CONCEPTOS DE SISTEMAS OPERATIVOS

P R I M E R A E D I C I Ó N

(corregida... herrar es umano...)

Pablo Bossi
Rodrigo Castro
Luján Del Río
Ariel Eidelstein
Gustavo González
Pablo Hidalgo
Ernesto Muñóz
Juan Pablo Proazzi
Gustavo Schmidt
Augusto Vega
Pablo Wolfus

INDICE

- Historia y comparaciones 4
- Procesos 9
- Link Edición 18
- Administración de memoria 23
- Graphical User Interfaces (GUI) 28
- Administración de archivos 31
- Sistemas operativos multimediales 37
- Palm OS 40
- Windows NT / 2000 44
- Real Time 49

PREFACIO

El presente trabajo fue realizado por un grupo de alumnos de la materia *Sistemas Operativos*, Facultad de Ingeniería (U.B.A.), con el único objetivo de proporcionar una guía clara, de fácil lectura, y completa sobre una gran variedad de temas del campo de la tecnología de los sistemas operativos. Comenzó como una necesidad en la etapa preparatoria de exámenes finales, y culminó como una interesante recopilación, estructurada en una dinámica "pregunta-respuesta", de manera tal que la lectura y comprensión de los temas sea lo más rápida y directa posible. Si bien el presente material es lo suficientemente completo, requiere de una base previa de conocimientos, y además este trabajo no pretende ser reemplazo de los libros de texto, sino simplemente una guía. Recomendamos entonces que, para una total comprensión de la materia, la presente guía sea leída acompañada de por lo menos algún libro sobre conceptos básicos de sistemas operativos.

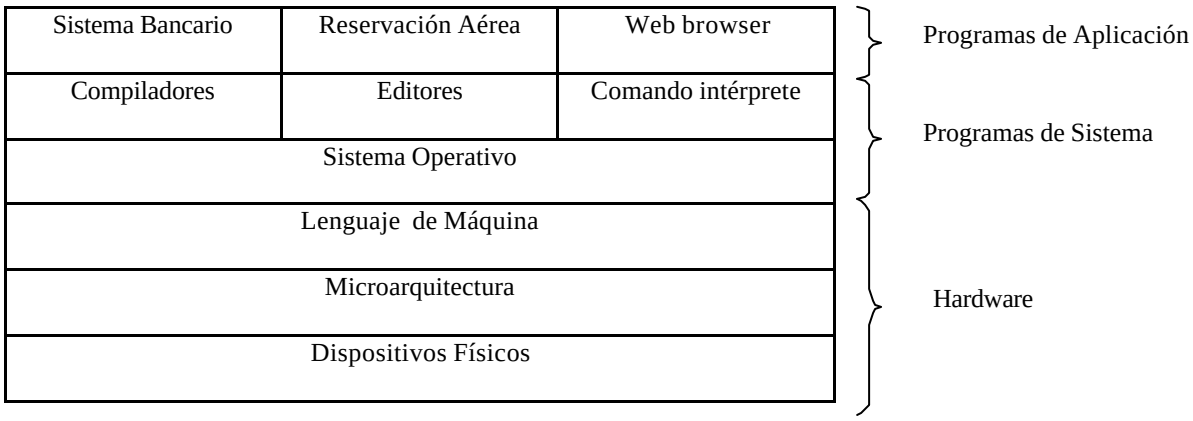
Deseamos que este trabajo sea lo más útil posible al interesado lector, y esperamos poder continuar enriqueciéndola con nuevos aportes de los autores presentes, de futuros alumnos de la materia *Sistemas Operativos*, o de quien se interese en la misma. No podemos cerrar este prefacio sin el explícito agradecimiento al Profesor Lic. Ing. Osvaldo Clúa, quien realizó un gran aporte para que este sueño se hiciera realidad.

Los autores
Invierno de 2002

HISTORIA Y COMPARACIONES

1) ¿En qué conceptos se basa el modelo de máquina multinivel?

Un actual sistema de computadora consiste en uno o más procesadores, memoria, discos, impresoras, teclado, pantalla, etc. Escribir programas que mantengan un orden en el uso de estos dispositivos de manera de poder usarlos en forma correcta, es una tarea difícil. Por esta razón las computadoras están equipadas con un sistema operativo, cuyo trabajo es manejar todos estos dispositivos y permitirle al usuario una interfaz más simple que el hardware.
En el siguiente diagrama se representa un posible modelo de máquina multinivel, en él puede verse además el lugar que ocupa el sistema operativo.



2) Describa las características de cada nivel, los elementos de interés y las actividades de quien diseña sobre ellos.

Debajo de todo se encuentra el hardware, el cual está compuesto por tres niveles. El primero contiene dispositivos físicos, como circuitos integrados, chips, cables, suministro de electricidad, tubos de rayos catódicos y dispositivos similares. El siguiente es el nivel de microarquitectura, en el cual los dispositivos físicos se agrupan para formar unidades funcionales. Este nivel contiene algunos registros internos de la CPU y un data path que contiene una unidad aritmética lógica. En algunas máquinas, la operación de el data path es controlada por software, el cual recibe el nombre de microprograma. En otras máquinas, éste es controlado por circuitos de hardware. El propósito de el data path es ejecutar conjuntos de instrucciones, las cuales quizás usan registros u otras facilidades del hardware.
El tercer y último nivel que constituye el hardware recibe el nombre de lenguaje de máquina, el cual típicamente tiene entre 50 y 300 instrucciones, la mayoría para mover datos a través de la máquina, para hacer operaciones aritméticas y también para comparar valores. En este nivel los dispositivos de entrada salida se manejan cargando valores dentro de registros especiales para esos dispositivos.
Para ocultar la complejidad que presenta el hardware, existe el sistema operativo. El cual además de ocultar al hardware, le entrega al programador un conjunto de instrucciones más convenientes para poder trabajar con él.
Arriba del sistema operativo están el resto de programas del sistema, aquí encontramos el shell, sistemas de ventanas, compiladores, editores y similares aplicaciones independientes. Es importante darse cuenta que estos programas no son parte del sistema operativo.
Por último y para finalizar, en el nivel superior están los programas de aplicación, los cuales son escritos por los usuarios con la finalidad de resolver problemas particulares.

3) Defina brevemente un Sistema Monotarea, un Sistema Batch, un Sistema de Time Sharing, una Workstation, un Sistema Personal, un Sistema Distribuido habilitado por Web y un Sistema Distribuido Orientado a Objetos.

<u>Sistema Monotarea</u>

Como su nombre lo indica es un sistema capaz de realizar una única tarea por vez, por lo que el tiempo de desperdicio de los recursos es demasiado grande comparado con el de los sistemas de multiprogramación.

Sistema Batch
La característica principal de este tipo de sistemas es que permite agrupar distintos trabajos a procesar en forma consecutiva, para que así la máquina los pueda ejecutar en forma secuencial. Su traducción al castellano sería algo así como sistema de procesamiento por lotes.

Sistema de Time Sharing
En castellano sería sistema de tiempo compartido. Permite que más de un usuario tenga acceso a la misma máquina. Esto se debe a que la interacción del usuario con la CPU es bastante pequeña en tiempo, por eso mientras un usuario A está pensando, otro usuario B está procesando un conjunto de datos X.

Workstation
Es una máquina personal de gran tamaño como una Sun, Apollo o Vaxstation, cuyo hardware es más poderoso, rápido y complejo que las computadoras personales comunes.

Sistema Personal
Al reducirse los costos de hardware, cada vez de se ha hecho más factible contar con un sistema de computación para un solo usuario, a este tipo de computadoras se las conoce comúnmente como computadoras personales.

Sistema Distribuido habilitado por Web
Los usuarios están enterados de la multiplicidad de máquinas y para el acceso a estos recursos necesitan conectarse a la máquina remota apropiada o transferir datos de la máquina remota a la propia.

Sistema Distribuido Orientado a Objetos
La más reciente innovación en el diseño de un sistema operativo es el uso de tecnologías orientadas a objetos. Una estructura como esta basada en objetos permite a los programadores construir a gusto un sistema operativo sin desestabilizar la integridad del sistema. Una orientación a objetos también facilita el desarrollo de sistemas operativos distribuidos.

4) Para cada elementos de los anteriores, indique cual de los elementos actuales de un Sistema Operativo aparece y como evoluciona a través de ellos.

Sistemas Operativos para Mainframes
Los sistemas operativos para Mainframes están fuertemente orientados hacia el procesamiento de muchos trabajos al mismo tiempo, muchos de los cuales necesitan grandes cantidades de E/S.
En la actualidad un sistema de estas características es el batch, en el cual procesa en forma continua y secuencial un gran número de trabajos sin interacción de usuario, como por ejemplo hacer un reporte de ventas de una cadena de almacenes o tiendas.
Otro ejemplo es un sistema de timesharing, el cual permite a una gran cantidad de usuarios a correr trabajos en una computadora al mismo tiempo, como por ejemplo consultas a una gran base de datos.

Sistemas Operativos para un Servidor
Este tipo de sistemas corren en servidores que pueden ser grandes pc, workstatios, o mainframes. Ellos prestan servicio a múltiples usuarios a la vez a través de una red y le permite a los mismos, compartir recursos de hardware y software. Los servidores pueden proveer servicios de impresión, de archivos, o de Web. Los proveedores de internet ponen en funcionamiento muchas máquinas para soportar a sus clientes y a los sitios Web que los mismos solicitan.

Sistemas Operativos para una Computadora Personal
En esta categoría el sistema operativo tiene como tarea principal proveer una buena interfaz a un único usuario. Ellos son usados ampliamente para procesadores de textos, planillas de cálculo y acceso a internet

Sistema Operativos de Red

La principal función de un sistema operativo de red es ofrecer un mecanismo para transferir archivos de una máquina a otra. En este entorno, cada instalación mantiene su propio sistema de archivos local y si un usuario de instalación A quiere acceder a un archivo en la instalación B, hay que copiar explícitamente el archivo de una instalación a otra.
La idea básica de la Web, es hacer un sistema distribuido que parezca una gran colección de documentos enlazados. Una segunda aproximación es hacer un sistema distribuido que parezca un gran file system.
Usar un modelo de file system para un sistema distribuido significa que hay un único file system, con múltiples usuarios, de todas partes del mundo, capaces de leer y escribir archivos para los cuales tengan autorización.

Sistemas Operativos Orientados a Objetos
Windows NT y otros sistemas operativos recientes confían plenamente en los principios del diseño orientado a objetos. El cual tiene como concepto principal el objeto. Un objeto es una unidad de software que contiene una colección de datos y procedimientos. Generalmente esta colección de datos y procedimientos no son visibles fuera del objeto. Para ello hay definido interacciones que permiten a otro software tener acceso a los datos y a los procedimientos.

5) ¿Cuáles son las tareas de administración de un Sistema Operativo? ¿Qué objetivos guían esta visión?

El sistema operativo básicamente desempeña dos funciones que no presentan relación alguna entre si y que son, extensión de la máquina y administración de recursos.
El sistema operativo como una extensión de la máquina es una definición que surge debido a la arquitectura que presentan la gran mayoría de éstas, en el nivel de lenguaje de máquina. Esta arquitectura es primitiva y complicada de programar. El sistema operativo le oculta el hardware al usuario y le presenta un punto de vista mucho más simple y entendible para leer y escribir archivos, y realizar distintas operaciones con ellos. También se encarga de esconder un montón de operaciones desagradables que tienen que ver con las interrupciones, la administración de memoria, etc. Desde este punto de vista el sistema operativo se presenta al usuario como el equivalente de una máquina extendida que es más fácil de programar que el subyacente hardware.
Una alternativa a esta definición, es pensar al sistema operativo como un administrador de recursos. Considerar que su trabajo es proveer una asignación, en forma ordenada y controlada, de los procesadores, memorias, y dispositivos de E/S entre todos los programas que compitan por ellos. Este punto de vista del sistema operativo nos muestra que su principal tarea es mantener conocimiento de quien está usando que proceso, para otorgar respuestas a los pedidos de recursos, y para mediar entre conflictos que puedan surgir entre diferentes programas y usuarios por los recursos pedidos.

6) Ensaye una justificación de la división en períodos de la historia de la Arquitectura de Computadoras como lo hace Tanenbaum en 1.2 o como se hizo en clase.

Períodos de la computación:

- Primera Generación 1945-1954. Tubos de Vacío y Plugboards.
- Segunda Generación 1955-1965. Transistores y Sistemas Batch.
- Tercera Generación 1966-1980. Circuitos Integrados y Multiprogramación.
- Cuarta Generación 1981-presente. Computadoras Personales.

7) ¿Qué grandes lineamientos de diseño aparecen en cada una de estas etapas?

Primera Generación:

- Desarrollada durante la guerra.
- Tubos de vacío.
- Mucha gente desarrolló máquinas de calcular automáticas.
- Las máquinas ocupaban gran espacio, llenaban habitaciones enteras.
- No había sistemas operativos.
- No había lenguajes de programación.
- Los usuarios alquilaban las computadoras por tiempo. Mayormente cálculos científicos y matemáticos.

Segunda Generación:

- Aparecen los transistores.
- Las máquinas se vuelven más confiables.
- Más factible para los productores vender computadoras a clientes.
- Se originó el trabajo de operador de computadora.
- Aparecieron los trabajos batch, que se introducían en las máquinas, vía tape para mejorar el rendimiento.
- Surge el primer sistema operativo.

Tercera Generación:

- Los circuitos integrados reemplazan a los transistores.
- Sistemas Mainframe.
- IBM crea System/360 corriendo como sistema operativo el OS/360.
- El comienzo de la multiprogramación, la cual apunta hacia la necesidad de un sistema operativo más complejo.
- Spooling. El código de un job puede leerse desde disco y cargarse a memoria cuando se necesite. Ambos input y output.
- Se introduce el concepto de time sharing el cual permite trabajo interactivo.
- Los sistemas operativos comienzan a volverse más complejos debido a que tiene que tratar con todos estos conceptos nuevos.

Cuarta Generación:

- Integración a gran escala (LSI).
- Gran desarrollo de la PC (personal computer).
- Uno de los requerimientos para la original IBM PC fue un sistema operativo - Bill Gates provee el MS-DOS
- Comenzaron a utilizarse en procesadores no-Intel, el sistema operativo UNIX.

8) ¿Cómo se organizaba el trabajo del Centro de Cómputos en la época de los procesadores periféricos y del Block Time?

Durante el primer período de la historia de la computación después de las primeras máquinas mecánicas, surgieron aquellas que se constituían por tubos de vacío. Estas máquinas eran enormes, tanto que llenaban habitaciones enteras con decenas de miles de tubos de vacío, pero aún así eran millones de veces más lentas que las máquinas personales de hoy en día.
En esos días, un único grupo de personas diseñaban, construían, programaban, operaban y mantenían cada máquina. Toda la programación era hecha absolutamente en lenguaje de máquina, operando con el cableado de las plugboards, para controlar las funciones básicas de la máquina. Los lenguajes de máquina eran desconocidos y ni hablar de un sistema operativo. El modo usual de operación para el programador era firmar por un bloque de tiempo, en una especie de pizarrón que había en la pared, y entonces iba a la sala de computadoras, insertaba su plugboard dentro de la computadora y pasaba las pocas horas que tenía esperando que ninguno de los 20.000 tubos de vacío, o más, no se quemase durante la ejecución del programa. Generalmente los principales problemas eran los cálculos matemáticos de senos, cosenos y logaritmos.

9) ¿Cómo se organizaba el trabajo en la época de un Mainframe de procesos Batch?

La aparición de los transistores a mediados de los 50 produjo un cambio radical a la hora de la construcción de máquinas. Estas se volvieron más confiables, por eso pudieron ser construidas esperando que alguien pague por ellas. Pero solamente grandes corporaciones, como agencias del gobierno o universidades, podían afrontar un costo de millones de dólares.
Estas máquinas, llamadas mainframes, se encontraban encerradas en habitaciones especiales con aire acondicionado, y eran manejadas por un staff de operadores profesionales. Para hacer correr un job, el programador debía en primera instancia escribir el programa en papel (en FORTRAN o en assembler), después pasarlo a tarjetas especiales que se usaban como input. Una vez hecho esto se dirigía a la sala de input y entregaba a uno de los operadores de la máquina, luego se iba y no volvía hasta que la máquina terminase su tarea.

Cuando la computadora terminaba cualquier trabajo que estuviese corriendo, un operador iba hacia la impresora, retiraba los resultados obtenidos y los llevaba hacia la sala de output, para que el programador pueda retirarlos. Como podrá apreciarse, se perdía mucho tiempo en el ida y vuelta de los operadores.
La solución adoptada fue el sistema de procesamiento por lotes o sistema batch. La idea consistía en juntar una pila llena de jobs, en la sala de input y pasarlos a una cinta magnética a través de una máquina pequeña y barata, como la IBM 1401. Mientras que luego una máquina mucho más cara, como la IBM 7094, realizase los cómputos.
Después de juntar los jobs en un tape, este se rebobinaba y se llevaba a la sala de máquina, donde se montaba en un tape drive. El operador entonces cargaba un programa especial (el ancestro del sistema operativo), el cual leía los jobs en forma secuencial y los corría. Cada vez que un job terminaba de ejecutarse, el sistema operativo automáticamente leía el próximo job del tape. Las salidas de cada job eran almacenadas en otro tape, el cual una vez que se terminaba con todos los jobs, se llevaba a la impresora.

10) ¿Qué papel juega Java en estas últimas etapas de la evolución?

Los sistemas operativos más pequeños corren sobre tarjetas inteligentes (smart cards), que contienen su CPU en un pequeño chip. Estos sistemas tienen un gran poder de procesamiento como también restricciones de memoria. Algunos de ellos pueden manejar solamente una función, pero otros pueden manejar múltiples funciones en la misma tarjeta.
Algunas de estas tarjetas son orientadas a Java, esto significa que la ROM de la tarjeta contiene un intérprete de applets (programas Java), que pueden ser bajados y corridos en estos sistemas. Muchas de estas tarjetas pueden manejar más de uno de estos applets al mismo tiempo, por lo que la administración y protección de recursos también se convierten en un punto importante cuando dos o más applets están presentes al mismo tiempo.

11) Defina un Sistema de Información Multimedial y dé algunos ejemplos de aplicaciones.

El término ‘multimedia’ hace referencia a la capacidad de presentar simultáneamente tipos variados de información, como parte de un diseño común.
Películas digitales, video clips, y temas musicales se están convirtiendo rápidamente en una manera entretenida de presentar información a través de la computadora. Los archivos de audio y video pueden ser almacenados en disco y reproducidos en cualquier momento. Sin embargo sus características son muy diferentes a los tradicionales archivos de textos que fueron diseñados por los actuales file systems. Como consecuencia de esto, nuevos file systems son necesarios para manejar este tipo de archivos. Más aún todavía, almacenar y reproducir audio y video exige nuevas demandas al scheduler y a otras partes del sistema operativo.
Los archivos multimedia consisten en múltiples procesamientos en paralelo, generalmente un video, un audio y a veces también un subtítulo. Todo esto debe ser sincronizado durante la reproducción. Por eso es necesario un sistema operativo capaz de manejar este tipo de archivos.

PROCESOS

1) Describa las diferencias entre *scheduling*, *dispatching* y *conmutación de contexto*.

En un sistema con *multiprogramación*, ***scheduling*** consiste en elegir un proceso entre varios en estado *ready*, para otorgarle tiempo de CPU, y que pueda realizar sus tareas. Para ello se aplica un algoritmo (***scheduling algorithm***), y el mecanismo que lo lleva a cabo se conoce como ***scheduler*** (Modern Operating Systems; A. Tanenbaum; pág. 132).
El *scheduler* tiene en cuenta los siguientes items:

- Cantidad requerida de recursos.
- Cantidad actualmente disponible de recursos.
- Prioridad del trabajo o proceso.
- La cantidad de tiempo de espera.

