.

3.8 LOS LENGUAJES

Para la elaboración de los primeros cuatro módulos de esta herramienta se utilizó Delphi, versión 4.0 (**Figura 10**), porque presenta las siguiente características:

- ✓ Provee todos los componentes necesarios para la realización de programas que ofrezcan al usuario una interfaz gráfica y sencilla de manejar.
- ✓ Permite trabajar con la POO (Programación Orientada a Objetos), arreglos dinámicos, parámetros por defecto, etc.
- ✓ Presentar objetos que ofrecen la facilidad de implementar aplicaciones Cliente/Servidor.

- ✓ Utiliza los *Sockets* para obtener comunicación entre varios programas, usando el protocolo TCP/IP.
- ✓ Brinda la posibilidad de programar con *Hilos*, para ejecutar secciones de código simultáneamente en forma paralela.

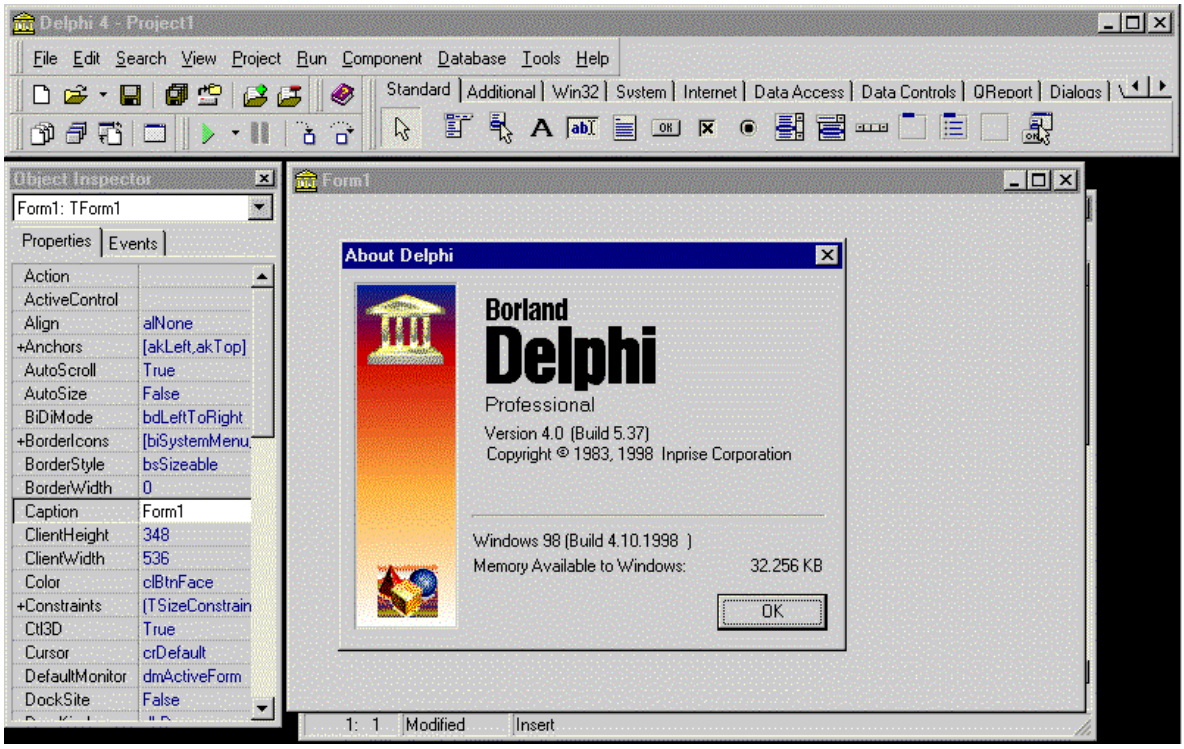

Figura 10. Entorno inicial de Delphi

En el último módulo, para la programación Paralela, se usó PVM (Parallel Virtual Machine), que es un sistema de software que habilita una colección de computadores heterogéneos para ser usado como una coherente y flexible herramienta computacional concurrente, la cual basa todo su funcionamiento en el paradigma de paso de mensajes.