Dispatching consiste en darle tiempo de CPU al proceso seleccionado por el *scheduler*, lo cual implica las siguientes operaciones:

- Conmutar el contexto (*context switching*).
- Cambiar a modo usuario (*user mode*).
- Saltar a la posición apropiada en el programa de usuario para ejecutar ese programa.

El tiempo es dividido en pequeños segmentos denominados *time slices* (que son variables en casi todos los sistemas). Cuando el *time slice* termina, el *dispatcher* le permite al *scheduler* actualizar el estado de cada proceso, entonces selecciona el siguiente proceso a ejecutar.

Context switching (conmutación de contexto) consiste en almacenar el estado de un proceso (cuando este sale del estado *running*), para poder luego reanudar dicho proceso (cuando vuelva al estado *running*). Luego de almacenar el estado (*contexto*), el scheduler le da CPU a otro proceso. El estado o contexto de un proceso consiste de la siguiente información: registros, mapas de memoria, cache de memoria, etc. El tiempo que dura el *context switch* es desperdiciado (*overhead*) ya que el sistema no puede aprovecharlo para hacer otra cosa.

2) Describa paso a paso el proceso de *conmutación de contexto*. Indique el rol de las interrupciones y en qué momentos el Sistema debe estar en modo protegido.

El proceso de *context switching* (conmutación de contexto) consta de los siguientes pasos:

1. Se almacena el contexto del procesador, incluyendo el contador de programa (*program counter*) y otros registros.
2. Se actualiza el *bloque de control del proceso* (PCB) que está actualmente en el estado *running*. Esto implica cambiar el estado del proceso a alguno de los otros estados (*ready*, *blocked*, etc.).
3. Se mueve el PCB de este proceso a la cola apropiada (*ready*, *blocked*, etc.).
4. Se selecciona otro proceso para ejecución (esto implica llamar al *scheduler*).
5. Se actualiza el PCB del proceso seleccionado. Esto implica cambiar el estado de este proceso a *running*.
6. Se actualizan estructuras de datos para administración de la memoria.
7. Se restaura el contexto del procesador que existía en el momento en que el proceso seleccionado fue sacado del estado *running* por última vez. Esto implica cargar los valores del contador de programa y otros registros.

Aclaración: los pasos anteriores fueron extraídos del libro de Stallings. Dicho autor llama al conjunto de pasos anteriores con el nombre de *process switching*, y lo diferencia del *context switching*. Sin embargo, Tanenbaum y otros autores no hacen diferencia alguna, y tanto *process switching* como *context switching* son la misma cosa.

Dentro del ciclo de instrucción (*instruction cycle*) se encuentra el ciclo de interrupción (*interrupt cycle*), en el cual el procesador chequea si ocurrió alguna interrupción. Si no hay interrupciones pendientes, el procesador carga la siguiente instrucción del proceso actual. Si una interrupción está pendiente, el procesador:

1. Guarda el contexto del programa que está siendo ejecutado.
2. Setéa el contador de programa (*program counter*) a la dirección donde comienza un programa de manejo de interrupciones (*interrupt handler*).

El procesador ahora carga la primer instrucción del programa de manejo de interrupciones, el cual atenderá la interrupción. Una interrupción podría estar seguida de un cambio del proceso en ejecución, aunque también podría suceder que, luego de la interrupción, vuelva a reanudarse el mismo proceso que venía ejecutándose (Stallings; pág. 129-130).

3) Indique qué información debe usar el Sistema Operativo para administrar procesos. ¿Cuáles de estos datos pueden estar en el área de usuario y cuáles en área del S.O.?

Cada proceso es representado en el sistema operativo por su propio *Process Control Block* (PCB), el cual es un registro que contiene la siguiente información:

- El estado del proceso (*process state*) podría ser *new*, *ready*, *running*, *idle*, o *halted*.
- El contador de programa (*program counter*) indica la dirección de la siguiente instrucción a ser ejecutada por este proceso.
- Los registros de la CPU varían en cantidad y tipo, dependiendo de la arquitectura. Incluyen acumuladores, registros índices, y registros de propósito general. Junto con el contador de programa, esta información de estado debe ser almacenada cuando ocurre una interrupción.
- Información de manejo de memoria.
- Información de estado de I/O.
- Información de *scheduling* de CPU, incluyendo la prioridad del proceso, punteros a las colas de *scheduling* y otros parámetros de *scheduling*.

En UNIX, un proceso debe contener la siguientes información:

Contexto a nivel de usuario
- Texto del proceso: instrucciones de máquina ejecutables del proceso.
- Datos del proceso: datos accesibles por el programa.
- Pila del usuario: parámetros, variables locales y punteros para funciones ejecutándose en modo usuario.
- Memoria compartida: memoria compartida por otros procesos.

Contexto de registro
- Contador de programa: dirección de la siguiente instrucción a ser ejecutada.
- Registro de estado del procesador.
- Puntero a la pila: apunta a la cima de la pila del kernel o del usuario.
- Registros de propósito general.

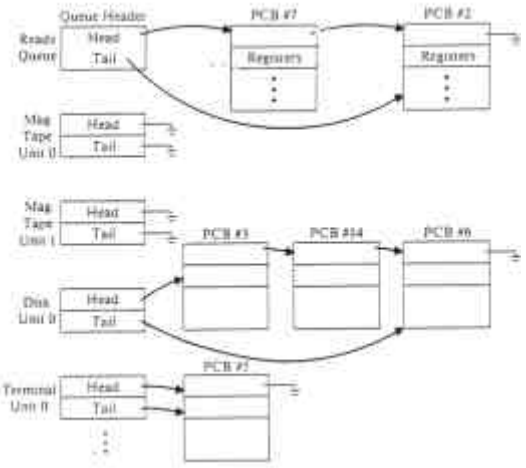
Contexto a nivel de sistema
- Entrada en la tabla de procesos: define el estado de un proceso.
- Área U: información de control del proceso.
- Tabla de región de preproceso: define el mapéo de direcciones virtuales a físicas.
- Pila del kernel.

(Stallings; pág. 143)

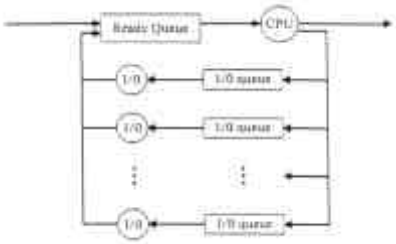
4) ¿Cómo encadena el S.O. los procesos para permitir su despacho?

Los procesos que están en estado *ready* y esperando a ser ejecutados son mantenidos en una lista llamada *ready queue*. El encabezado de la lista contendrá punteros al primer y último PCB en la lista. La *ready queue* podría ser implementada como una cola FIFO, una cola de prioridad, un árbol, una pila, o simplemente una lista desordenada. Sin embargo,

conceptualmente todos los procesos en la *ready queue* están listos y esperando por una chance para usar la CPU. La siguiente figura muestra una representación de una *ready queue* y varias *device queue*:



Si un proceso que se está ejecutando debe esperar que se complete una operación de I/O, entonces es colocado en otra cola, llamada *device queue*. Cada dispositivo tiene su propia *device queue*.



5) Explique paso a paso el proceso de *dispatching*.

El *dispatcher* le da control de la CPU al proceso seleccionado por el *scheduler*; esto implica las siguientes operaciones:

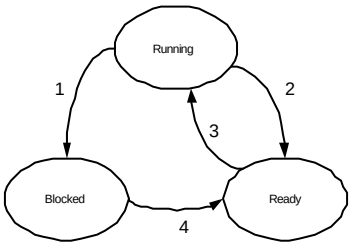
- Conmutar el contexto (*context switching*).
- Cambiar a modo usuario (*user mode*).
- Saltar a la posición apropiada en el programa de usuario para ejecutar ese programa.

6) ¿Cómo nace un proceso en un sistema operativo *batch* y en uno de *time sharing* como UNIX?

En un sistema *batch* los usuarios le presentan al sistema los trabajos a realizar (*jobs*). Cuando el sistema operativo decide que dispone de los recursos para ejecutar un trabajo, crea un nuevo proceso y ejecuta el trabajo (que se toma de la cola de trabajos). A veces los procesos se crean automáticamente al iniciarse el S.O., con el nombre de "particiones". Es importante recalcar que en un sistema *batch* perfectamente puede haber *multriprogramación* (es un error clásico relacionar *batch* con *monoprogramación*).

En un sistema *time sharing*, cada usuario de cada terminal solicita la ejecución de programas (tipeando un comando o *clickeando* un icono), con lo cual se ejecuta un nuevo proceso.
En líneas generales, un sistema *batch* no presenta interacción con el operador, mientras que un sistema *time sharing* sí la tiene.

7) Describa un diagrama de estados de procesos e indique cómo se operan las transiciones entre ellos.



1.- El proceso no puede seguir ejecutándose porque, por ejemplo, está esperando por datos de entrada que aún no fueron ingresados, y pasa al estado *blocked*.
2.- El sistema operativo decide dar CPU a otro proceso. Por lo tanto, el proceso que se está ejecutando en ese instante pasa al estado *ready* (esto significa que no se ejecuta, pero está listo para cuando el SO le vuelva a dar tiempo de CPU).
3.- Todos los procesos han tenido tiempo de CPU, y por lo tanto el *scheduler* le da tiempo de procesamiento al primer proceso nuevamente (de todos los que están en estado *ready*).
4.- Se produce un evento externo que el proceso bloqueado estaba esperando (por ejemplo, se dio entrada a la información que el proceso necesitaba para ejecutarse). Además, si no hay ningún proceso ejecutándose en ese instante, entonces se disparará automáticamente la transición 3.

8) ¿Qué objetivos de diseño puede tener el *scheduler*?

Los objetivos de diseño dependen de cada caso, como se muestra en la lista siguiente:

En todos los sistemas:
Equidad: dar a cada proceso equidad en cuanto al uso de la CPU (ningún proceso debería aplazarse en forma indefinida).
Cumplimiento de una política: controlar que la política establecida sea llevada a cabo.
Balance: mantener ocupadas todas las partes del sistema. Debe favorecerse a aquellos procesos que requieren recursos poco utilizados.

Sistemas batch:
Rendimiento: maximizar los trabajos (*jobs*) por hora.
Tiempo de retorno (turnaround time): minimizar el tiempo promedio entre que el trabajo comienza hasta que es completado.
Utilización de la CPU: mantener la CPU ocupada todo el tiempo.

Sistemas interactivos:
Tiempo de respuesta: responder rápidamente a los requerimientos (minimizar el tiempo entre que se ejecuta un comando y se obtiene un resultado).
Proporcionalidad: cumplir las expectativas de los usuarios. Está relacionado con la idea (no siempre correcta) que tienen los usuarios de que *"las cosas que parecen más complejas requieren más tiempo, y cosas que parecen más simples deben requerir poco tiempo"*. Así, el scheduler se debe ajustar a esta premisa.

Sistemas de tiempo real:
Cumplir con los plazos: evitar la pérdida de información.
Ser predecible: una tarea debe ejecutarse aproximadamente en el mismo tiempo y casi al mismo costo sea cual sea la carga del sistema.

9) Usando los simuladores, obtenga un diagrama de Gannt de a) *Shortest Job First* b) *First Come First Served* c) *Round Robin* d) *Round Robin con prioridades*.

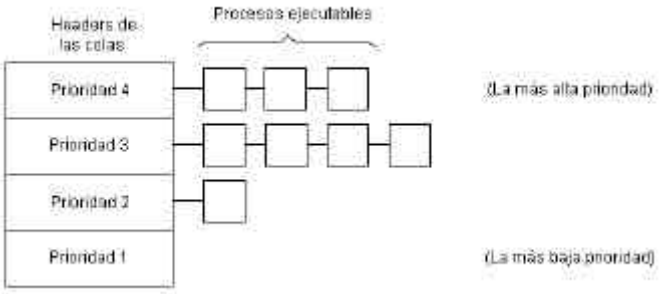
No desarrollada (implica el uso de simuladores, que deben bajarse de Internet).

10) Para el ejercicio 9), explique y calcule valores que se relacionen con los objetivos de 8) (Ej: tiempo medio de la tarea en el sistema, para el objetivo de maximizar tiempo de respuesta).

No desarrollada (implica el uso de simuladores, que deben bajarse de Internet).

11) Describa el funcionamiento de un *scheduler de colas múltiples* ¿Puede simularlo?

En el modelo de scheduler de *colas múltiples*, existen "clases de prioridad". Los procesos que están en la primer clase de prioridad son ejecutados por un *quantum.* Los que están en la segunda clase de prioridad se ejecutan por dos *quantums*, los que están en la tercer clase de prioridad se ejecutan por cuatro *quantums*, y así sucesivamente en potencias de dos. Cuando un proceso consume todos los *quantums* que le fueron cedidos, entonces se lo mueve a la clase siguiente. Además, a medida que el proceso se "hunde" más profundamente en la jerarquía de clases de prioridad, entonces será ejecutado menos frecuentemente, ahorrando CPU para procesos interactivos cortos (Modern Operating Systems; A. Tanenbaum; pág. 144).



12) ¿Qué son las *threads*? Describa sus variantes de implementación.

Un ***thread*** es similar a un proceso (en cuanto que comparten varias características). Cada uno contiene un *program counter*, registros, un *stack*, y se pueden ver como "entidades" que se ejecutan en la CPU. Un proceso puede estar compuesto por uno o más *threads* (en este último caso hablamos de *multithreading*). Los *threads* permiten ejecuciones múltiples dentro del mismo proceso, con un alto grado de independencia unos de otros. Dentro del mismo ambiente de proceso, los *threads* comparten el espacio de direcciones de memoria (*address space*), archivos abiertos, y otros recursos.
Los *threads* pueden implementarse en el espacio de usuario (*user space*) o en el *kernel*.

Implementación de *threads* en el espacio de usuario:
El paquete de threads se coloca completamente en el espacio de usuario (el kernel no sabe nada acerca de su existencia). La principal ventaja es que un paquete de threads de usuario pueden ejecutarse en sistemas operativos que no soportan threads. Cada proceso a nivel usuario cuenta con un sistema de tiempo de ejecución (*run-time system*), que maneja los threads, y por consiguiente cada proceso necesita su propia tabla de threads (*threads table*). El manejo que hace el sistema de tiempo de ejecución con los threads a nivel usuario, es similar al manejo hecho por el *scheduler* con los procesos a nivel kernel. El cambio de threads hecho a nivel usuario es algunos órdenes de magnitud más rápido que el hecho a nivel kernel, ya que entre otras cosas, no se requiere *conmutación de contexto* (*context switching*). Otra ventaja consiste en que cada proceso puede tener su propio algoritmo de *scheduling*.
Uno de los problemas que presenta esta opción es respecto al bloquéo de *system calls*. Dejar que los threads ejecuten los *system calls* es inaceptable, ya que podrían parar a todos los threads. La solución consiste en agregar código a los *system calls* que determinen si dicha llamada se bloqueará (en cuyo caso no se ejecuta). Otro problema tiene que ver con los *page faults*. Si el programa necesita una instrucción que no está en memoria, el sistema operativo deberá obtener dicha instrucción desde disco. El proceso se bloquea completamente hasta que termine la operación de I/O, aunque haya algún thread en estado *ready* (ya que el kernel no conoce sobre la existencia de los threads). Otro problema consiste en que un thread que está ejecutándose podría no ceder nunca la CPU a otro thread que no lo está, ya que no existe ningún reloj que se encargue de esta tarea. Es el propio thread quien debe entrar al sistema de tiempo de ejecución (*run-time system*). El problema tal vez más importante es que los programadores generalmente necesitan threads en aplicaciones donde los estos se bloquéan muy frecuentemente (por ejemplo, en un servidor web con múltiples threads).

Implementación de *threads* en el *kernel*:
En este caso no se necesita el sistema de tiempo de ejecución (*run-time system*) ni la tabla de threads en cada proceso. El kernel cuenta con su propia tabla de threads, en donde mantiene la información de todos los threads del sistema. Cuando un *system call* bloquéa un thread, el kernel puede ejecutar otro thread del mismo proceso o de otro proceso. Debido al alto costo que presenta la creación y destrucción de threads a nivel de kernel, es posible que estos se reciclen. Cuando un thread es destruido, se lo marca como "no ejecutable", pero todas sus estructuras son mantenidas. Luego, cuando debe ser creado un nuevo thread, se reactiva uno de estos threads, reduciendo el costo. Por otra parte, si se produce un *page fault*, el kernel puede cambiar la ejecución a otro thread, mientras el primero espera que se complete la operación de I/O.
La principal desventaja de esta implementación es que el costo de los *system calls* es elevado.

Implementación híbrida de *threads*:
En este caso, el kernel solo conoce y maneja los threads a su nivel, algunos de los cuales podrían tener múltiples threads a nivel de usuario. Estos threads a nivel de usuario son manejados de la misma forma que en la implementación de threads en el espacio de usuario.
(Modern Operating Systems; A. Tanenbaum; pág. 90-94)

13) ¿Qué campos de los mencionados en 3 pasan al *Thread Control Block*?

En el *Thread Control Block* (TCB) encontramos la siguiente información:

- Estado de ejecución: registros de la CPU, contador de programa (*program counter*), puntero al *stack*.
- Información de *scheduling*: estado (*new*, *ready*, *running*, *waiting*, o *terminated*), prioridad, tiempo de CPU usado.
- Varios punteros (para implementar colas de *scheduling*).

14) ¿Qué opciones hay para la interacción del *scheduler* con las *threads*?

Cuando varios procesos tienen múltiples *threads* cada uno, tenemos dos niveles de paralelismo: procesos y *threads*. El *scheduling* en ambos sistemas difiere sustancialmente, dependiendo si son soportados los *threads* a nivel de usuario o los *threads* a nivel de kernel (o ambos).
En el primer caso (*threads* a nivel de usuario), el kernel no tiene conocimiento de la existencia de los *threads*, y por tanto toma cada proceso y lo ejecuta en un *quantum* (intervalo de tiempo que se le asigna a un proceso para que se ejecute); por ejemplo, ejecuta el proceso *A*. El *scheduler* de *threads* dentro del proceso *A* decide que *thread* ejecutar, por ejemplo *A1*. Este *thread* se ejecutará tanto como él quiera (ya que los *threads* no son interrumpidos), hasta que se consuma el *quantum*, y el kernel decida tomar otro proceso para ejecutar. Cuando el proceso *A* vuelva a ser ejecutado, el *thread A1* continuará hasta volver a consumir todo el *quantum* del proceso, o hasta que termine su trabajo. Se debe notar que el *scheduler* del kernel no sabe (ni tampoco le interesa) qué está ocurriendo dentro del proceso *A*.
En el segundo caso (*threads* a nivel de kernel), el kernel toma un *thread* y lo ejecuta. No tiene en cuenta a qué proceso pertenece dicho *thread*. Al *thread* se le da un *quantum*, y es forzado a suspenderse en caso de exeder dicho intervalo de tiempo. El kernel sabe que cambiar entre dos *threads* pertenecientes a distintos procesos es más costoso que si los *threads* pertenecen al mismo proceso (porque se requiere de un *context switch* completo). Por lo tanto, puede tener en cuenta esta información para decidir qué *thread* ejecutar en un determinado momento.

15) ¿Cómo se inserta el Modelo de Objetos en esta evolución?

En esta evolución, los objetos pueden verse como unidades de agrupamiento. En un caso concreto (Windows NT), tanto los procesos como los *threads* son objetos. Cada proceso se define por una cierta cantidad de atributos y encapsula una cierta cantidad de acciones (o servicios). Un proceso en Windows NT debe contener al menos un *thread* para ejecutar. Ese *thread* podría ejecutar otros *threads*. Algunos de los atributos del *thread* son derivados del proceso.

Sin embargo, los objetos de Windows NT son solamente unidades de encapsulamiento, y no tienen nada que ver con los objetos de OOP. No hay herencia ni polimorfismo ni *frameworks* de integración (conjunto de clases cooperantes). En un ambiente orientado a objetos "nativo" (no hay comerciales, pero la *Java Virtual Machine* provee parte de dicho ambiente)

cada objeto "puede" poseer al menos un *thread* propio. Si no lo hace es por razones de performance. En este ambiente, los procesos se "diluyen" ya que cada objeto tiene "vida propia".

16) Ensaye una justificación tecnológica del uso de multiprogramación y multithreading.

La *multiprogramación* es una manera de ejecutar procesos en un sistema, por la cual cada proceso hace uso de la CPU por un determinado intervalo de tiempo. Si el sistema operativo tiene la habilidad de intercambiar rápidamente la ejecución de los procesos, entonces dará la sensación de ejecución simultánea o paralela de los mismos. Algunas ventajas y desventajas de esta técnica son:

Ventajas
- Puede mantener la CPU ocupada.
- Puede superponer la ejecución de tareas, incrementando el rendimiento.

Desventajas
- Aumenta la complejidad del sistema.
- Es potencialmente propensa a sufrir daños en la seguridad.

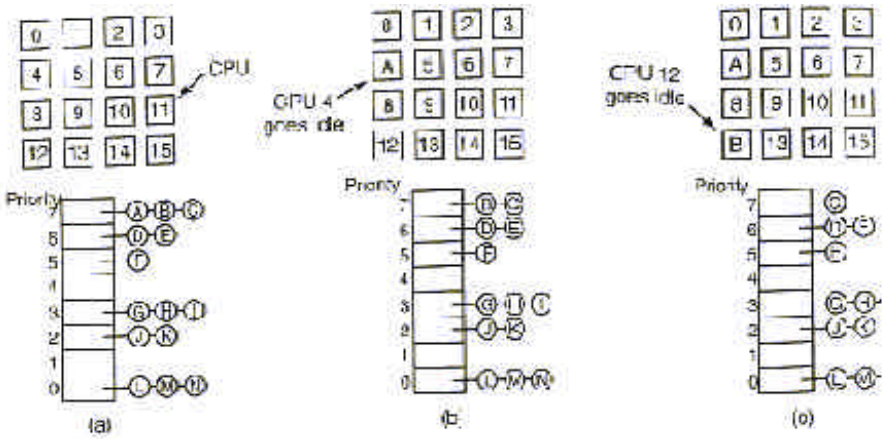
En una tarea *multithreading*, mientras un *thread* servidor está bloqueado y esperando, un segundo *thread* en la misma tarea puede ejecutarse. La cooperación de múltiples *threads* en el mismo trabajo brinda mayor rendimiento y mejora la performance. Además, el uso de *multithreading* permite separar las tareas minimizando los puntos de contacto ("cohesión" y "acople" dirían los seguidores de la programación estructurada). Por ejemplo, se maneja la *GUI* por un *thread* y el proceso por otro *thread*, sin "contaminar" la programación.

Aclaración: en esta pregunta, se pedía que se "ensaye" una justificación tecnológica del uso de *multiprogramación* y *multithreading*. Lo anterior son conceptos, con lo cual, el ensayo queda a criterio de cada alumno. Sin embargo, el Ing. Osvaldo Clúa me pidió que resaltara lo que está subrayado.

17) ¿Cómo impacta el multiprocesamiento en las operaciones de *scheduling*, *dispatching* y *conmutación de contexto*?

En multriprocesamiento, el *scheduling* tiene dos dimensiones: debe decidir qué proceso ejecutar y en qué CPU ejecutar dicho proceso. Veamos algunos casos:

Timesharing: en este caso, los procesos a ejecutar no mantienen ninguna relación entre ellos. El sistema mantiene un conjunto de listas de acuerdo a prioridades (es una lista de listas) con los procesos en estado *ready* (esta estructura es compartida por todas las CPU). Así, cada procesador que se vaya desocupando, tomará el proceso con mayor prioridad en ese conjunto de listas. Hacer el *scheduling* de esta forma es una opción razonable.



Space Sharing: en este caso los procesos están relacionados de alguna forma. Hablando a nivel de *threads*, todos aquellos que pertenecen a un mismo proceso están relacionados. El *scheduling* de múltiples *threads* al mismo tiempo a través de varias CPU es llamado *space sharing*. Cuando un conjunto de *threads* debe ser ejecutado, el *scheduler* chequéa si existen tantas CPU libres como *threads* a ejecutar. En caso afirmativo, a cada *thread* se le asigna su propia CPU (notar que no hay *multiprogramación* en cada CPU), y todos comienzan a ejecutarse. Si no hay suficientes CPU libres, entonces ningún *thread* comienza a ejecutarse. Cuando un *thread* termina de ejecutarse, la CPU es colocada en un pool de CPU disponibles.

Gang Scheduling: una clara ventaja de *Space Sharing* consiste en la eliminación de la *multiprogramación* (porque en una CPU se ejecuta un único proceso o thread), con lo cual se evita la carga provocada por el proceso de *conmutación de contexto*. Sin embargo, de igual forma, una clara desventaja de dicha técnica es el tiempo perdido cuando una CPU se bloquea y no tiene nada para hacer hasta que vuelva al estado *ready*. Así, se ha llegado a esta alternativa que junta las ventajas de *Timesharing* y *Space Sharing*, y que consta de tres partes:

- Los grupos de *threads* relacionados son tratados por el *scheduler* como una unidad (*gang*).
- Todos los miembros de un *gang* se ejecutan simultáneamente (en diferentes CPU cada miembro).
- Todos los miembros de un *gang* comienzan y terminan juntos sus intervalos de tiempo.

La idea de *gang scheduling* es tener todos los *threads* de un proceso ejecutándose juntos, así si uno de ellos envía un pedido a otro *thread*, este tendrá el mensaje inmediatamente y responderá inmediatamente. Además, el tiempo es discretizado en *quantums*, de manera tal que si un *thread* se bloquéa, su CPU estará en desuso hasta que termine el *quantum.*

18) ¿Cómo es la técnica de programación *pseudoconversacional*, *transaccional* o usando *autómatas finitos* (son distintos nombres de la misma técnica)?

Una *transacción conversacional* es aquella que implica más de una entrada desde la terminal, de manera tal que la transacción y el usuario establecen una especie de "conversación". Una *transacción no-conversacional* tiene solo una entrada (la que invoca a la transacción). Veamos un ejemplo de *transacción conversacional*:

Programa: Pedir DNI, esperar respuesta.
Usuario: Ingresar DNI.
Programa: Pedir password, esperar respuesta.
Usuario: Ingresar password.
Programa: Enviar menú, esperar respuesta.
Usuario: Enviar ítem de menú.
Programa: Enviar resultados, terminar.

El programa ha "saltado" de estado a estado. Estuvo ocupando recursos (sobre todo espacio) durante toda la "conversación", y si el usuario desaparece, el programa queda "colgado".

La *programación pseudoconversacional* se basa en el hecho de que, en una *transacción conversacional*, la cantidad de tiempo gastado procesando cada respuesta al usuario es extremadamente corto comparado con la cantidad de tiempo esperando la entrada del usuario. Una secuencia en una *transacción pseudoconversacional* contiene una serie de *transacciones no-conversacionales* que ven al usuario como una simple *transacción conversacional* con varias pantallas de entrada. Cada transacción en la secuencia maneja una entrada, responde, y termina.
Antes de que una *transacción pseudoconversacional* termine puede pasar datos a la siguiente transacción, iniciada desde la misma terminal. Veamos un ejemplo de *transacción pseudoconversacional*:

Programa 1: Pide DNI, y termina.
Usuario: Ingresa DNI.
Programa 2: Recibe DNI, pide password, envía DNI (que el usuario puede o no ver), y termina.
Usuario: Ingresa password.
Programa 3: Recibe DNI y password, envía menú, DNI y password (posiblemente oculto, hay problemas de seguridad pero es un ejemplo) y termina.
Usuario: Envía ítem de menú.
Programa 4: Recibe DNI, password e ítem de menú, verifica y envía resultados, y termina.

El programa se abrió en 4 programas "invertidos" o "Data Driven". En cada interacción se llama a un programa distinto. Además hay transferencia de datos ocultos al usuario (el tema de seguridad se trata de otra forma, no viajan varias veces los pares usuario-password, sino un identificador de corta vida llamado *ticket*).
Un sistema así admite más cantidad de usuarios simultáneos, y los problemas de espacio y de “cuelgues” se desvanecieron.

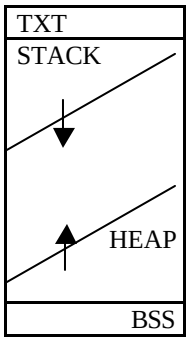
LINK EDICION

1) ¿Qué es el binding? En qué momentos del ciclo de desarrollo de un programa ocurre?

Binding: Es ligar las direcciones simbólicas de un programa para referenciar direcciones a memoria y las propias direcciones de memoria que la máquina interpreta inmediatamente. Puede darse en tiempo de compilación, en tiempo de carga, en tiempo de ejecución (donde la ubicación puede cambiar). Durante la ejecución hay un record de activación que retoca los valores en TXT (área de etiquetas) para poder redirigir el área de datos.

Binding en compilación: vincula las variables estáticas (área BSS).
Binding en ejecución:

- Automático: se retoca el área TXT (llamadas recursivas). Eso es el stack. El SO asigna la dirección inicial del stack para el programa.
- Controlado: la memoria dinámica. El SO debe proveer memoria para que crezca el espacio de variables controladas (heap).



Binding estático: se enlazan las bibliotecas externas precompiladas.
Binding dinámico: enlaza las cosas al vuelo, pero sólo si se usan (especie de dll). El linker pone una llamada a un programita que carga la biblioteca dinámica. Es bueno porque ahorra memoria (pero puede ser una puerta para hackers =). Se da en tiempo de carga y en tiempo de ejecución.

2) ¿Qué tipo de información debe guardarse en los módulos objeto? Describa el uso de cada tipo.

Módulos Objeto:

La estructura de un módulo objeto producido por un traductor es:

- *Header information:* información general del archivo, como el tamaño del código, nombre del código fuente, fecha de creación.
- *Object code:* instrucciones y datos generados por el compilador o ensamblador.
- *Relocation:* lista de lugares en el código objeto que deben ser fijados cuando el linker cambia las direcciones del código objeto.
- *Symbols:* símbolos globales definidos en este modulo, símbolos que se importan de otros módulos o definidos por el linker.
- *Debugging information:* información adicional utilizada por el debugger. Esto incluye archivo fuente y número de línea de la información, símbolos locales, descripción de las estructuras de datos usadas por el código objeto, como las definiciones de estructuras en C.

3) ¿Qué soluciones se adoptan para guardar esa información en los distintos formatos de archivos objeto y ejecutables como coff, elf y pe?

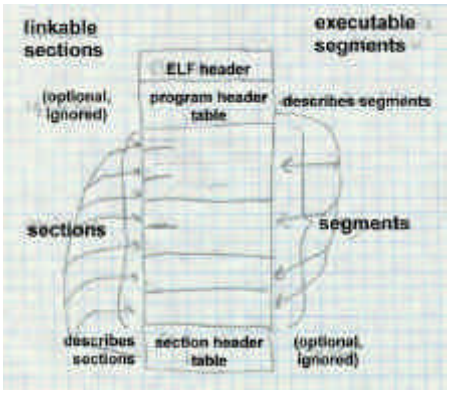
Objetos ejecutables:

ELF (Executable and Link Format): Se usa mucho en Linux, en los BSD actuales.

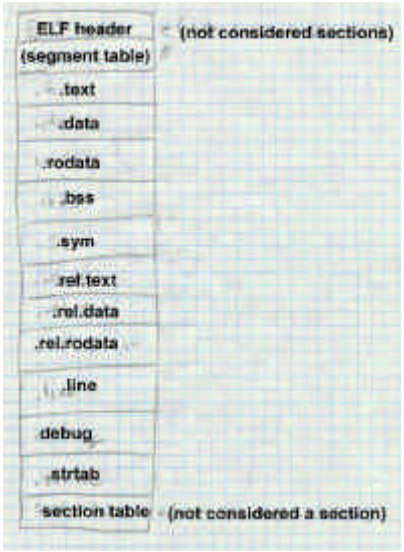
Es un formato ejecutable eficiente para memoria virtual con dynamic linking, y hace fácil el mapeo de páginas ejecutables directamente en el espacio de direcciones del programa. Además permite cross-compilation y cross-linking de una plataforma a otra, con la información suficiente en cada archivo ELF para identificar la arquitectura.

El formato ELF se usa para archivos relocatables, ejecutables y objetos compartidos. Archivos relocatables son creados por compiladores y ensambladores pero necesita ser procesado por un linker antes de correr. Archivos ejecutables tienen toda la relocation hecha y los símbolos resueltos excepto, tal vez, algunos símbolos de libraries compartidas que son resueltos en runtime. Los objetos compartidos son librerías compartidas, que contienen información de los símbolos para el linker y código directo para runtime.

Compilers, assemblers, and linkers toman al archivo como un set de secciones lógicas descritas por a section header table; mientras que el system loader lo toma como un set de segmentos descritos por a program header table. Un segmento puede contener varias secciones. Por ejemplo, un segmento de ``loadable read-only'' puede contener secciones de código ejecutable, RO data y símbolos para el dynamic linker. Archivo relocatable tiene section tables, archivos ejecutables tienen program header tables, y los objetos compartidos tienen ambos.



Un archivo relocatable o de objetos compartidos es considerado una colección de secciones definidas en section headers. Cada sección contiene un solo tipo de información como código del programa, RO o RW data, relocation entries, o símbolos. Cada símbolo definido en el módulo es definido relativo a la sección, con lo cual la entrada a un procedimiento va a ser relativo a la sección del código de programa que contiene el código de ese procedimiento.



Un archivo ejecutable tiene el mismo formato que el relocatable, pero los datos están de tal forma que el archivo pueda ser mapeado en memoria y corrido. El archivo contiene un header del programa que sigue al header ELF en el archivo. Ese header del programa define los segmentos que deben ser mapeados.
Objetos compartidos: tiene el program header table en el comienzo, seguido por las secciones en los loadable segments, incluyendo la información para dynamic linking. Las secciones siguientes que abarcan los loadable segments es la tabla de símbolos relocatables y otra información que necesita el linker para crear programas ejecutables que refieren a los objetos compartidos, con la tabla de secciones al final.

PE (Portable executable): Es un formato que se usa en Windows junto con Coff.
Puede contener un directorio de recursos (cursors, icons, bitmaps, menus, and fonts) para todos aquellos recursos que el código usa en ese archivo.
Los archivos ejecutables PE están pensados para la paginación. Las páginas son mapeadas directamente a memoria y corridas. Pueden ser archivos EXE o DLL (dynamic link libraries). Cada uno contiene una lista de funciones que exporta y datos que pueden ser usados por otros archivos PE que estén cargados en el mismo espacio de direcciones; y una lista de las funciones que importa y datos que necesita resolver en tiempo de carga. Cada archivo está formado por un set de divisiones análogos a los segmentos de archivos ELF, que son llamados indistintamente secciones, segmentos u objetos.
Secciones Especiales en archivos PE.

- *Exports*: lista de símbolos definidos en este modulo y que pueden ser vistos por otros. Los archivos EXE típicamente no exportan símbolos, en cambio las DLLs exportan los símbolos para las rutinas y datos que proveen.
- *Imports*: tabla con todos los símbolos de DLLs que necesita resolver en tiempo de carga. El linker predetermina qué símbolos va a encontrar en qué DLLs, con lo cual la tabla comienza con un directorio de entradas a cada DLL referenciada. Cada entrada contiene el nombre de la DLL, identificación del símbolo requerido y el lugar donde va a ser guardado el valor.
- *Resources*: tabla de recursos, organizada como árbol.
- *Thread Local Storage*: Windows soporta múltiples threads de ejecución por proceso. Cada thread puede tener su propio lugar de almacenamiento, Thread Local Storage or TLS. Esta sección apunta a una sección del image usado para inicializar el TLS cuando el thread empieza, y también contiene punteros a rutinas de inicialización para llamarlas cuando cada thread empieza. Generalmente está presente en los EXE pero no en archivo DLL, porque Windows no aloca TLS storage cuando un programa linkea dinámicamente una DLL.
- *Fixups*: If the executable is moved, it is moved as a unit so all fixups have the same value, the difference between the actual load address and the target address. The fixup table, if present, contains an array of fixup blocks, each containing the fixups for one 4K page of the mapped executable. (Executables with no fixup table can only be loaded at the linked target address.)

COFF (Common Obj. File Format): Es un formato que en Unix se usó entre el a.out y elf; y que se usa en Windows.
Un archivo relocatable tiene el mismo header que los archivos PE, pero su estructura es más similar a los archivos ELF.
Cada sección de datos o código lleva información de reubicación y numero de línea. Objetos COFF tienen section-relative relocations, como los archivos ELF, y una tabla de símbolos con aquellos que necesita.
Los archivos COFF provenientes de compiladores de lenguaje no contienen, típicamente, ningún recurso. Por lo general están en un archivo objeto separado creado por un compilador de recursos.
Tiene algunas secciones que no usan los archivos PE. La mas notable es .drective que contiene los comandos de texto para el linker. Los compiladores usan esta sección para informar al linker de buscar las librerías propias del lenguaje. Algunos compiladores que incluyen MSVC también tienen directivas para el linker para exportar código y símbolos al crear una DLL.

Microsoft PE and COFF file
DOS header (PE only)
DOS program stub (PE only)
PE signature (PE only)
COFF header: describe el contenido del archivo (más importante: número de entradas en la tabla).
Optional header (PE only) Contiene punteros a las secciones más usadas.
Section table
Mappable sections (pointed to from section table)
COFF line numbers, symbols, debug info (optional in PE File)

4) ¿Pueden dos sistemas operativos distintos para la misma arquitectura y con el mismo file format de archivo ejecutable ejecutar los mismos programas ejecutables?

No, porque pueden tener diferentes llamadas al sistema.

5) Defina el concepto y el uso de un símbolo relocatable. Dé algunos ejemplos.

Un símbolo es ***relocatable*** cuando la dirección absoluta todavía no está especificada, sino que tiene una dirección que es relativa a una dirección base.
Ejemplo: Una función en un módulo A comienza en la dirección 20. Supongamos que se linkearon otros módulos encima de A tal que la dirección de inicio de A es ahora 50. Entonces la dirección de la función ahora es 70, lo cual implica que todas las referencias a esa función deben ser actualizadas (con el base que es 50).

6) ¿Qué es una referencia externa?

Es una llamada a un símbolo definido en otro módulo objeto, que no coincide con el que realiza la llamada. Como los módulos se compilan por separado, no se puede saber hasta el momento del enlace cuál es la dirección de ese símbolo.

7) ¿Qué es una library, qué símbolos exporta y cuáles importa?

Una library es fundamentalmente una colección de archivos objeto, usualmente con información adicional para hacer más rápida la búsqueda. En otras palabras, es un módulo que contiene subrutinas. Los símbolos que exporta son los puntos de entrada a las subrutinas. Los símbolos que importa son los de las otras libraries que utiliza.