4. DESCRIPCIÓN DE ASPECTOS DE EJECUCIÓN DEL SOFTWARE

4.1 ASPECTOS GENERALES

- ✓ Esta herramienta esta compuesta por una interfaz gráfica que permite al usuario navegar de una manera fácil y sin complicaciones.

- ✓ Al dar inicio a la ejecución, se tiene la posibilidad de escoger entre los cinco módulos operativos que representan los capítulos a tratar en esta fase, los cuales están documentados por medio de conceptos básicos, tablas y gráficas.

- ✓ Algunos módulos están diseñados para que el usuario pueda obtener datos importantes en cualquier instante de la simulación, ya que contienen un *Visor de Log* que los captura mientras este en ejecución dicho programa.

- ✓ Se presentan una serie de mensajes que indican algunas operaciones que están ocurriendo en la simulación.

- ✓ Existen simulaciones que por la naturaleza del tema, requieren para su ejecución, dos máquinas que estén conectadas en red.

- ✓ Las simulaciones pueden ser interrumpidas en cualquier momento por el usuario, pero los datos obtenidos hasta ese momento no podrán ser recuperados.

- ✓ En el último modúlo, como el programa se ejecuta en una máquina en Background no es llamativo su entorno inicial.

- ✓ No posee interfaz gráfica.

- ✓ Para su ejecución es indispensable poseer conocimiento de los comandos en Linux.

4.2 ASPECTOS PARTICULARES

4.2.1 Planificación de Procesos.

- ✓ La ventana principal ofrece dos alternativas, bien sea accesar a las definiciones o dar inicio a cada una de las cuatro simulaciones de Comunicación de procesos.
- ✓ Cada simulación se muestra en ventanas diferentes e independientes.
- ✓ Cada una presenta *Marcos de Páginas* en los cuales se pueden observar el Bloque Descriptor de Procesos (BDP), el autómata y el Visor de Log en forma simultanea.
- ✓ Al finalizar, tiene la posibilidad de obtener resultados como son la eficacia y el tiempo de regreso.
- ✓ No se pueden ejecutar dos simulaciones al mismo tiempo.

4.2.2 Simulación de un Autómata que representa la dinámica de los procesos de un sistema Operativo (UNIX).

- ✓ Es la simulación más sencilla de la herramienta. Sólo cuenta con las definiciones y el programa del Autómata en UNIX.
- ✓ La ventana del Autómata describe brevemente y paso a paso cada unos de los estados por los que pasa un proceso en este sistema Operativo.
- ✓ Como parte de las definiciones se presentan varias ventanas en donde se puede observar en forma ampliada y detallada cada unos de los estados.

4.2.3 Sincronización entre Procesos.

- ✓ La forma principal presenta las opciones para que el usuario decida que simulación acerca de los Problemas Clásicos desea ejecutar.
- ✓ Las simulaciones se encuentran en ventanas diferentes.
- ✓ Cada una, presentan tres *Marcos de páginas:* formulación del Problema, tablas que indican el estado de los hilos con el Visor de Log, y por último, la simulación en forma gráfica Paso a paso.
- ✓ En el programa de *Productores y Consumidores*, el usuario tiene la facilidad de decidir las velocidades de ejecución de los dos hilos.
- ✓ En este módulo no se pueden ejecutar dos problemas Clásicos al mismo tiempo.
- ✓ Durante la ejecución de los programas, se podrá percibir como interactúan los hilos en forma simultanea.

4.2.4 Modelo Cliente/Servidor.

- ✓ Es el módulo que tiene dos posibilidades de ejecución, ya sea en una misma máquina que posea tarjeta de red en dos aplicaciones diferentes o en equipos que estén conectados en red.
- ✓ La ventana principal posee un controlador (botón) que despliega las definiciones necesarias para comprender este módulo. Así como también de las opciones de las diferentes simulaciones.
- ✓ Las simulaciones necesitan de un puerto establecido previamente y de una dirección IP que el usuario digita durante la ejecución.
- ✓ Para la transferencia de archivos de imágenes, no existe ninguna clase de limitaciones.
- ✓ El programa del Protocolo UDP presentan un *Marco de Página,* en donde muestra el envió y la recepción del mensaje.

4.2.5 Algorítmica Paralela y Concurrencia.

- ✓ Este último módulo de la herramienta, requiere la plataforma Linux, versión 6.1.
- ✓ Antes de ejecutar los programas, estos deben compilarse.
- ✓ Hay que verificar que el demonio este activo.

- ✓ Para la ejecución total del programa es necesario, ejecutar primero el maestro y luego el esclavo.

4.3 MENSAJES EN LAS SIMULACIONES

Existen tres clases de mensajes en esta herramienta:

1. **Mensajes de Información:** como su nombre lo indica, informan al usuario acerca de alguna operación que este ocurriendo en un instante determinado. Entre ellos están:

   - ❖ Ha finalizado la Simulación.
   - ❖ Notas importantes.
   - ❖ Mensaje enviado al Servidor.
   - ❖ Falló la Conexión.

2. **Mensajes de Decisión:** ofrecen al usuario la posibilidad de elegir entre dos alternativas. Sólo hay uno:

   - ❖ ¿Desea realizar Paso a Paso para ver el estado de los mensajes?.

3. **Mensajes para completar información:** son aquellos que le piden al usuario alguna clase de información para llevar a cabo su ejecución con éxito:

- Especificar dirección IP.

5. DESCRIPCIÓN DE ASPECTOS DEL DISEÑO DEL SOFTWARE

5.1 ASPECTOS GENERALES

- ✓ La aplicación provee una interfaz amigable que facilita el manejo al usuario final.
- ✓ La simulación no es en tiempo real.
- ✓ Los programas están diseñados de tal forma que todos los parámetros son elegidos aleatoriamente, así cada vez que se inicie una simulación se presentarán nuevas características durante la ejecución.
- ✓ El software está basado en reguladores de tiempo (objetos timers) que, junto con algunos objetos permiten detener y continuar la simulación en cualquier instante.
- ✓ Se manejas diferentes paradigmas de programación como son: hilos, paso de mensajes y sockets.
- ✓ Todas los módulos constan de ayudas que ofrecen una breve descripción de las definiciones utilizadas durante el diseño.

5.2 ASPECTOS PARTICULARES

5.2.1 Planificación de Procesos. Este primer módulo comprende básicamente cuatro simulaciones de las cuales se destacan los siguientes aspectos:

- ✓ Cuando se da inicio a las todas las simulaciones se crean aleatoriamente los parámetros iniciales: número de procesos, sus tamaños, las entradas/salidas y el quamtun.
- ✓ Cada ventana consta de tres *Marcos de Páginas*, en los cuales se puede observar la Máquina de estados Finitos, el Bloque Descriptor de Procesos (BDP) y las barra de prograsos de cada proceso.
- ✓ Para llevar un control más preciso acerca de la ejecución de los procesos, las simulaciones están dotadas también por un *Visor de Log*, que se actualiza constantemente.
- ✓ Todos los datos almacenados en cada uno de los marcos de páginas se perderán y no podrán ser recuperados, al cerrar las simulaciones.
- ✓ En el diseño de todos los programas se utilizaron tres relojes: uno para manejar toda la simulación, otro que manipula las entradas/salidas y uno totalmente independiente, quien se encarga de la Máquina de Estados Finitos.
- ✓ Los planificadores, *El Trabajo más corto* y *Prioridad*, poseen un procedimiento de ordenación que permite el manejo más eficiente y descomplicado de todas las operaciones realizadas sobre la estructura de datos para efectos de la simulación.
- ✓ Para el diseño del planificador *Colas Múltiples con Prioridad* se utilizaron un arreglo y tres clases de prioridad.
- ✓ El diseño de la Máquina de Estados Finitos para la simulación de Colas Múltiples con Prioridad es totalmente diferente a las demás simulaciones de planificación, debido a la naturaleza misma del metodo de planificación de procesos.

- ✓ Al finalizar las simulaciones se presenta una tabla con los indicadores de gestión y rendimiento más relevantes para cada simulación.
- ✓ La ayuda de este módulo consta de cuatro opciones básicas: Planificación, los algoritmos de planificación, la Máquina de Estados Finitos y el Bloque descriptor de Procesos.