8) ¿Qué es dynamic linking, cómo se diferencian los mecanismos para hacerlo en tiempo de carga y en tiempo de ejecución?

Es la técnica por la cual se enlazan los procedimientos en el momento de ser referenciados (en que se llama) por primera vez. En tiempo de carga, se conoce la dirección del procedimiento porque se está cargando el programa y la biblioteca. En tiempo de ejecución, se pone una dirección inválida en el segmento del enlace y cuando se referencia por primera vez al procedimiento, se carga y se cambia la dirección inválida por la correcta.

9) ¿Cómo se comparten las bibliotecas dll?

El linker busca en las librerías los módulos que resuelven los símbolos externos. Pero en vez de copiarlos, anota en qué library está el módulo, y escribe una lista de libraries en el ejecutable. Cuando se carga el programa, el startup code encuentra esas libraries y las mapea en el espacio de direcciones del programa antes del comienzo del programa.

10) ¿Por qué es necesario y cómo puede implementarse un control de versiones en bibliotecas compartidas?

Es necesario para que cada programa pueda utilizar la versión para la cual fue preparada. Los cambios menores que mantienen compatibilidad conservan el número de versión, pero los cambios mayores, no. Una manera de implementarlo sería proveer un stamp de biblioteca en el archivo ejecutable, entonces se comparan los stamps de los archivos y de las bibliotecas y se elige la correcta.

11) ¿Cómo se cargan las clases en Java?

El runtime environment de java carga, en tiempo de ejecución, las clases que se necesiten. Este componente llamado 'Dinamic class loader' se ocupa de buscar el método solicitado por su nombre con el path de clases del file system.

12) ¿Cómo se mantiene la relación de herencia en memoria para poder hacer un binding en un Lenguaje Orientado a Objetos?

Las relaciones de herencia en los lenguajes orientados a objetos requieren de un binding bidireccional porque un método de una librería puede enviar mensajes a otro del programa que la utiliza. Cuando se invoca un método virtual se exige 'Dynamic Binding', que consiste en hacer búsquedas en la TMV de la clase que invocó al método.

13) Usando la respuesta anterior, explique cómo se realiza un despacho dinámico.

Es el mecanismo que realiza el 'Dynamic Binding'. Este difiere de un lenguaje a otro. En general, son optimizaciones del método virtual, en caso de no ser exitosa la búsqueda en la TMV de la clase que invocó al método, se itera a través de la TMV de los padres hasta encontrarlos.

ADMINISTRACION DE MEMORIA

1) Cómo era el esquema de administración de memoria en particiones fijas? ¿Por qué aparece la protección de memoria?

Monoprogramación: Un solo programa corriendo a la vez compartiendo la memoria con el S.O., de la siguiente forma:

- Usuario tipéa un comando.
- S.O. copia el programa solicitado de disco a memoria y lo ejecuta.
- S.O. imprime el promt y espera otro comando.
- Cuando el usuario ingresa un nuevo comando, el S.O. levanta el nuevo programa a memoria sobrescribiendo el anterior.

Multiprogramación con particiones fijas compartidas: Se divide la memoria en *n* (no iguales) particiones. Estas particiones pueden crearse durante la inicialización del sistema. Cuando un trabajo arriba, éste es puesto en una cola, esperando que una partición que pueda contenerlo se libere. En ese momento el trabajo es levantado a memoria y ejecutado hasta finalizar. Este modelo puede realizarse utilizando una cola de entrada por partición (acolando según el tamaño) o una única cola de entrada para todas. La primera tiene como desventaja que si una cola es muy larga, varios trabajos pueden estar esperando por ser levantados en memoria teniendo gran parte de la misma en desuso. En el segundo caso, es necesario aplicar un criterio para no desfavorecer a los trabajos pequeños y a la vez no desperdiciar particiones grandes en trabajos chicos.

Dado que los programas comparten la memoria, cualquier programa malicioso podría construir una instrucción que le permita saltar a una dirección de memoria fuera de su partición. Algunas posibles soluciones son:

- Clave de protección (en la PSW de IBM360).
- Uso de dos registros especiales, registro base y límite.

2) ¿Cuál es el principio de Swapping?

Consiste en levantar enteramente un proceso a memoria, correrlo durante un determinado tiempo y luego bajarlo nuevamente a disco. Permite variar la cantidad de particiones, su ubicación y tamaño dinámicamente. Cuando un proceso es copiado en memoria, es necesario realocar todas las referencias a memoria. Swapping crea agujeros en la memoria cuando un proceso termina ó crece y debe ser realocado (crece el área de datos o el stack). Estos agujeros pueden ser combinados entre sí para crear otros más grandes. El proceso se denomina compactación y no es usado en la práctica por el alto costo de CPU.

3) Describa algunos métodos de administrar la memoria usando particiones variables.

Bitmaps: Memoria dividida en unidades de alocación, para cada una corresponde un bit en el Bitmap. 0 indica que está libre, 1 ocupado (o viceversa). El tamaño de la unidad es importante. Si es grande se desperdicia espacio en cada unidad. Por el contrario si la unidad es pequeña, el bitmap crece y las búsquedas resultan muy lentas.

Listas enlazadas: Lista doblemente enlazada donde cada entrada corresponde a un segmento, el cual puede ser un proceso o un agujero. Se mantiene ordenada por dirección de memoria (costo de mantenimiento). Algoritmos para alocar un nuevo proceso:

- *First Fit:* El memory manager recorre la lista hasta que encuentra un segmento en el que cabe el proceso. Parte el segmento en dos, ubicando el proceso en uno y dejando un agujero en el otro (en caso que el segmento tenga un tamaño mayor del que el proceso necesita). Este método realiza la búsqueda más corta posible.
- *Next Fit:* Idem first fit, salvo que mantiene un registro de donde terminó la búsqueda anterior, en la siguiente búsqueda comienza desde allí. Tiene peores resultados.

- *Best Fit:* Busca el menor agujero donde quepa el proceso. Es más lento, ya que recorre toda la lista y produce un mayor gasto de memoria, ya que llena la misma de pequeños agujeros donde no cabe ningún proceso.
- *Worst Fit:* El inverso del anterior. No produce buenos resultados..
- *Quick Fit:* Separa en listas, agujeros de algunos de los tamaños mas frecuentemente usados. Es muy rápido pero desperdicia memoria como el best fit.

4) ¿Cómo funciona la Memoria Virtual?.

La idea se basa en que el espacio ocupado por un programa, los datos que maneja y su stack pueden superar el tamaño de la memoria disponible. Por lo tanto el S.O. mantiene en memoria aquellas partes del programa que están siendo actualmente usadas y el resto las guarda en disco. La mayoría de los sistemas con memoria virtual usan una técnica llamada *paging:* existe un set de direcciones de memoria que los programas pueden generar (espacio de direcciones virtuales) usando índices, registro base y otras técnicas. Dichas direcciones son mapeadas a direcciones físicas en la memoria. Desde luego el espacio de direcciones virtuales es significativamente mayor que espacio de direcciones de la memoria. Ambos se dividen en unidades, de igual tamaño, llamadas virtual pages y page frames respectivamente. Para realizar el mapeo entre virtual pages y page frames se utiliza una tabla de páginas o page table. La dirección virtual está formada por el número de página virtual (índice de entrada a la tabla de páginas) y por un offset. El mapeo se lleva a cabo de la siguiente forma:

- Con los primeros *n* bits de la dirección virtual (virtual page) se accede a la correspondiente entrada en la page table.
- Dada la entrada, se verifica si el page frame correspondiente a dicha virtual page se encuentra en memoria (bit presente/ausente).
- De ser así, la dirección física en memoria se obtiene concatenando el número de page frame y el offset.
  En caso de no encontrarse en memoria el page frame (page fault), es necesario leerlo de disco. Previamente debe elegirse otro page frame (que no haya sido usado recientemente), grabarlo en disco (si fue modificado), para escribir el nuevo page frame en ese lugar de memoria.

Lo importante a recalcar es que la memoria virtual (por como está diseñada) hace una separación entre direccionamiento y almacenamiento.

5) Describa el hardware asociado al manejo de las direcciones en memoria virtual.

MMU: Memory Management Unit. Componente del hardware que se encarga de realizar el mapeo de las direcciones virtuales a direcciones físicas en la memoria, verificando si el page frame esta mapeado en memoria y llamando al S.O. en caso de que ocurra un page fault. La MMU puede encontrarse en el CPU o cerca del mismo.
La respuesta 6 puede ser parte de ésta respuesta ya que la TLB es otro mecanismo, o es parte del hardware asociado al manejo de las direcciones en memoria virtual.

6) ¿Para que se usa la memoria asociativa (TLB) en un esquema de memoria virtual?

Se usa para evitar el overhead que se produce por estar obligado a realizar múltiples referencias a memoria por cada dirección virtual de un programa. La solución parte del siguiente fenómeno observado: la mayoría de los programas poseen una gran cantidad de referencias a una pequeña cantidad de páginas. Hay solo un reducido número de páginas que son intensamente leídas. La solución consiste en un pequeño dispositivo de hardware que mapee direcciones virtuales en direcciones físicas en memoria sin utilizar la page table. Translation Lookaside Buffer (TLB). Generalmente se encuentra dentro de la MMU y consiste en un pequeño número de entradas conteniendo la misma información que una entrada en la page table. Como funciona:

- Cuando una dirección virtual se presenta en la MMU, el hardware chequea si el virtual page number se encuentra en la TLB, comparando todas las entradas simultáneamente.
- Si es así, y el acceso no viola los bits de protección, el page frame es extraído directamente de la TLB sin usar la page table.

- Si la página está presente pero hay una violación de permisos, se produce un protección fault tal como hubiera ocurrido con la page table.
- Si la página no se encuentra, la MMU hace una búsqueda ordinaria en la page table, luego sobrescribe alguna entrada de la TLB con la nueva página solicitada.

7) Describa algunos algoritmos de paginación.

Optimal: Imposible de implementar. Numera cada página teniendo en cuenta cuantas instrucciones faltan ejecutarse para que dicha página sea necesaria (no hay forma de saberlo). Al producirse el page fault, remueve de memoria la página cuyo uso es menos próximo. Sirve solo como guía para evaluar otros métodos.

Not Recently Used: Cada página en la page table posee dos bits: R, indicando si la página está siendo referenciada y M cuando la página está siendo modificada. Cuando se inicializa el sistema ambos bits están en cero en todas las entradas de la tabla. En cada interrupción del reloj el bit R es borrado para diferenciar las páginas que fueron recientemente utilizadas. Cuando ocurre un page fault, el sistema inspecciona las entradas clasificándolas según cuatro clases (posibles combinaciones de bits R y M, Clase1: 00, Clase2: 01, Clase3: 10 y Clase4: 11). El algoritmo remueve una página, en forma random, de la clase con menor número.

FIFO: El sistema mantiene una cola con referencias a todas las páginas que actualmente están en memoria. Al producirse un page fault se remueve la página mas vieja (cabeza) y la nueva pagina (página solicitada) se atacha al final (cola). Este método no es aplicable ya que no tiene en cuenta el uso reciente de las páginas, sino el tiempo que llevan en memoria, removiendo páginas que están siendo intensamente usadas.

Second Chance: Idem que FIFO, salvo que inspecciona el bit R, en caso de estar en 1, la página no es removida, sino que es retrasada en la lista hasta el final de la misma. Por lo tanto la búsqueda en la lista continúa hasta encontrar una página no referenciada. De no haber ninguna, se remueve la que inicialmente estaba el comienzo (FIFO).

Clock Page: Mejora para Second Chance: "se usa una lista circular". Cuando ocurre un page fault, se inspecciona el bit R de la página que está siendo apuntada. Si él es 0, se remueve la página, se ubica la nueva en esa posición y se avanza el puntero a la siguiente página. Si es 1, se avanza el puntero y se inspecciona la siguiente página hasta encontrar una que no esté siendo referenciada.

Least Recently Used: Se basa en mantener una lista enlazada ordenada por el tiempo que transcurrió desde la última vez que una página fue usada. Los resultados del método son muy cercanos al óptimo, sin embargo el mantenimiento de la lista (en cada referencia) es demasiado costoso. Existen algunas posibles implementaciones usando hardware que eliminan el problema.

Working Set: (ver 9).
Conceptos:

- Paginación en demanda: una pagina es alocada en memoria sólo cuando se produce un page fault.
- Pre-paginación: las páginas son alocadas entes de que las mismas sean solicitadas evitando un page fault.

El modelo funciona de la siguiente manera: el sistema mantiene registro del working set de un proceso y se asegura de que el mismo esté completamente en memoria antes de dejar ejecutarse el proceso. Basados en el fenómeno antes mencionado de que un proceso tiene una gran cantidad de referencias a un pequeño set de direcciones, puede decirse que a medida que el proceso de ejecución alanza el working set tiende a mantenerse invariante. Por esto último se consigue reducir la cantidad de page faults. Puede producir *trashing* (ver 10).

WSClock: Idem anterior, además maneja una lista circular para mejorar el rendimiento del método a la hora de buscar una página para remover.

8) Describa alguna forma de mantener el tamaño de las page tables en un valor manejable.

Dado que el espacio de direcciones virtuales puede ser muy grande la page table también resulta serlo. En un modelo en el que la tabla, en su totalidad, se encuentra en memoria, esto se traduce en un gasto inmenso de recursos. Como solución puede implementarse el uso de Multilevel Pages. Consiste en una tabla con varios niveles de indirección (ver gráfico pag.208). La ventaja es que si bien la tabla puede tener millones de entradas, la misma puede usarse teniendo solo un reducido número de ellas en memoria.

9) ¿Qué es el working set?

Es el set de páginas que un proceso está utilizando en ese momento. Si un proceso tiene la totalidad de su working set en memoria, el mismo podrá ejecutarse sin producir ningún page fault, hasta pasar a otra fase de ejecución.

10) ¿Qué es el trashing?

Se produce cuando el espacio libre en memoria es demasiado pequeño, produciéndose un page fault por cada instrucción que contenga referencias. En un modelo de working set, un proceso podría causar un page fault cada vez que el sistema intenta levantarlo.

11) ¿Qué problemas debe tener en cuenta un diseñador de un sistema de paginación?

Alocación Local vs. Global: Determinar si conviene usar algoritmos de reemplazo locales (busca solo en las páginas alocadas por el proceso) o globales (busca en toda la memoria). En general los algoritmos globales funcionan mejor. Los algoritmos locales hacen crecer el working set produciendo trashing. Los algoritmos globales obligan al sistema a decidir cuantos page frames por proceso mantener en memoria (inspeccionando el tamaño del working set). Por lo general el buen funcionamiento de un reemplazo local o global varía según el método de paginación que se use.

Load Control: Cuando el sistema comienza a hacer trashing puede tomarse la decisión de remover algunos procesos de la memoria, en forma temporal (bajarlos a disco) para mejorar el rendimiento de la paginación.

Tamaño de la página: Para determinar el tamaño de la página es necesario balancear ciertos factores que compiten entre sí. Pequeño: Reduce la fragmentación interna (desperdicio). Reduce la cantidad de datos que se mantienen en memoria sin ser usados realmente. Aumenta el tamaño de la page table, generando incluso mas desperdicio de memoria que la fragmentación interna.

Separar instrucciones de datos: Amplía el espacio disponible en la memoria. Cada espacio, el de instrucciones y el de datos maneja su propio page table. Fácil de implementar.

Compartir Páginas: Permite eliminar el desperdicio de tener dos copias de la misma página en forma simultanea en la memoria. Por lo general se comparten las páginas de código.

Política de limpieza: Utilización de procesos en background (demonios) que duermen la mayoría del tiempo. Cuando despiertan inspeccionan el estado de la memoria. Mantienen un pool de page frames no referenciados (cuyas copias en disco fueron actualizadas si fue necesario) que serán removidos por el algoritmo de reemplazo. El page frame removido puede ser obtenido nuevamente del pool de frames.

Interface de VM: la MV no es del todo transparente al programador. Existe la posibilidad de permitirle al mismo manipular el mapeo de memoria. Por ejemplo:

- Permitir que dos o más procesos compartan la memoria.
- Memoria Compartida Distribuida (a través de una red).

12) Describa la memoria segmentada.

Memoria dividida en espacios de direcciones totalmente independientes unos de otros. Cada segmento puede crecer o achicarse sin afectar a los demás. Una dirección se compone de un número de segmento y una dirección dentro del segmento. Los segmentos pueden contener procedimientos, un arreglo, una pila, variables y demás, pero es raro que contenga una mezcla de diferentes tipos. Los segmentos simplifican el manejo de estructuras de datos que crecen y se achican. Permite compartir librerías y aplicar protección a nivel del segmento.

13) ¿Cómo usa el programador la memoria segmentada?

El programador puede decidir qué datos colocar en qué segmentos. Puede aplicar a los datos una protección en forma individual. Puede utilizar los segmentos para compartir librerías o memoria entre procesos. La forma de hacer uso de la memoria segmentada es desde luego mediante un lenguaje de bajo nivel, un lenguaje ensamblador. Por lo tanto, cuando se habla de programador, se refiere a un programador de lenguaje ensamblador o a un compilador de un lenguaje de cuarta generación que hace uso de la segmentación.

14) ¿Cómo puede combinarse segmentado con paginación?

La combinación surge del problema que se plantea cuando un segmento crece lo suficiente como para no caber en memoria. Sistemas que lo aplican:

MULTICS:

- Trata a cada segmento como una MV.
- Posee una tabla de segmentos, con descriptor por cada segmento. Dado que la cantidad de segmentos puede ser muy grande, la tabla en sí es un segmento y está paginada.
- El descriptor indica si el segmento está en la memoria (o parte del segmento). Si es así el descriptor apunta a la dirección de su page table (cada segmento tiene una page table).
- Cada dirección se compone de dos partes: un número de sector y una dirección en el sector. A su vez la dirección en el sector está formada por un número de virtual page y un offset en la página.
- Además usa una TLB.

PENTIUM: Ver directamente el libro.

Lo que hay que destacar, más allá de cómo lo implementa cada sistema, es el hecho de que todos resuelven primero la SEGMENTACIÓN y luego la PAGINACIÓN, en ese orden.

15) ¿Cómo cambian los principios de paginación en el modelo de Objetos?

Cada objeto tiene su propia *page table*, cada uno usa autopaginación. Los objetos paginan sobre si mismos usando su propio algoritmo. Por lo general usan uno standard (*safe paging*).

16) ¿Cómo afecta la multimedia a la Paginación?

Multimedia y Paginación no son compatibles ya que lo primero necesita una velocidad de proceso constante que la Paginación no puede brindarle.

GRAPHICAL USER INTERFACES

1) Describa como traduce el Sistema Operativo la entrada por teclado y por mouse.

Keyboard: Cada vez que se suelta o aprieta una tecla del teclado se manda una interrupción a la CPU, y el hardware provee el numero de esa tecla (que **no** es el código ASCII) depositándolo en el registro de E/S, le corresponde al driver del teclado extraer el carácter tipeado leyendo ese puerto e interpretar la acción a llevar a cabo. Todo lo demás ocurre por software, la mayoría por el driver del teclado. Si por ejemplo si se apretó la tecla A, se le coloca el código de la tecla A que es 30 en el registro E/S, y el driver debe determinar si es una A mayúscula o minúscula (shift + A) o que combinación de teclas se apretaron , esto lo hace porque guarda registro de qué teclas se apretaron y todavía no fueron soltadas.
En las Pentium, hay un microprocesador en el teclado que se comunica a través de un puerto serial especializado con un chip controlador en el parentboard.