5.2.2 Simulación de un Autómata que representa la dinámica de los procesos de un sistema Operativo (UNIX). Este capitulo consta de una sola simulación en la que se sobresalen los siguientes aspectos:

- ✓ En la ventana inicial se puede observar las ayudas del módulo como la simulación.
- ✓ La ayuda esta compuesta por la definición de la Dinámica, el diagrama de estados y un autómata ampliado.
- ✓ En el diseño del programa, se hizo sólo un intercambio de imágenes. Cada una de ellas representan un estado del autómata en UNÍX y son cargadas desde disco cuando son requeridas.
- ✓ La simulación se puede detener y/o continuar en cualquier instante de tiempo.

5.2.3 Sincronización entre Procesos. Esta compuesto por cinco simulaciones que representan cinco de los Problemas Clásicos:

- ✓ En este capitulo se puede apreciar el funcionamiento de los Hilos.

- ✓ Cuando se da inicio a las todas las simulaciones se ejecutan aleatoriamente los hilos y se crean los parámetros de tiempo.
- ✓ Cada ventana consta de tres *Marcos de Páginas*, en los cuales se puede apreciar la definición del Problema, el estado de los hilos y una interfaz opcional para observar el comportamiento de los semáforos y/o los mensajes de los procesos simulados en la aplicación de una manera interactiva.
- ✓ Todas los programas estan compuestos por *Timer*, que simulan el tiempo en el cual un proceso hilo se encuentra realizando alguna actividad.
- ✓ Durante el diseño se hace uso de temas tan importantes como son el uso de semáforos, manejo de la región crítica, paso de mensajes y la programación Orientada a Objetos.
- ✓ En la simulación del *Barbero Dormilón*, sólo se muestra un *Visor de Log*, debido a la naturaleza misma del problema, ya que se quiere que el usuario final generé el número de clientes al azar y no se puede saber con precisión la cantidad de hilos activos durante un instante de tiempo determinado.
- ✓ La simulación de los *Fumadores* es la única que utiliza el paradigma del paso de mensajes, las demás, se basan en los semáforos.

5.2.4 Modelo Cliente/Servidor. Este cuarto y último módulo en Delphi, comprende tres simulaciones de las cuales se destacan los siguientes aspectos:

- ✓ En este capitulo se puede apreciar el funcionamiento de los sockets.
- ✓ La *Ayuda* esta compuesta por las definiciones utilizadas durante todo el capitulo y que facilitan la comprensión de los temas.

- ✓ La simulación del *Protocolo Udp* utiliza el objeto "UDP" de la paleta Internet, esta compuesto por dos *Marcos de Páginas,* el primero envía un mensaje *Encriptado* y en el segundo se recibe y *Desencripta* a la vez, aquí se hace uso de algunos conceptos y algoritmos de Criptografía y Seguridad en Redes.
- ✓ El programa de *Transferencia de archivos de imágenes* está compuesto por un Servidor y un Cliente que se ejecutan independientemente. La imagen a transferir puede ser cualquiera que se encuentre en el disco duro del cliente. Se hace uso del protocolo TCP/IP.
- ✓ La simulación de los *Sockets,* también está compuesto por un Servidor y un cliente que deben ejecutarse en forma independiente. Aquí el mensaje envíado es una cadena de caracteres que son recibidos por un servidor y escritos en un *Memos* "al revés".

5.2.5 Algorítmica Paralela y Concurrencia.

- ✓ La simulación está compuesta por un maestro y un esclavo, basados en la comunicación con paso de mensaje.
- ✓ Se utilizó la herramienta de PVM para lograr el paralelismo.
- ✓ Como lenguaje de programación se usó C ANSI enriquecido con las librerías de PVM.

5.3 LIMITACIONES Y SUPUESTOS

Los términos *Limitaciones* y *Supuestos* son utilizados en el desarrollo de este proyecto con el fin de facilitar el diseño de las diferentes Simulaciones. Por ejemplo, las Llamadas al Sistema o *System Call* por tener máxima prioridad se han omitido en el desarrollo de todos los programas.

5.3.1 Algoritmos de Planificación. En el programa de Colas Múltiples con Prioridad, se ha supuesto que las E/S es un mismo dispositivo independientemente del nivel de prioridad donde se encuentre un proceso.

Tabla 1. Limitaciones.