Mouse: La mayoría de los mouse poseen un bola que sobresale en la parte inferior cuyo movimiento representa una traslación en la pantalla; cuando la bola gira, al mover el mouse, esta hace fricción sobre dos rodillos o ejes ubicados ortogonalmente (a 90° entre si) uno paralelo al eje X y otro paralelo al eje Y, como resultado estos ejes giran y representan los movimientos en X (este-oeste) y en Y(norte-sur).
Cada vez que el mouse se mueve una distancia mínima en cualquier dirección o se aprieta o suelta un botón del mouse, un mensaje es enviado a la computadora. Esa distancia mínima se llama mickey, es de 0.1 mm generalmente aunque puede ser modificada por software.
El mensaje enviado está compuesto por tres datos Δx, Δy, botones, se manda la distancia recorrida o variación en **x** y en **y** desde el último mensaje y el estado de los botones. El formato del mensaje dependerá del sistema y del numero de botones que posea el mouse (en general ocupa 3 bytes). El doble clic se produce si los dos clic están los suficientemente cerca en espacio y en tiempo, el software lo decide.

2) ¿Qué diferencias hay entre los gráficos vectoriales y los rasterizados?

Los dispositivos para gráficos vectoriales representan el dibujo como puntos en x y en y, por eso pueden dibujar puntos, líneas, figuras geométricas y texto(plotters). Mientras que los dispositivos para gráficos rasterizados (casi todos) representan el área de salida como una grilla rectangular de puntos llamados píxeles, cada uno de los cuales tiene algún valor de la escala de gris o algún color.
Para el despliegue de gráficos rasterizados se utiliza el adaptador grafico, el cual contiene una memoria RAM en donde se almacena la imagen de la pantalla en modo carácter o modo bit.

3) ¿Cómo es el tratamiento del color en los monitores, que relación tiene con la percepción?

Para el despliegue de gráficos rasterizados se utiliza el adaptador grafico, el cual contiene una memoria llamado video RAM (forma parte del espacio de direcciones que usa la CPU) en donde se almacena la imagen de la pantalla en modo carácter o modo bitmap (cada píxel es representado por 1 , 16 o 24 bits). El chip controlador de video es el que saca los caracteres o bits de la video RAM y genera la señal de video usada por el monitor. El monitor genera tres rayos de electrones que impactan sobre la pantalla horizontalmente dibujando líneas sobre él, estos tres rayos corresponden al rojo, al verde y al azul. Este escanéo que se realiza sobre la pantalla se hace hasta 100 veces por segundo. En un instante dado cada píxel esta formado por cierta intensidad distinta e independiente entre sí de esos tres rayos.
Es posible usar una paleta de colores, por ejemplo utilizando 16 bits por píxel para representar el color, de manera que solo hasta 65536 colores puedan ser vistos al mismo tiempo en el monitor. Esos 16 bits pueden ser utilizados para guardar el valor RGB (red, green, blue) con 5 bits para cada uno, o 6 para el verde (ya que es el color que nos parece más fuerte al lado del rojo y el azul, nuestro ojo es más sensible a él por lo que podemos distinguir más matices).
Este es lo que justamente nuestro sentido visual necesita para ver imágenes, ya que el ojo esta compuesto por bastoncillos y conos, los primeros son sensibles a la luminancia y los segundos están divididos en tres clases, los que son sensibles al rojo, los que lo son al verde y los que lo son al azul. Esto permite la visión en colores ya que la combinación de esos colores

primarios genera todos los demás colores. Pero debemos tener en cuenta que por ejemplo no existe la longitud de onda del color púrpura sino que para los humanos este color es solo una sensación, una combinación de longitudes de onda (la verde, roja y azul con cierta intensidad). Otro asunto a mencionar es que los humanos no ven colores o longitudes de onda, sino que distinguimos colores, porque si estuviésemos en una habitación con luz roja, lo que fuese verdaderamente rojo no lo podríamos identificar. Además ciertos colores nos parecen más fuertes que otros, por ejemplo el verde nos parece mas fuerte que el rojo, y este más fuerte que el azul.

4) Describa el modelo WIMP. ¿Cómo se implementa el WIMP en Win32, y en el X-Windows System?

La GUI (graphical user interface) fue inventada por Engelbart y posee cuatro elementos esenciales denotados por los caracteres WIMP: Windows, Icons, Menus y Pointing device. Los Windows son bloques rectangulares de área de pantalla usados para correr programas. Los Icons son símbolos que pueden ser clickeados para que alguna acción tenga lugar. Las Menus son listas de acciones en las que se puede elegir una. Los Ponting device son el mouse, trackball o cualquier otro dispositivo que permite mover un cursor sobre la pantalla para seleccionar ítems.
El software GUI es implementado en el código de nivel de usuario en los sistemas UNIX, mientras que es parte del sistema operativo en las WINDOWS.

En WINDOWS: la ventana es un área rectangular definida por su posición y tamaño dando las coordenadas en píxeles de dos esquinas opuestas, la izquierda superior y la derecha inferior. Una ventana puede tener title bar, menú bar, tool bar, vertical and horizontal scroll bar y puede ser cambiada de lugar, de tamaño o mini y maximizada. Los programas deben ser informados de estos cambios en el tamaño de la ventana, ya que deben redibujar el contenido de su ventana en cualquier momento. Como consecuencia los programas son orientados a eventos.
El dibujado en pantalla de cualquier cosa es controlado por un paquete de procedimientos llamado GDI (graphical device interface), el cual esta diseñado para ser independiente del dispositivo y de la plataforma. Antes que un programa pueda dibujar en una ventana necesita adquirir un device context que es una estructura de datos interna que contiene propiedades de la ventana como el text color, el font, background color, etc. , la mayoría de las llamadas al GDI usan el device context ya sea para setear u obtener propiedades como para dibujar.
La GDI contiene varias llamadas a procedimientos para obtener y liberar el device context, para obtener información sobre ellos, para fijar y obtener los atributos del device context, para manipular los objetos GDI como pens, brushes y fonts y por supuesto llamadas para realmente dibujar en la pantalla. Estas últimas llamadas caen dentro de cuatro categorías, dibujo de líneas y curvas, dibujo de áreas rellenadas, manipulación de bitmaps y despliegue de texto.
Los procedimientos GDI son ejemplos de gráficos vectoriales. Una colección de llamadas a procedimientos GDI puede ser agrupada en un archivo de extensión wmf, este es un archivo metafile ya que describe un gráfico. Las fotos y los videos no son gráficos vectoriales, sino que sus elementos son escaneados sobreponiendo una grilla encima de la imagen, el valor promedio de rojo, verde y azul de cada cuadrado de la grilla son tomados como muestra y representan el valor de un píxel, tales gráficos se llaman bitmaps. El problema con estos gráficos es que no pueden cambiar de tamaño conservando la forma y son problemáticos si se copian imágenes entre distintos dispositivos, por eso surge la estructura DIB (device independent bitmap) con extensión .bmp que soluciona este problema.
Para las fuentes antes se utilizaban los bitmaps, pero esto requería tener un bitmap por cada letra de tamaño diferente. Ahora se usa las fuentes TrueType que no son bitmaps sino siluetas de caracteres, cada carácter es definido por una secuencia de puntos alrededor de su perímetro. Esto permite que sean fácilmente escalables o cambiados de tamaño, solo se tiene que multiplicar cada coordenada de los puntos por el factor de escala.

X-WINDOWS: Hay dos filosofías sobre como debe ser una terminal de red, un punto de vista dice que la terminal debe tener una gran cantidad de poder de procesamiento y memoria para comunicarse a través de la red con protocolos, el otro punto de vista dice que la terminal debe ser extremadamente simple, básicamente debe desplegar píxeles y no hacer muchos cálculos para hacerla barata. X-WINDOWS pertenece a la primer filosofía y la SLIM a la segunda.

Una terminal X es una computadora que corre programas X y se comunica con otras aplicaciones corriendo en computadora remotas. El programa dentro de la terminal X que recoge el input del teclado, el mouse y acepta comandos provenientes de computadoras remotas se llama **X server**. El servidor se halla en la terminal y se conecta con usuarios remotos llamados **X clients** a través de la red enviándoles los eventos de teclado y mouse y aceptando sus comandos para dibujar en pantalla. Por eso el server debe saber qué ventana está activa y dónde está el puntero del mouse para enviar los eventos. Desde el punto de vista del programa, este es un cliente diciéndole al servidor que haga cosas como desplegar figuras o texto por pantalla.
Es posible montar **X Windows System** sobre un UNIX, lo que hace X realmente es definir un protocolo de comunicación entre el **X client** y el **X server**, sin importar que estén en la misma maquina o separados por km, el protocolo y la operación del sistema es el mismo en todos los casos.
X no es un GUI completo solo es un sistema de ventanas, para completarlo otras capas de software se necesitan, la primera se llama **Xlib** que es un conjunto de procedimientos para acceder a la funcionalidad del X. Otra capa por encima de **Xlib** se llama **Intrinsics**, esta capa maneja botones, scroll bars y otros elementos GUI llamados widgets. Para hacer una buena GUI interface se necesita otra capa por encima, la mas popular se llama **Motif** con el cual los programas se comunican.
La administración de ventanas no es parte del X, se encuentra aparte, por lo que un proceso del cliente llamado windows manager (KDE, GNOME) debe hacerse cargo de la creación, borrado y movimiento de las ventanas en la pantalla. Para hacer esto el windows manager manda comandos al X server diciéndole que hacer. El que se encuentre separado permite que corra remotamente.
Este diseño modular con varias capas y múltiples programas lo hacen muy portable entre diferentes UNIX. Mientras que el GDI es muy complicado de mantener porque los sistemas de GUI y de ventanas están mezclados entre si y colocados en el kernel.
Los mensajes por la red por TCP/IP entre el X program y el X server (en diferentes maquinas) pueden ser de cuatro tipos: comandos de dibujo desde el programa hacia el server, respuestas del server a los pedidos del programa, anuncios de eventos (mouse, teclado, etc.) y mensajes de error.

5) ¿Qué facilidades adicionales provee el X-Window System?

Fue respondida en el punto anterior.

6) Describa las diferencias entre los modelos de eventos de win32 y de X-Window. Compare con el modelo observador-observable de Java.

En WINDOWS los programas son orientados a eventos, las acciones del usuario que envuelven el teclado o el mouse son capturadas por el SO y convertidas en mensajes a la aplicación dueña de la ventana activa. Cada programa posee una cola de mensajes a la cual van todos los mensajes relativos a sus ventanas y posee un loop principal que se va fijando en el próximo mensaje y lo procesa llamando al procedimiento apropiado para ese evento o mensaje. En ciertas circunstancias el SO mismo puede pasar por alto la cola de mensajes y ejecutar directamente algún procedimiento del programa. Cada ventana debe estar asociada a un objeto clase que define sus propiedades, tal objeto se llama wndclass, la más importante de sus propiedades se llama lpfnWndProc que es un puntero a la función que maneja los mensajes dirigidos a sus ventana.
Entonces un programa esta compuesto por un main y una función que maneja los eventos. En el main después de definir las propiedades de la ventana se registra el objeto clase para que WINDOWS sepa a que procedimiento llamar para manejar los mensajes, luego se crea la ventana en la pantalla visualmente. Después viene el loop principal para traducir los mensajes recibidos y despacharlos al SO para que este llame al WndProc, la función que maneja los mensajes recibidos. Esta función recibe eventos como la ventana se crea, se destruye, se repinta, etc.

Como WINDOWS, X es un sistema manejado altamente por eventos, el programa puede especificar qué eventos quiere escuchar o quiere que le manden y cuales no, hay una cola de eventos de la cual el programa lee, pero el SO jamás puede llamar a procedimientos dentro del programa por si mismo como ocurre con windows.

Un concepto clave en X es el recurso, que es una estructura de datos que contiene cierta información como una ventana, fuente, colormap (color palettes), pixmap(bitmaps), cursor y graphic context (similares a los device context de windows). Las aplicaciones crean recursos en X server que pueden ser compartidos por múltiples procesos y que son de vida corta.
Un programa X debe conectarse con el X server, reservar memoria para la ventana, decirle al windows manager de la existencia de su ventana para que la conozca y la administre, crear un graphic context, especificar los tipos de eventos que quiere escuchar, desplegar la ventana por pantalla y entrar dentro de un while donde hay un switch para el manejo de los diferentes tipos de eventos.

Modelo JAVA: cada vez que se crea un elemento del GUI como un botón, JPanel, etc. se debe añadir un listener al elemento si uno quiere que escuche los eventos que ocurren y actuar en consecuencia, por supuesto que se puede agrupar elementos con un mismo listener. Esto se codifica y es interpretado por la JVM.

ADMINISTRACION DE ARCHIVOS

1) ¿Cómo influye la estructura de directorios sobre el problema de naming de los archivos?

Primero veamos que es el problema de naming: En un principio, los sistemas tenían una estructura de un solo directorio, el problema que se presentaba era que los archivos eran identificados por su nombre y en máquinas que utilizaban mas de un usuario, podía darse el caso de que dos usuarios pusieran el mismo nombre a sus archivos sobrescribiéndose los datos uno a otro, la solución a este problema fue crear una estructura de directorios, en un principio fue uno para cada usuario de manera que los usuarios podían en ese directorio nombrar a sus archivos como quisieran, sin temor de sobrescribir los de otro usuario, dado que dos archivos con el mismo nombre pero en distintos directorios no se interfieren; de esta forma la estructura de directorios soluciona el problema de naming, brindando distintos contextos a los usuarios, donde estos pueden repetir nombres que están en otros directorios.

2) ¿Qué tipos de archivos reconoce Windows y cuales reconoce UNIX?

Hay dos visiones para esto, según la primera visión (TANENBAUM), en Windows se reconocen dos tipos de archivos:

Directorios*:* Son archivos de sistema para mantener la estructura del File System.
Archivos regulares*:* Son los archivos que contienen los datos del usuario, y a su vez se dividen en archivos *ASCII* y *binarios.*

En UNIX, además de los dos tipos que ve Windows, hay dos tipos de archivos mas, estos son:

Character special files*:* están relacionados a la Entrada/Salida y se usan para modelar dispositivos seriales.
Block special files*:* Se usan para modelar discos.

El otro punto de vista (dado en clase y en TANENBAUM), refiere a tipos de archivo en cuanto a su estructura, ejemplos de estos son los BMP asociados a un BITMAP, o los .C asociados a un código fuente en lenguaje C. Desde este punto de vista UNIX no maneja tipos y en Windows los tipos se manejan como asociaciones en el registry entre una aplicación y una extensión de esta forma todos los archivo de extensión cpp estarán relacionados a un compilador de C++, pero windows no se ocupa de controlar el formato, son las aplicaciones las que esperan que los archivos tengan un formato específico.
Con esto también se está protegiendo a los archivos de un uso indebido como puede ser deszipear un archivo que no es ZIP.

3) Dé un ejemplo de programas que obliguen al uso de tipos. Indique como se soluciona el soporte de tipos.

Son varios los programas que obligan al uso de tipos, por ejemplo un compilador de C++ solo aceptará archivos que tengan la extensión C o CPP, de esta manera esta manejando los tipos por medio de su extensión, esto es particularmente útil, dado que de acuerdo al tipo de archivo el compilador deberá realizar operaciones diferentes. Otro programa que exige el uso de tipos es el compresor ZIP.
Las formas en que se resuelve el uso de tipos son principalmente dos:
Con las extensiones de los archivos, esto en Windows sirve, pero no en UNIX, donde las extensiones son convenciones para el usuario, pero no significan nada para el sistema operativo.
La otra forma es colocando un número al principio de los archivos llamado *magic number* donde cada número esta relacionado con un tipo de archivo específico, de esta forma las aplicaciones abren el archivo, y leyendo ese número saben si el archivo es del tipo correspondiente.

4) Dé ejemplos de atributos de un archivo.

Todos los archivos tienen un nombre y sus datos. Además de esto, todos los sistemas operativos los asocian con otra información, esta información extra son los atributos del archivo, los atributos de los archivos varían de un sistema operativo a otro, pero algunos de los que se pueden encontrar son los siguientes:

Atributo	*Significado*
Protección	Quien puede acceder y de que forma
Password	Password necesario para acceder a un archivo
Creator	ID del creador
Owner	ID del actual owner
Marca de solo lectura	0 para lectura escritura, 1 para solo lectura
Marca de Oculto	0 para normal, 1 para oculto
Marca de archivo de sistema	0 para archivos normales, 1 para los de sistema
Marca de ASCII/binario	0 para ASCII, 1 para binario
Marca de acceso random	0 para solo acceso secuencial, 1 para acceso random
Marca de temporario	0 para normal, 1 para borrar el archivo al final del proceso
Marca de bloqueo	0 para desbloqueado, distinto de 0 para bloqueados
Longitud de registros	Cantidad de bytes en un registro
Posición de la clave	Offset de la clave dentro de cada registro
Longitud de la clave	Cantidad de bytes en el campo clave
Fecha de creación	Fecha y hora de creación del archivo
Fecha de último acceso	Fecha y hora del último acceso
Fecha de última modificación	Fecha y hora de la última modificación
Tamaño actual	Cantidad de bytes en el archivo
Tamaño máximo	Máxima cantidad de bytes para el archivo

5) Indique que hacen las siguientes operaciones sobre los archivos: *Create, Delete ,Open, Close, Read, Write, Append, Seek, Get Attributes, Set Attributes, Rename.*

Create: El archivo se crea sin datos. El propósito de este system call es avisar que se usará un archivo y setear algunos de sus atributos.
Delete: Cuando el archivo ya no es necesario, debe ser borrado para liberar espacio en disco.
Open: Antes de usar un archivo un proceso debe abrirlo. El propósito de este system call es permitirle al sistema recuperar los atributos del archivo, y la lista de direcciones de disco en memoria principal, para un acceso mas rápido en posteriores llamadas.
Close: Cuando todos los accesos terminan, los atributos y las direcciones de disco ya no son necesarias, entonces se debe cerrar el archivo para liberar el espacio que estos ocupan en las tablas internas. Cerrar el archivo fuerza a que se escriba a disco.
Read: Para leer datos de un archivo. Generalmente se lee desde la posición actual, la llamada debe proveer la cantidad de datos a leer y un buffer donde ponerlos.
Write: Los datos son escritos a un archivo, generalmente en la posición actual, si esta es el final del archivo, el tamaño del mismo crece, si es en el medio, los datos anteriores se sobrescriben.
Append: Es una forma restringida del write que solo permite escribir al final del archivo, muchos sistemas operativos no la proveen, ya que puede hacerse con un seek y un write.
Seek: Para archivos de acceso random es necesaria una forma de decirle de donde debe tomar los datos. Este system call mueve el file pointer a la posición indicada.
Get attributes: Sirve para obtener los atributos de un archivo.
Set attributes: Muchos de los atributos de un archivo son seteables, un ejemplo son los permisos, para poder realizar ese seteo está este system call.
Rename: Frecuentemente un usuario necesita cambiar el nombre de un archivo, para realizar esta operación está este system call.

6) Indique qué hacen las siguientes operaciones sobre directorios: *Create, Delete, Opendir, Closedir, Readir, Rename, Link, Unlink.*

Create: Crea un directorio que solo contiene las entradas . y .. referidas a sí mismo y al padre respectivamente.