Parámetros	Intervalo
Tamaño del Quantum.	50 - 150
Número de Trabajos (procesos) aleatorios a crear.	4 - 10
Tamaño de cada Trabajo (proceso)	500 - 1000
Número de E/S	4
Duración (tiempo) de las E/S	12 segundos aproximadamente

Tabla 2. Aleatoriedad para las E/S.

Entradas/Salidas	Intervalo de Posición

1	0 - 250
2	251 - 500
3	501 - 750
4	751 - 1000

5.3.2 Simulación de un Autómata que representa la Dinámica de los procesos de un Sistema Operativo (UNIX). El programa hace su recorrido por el autómata describiendo cada estado para un solo proceso en UNIX. Las interrupciones o Llamadas al sistema se presentan una sola vez.

5.3.3 Algoritmos de Sincronización entre Procesos.

5.3.3.1 Productores y Consumidores. Se manejan velocidades tanto para el Productor como para el Consumidor. Está velocidad no debe ser mayor a 900. El programa está diseñado para que el Consumidor no tenga una velocidad mayor que el Productor.

5.3.3.2 El Barbero Dormilón. Debido a la naturaleza Estocástica de este problema hay eventos, como la interacción Paso a Paso y la tabla de estados, que han sido omitidos para poder mantener la integridad de la aplicación. La simulación sólo soporta máximo 100 clientes en la peluquería.

5.3.3.3 Lectores y Escritores. El programa posee dos Escritores y tres Lectores, los cuales realizan sus actividades en forma aleatoria. Los Escritores no pueden escribir al mismo tiempo, en cambio los Lectores pueden acceder a las bases de datos en cualquier instante.

5.3.4 El Modelo Cliente/Servidor.

5.3.4.1 **UDP.** Utiliza un protocolo no confiable.

5.3.4.2 Simulaciones con Cliente/Servidor. Algunas validaciones de direcciones IP y puertos de comunicación no fueron tomadas en cuenta, por que no son el propósito final de este trabajo de grado.

Además, estas simulaciones van dirigidas a una población con un nivel básico de conocimientos del mundo de las computadoras.

5.4 ESTRUCTURAS DE DATOS RELEVANTES

5.4.1 **Algoritmos de Planificación.** Para las cuatro simulaciones se utilizaron las mismas estructuras de datos, sólo se diferencian en algunos de sus campos

1. ***Estructura de datos: Proc***

Descripción: Define los procesos que serán atendidos por el sistema operativo, dependiendo del tipo de planificador utilizado.

Definición de los campos:

Tabla 3. Campos para la Estructura Proc.

Nombre del Campo	Descripción	Tipo
Pc	Es el contador del Programa	Entero
PcR	Lleva el contador de programa Relativo a cada planificación	Entero
Tam	Almacenar el tamaño escogido aleatoriamente para cada proceso.	Entero
Es1	Determina en que línea se encuentra la E/S1	Entero
Es2	Determina en que línea se encuentra la E/S2	Entero
Es3	Determina en que línea se encuentra la E/S3	Entero
Es4	Determina en que línea se encuentra la E/S4	Entero
Est	Establece en que estado se encuentra cada proceso.	Entero
Orden	Captura el orden de numeración lógica que tienen los procesos.	Entero
Prior	Organiza la prioridad de los procesos.	Entero
Asc	Determina si la prioridad asciende o desciende.	Entero
ConQuan	Contador del Quantum para cada ejecución.	Entero

Nombre	Es el identificador del proceso.	Cadena

2. ***Estructura de datos: Procesp***

Descripción: Define los procesos que se encuentran en los estados de E/S y en espera de E/S.

Definición de los campos:

Tabla 4. Campos para la Estructura Procesp.

Nombre del Campo	Descripción	Tipo
Est	Establece si se encuentra en E/S o en espera de E/S	Entero
Orden	Captura el orden de numeración lógica que tienen los procesos.	Entero
Clase	Permite estable la clase de prioridad en que se encuentra un proceso.	Entero

5.4.2 Algoritmos de Sincronización entre Procesos. Para el diseño de los problemas Clásico se utilizaron las siguiente operaciones sobre semáforos:

Operación Down:

```
procedure down(var t:boolean);
begin
 while (true) do
   begin
```

```
   if (t=true) then
    begin
      t:=false;
      exit;
    end;
   end;
end;
```

Operación Up:

```
procedure up(var t:boolean);
begin
 while (true) do
   begin
   if (t=false) then
    begin
      t:=true;
      exit;
    end;
   end;
end;
```

5.5 PARADIGMAS DE PROGRAMACIÓN

A continuación se mencionarán los paradigmas de programación utilizados durante el diseño de todas las simulaciones, acompañados de un ejemplo práctico o código fuente.