Delete: Borra un directorio, solo se pueden borrar directorios que están vacíos. Un directorio se considera vacío si solo tiene las entradas . y .. .
Opendir: Los directorios pueden ser leídos, por ejemplo para listar su contenido, para ello primero deben ser abiertos como los archivos, esta apertura la realiza este system call.
Closedir: Cuando un directorio ha sido leído debe cerrarse, para liberar el espacio, esto se hace con este system call.
Readdir: Este system call, devuelve la próxima entrada en el directorio, formalmente se puede hacer con el read de archivos, pero obliga al usuario a conocer la estructura del directorio, en cambio esta devuelve siempre una entrada del directorio independientemente de la estructura.
Rename: Sirve para cambiarle el nombre a un directorio.
Link: Linkear es una forma de hacer que un archivo aparezca en varios directorios. Este system call especifica un archivo existente y un Path Name, y realiza un link entre ambos. De esta forma el mismo archivo aparece en varios directorios y con nombres distintos.
Unlink: Se da de baja una entrada del directorio, si ese archivo aparecía solo en el directorio, también se da de baja el archivo, caso contrario solo se da de baja la referencia.

7) Para que se usan los buffers.

Los buffers son capas de memoria que se usan como intermediarios entre Virtual File Systems, amortiguando diferencias de velocidad, y en algunos casos como la grabación de CD´s, asegurando una continuidad y uniformidad de transferencia.

8) Describa un ejemplo del uso de buffers.

Un ejemplo de uso de buffers es la Entrada/Salida a través de una red, vamos a ver un par de formas distintas para ver la utilidad y los inconvenientes solucionados por el uso de buffers:

Entrada:

- *Sin uso de buffers*: El proceso hace un system call Read y se bloquea esperando por un caracter, cada caracter que llega produce una interrupción que le entrega el carácter al proceso y lo desbloquea, el proceso guarda el carácter y repite el system call, realiza este loop hasta que recibe el mensaje completo. El problema de este método es que es muy poco eficiente.
- *Con un buffer de n caracteres:* El proceso es parecido al anterior, solo que en este caso, la interrupción va poniendo los caracteres que llegan en el buffer del usuario, y desbloquea el proceso, recién cuando este buffer se llena, de esta manera se logra mejorar un poco la performance; pero tiene el problema de que la página donde está el buffer debe quedar bloqueada en memoria, y si hay muchas páginas en este estado, la performance de la máquina vuelve a decaer.
- *Un buffer de usuario y un buffer del kernel:* La forma de solucionar el caso anterior es poner un buffer del kernell y cargar ahí los datos entrantes, y cuando este se llena, recuperar la página del buffer del usuario y copiar ahí el buffer del kernell, el problema que presenta esta forma es que si mientras se recupera la página del buffer entran nuevos caracteres, no hay donde ponerlos y se perderían.
- *Doble Buffering:* Para resolver el problema del modelo 3, lo que se hace es agregar un buffer mas en el kernell, de esta forma, cuando se busca la página del buffer del proceso de usuario, los nuevos caracteres que entran se guardan en este nuevo buffer, que luego cuando se llene completará el buffer del usuario, mientras que el buffer original le hace de respaldo, y así van intercambiando los roles.

Salida:
El proceso hace un write para dar salida a n caracteres; en este punto el sistema puede tomar 3 decisiones:

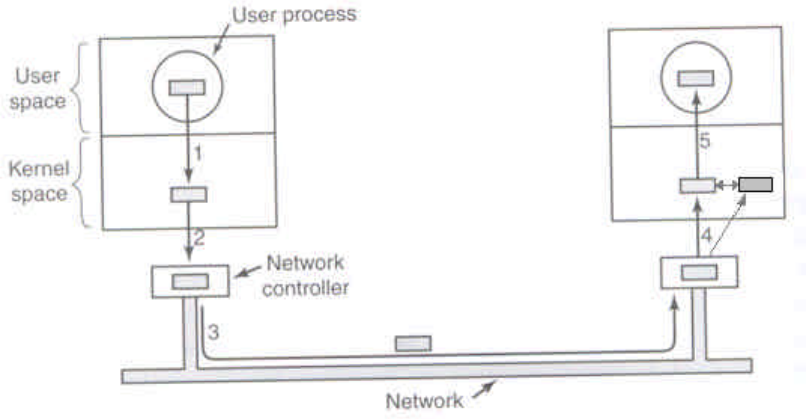
- Bloquear el proceso hasta que todos los caracteres hayan sido transferidos, esto puede ser muy lento si el medio de transmisión es una línea de teléfono
- Puede liberar al proceso para que haga otras cosas mientras termina la transferencia, pero presenta el inconveniente, que para avisarle al proceso que puede reutilizar el buffer hay que realizar interrupciones por software, lo que lleva a una programación complicada y que propicia las Race Conditions.

- Una mejor solución es usar un buffer en el kernell donde se copia el buffer a transmitir, y se libera al proceso, permitiéndole reutilizar el buffer.

Para completar y clarificar hacemos un ejemplo completo de comunicación entre dos máquinas:

1- El proceso hace un write, y el Kernell copia el buffer al buffer del kernell.
2- Cuando se llama al *driver* este copia el buffer del kernell en el buffer del controlador, hace esto para asegura que la transferencia se haga a una velocidad uniforme.
3- El controlador copia el paquete a la red donde la otra terminal va guardando en el buffer del controlador los bytes que llegan
4- El paquete se copia al buffer principal del kernell.
5- Finalmente el buffer se copia en el buffer del proceso receptor.

Generalmente el proceso receptor devuelve una señal de recepción, cuando el emisor del mensaje recibe esta respuesta, puede mandar el siguiente paquete.
Es importante aclarar, que todo este movimiento entre buffers retrasa un poco la transmisión ya que debe hacerse en forma secuencial.



9) Indique las funciones de las siguientes partes del software de I/O: Software de I/O de Nivel del Usuario; Software de I/O independiente del dispositivo; Software dependiente del dispositivo; Software de atención de interrupciones relacionado.

Software de I/O de Nivel del usuario: Tiene como principales funciones a las siguientes:

a) Hacer las llamadas para Entrada/Salida, mediante System calls, los cuales generalmente se hacen en librerías de procedimientos.
b) Dan formato a la Entrada/Salida.
c) Realizan Spooling: Esto es una forma de tratar a los dispositivos de E/S dedicados en un sistema multiusuario, el problema que se presenta es que si el uso de estos dispositivos se deja libre al usuario, puede pasar que un proceso tome control de un dispositivo y lo retenga sin hacer nada, interrumpiendo a los otros procesos; esto se soluciona con este método, que crea un *daemon* y un directorio especial llamado *directorio de spooling* donde se ponen los archivos que requieren hacer uso del dispositivo, y luego es el *daemon* el único que tiene permiso para usar el dispositivo, y este va tomando los archivos en el directorio especial y los procesa en el dispositivo, de esta forma se evita que un proceso acapare un dispositivo.

Software Independiente del dispositivo: Mucho del software de E/S depende del dispositivo, pero hay otra parte que es independiente y es la que implementa esta capa; el borde entre esta capa y los drivers depende del dispositivo y del sistema, dado que hay muchas operaciones que podrían ser independientes, pero que por ciertas causas las implementa el driver. Las funcionalidades que debe cumplir esta capa son las siguientes:

a) Proveer una interfase uniforme para los drivers: Debe proveer una interfase uniforme para cada tipo de dispositivo, de manera que los fabricantes de drivers sepan que funciones deben implementar. Otro aspecto de esta interfase es dar un estándar de cómo se nombraran los dispositivos, por ejemplo en UNIX, un dispositivo de nombre /dev/disk0 apunta a un I-Nodo de un archivo especial a partir del cual se puede acceder al driver específico; con esto de los nombres también se logra proteger a los dispositivos de accesos no permitidos, utilizando el mismo sistema de protección que para los archivos.
b) Buffering: Debe proveer todos los mecanismos de buffering.
c) Reporte de Errores: En la E/S los errores son muy frecuentes, en esta capa se debe dar la estructura del manejo de errores, los errores pueden ser de programación en cuyo caso simplemente se devuelve un código de error al system call, o los errores pueden ser propios del dispositivo y que el driver no pueda resolverlos en cuyo caso la solución puede ser un mensaje preguntando que hacer, o abortar la operación, etc y hay errores sobre estructuras críticas de datos como el directorio raíz, el sistema debe enviar un mensaje de error y abortar.
d) Alocar y liberar dispositivos dedicados: hay dispositivos que solo pueden ser usados de a uno por vez , es función del sistema analizar si los pedidos son aceptados o rechazados dependiendo de si el dispositivo esta disponible o no.
e) Proveer un tamaño de bloque independiente del dispositivo: Los dispositivos trabajan con distintos tamaños de bloque, el trabajo de esta capa es ocultar esta diferencia y proveer un tamaño de bloque uniforme para las capas superiores.

Software dependiente del dispositivo(Drivers): Un driver de dispositivo tiene las siguientes funciones:

a) Aceptar pedidos de lectura y escritura de la capa independiente, y hacer que se lleven a cabo.
b) Inicializar los dispositivos si es necesario
c) Debe manejar los requerimientos de energía del dispositivo.

Todo esto lo hace con conocimiento de la estructura del dispositivo, por ejemplo para lectura o escritura de un disco, el driver debe convertir un número de bloque en los parámetros Cabeza, Cilindro y Sector, y manejar el dispositivo escribiendo y leyendo los registros del controlador del dispositivo.
Los drivers no pueden utilizar system calls, sin embargo en ocasiones necesitan llamar al kernell, para alocar memoria, solicitar un DMA, etc., por ello hay ciertos procesos del kernell a los cuales les es permitido acceder.

Software de atención de interrupciones: Se busca que las interrupciones estén lo más ocultas posibles, para esto lo que se hace es que el driver que lanza la operación de E/S se bloquee a sí mismo.
La función de las interrupciones será desbloquear el driver que disparó la operación de E/S.

10) Describa brevemente como se accede a un dispositivo usando: Modos de I/O programados (PIO), Acceso directo a memoria (DMA), Manejo por Interrupciones.

PIO (Programmed I/O): Con este método la CPU hace todo el trabajo. En este método, la CPU se encarga de hacer la transferencia, quedando esta pendiente de la transferencia que se hace de la siguiente manera:
El acceso consiste en que la CPU entra en un loop en el cual va revisando los registros de los controladores, fijándose cuando está listo para recibir un caracter para mandar a salida, o cuando recibió un caracter de la entrada, y cuando el dispositivo esta listo lee del buffer del controlador, o escribe en el mismo; de esta forma la CPU, esta constantemente revisando los registros del controlador en un estado de *pooling o busy waiting*, no pudiendo hacer otra cosa. Cuando la operación de I/O termina, la CPU devuelve el control al proceso que se estaba ejecutando.
Claramente la desventaja es que ocupa todo el tiempo de CPU hasta que la transferencia termina.
Otra desventaja es que los "modos" PIO indican cuanto tiempo debe transcurrir entre comando y comando de la operación. Esto complica mucho los drivers ya que dicho timing debe ser "calculado" en función de la duración de las instrucciones y esta duración varia a medida que aumenta la velocidad del clock.

DMA (Acceso directo a memoria): Para comprender esto veamos cómo trabaja un control básico de DMA:

a) La CPU setea los registros del controlador de DMA , tales como la dirección base de memoria principal, la cantidad de bytes, en que sentido es la operación (Entrada o Salida). También envía un pedido de lectura o

escritura al control de disco. En escritura, cuando en el buffer del controlador del dispositivo hay datos válidos, comienza el DMA

b) El DMA inicia la transferencia, solicitando una lectura al controlador de disco y pone en el bus de direcciones la dirección donde debe escribirse en memoria. En el caso de escritura, manda el WORD al buffer del controlador de disco.
c) Se realiza la escritura a memoria o a disco
d) El controlador de disco manda una señal de acknowledgement al controlador de DMA. El controlador de memoria aumenta la dirección de memoria y decrementa la cantidad de bytes a leer, y repite los pasos b, y d hasta que la cantidad de bytes a copiar sea 0, en ese momento se emite una interrupción para avisar que se completó la transferencia y desbloquear el proceso que disparó la operación de E/S.

La idea es usar el DMA explicado arriba para realizar la transferencia, en esencia es una transferencia tipo PIO, con la diferencia de que el procesador que se encarga de la transferencia ya no es la CPU, que está haciendo otras cosas, si no el controlador de DMA. Este método es bueno, siempre que el controlador de DMA pueda manejar el bus a alta velocidad y la CPU tiene otros procesos para correr, dado que el controlador de DMA es mas lento que la CPU.

Manejo por interrupciones: La CPU, inicializa el dispositivo con los parámetros necesarios y los buffers; luego de hacer esto la CPU bloquea el proceso que llamo a la E/S y llama al Scheduler para ejecutar otro proceso.
Cuando el dispositivo recibió un nuevo caracter o está listo para sacar otro carácter emite una interrupción. Esta interrupción frena el proceso que estaba corriendo y guarda su estado. Luego el proceso de la interrupción comienza a correr, si no quedan mas caracteres para sacar o recibir, desbloquea el proceso que llamo a la E/S, caso contrario continua con el siguiente byte y retorna el control al proceso que estaba corriendo al momento de la interrupción.
De esta forma la CPU solo se ocupa de la transferencia cuando es necesario.
La desventaja de este método es que tiene muchas interrupciones que desperdician mucho tiempo de CPU.

11) Intente explicar paso a paso como los datos se transfieren desde una aplicación hasta el disco.

Basándonos en las respuestas anteriores el proceso sería de la siguiente forma atravesando por todas las capas de software de I/O:

a) La aplicación en algún momento llama al system call write.
b) Esta llamada será atendida por el software independiente del dispositivo, en esta etapa el buffer a copiar a disco se copiara en un buffer del kernel y se produce una llamada al driver del dispositivo usando la interfase definida en este nivel.
c) El driver del disco toma los parámetros que le pasa la capa independiente y transforma los datos para acomodarlos al dispositivo y completar los registros del controlador de disco (Ej: Transformar a la geometría del disco, la dirección lógica que se le pasa) Cabe destacar, que todas estas transformaciones las puede realizar mediante los datos obtenidos del File System durante una operación de montaje del mismo (En Unix mount). Luego manda mensajes al kernell para solicitar un DMA (en el caso que se use esta transferencia, se pedirán otros servicios en otros casos).
d) Una vez que el driver termina su trabajo, comienza la escritura, durante ese lapso el controlador de DMA le irá proveyendo al controlador de disco los datos a escribir; cuando se termina la transferencia, el DMA emite una interrupción y desbloquea el driver y el proceso que pidió la E/S.

SISTEMAS OPERATIVOS MULTIMEDIALES

1) ¿Qué se entiende por multimedia? ¿Y por sistema de información multimedial?

El término 'multimedia' hace referencia a la capacidad de presentar simultáneamente tipos variados de información, como parte de un diseño común.
La mayoría de la gente emplea la palabra multimedia para indicar un documento con dos o mas formas de "media" (video, audio, etc), esto es media que puede ser reproducida en un intervalo de tiempo.

2) ¿Cuáles son las exigencias de soporte multimedial para los sistemas operativos?

Las áreas donde la multimedia debe ser soportada es en la reproducción y edición de video, música, en juegos de computadora y en la provisión de video por demanda, teniendo en cuenta que la multimedia maneja un promedio de datos extremadamente alto y requiere de reproducción en tiempo real.
Para soportar todo esto se necesita un sistema operativo con un adecuado file system, scheduling de proceso y scheduling de disco.

3) ¿Qué es la calidad de servicio en un sistema operativo para multimedia? ¿Qué nuevas operaciones de planificación aparecen?

La calidad se servicio (QoS) consiste en un conjunto de parámetros que se deben prever y satisfacer para reproducir multimedia en condiciones aceptables, incluye parámetros como promedio de ancho de banda disponible, pico de ancho de banda disponible, "delay" mínimo y máximo, y probabilidad de pérdida de bit (bit loss).
Las operaciones de planificación consisten en reservar capacidad (o recursos) para aceptar nuevos usuarios. Los recursos reservados incluyen una porción de CPU, buffer de memoria, capacidad de transferencia de disco, y ancho de banda de la red. Los servidores multimediales necesitan esquemas de reserva de recursos y algoritmos de control de admisión para decidir cuando pueden tomar mas trabajo.

4) Describa una posible organización en subarchivos de un archivo multimedia. ¿Qué nuevas operaciones sobre los archivos aparecen?

Si tenemos en cuenta que una película viene acompañada de sonido y subtítulos podemos pensar que esta está representada por un archivo de video, varios archivos de sonido y de texto. En consecuencia el sistema de archivos necesita mantener múltiples subarchivos por archivos. Una posible organización es manejar cada subarchivo como un archivo tradicional (ejemplo: con inodes) y tener una nueva estructura de datos que liste todos los subarchivos por archivo multimedia. Otra forma es inventar una especie de inodo de dos dimensiones, con cada columna listando los bloques de cada subarchivo. En general la organización debe ser tal que haya opción para cambiar dinámicamente de subtítulos e idioma mientras se reproduce la película.

5) Describa someramente la codificación de audio y video.

Codificación de audio: las señales de audio analógicas pueden ser convertidas a digital con un conversor analógico/digital, este toma una señal analógica y la convierte en una salida binaria. Para generar una señal digital debemos "muestrear" la señal analógica cada delta T segundos. La representación digital no es exacta, el error introducido por el número finito de bits por muestra se llama error de cuantización, si este es muy grande el oído puede percibirlo.

Codificación de video: para comprender la codificación de video, debemos empezar por video en blanco y negro (analógico). Para representar la imagen la cámara escanea un haz de electrones rápidamente a lo largo de la imagen grabando la intensidad de luz. Al finalizar el escaneo, llamado frame, el haz vuelve a comenzar. La intensidad es transmitida y los receptores repiten el proceso de escaneo para reconstruir la imagen. El sistema NTSC tiene 525 líneas de escaneo, y 30

cuadros por segundo. Existen dos técnicas para mostrar la imagen, la primera reconstruye la imagen presentando las líneas escaneadas desde la parte superior hasta la inferior y se la conoce como progresiva, mientras que la técnica de entrelazado (interlacing), primero muestra las líneas impares y luego las pares.
El video a color (analógico) utiliza la misma técnica, pero utiliza tres haces que representan los colores rojo, verde y azul (RGB), que son combinados para mandar una única señal.
La representación más fácil de video digital es una secuencia de cuadros que consiste en una grilla rectangular de pixeles, con 8 bits por pixel es suficiente para representar los colores RGB dando 16 millones de colores. Para dar una buena imagen con movimientos suaves es suficiente mostrar 25 cuadros por segundo. Los monitores de computadora utilizan la técnica progresiva para mostrar la imagen.

6) Describa someramente las operaciones de la técnica de compresión MPEG.

MPEG consiste en una técnica para comprimir video, es un estándar y presenta los algoritmos MPEG-1 y MPEG-2. Ambas versiones aprovechan las 2 principales redundancias que existen en video, espacial y temporal. La redundancia espacial se lleva a cabo comprimiendo cada cuadro por separado con JPEG, mientras que la redundancia temporal consiste en la comparación entre cuadros consecutivos.
Se distinguen tres tipos de cuadros:

- *I frames:* cuadros codificados con JPEG
- *P frames:* diferencias en bloques entre el cuadro actual y el anterior
- *B frames:* diferencias entre el actual, el anterior y el siguiente cuadro.

P frames y B frames se basan en la idea de macrobloques (grilla de bloques que cubren toda la imagen) para detectar qué partes son iguales y cuales son distintas en la sucesión de cuadros.

7) ¿Qué es el video on demand y el near video on demand? ¿Qué exigencia aparece sobre el sistema operativo?

Video on demand consiste en un servicio en donde los consumidores seleccionan un video usando el control remoto (o mouse) y este se reproduce en su televisión (o monitor).
Near video on demand, es similar pero la reproducción se lanza a un determinado horario y se relanza cada delta T minutos (comienza a las 8, se relanza 8:05, 8:10 etc.), o sea que el video no comienza al instante que es pedido sino casi "cerca" al momento del pedido.
La exigencia que aparece en el sistema operativo es una adecuada administración de los streams transmitidos, reserva de recursos y provisión de los bloques de datos sin demora.

8) ¿En la jerga de streaming, qué se conoce como tecnología push y pull?

Pull: los usuarios hacen llamados reiterados para que los datos sean entregados. Es una secuencia donde para que un bloque de datos sea entregado primero debe ser requerido.
Push: el usuario hace un system call de start y el servidor de video comienza a entregar los datos sin necesidad de ser requeridos.

9) Compare los modelos de archivos multimediales con bloques pequeños y con bloques grandes (con y sin fragmentación interna) en cuanto a: uso de RAM, desperdicio de espacio en disco, tiempo I/O, cue, rew.

Organización con bloque pequeño: los bloques de disco son mas chicos que el tamaño promedio de los frames. La idea es tener una estructura de dato con un índice de frame apuntado al inicio del frame. Cada cuadro consiste de audio, video y texto como una secuencia contigua de bloques de disco.
Organización con bloque grande: pone múltiples frames en cada bloque, requiere de un índice de bloques. Un bloque puede no contener una cantidad pareja de cuadros, dando lugar a dos opciones, como primer opción cuando un frame no entre en el bloque, se deja el espacio libre. Como segunda opción se puede llenar cada bloque hasta el final, partiendo los frames entre los bloques, introduciendo la necesidad de hacer seeks entre frames.

Comparación:

	Bloque pequeño	Bloque Grande sin fragmentación	Bloque grande con fragmentación
RAM	Mucho uso	Poco uso	Poco uso
Desperdicio espacio	Bajo	Mayor	No hay
Tiempo I/O			
Fast forward y fast backguard	Es posible a través de los I frames	No es posible de hacer, directamente, solución: otro archivo que cuando se reproduce de la sensación que lo hace en 10x	No es posible de hacer directamente, solución: otro archivo que cuando se reproduce de la sensación que lo hace en 10x

10) ¿Cómo varía la exigencia de uso de los archivos en un servidor de near video on demand?

11) ¿Cómo se usa la ley de Zipf para colocar varias películas en el mismo disco?

Para películas en servidores, la ley de Zipf dice que la película más popular es generalmente dos veces más elegida que la segunda más popular, tres veces mas que la tercera más popular y así sigue.
Una conclusión es que se necesita tener muchas películas para establecer el ranking de popularidad.
Conociendo la popularidad relativa de las diferentes películas es posible modelizar la performance de un servidor y utilizar esa información para colocar los archivos de video. La estrategia es simple, consiste en colocar la película más popular en el medio del disco, con la segunda y tercer película a ambos "costados", mas afuera de estas la cuarta y la quinta etc.

12) Describa el proceso por hard de la multimedia según tecnologías conocidas como DSP, ASIC, y Stream processor.

Para la próxima edición ...

PALM OS

1) Describa los componentes de hard de una plataforma Palm.

Componentes de hard de una Palm:

- Pantalla que recibe eventos (taps) del usuario por medio de un lápiz.
- Zona de la pantalla permitida para escritura, mediante ciertos jeroglíficos.
- Cuatro botones programables, que de fábrica, llaman a las aplicaciones comunes (agenda, schedule, memo, To Do).
- Botón de encendido (que también sirve para encender la luz fluorescente).
- Sócalo de sincronización, para conectarla con la PC.
- Conector infrarrojo para comunicarse con otra Palm u otros dispositivos (ejemplo: Televisión).

2) Cuales son los modos de consumo de potencia de una Palm.

Modo Sleep: se mantiene el flujo de energía a la memoria, pero se la quita a la pantalla y a los circuitos de entrada.
Modo Doze (siesta): el procesador se detiene después de ejecutar cada instrucción (debe tener un micro específicamente diseñado para ese propósito).
Modo Running: mientras se esta ejecutando una instrucción.

3) Que hay en RAM y en ROM, que parte puede asimilarse a un file system y que parte a la memoria RAM de una computadora de propósito general.

En ROM esta el sistema operativo y las aplicaciones básicas que vienen por default en la palm. En la RAM se cargan los programas nuevos del usuario (que persisten mientras se le provea energía).
La RAM se comporta (a diferencia de la pc) de manera similar a un file system, posee las primitivas conocidas y es administrada de esa manera. El análogo a la RAM de la computadora personal es la ROM de la Palm ya que esta si esta organizada de esa manera.
En algunas Palm, el SO se aloja en la RAM, con lo cual es posible cargarle una nueva versión del SO.

4) ¿Qué es un *soft reset*, *soft reset+page up* y un *hard reset*?

Tipos de Reset: Soft, Warm, Hard.

Soft reset: vuelve a lanzar todas las aplicaciones que estaban lanzadas (esto se hace principalmente cuando se colgó o no funciona algo en particular pero se desea seguir trabajando en el ambiente actual).
Hard reset: borra toda la ram (esto se hace principalmente cuando se desea dejar la configuración de la palm en cero para cargarle otros programas).
Warm reset: a diferencia del soft reset, las aplicaciones quedan cerradas (esto se hace principalmente después de un cuelgue "importante" y cuando no se desea seguir en el mismo ambiente).
Igualmente, como recomendación del profesor, habria que bajarse un POSE para ver mejor la funcionalidad de la Palm para los "Palm-disabled" (http://www.palm.com).

5) ¿Qué es POSE?

POSE (Palm Operating System Emulator) es un programa que se ejecuta en una pc de escritorio que emula la palm. Se usa generalmente para desarrollo (prueba de aplicaciones) y se le debe cargar la ROM del modelo de Palm requerido antes de ser usado.

6) ¿Cómo es la sincronización, backup e instalación de aplicaciones en una palm?

El objetivo de la sincronización de una Palm con la PC es la de tener la misma información en ambos lados, tanto para consulta (en la PC mediante un soft) como para recuperación de datos en caso de pérdida.
La comunicación entre la Palm y la PC se hace mediante un programa sincronizador, y apoyando la Palm en una plataforma que se conecta al puerto serie de la PC. La transmisión de datos puede ser de las siguientes formas:

- Palm a PC: Lo que está en la Palm es lo nuevo, y pisa a lo que está en la PC
- PC a Palm: (lo opuesto)
- Sincronización: en este caso, el programa que sincroniza lleva un registro de qué cambios hubo tanto en la Palm como en la PC y actualiza a ambos con la última información. Esto funciona bien cuando se sincroniza con una sola PC, pero si a la misma Palm se la quiere sincronizar con más de una PC, habrá que tener ciertas precauciones para no perder información.

7) ¿Cómo es el lanzamiento de una aplicación Palm? ¿Qué son los launch codes?

Launch code: es lo que se le da a la aplicación cuando se la ejecuta (lanza).

Tipos de lanzamiento:

- Normal: la aplicación da entrada a su Stara app
- Initializing: aparecieron nuevos datos globales para la aplicación a traves de la sincronización.
- Notification: pueden o no interesar a las aplicaciones (sync, alarmas, resets)

8) ¿Cómo es la arquitectura de eventos y las categorías Pen Event, Key Event y UIEvent?

El Event Manager es el que maneja todos los tipos de eventos en la Palm. El GetEvent (en el Event Manager) es el que espera un evento sobre la parte de digitalización y ejecuta lo que corresponda dependiendo del evento. Si recibe más de un evento, decide cuál tiene mayor prioridad. PenEvent, KeyEvent y UIEvent son eventos generados por el lápiz, por una tecla presionada o mediante la interfaz de usuario, respectivamente.

9) ¿Qué son los Hooks o Hacks y que es un hackmaster?

Hooks o Hacks son programas que están por debajo de los eventos, los captura antes que el GetEvent y después genera el evento que corresponda.
El Hackmaster es el que maneja las secuencias de los hacks.

10) Describa las particularidades de la administración de memoria como File System.

Las particularidades se enumeran a continuación:

- El acceso se hace **sin buffers** (obvio pero hay que dejarlo claro)
- Los datos se dividen en registros dispersos en la memoria (como los de un file system en disco). Estos registros se llaman **chunks** (espacios de datos contigua) Una Data-Base es una lista de "**chunks**".
- Los frames de memoria se llaman "**heaps**".
- La alocación de una Data-Base es una lista de pares de mapeo (chunk, heap).
- La defragmentación consiste en mover chunks a heaps contiguos.

Notar la similaridad con los file-systems y la "naturalidad" del concepto de Virtual File System como "super-clase" de las distintas implementaciones de file-systems.

11) ¿Qué diferencia hay entre Databases y Resources?

Ambos sirven como soporte de información, y la diferencia principal radica en que el primero (pdb) es para datos ordenables y el segundo (prc) no lo es.

12) Describa las facilidades de expansion.

El Expansion Manager (con funcionamiento similar al autorun en los CDs) es el que detecta hardware de expansión, y le manda información Slot Driver que es el driver físico que provee las operaciones de bajo nivel.
Hay interfaces de expansión series (bit a bit) y de bloques. Se accede a través de un Virtual File System ya que las "tarjetas" de expansión tienen su propio file system y "responden" a un conjunto de primitivas más o menos standard (en general, son VFAT con una limitación en la cantidad de subdirectorios). Las aplicaciones (y los datos) **no se leen directmaente** (por performance y gasto de potencia) **sino por copy** (posiblemente parcial) & run. Hay un Launch-code especial para lanzar aplicaciones por Copy&Run. y otro para notificar inserción.

13) Describa las facilidades de intercambio de objetos.

Se utiliza el "planchado" o Flatening de objetos, que es como la "serialización" en Java, para poder manejar los objetos.
Utiliza el Typed Data Objects, que es el mismo estándar que usa MIME.
La aplicación se comunica con el Exchange Library para el intercambio de objetos.
Algunas Exchange Librarys:

- Infrarrojo (IrDA).
- Local Exchange.
- BlueTooth (para comunicarse con dispositivos celulares).
- Messaging System (ej TCP/IP).

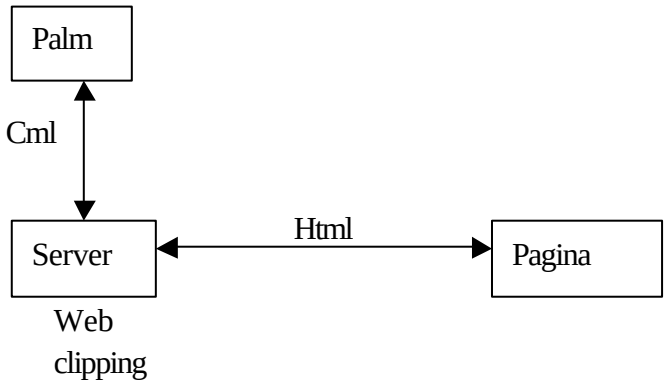
(para más información: http://www.imc.org/pdi).

14) Describa la arquitectura general de IrDA y de las Comunicaciones Series (probablemente deba tener conocimientos de redes para esto).

No es tomable, me interesaba que supieran que IrDA es un Standard y que **NO es el usado por Win KK**.

15) ¿Qué es y como funciona el web clipping?

Para poder adaptar las páginas web creadas para pc de escritorio a su visualización en palm se le deben hacer modificaciones para poder soportar su interfaz grafica y además achicar el tamaño de la misma para ser enviado en forma inalámbrica (por su alto costo).
Esto se hace con un programa en el servidor que toma las paginas web y les aplica un "filtro" convirtiéndolas en paginas CML (Compressed Markup Languaje).



16) ¿Qué es un *conduit*?

Un Conduit es una DLL de Windows que proporciona un puente entre el uso de la PALM y el de una PC de manera conjunta. Es responsable de los datos de la aplicación durante una sincronización entre la PALM y la computadora personal.

WINDOWS NT / 2000

1) Indique en que *feature* (característica) de NT aparecen visibles los siguientes criterios de diseño: extensibilidad, portabilidad, privacidad, compatibilidad (según Microsoft).

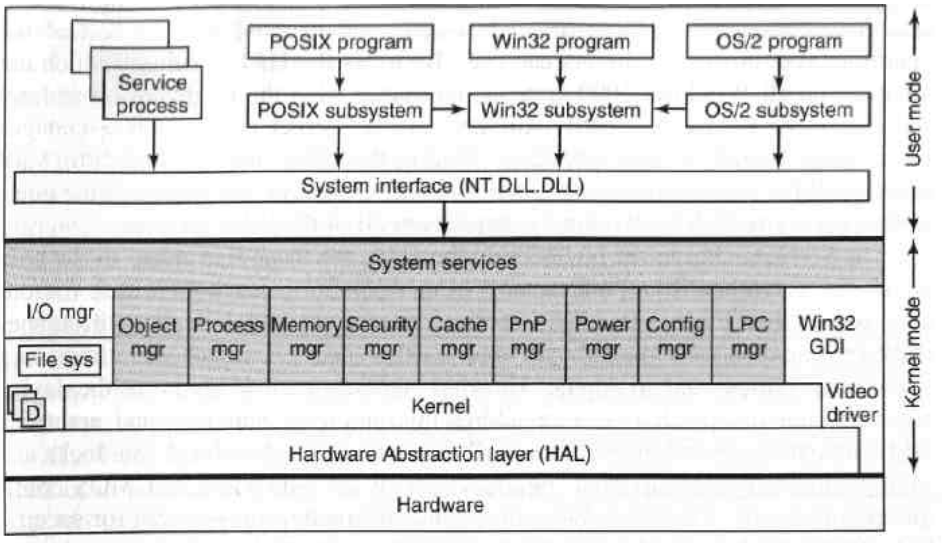
Extensibilidad: permite adaptarse a desarrollos de hardware y a la creación de nuevos dispositivos. NT (original) lograba la extensibilidad por medio del concepto de subsistemas que permitiría agregar nuevos ambientes de ser necesario y de los "drivers" cargables en memoria por medio del conocido mecanismo del Config.sys.

Portabilidad: permite migrar de plataforma. El concepto de HAL (*Hardware Abstraction Layer*) es el que libera al hard de las diferentes arquitecturas (en realidad del "chipset" ya que la diferencia del procesador viene dada por la compilación del Kernel para diferentes plataformas). Al quedar NT solo para plataformas Intel (no soporta mas Alpha ni Power-PC) la portabilidad queda restringida a Intel 32 (IA32) o Intel 64 (IA64, si es que esto existe).

Privacidad: cumple con los requerimientos de seguridad del Libro Naranja a un nivel C2.

Compatibilidad: Las aplicaciones de Win16 que no acceden al hard en forma directa se soportan. No debería haber aplicaciones Win32 que lo hicieran para permitir la unión de las plataformas Win kk con las NT para el 2000. Pero la historia dijo otra cosa. El intento de soportar Subsistemas OS/2 y Posix es también otro ejemplo de como Microsoft uso estos objetivos de diseño como propaganda para "vender" NT a los técnicos que no le creyeron y que no se equivocaron. Una vez impuesto el marketing, son solo recuerdos y papeles en el escritorio de algún Juez (demócrata).

2) Indique en un diagrama la estructura de la Arquitectura de NT y las funciones generales (una línea) de cada elemento.



The executive:
Se llama así a el conjunto de funciones integradas por los “manager”, el “system services” y el “system interface”.
Object Manager:
Maneja todos los objetos conocidos en el sistema, esto incluye los procesos, threads, archivos, directorios, semáforos, dispositivos de i/o, timers, etc.
Process Manager:
Es el encargado de manejar los procesos y los threads, incluyendo su creación y su destrucción.
Memory Manager:
Es el encargado de administrar la demanda y paginado de la memoria virtual.

Security Manager:
Hace cumplir el mecanismo elaborado de la seguridad de Windows 2000, que resuelve los requisitos de U.S. Dept. of Defense´s Orange Book C2.
Cache Manager:
Mantiene los bloques de disco más recientemente usados en memoria para acelerar el acceso a ellos en caso que se necesiten otra vez .
PnP Manager:
Se encarga de todos los dispositivos plug and play y de notificar sus acoplamientos al sistema.
Power Manager:
Administra el uso de la energía. Esto consiste en apagar el monitor y los discos después de que hayan estado inactivos por un cierto tiempo.
Config Manager:
Está a cargo de la administración del ”registry”. Agrega nuevas entradas y devuelve las claves cuando se hace su pedido.
LPC Manager:
El LPC (*Local Procedure Call*) prevé una comunicación entre procesos altamente eficiente usada entre los procesos y sus subsistemas.

3) ¿Qué se almacena y como es la estructura del “registry”?

Windows 2000 necesita guardar una gran cantidad de información, que es crucial para su funcionamiento, que tiene que ver con el hardware, software y los usuarios. A partir de Windows 95 se decidió juntar todos los archivos .ini (que era donde estaba guardada toda esta información) en una base de datos central llamada “Registry”.
La estructura del “registry”:

Figura Pág. 776

4) ¿Qué es un Sistema Operativo de Microkernel? ¿Cómo se implementa y cambia el concepto de NT?

Una tendencia en sistemas operativos modernos es tomar la idea de mover la mayor cantidad de código posible a capas más altas y quitar tanto como posible desde modo del “kernel”, dejando un “microkernel” mínimo.
El acercamiento a esto se hace implementando la mayor parte del sistema operativo en procesos del usuario. Para solicitar un servicio, tal como lectura de un bloque de un archivo, un proceso del usuario (ahora llamado “Proceso del Cliente”) envía la petición a un “Proceso del servidor”, que después hace el trabajo y envía la respuesta.
NT lo implementa en su primera versión (3.5) logrando un sistema muy estable e imposible de correr en aplicaciones comerciales. Con NT 4.0 abandona la implementación de Microkernel (aunque queda como figura en los libros y como "modelo de diseño" pero no de implementación) y vuelve a los sistemas Monolíticos. No hay un sistema comercial de microkernel (la palabra clave es “comercial”) en al actualidad. La "tendencia" está marcada por el mundo académico y, a pesar que los académicos estamos todos de acuerdo con esos principios, el divorcio con la realidad de lo que se usa comercialmente es para pensarlo. Recién cuando se monte algo así como "virtual PC" sobre un sistema con Microkernel se verá si es algo mas que una preciosista moda académica. A pesar del Ghz de reloj y de los poderosos RISCs y mainframes, sigue habiendo dos problemas:

- *Performance:* en monolítico se comparten datos de una forma que cualquier profesor de diseño calificaría como pésimo acople, que en Microkernel se transforman en burocráticos mensajes) y
- *$$$*: para lograr un Sistema Operativo estabilizado hay que pensar en al menos 5 años con equipos a full, y encima hay que venderlo.

5) ¿Cuál es el concepto de Objeto usado en NT/2000? ¿Cuáles son Control Objects y Cuales Dispatcher Objects?

NT no esta orientado a Objetos en la forma entendida por la Programación Orientada a Objetos.
Un objeto para Windows 2000 NT es una unidad de encapsulamiento y una unidad de Acceso Uniforme. Se clasifican en 2 tipos básicos, que son Control Objects y Dispatcher Objetcs. Los primeros incluye los objetos que modifican el flujo de control, los objetos de interrupción y los DPC (Deferred Procedure Call) y APC (Asynchronous Procedure Call). Los DPC se encargan del manejo de las tareas diferidas, por ejemplo al presionarse una tecla, el proceso de interrupción del teclado

lee el código de la tecla desde un registro y lanza la interrupción, esta interrupción puede ser menos importante que un proceso que se esté corriendo, con lo cual la ejecución de la misma se hace en un tiempo diferido.
Los APC son como los DPC excepto que se ejecutan en el contexto de un proceso especifico. Son objetos asincrónicos que no están en modo kernel.
Los Dispatcher Objetcs incluyen los semáforos, mutexes, eventos, timers y otros objetos en que los threads pueden esperar (Ej. Cola, pila). Estos objetos pueden causar el bloqueo de los procesos.