5.5.1 Hilos. Un programa construido mediante hilos tiene el potencial de incrementar el rendimiento total de la aplicación en cuanto a productividad y tiempo de respuesta mediante la ejecución de un código asíncrono y paralelo

Otros beneficios de los hilos están relacionados con los recursos compartidos y el escaso tiempo empleado en su creación, terminación y cambio de contexto. Todo esto contribuye a incrementar el rendimiento de la aplicación así como conservar los recursos del sistema.

Uno de los significados exactos del término Hilos denota a procesos ligeros (con flujo de control secuencial) que comparten un espacio de direcciones y algunos otros recursos, y para los cuales el tiempo empleado en el cambio de contexto es mucho menor que el empleado en los procesos pesados (procesos soportados por el kernel del sistema operativo).

En una máquina con un sólo proceso que tiene varios hilos, estos suelen llamarse procesos ligeros. Los hilos son como miniprocesos, ya que cada hilo se ejecuta de forma estrictamente secuencial.

Los hilos comparten la CPU al igual que lo que los procesos en un sistema de tiempo compartido y pueden crear hilos hijos. Al igual que los procesos tradicionales, los hilos poseen los siguientes estados:

1. En ejecución (Running).
2. Bloqueado (Suspend).
3. Listo o preparado (Ready).
4. Terminado (Finished).

Los hilos son un mecanismo que permite mejorar el rendimiento de los Sistemas Operativos tratando de reducir la sobrecarga producida por el cambio de contexto entre procesos.

Ejemplo: La Cena de los Filosofos;

```
procedure TFFilosofo.Button1Click(Sender: TObject);
begin

  {Número de hilos a crearse}
  hilosnum := 5;

  {Crea al hilo Filósofo 1}
  x0:=TFilosofo.Create(0);
  with x0 do
  OnTerminate := Terminado;

  {Crea al hilo Filósofo 2}
  x1:=TFilosofo.Create(1);
  with x1 do
  OnTerminate := Terminado;

  {Crea al hilo Filósofo 3}
  x2:=TFilosofo.Create(2);
  with x2 do
  OnTerminate := Terminado;

  {Crea al hilo Filósofo 4}
  x3:=TFilosofo.Create(3);
  with x3 do
  OnTerminate := Terminado;

  {Crea al hilo Filósofo 5}
  x4:=TFilosofo.Create(4);
  with x4 do
  OnTerminate := Terminado;

end;

{Destrucción de los Hilos}
procedure TFFilosofo.BitBtn1Click(Sender: TObject);
begin
  if (button1.enabled=false)then
```

```
    begin
     x1.destroy;
     x2.destroy;
     x3.destroy;
     x4.destroy;
     x0.destroy;
   end;
end;

{Botón de Continuar la Animación}
procedure TFFilosofo.Button2Click(Sender: TObject);
begin
  x0.Resume;
  x1.Resume;
  x2.Resume;
  x3.Resume;
  x4.Resume;
end;

{Detiene la Simulación}
procedure TFFilosofo.Button5Click(Sender: TObject);
begin
x0.Suspend;
x1.Suspend;
x2.Suspend;
x3.Suspend;
x4.Suspend;
end;
```

Unit CDeclara;
Se especifican las clases y objetos que se manejan en la Simulación.

```
interface

type
  TFilosofo = class(TThread)
  public
   Id:byte;
   constructor Create(i:byte);
  private
  protected
    procedure Execute; override;
  end;

implementation

uses  Filosofos;
```

```
{Crea los Hilos}
constructor TFilosofo.Create(i:byte);
begin
 Id:=i;
 FreeOnTerminate := True;
 inherited Create(False);
end;

{Ejecuta los Hilos}
procedure TFilosofo.Execute;
 Const
   N = 5;
   Pensando = 0;
   Hambriento = 1;
   Comiendo = 2;
 Var
   IZQ,DER:shortint;
begin
   IZQ:=(id-1) mod N;
   if (IZQ=-1)then
      IZQ:=4;
   DER:=(id+1) mod N;
  while (true) do
    begin
      {Llamado al Procedimiento Pensar}
      Pensar(id);
      {Llamado al Procedimiento TomaTenedor}
      TomaTenedor(id);
      {Llamado al Procedimiento Comer}
      Comer(id);
      {Llamado al Procedimiento PoneTenedor}
      PoneTenedor(IZQ,DER,id);
    end;
end;
```

5.5.2 Sockets. Son una generalización del mecanismo de acceso a archivos de UNIX que proporciona un punto final de la comunicación. Al igual que el acceso a archivos, cuando se crea un socket el sistema devuelve un entero pequeño que será el descriptor del socket utilizado en el cuerpo de la aplicación.

La principal diferencia entre un descriptor de archivo y un descriptor de socket es que el descriptor de archivo es asociado a un archivo o dispositivo especifico, mientras que al crear un socket no en ese momento se asocia a un punto de comunicación, ello se realiza después con algunas primitivas disponibles para este propósito.

Ejemplo: Servidor

```
{Activa el Servidor Sockets}
procedure TFServidor.FormCreate(Sender: TObject);
begin
  ServerSocket1.Active := True;
end;

{Desactiva el Servidor Sockets}
procedure TFServidor.FormDestroy(Sender: TObject);
begin
  ServerSocket1.Active := False;
end;

{Procedimiento que recibe el mensaje enviado por el cliente}
procedure TFServidor.ServerSocket1ClientRead(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  Mensaje := Socket.ReceiveText;
end;

{Establece la conexión con el Cliente}
procedure TFServidor.ServerSocket1ClientConnect(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  Memo1.Lines.Add('En Conexión...');
end;

{Se desconecta del Cliente}
procedure TFServidor.ServerSocket1ClientDisconnect(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  Memo1.Lines.Add('Desconectado');
end;
```

```
{Procedimiento para escuchar las peticiones de los Clientes}
procedure TFServidor.ServerSocket1Listen(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  StatusBar1.Panels.Items[0].Text := 'Escuchando...';
end;
```

Cliente

```
{Establece conexión con el Servidor}
procedure TFCliente.ClientSocket1Connect(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  Statusbar1.Panels[0].Text := 'Conectado a ' + ClientSocket1.Host;
end;

{Desconectado del Servidor}
procedure TFCliente.ClientSocket1Disconnect(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  Statusbar1.Panels[0].Text := 'Desconectado';
end;

{Procedimiento que indica que hubo un error al conectar}
procedure TFCliente.ClientSocket1Error(Sender: TObject;
  Socket: TCustomWinSocket; ErrorEvent: TErrorEvent;  var ErrorCode: Integer);
begin
  Memo1.Lines.Add('Error al conectar a ' + ClientSocket1.Host);
  ErrorCode := 0;
end;

{Escribir la dirección IP con quien se quiere establecer conexión y activa al Cliente}
procedure TFCliente.ConectarClick(Sender: TObject);
begin
  if ClientSocket1.Active then
  ClientSocket1.Active := False
   else begin
      ClientSocket1.Host := Edit1.Text;      ClientSocket1.Active := True;
   end;
end;

{Cuando el usuario presiona Enter, se envia la línea anterior}
procedure TFCliente.UsuarioKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if Key = VK_Return then
```

```
    ClientSocket1.Socket.SendText(Usuario.Lines[Usuario.Lines.Count - 1]);
end;

{Datos que han sido recibidos del Servidor}
procedure TFCliente.ClientSocket1Read(Sender: TObject;  Socket: TCustomWinSocket);
begin
  Memo1.Lines.Add(Socket.ReceiveText);
end;
```

6. RECOMENDACIONES

- ✓ Para incrementar la dinamica de las simulaciones del capitulo de Planificación de Procesos se podria pensar en ampliar el nivel de parametrización de las mismas.

- ✓ Con el objetivo de aumentar la variedad de algoritmos de Planificación de Procesos se podrian implementar aquellos que no fueron realizados en este trabajo.

- ✓ Se podría reforzar la capacidad didáctica de las simulaciones de Sincronización de Procesos si se implementan los Problemas Clásicos por medio de otros mecanismos, tales como monitores o contadores de eventos.

- ✓ Con el propósito de conseguir una real Dinámica del Autómata de los procesos en UNIX se podrían incluir un mayor número de procesos dentro de la simulación.

- ✓ En el capitulo de Cliente/Servidor, debido a que este es un trabajo orientado al área de los Sistemas Operativos, sólo se quiso dejar en claro el concepto, por tal razón se podría ampliar y mejorar en estas simulaciones.

- ✓ Si personas que consulten esta tesis tienen interés en ampliar los conocimientos respecto a Programación Paralela, pueden revisar otros trabajos de grado existentes en la biblioteca de la CUTB.

- ✓ Los mecanismos de Administración simulados no son los únicos, existen otros que resultan de la variación o combinación de ellos; por tanto se recomienda que las personas interesadas en este estudio y que hagan uso de esta herramienta propongan sus propias variantes las cuales a su vez podrían ser simuladas.

- ✓ Una opción alterna a este software podrían ser la elaboración de un tutor con fines netamente académicos que posea el soporte de una metodología educativa.

- ✓ Para estudios posteriores sería interesante llevar a cabo el Diseño e Implementación de un prototipo de Sistema Operativo.

7. CONCLUSIONES

- ✓ Con la implementación de cada uno de los algoritmos de Planificación de Procesos se afianzarón los conceptos teóricos y se comprobarón las ventajas y desventajas de cada uno.

- ✓ Se logró ilustrar de una manera sencilla, clara y didáctica la dinámica que siguen los procesos dentro del Sistema Operativo UNIX.

- ✓ Por medio de la solución dada a los diversos Problemas Clásicos, se consiguió describir la Sincronización entre Procesos a través de aplicaciones que simulen los diversos mecanismos, como *Semáforos* y *Paso de mensajes*, y así lograr el acceso concurrente a un recurso compartido definido como región crítica de una manera eficiente.

- ✓ A través de las simulaciones del modelo Cliente/Servidor se pudo establecer una comunicación y a la vez clarificar el concepto del modelo.

- ✓ Después de este trabajo tenemos una visión más clara del paradigma de programación concurrente y de la herrramienta de progración PVM.
- ✓ Se alcanzarón todos los objetivos propuestos al inicio de este trabajo, en un cien por ciento, ya que se consiguió la implementación de está herramienta de Simulación, en donde se podrán verificar y afianzar todos los conociminetos teóricos de la materia de Sistemas Operativos.

BIBLIOGRAFÍA

TANENBAUM, Andrew S. Sistemas Operativos Modernos. México : Editorial Prentice Hall Hispanoamericana, S.A., 1.993. Págs. 1 - 80.

MILENKOVIC, Milán. Sistemas Operativos, Concepto y Diseño. Madrid, España : Editorial McGraw Hill, 1.994. Págs. 3 - 199.

MÁRQUEZ GARCÍA, Francisco Manuel. Programación Avanzada. USA, 1994. 484 p.

WEB SITES:

www.epm.ornl.gov/pvm

www.labvis.unam.mx/~dulce/docs/pvm_descripcion.html

www.mac.cie.uva.es/pvm.html

www.lola.ii.uam.es/~juan/docencia/pvm/tutorial.html

www.fciencias.ens.uabc.mx/~malba/school/pvm/html

www.ins.udc.cl/~chernand/sist_computacion/c2.3.html

www.ins.udc.cl/~chernand/compdrs/cap2.html

www.fciencias.ens.uabc.mx/notas_cursos/so2/exp2.html

www.taquion.ivic.ve/tutoriales/load/node15.html

www.agamenon.uniandes.edu.co/infocur

MATERIAL ACOMPAÑANTE

Además del documento final del proyecto, se entregará a la Corporación Universitaria Tecnológica de Bolívar:

- Software en formato CD que contine documentos, gráficos, instaladores y simulaciones con su respectivo código fuente.
- Manual del Usuario que hacen parte del anexo de este documento.