6) ¿Cómo es el mecanismo de ejecución usando subsistemas?

Figura Pág. 794

Los procesos de usuarios tienen llamadas a la WIN32 API a través de las dll. Estas dll pueden interactuar directamente con el sistema operativo o bien enviar un mensaje al subsistema de win32, que se encarga de hacer un cierto trabajo, para luego llamar al sistema operativo. En algunos casos puede pasar que el subsistema realice todo el trabajo en modo usuario sin pasar a la llamada al sistema operativo que esta modo kernel.

7) ¿Cuál es la relación entre LPC y la API de NT y WIN32?

LPC (Local Procedure Call) es la forma en que se comunican las distintas partes del Executive. Es una comunicación vía pasaje de mensajes en la que el llamador se bloquea hasta que el llamado tenga un resultado y la devuelva. Técnicamente se llama "pasaje sincrónico de mensajes". Las API encapsulan todo esto en una serie de funciones de biblioteca y "serializan" las llamadas, aumentando la estabilidad a costa de la performance. Las aplicaciones de Microsoft usan LPC pero la interface de LPC no esta documentada y Microsoft se reserva el derecho de cambiarla. Los desarrolladores deben usar las API. (Esto es la continuación de la política Undocumented Windows). En el escritorio de un juez están los pedidos del management de Microsoft para cambiar la arquitectura externa de las API de forma tal de perjudicar a ciertos programas de Groupware (¿Lotus Notes?) que el Juez decidió mantener en secreto por razones obvias (es lo mismo que decirle a todo el mundo que NO use esa aplicación ya que NO va a andar por diseño).

8) ¿Qué relación hay entre Jobs, Process, Threads y Fibers?

Primero veamos rápidamente que es cada una de estas:

Figura Pág.. 797

La relación que hay en Windows 2000 es que cada proceso contiene por lo menos un thread, el cual contiene por lo menos un Fiber. Además los procesos pueden agruparse en Jobs por ciertos propósitos de la administración de recursos.
Cambiar de threads en Windows 2000 es relativamente costoso ya que esto requiere entrar y salir del modo kernel. Para simular un paralelismo mas rápido, Windows 2000 provee los fibers, que son como threads, pero solo en modo usuario.

9) ¿Cómo esta organizada la memoria de un proceso?

En Windows 2000 cada proceso tiene su espacio de direcciones virtuales. Estas son un número de 32 bits, con lo cual cada proceso tiene 4 GB de memoria virtual.
Hasta los 2 GB de memoria están disponibles para el código y los datos de proceso. Los 2 GB hasta completar los 4 GB están protegidos para el kernel.
El espacio de dirección virtual es demanda por páginas, con un tamaño fijo de la página. (de 4KB en los Pentium). Un ejemplo:

Figura Pág. 813

10) ¿Cuáles son los estados de una página virtual?

Hay tres estados:

Free pages: esta sin uso y usarla causaría un “page fault”.
Committed: cuando los datos o código ya están en la pagina.
Reserved: significa que no se puede usar hasta que no se quite la reservación.

11) Describa el algoritmo de balanceo de paginas.

El algoritmo de balanceo de paginas utilizado por Windows 2000 funciona de la siguiente manera:
El sistema hace una tentativa de mantener un número substancial de páginas libres en memoria para que cuando ocurre una “page fault” poder demandar una página libre, sin la necesidad de escribir una página al disco.
Como consecuencia de esta estrategia, la mayoría de las “page fault” se pueden satisfacer con una sola operación del disco (lectura en la página), en vez de dos (escritura de una página cambiada y después una lectura de la página necesaria).
Por supuesto, las páginas en la lista disponible tienen que venir de alguna parte, así que el algoritmo depende de las paginas que tiene el proceso que pide una pagina.
Si el proceso tiene menos paginas que el mínimo establecido, el algoritmo le da una pagina libre al suceder el “page fault”, si el proceso esta entre el mínimo y el máximo, el algoritmo le da una pagina y le quita otra al suceder el “page fault”, y finalmente si el proceso esta por encima del máximo de paginas el algoritmo le saca 2 paginas y le da una al suceder el “page fault”.

12) Describa el proceso de Login de un usuario NT

Si Windows 2000 NT está configurado con conexión segura, esto significa que el administrador de sistema puede requerir a todos los usuarios tener una contraseña para abrirse una sesión. Si el usuario y contraseña ingresados son válidos, el sistema crea un proceso con un access token asociado. Entre las elementos que tiene el access token, esta el SID que es el identificador que representa al usuario en el sistema por razones de seguridad. Todos los procesos, que de aquí en mas lance este proceso inicial, heredaran el mismo access token.

13) ¿Cómo Funciona el Local Procedure Call?

Quien quiere un servicio conoce de antemano a quien enviarle un mensaje. Formatea (encapsula) el pedido en un formato standard donde indica entre otras cosas quien es (origen), a quien va dirigido (destino) el tipo de servicio (inmediato, diferido, diferible o urgente) y los datos referentes al servicio. El Kernel ubica al destino y le entrega el mensaje. De acuerdo con el tipo, el kernel pasa el llamado a bloqueado y le da control al scheduler. Lo que hace el proveedor del servicio depende del tipo de servicio, puede atenderlo de inmediato, diferirlo o atenderlo pero en modo interrumpible. Tarde o temprano el proveedor llega a un resultado, lo formatea en una cápsula similar a la que recibió y el Kernel lo entrega al que pidió. Hasta aquí es un Cliente Servidor con algunas peculiaridades. Esto implica copiar los pedidos del espacio del cliente al del Kernel, del Kernel al servidor y otras dos copias para la vuelta. Si el llamado se hace vía API, hay que sumarle las copias clásicas de los llamados a funciones. Para el caso de grandes cantidades de datos (como el video o el acceso a disco) estas copias son prohibitivas por el espacio y el tiempo que requieren. En la mayor parte de los LPC se pasan entonces punteros en vez de datos perdiéndose todas las ventajas del encapsulamiento y volviendo al sistema de la "gran área (bodoque) común de datos" que era denostado en la propaganda de NT 3.5. Por supuesto no hay documentación para que una aplicación de alguna otra parte que no sea de Redmont use LPC, o sepa bien qué hace cuando usa el área común de datos.

14) Describa los elementos que intervienen en la seguridad.

Elementos:

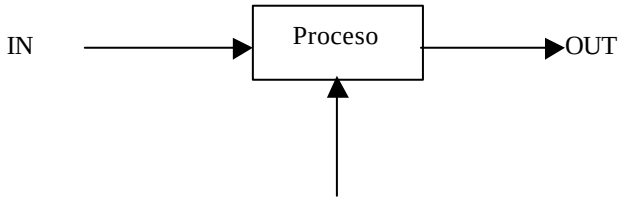
Todos los Usuarios y Grupos están identificados por un SID (Security ID). Estos son números binarios con un breve encabezamiento seguido de un gran componente al azar.

Cada proceso tiene un Access Token, que especifica su SID y otras propiedades. Tiene dos propósitos: uno es tener en un solo lugar toda la información necesaria para la seguridad para apresurar la validación de los accesos, y permite que cada proceso modifique sus características de seguridad de manera limitada sin afectar otros procesos del mismo usuario. Su estructura es: Figura Pág. 846.
La DACL por defecto (Discretionary Access Control List): determina qué usuarios y grupos pueden tener acceso a este objeto y para que operaciones
Security Descriptor.
SACL (System Access Control List): especifica qué clases de operaciones en el objeto deben generar mensajes que se registran en el log.

REAL TIME

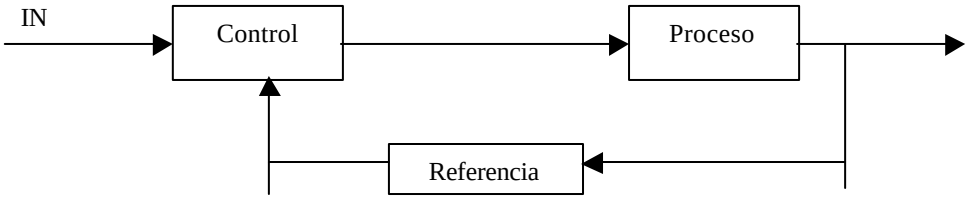
1) De ejemplos de los modelos generales de un Sistema de Control de Procesos. Con y sin realimentación.

Sin realimentación (lazo abierto):



Ejemplo: bajar la temperatura cuando calienta el termotanque. Hay un proceso que hace variar la salida de acuerdo a la entrada, pero en ningún momento se verifica si la salida es la deseada.

Con realimentación (lazo cerrado):



Ejemplo: combinar el agua fría y el agua caliente en una ducha de manera de obtener la temperatura correcta. Se va agregando agua caliente (o fría) y se va "testeando" la salida de la ducha mediante el tacto para comprobar si se tiene la salida requerida. El resultado del "test" se utiliza para saber si al instante siguiente se agregará mayor cantidad de agua caliente o de agua fría.

2) De ejemplos de un modelo de Control de Procesos en el que el ser humano intervenga en el lazo.

En el ejemplo anterior de la ducha interviene el ser humano.

3) De ejemplos de un modelo de Control de Procesos en el que la simulación y el ser humano intervengan en el lazo.

4) ¿Qué características presenta un Sistema de Tiempo Real? Explique la división entre *Hard, Soft* y *Firm Real Time* (RT).

Características de un sistema de RT:

- Es cualquier actividad de proceso de información que tiene que responder a estímulos generados externamente dentro de un plazo especificado y finito.
- El instante en que se produce el resultado del sistema es significativo.
- Debe ser lo suficientemente rápido para responder aquello para lo cual está diseñado.
- El tiempo interno es igual al tiempo externo.

Hard RT (Crítico): es inadmisible que se pierda algún plazo.
Soft RT (Acrítico): ocasionalmente puede perderse algún plazo.
Firm RT (Firme): el plazo no es crítico, pero una respuesta tardía no sirve.

5) ¿Qué es una *Task* desde el punto de vista de un Sistema de RT? ¿Cuáles son los atributos importantes?

En un sistema de RT cada estímulo del entorno activa una o más tareas. Una tarea es una secuencia de instrucciones que se ejecuta en concurrencia con otras tareas. La ejecución de las tareas se multiplexa en el tiempo en uno o más procesadores.

Existen 3 tipos de tareas:

- *Periódicas:* se repiten cada un tiempo específico y determinado.
- *Aperiódicas:* surgen en cualquier momento.
- *Esporádicas:* se sabe que deben aparecer en un intervalo de tiempo, pero no se sabe bien cuándo.

Los atributos importantes de una tarea periódica son: tiempo de ejecución (o de utilización del procesador), release time (tiempo en que es lanzada la tarea) y deadline (tiempo en que finaliza la tarea). Las tareas esporádicas son tratadas como periódicas en el peor caso, y para atender a las aperiódicas se pueden usar los ciclos de procesador que no tienen uso en el diagrama de tareas a través del tiempo, o sino se puede "crear" una tarea periódica cuya tarea sea atender a la aperiódica. Por lo tanto los atributos importantes de las tareas periódicas son importantes también para las demás tareas.

6) ¿Cuándo un proceso es controlado por tiempo y cuándo por eventos?

Existen tres tipos de tareas o procesos: periódicos, aperiódicos y esporádicos.

- *Periódicos:* generalmente muestran datos o forman parte de un bucle de control. Son controlados por tiempo.
- *Aperiódicos:* son controlados por eventos externos que se producen en forma aleatoria.
- *Esporádicos:* se activan en respuesta a eventos externos al sistema de RT.

7) Describa los pasos de un diseño de un Sistema de RT. Qué es *time constraining*?

Para empezar se necesita que el hardware y el software sean confiables y seguros.
Se deben conocer las facilidades de tiempo real, es decir, los tiempos en que las operaciones (tareas) deben ejecutarse y completarse.
Requisitos para un Sistema Operativo de Tiempo Real: multitarea con sincronización (ya que la concurrencia de procesos es muy importante en RT), cambio de contexto veloz, rápida respuesta a interrupciones, no debe usar memoria virtual, debe poseer un scheduler de RT, time out, alarmas, primitivas de demora, y debe ser previsible y adaptable al hard.
El scheduler del Sistema Operativo debe garantizar que una tarea vaya a realizarse, ya que esto es tan importante como realizarla.
Requisitos para un Lenguaje de Programación de Tiempo Real: debe poseer tipos fuertes (ya que de lo contrario es más probable el hecho de cometer errores), asignación dinámica de memoria, manejo de excepciones (para todo aquello que no es normal), multithreading, time constrains, soporte de distribución, etc.

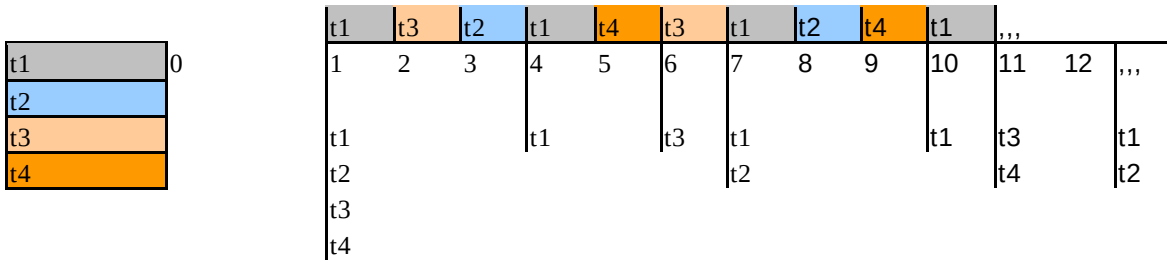
8) Cómo funciona el scheduling *Rate Monotonic*? Cuál es la prueba de *schedulability*?

Asigna a cada tarea a realizar una prioridad inversamente proporcional a su período, es decir, que las tareas de menor período tendrán mayor prioridad.

Prueba de schedulability:

Ejemplo 1:

4 tareas: t1 (3,1) - t2 (6,1) - t3 (5,1) - t4 (10,3) → (Período,Tiempo de Ejecución)
→ orden de prioridad: t1 > t3 > t2 > t4

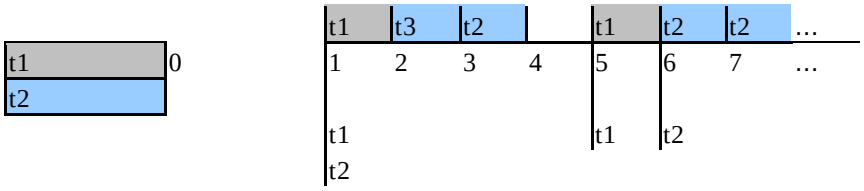


Nunca se llega a cumplir con la tarea número 4, que es la de menor prioridad y mayor duración.

Existe una manera de medir la carga de un procesador para un conjunto de tareas, y es mediante el factor de utilización del procesador: $U=\sum(e_i/T_i)$, siendo e_i el tiempo de ejecución de la tarea i y Ti el período de la misma. Para el Rate Monotonic, los plazos están garantizados para todas las tareas si se cumple que $U <= N*(2^{1/N}-1)$, siendo N la cantidad de tareas (esta condición es suficiente pero no necesaria). En el caso anterior se tiene que: U=1/3+1/6+1/5+3/10=1 >= 0,757, por eso mismo es que no se puede garantizar que todas las tareas se cumplan en los lapsos estipulados (ej: t4).

Ejemplo 2:

2 tareas: t1 (4,1) - t2 (5,2) → orden de prioridad: t1 > t2

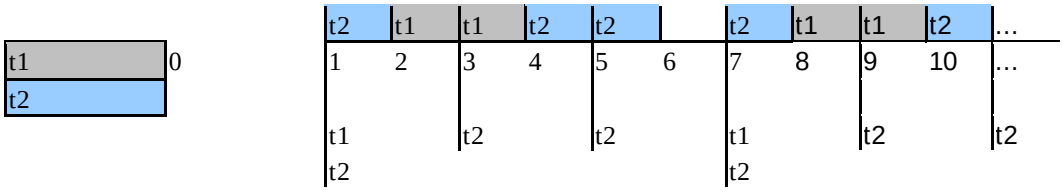


Aquí las tareas se cumplen correctamente en los plazos que se pide.
U=1/4+2/5=0,65 <= 0,828

9) Planifique por RM 2 tareas con Períodos 4 y 5 y tiempo de uso 1 y 2 respectivamente. Idem para Períodos 6 y 2 y utilización 1 y 1 respectivamente.

El primero es el ejemplo 2 del punto 9: t1 (4,1) - t2 (5,2)

Para el segundo: t1 (6,2) - t2 (2,1) → orden de prioridad: t2>t1
U=2/6+1/2=0,833 > 0,828 → no se garantizan los plazos.

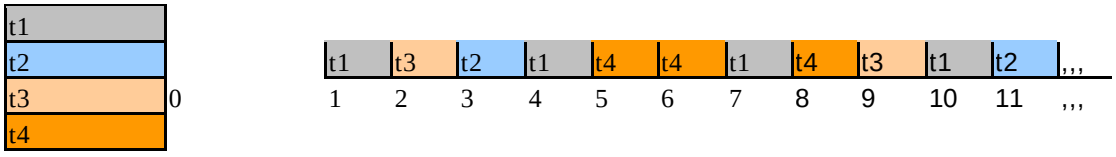


Como se puede apreciar, no se cumple la condición del factor de utilización del procesador, pero igualmente se garantizan las tareas en sus plazos específicos.

10) ¿Cómo funciona *Earliest Deadline First*? ¿En qué sentido es óptimo y cuál es su prueba de schedulability?

Asigna prioridades dinámicamente, se basa en asignar mayor prioridad a las tareas que tienen el plazo (deadline) más cercano siempre. EDF es óptimo, ya que cualquier secuencia planificada con otro algoritmo EDF también lo hace. Optimiza el uso del CPU.

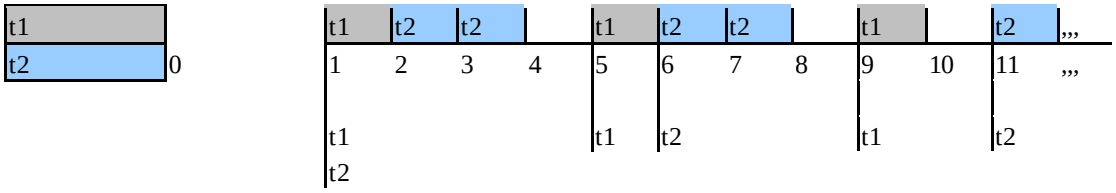
Prueba de Schedulability:
Ejemplo:
4 tareas: t1 (3,1) - t2 (6,1) - t3 (5,1) - t4 (10,3) → (Período,Tiempo de Ejecución)
→ orden de prioridad: t1 > t3 > t2 > t4



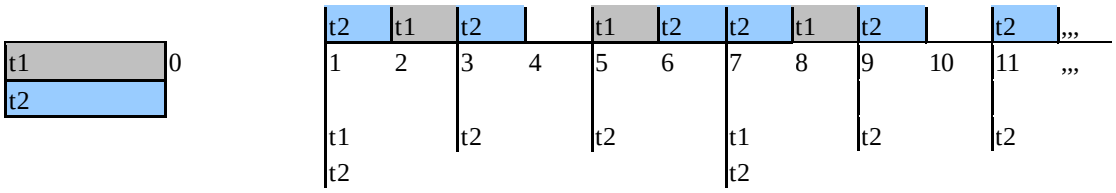
Como puede verse las tareas se garantizan.

11) Planifique por EDF las tareas de 9.

Para el primero:
t1 (4,1) - t2 (5,2)



Para el segundo:
t1 (6,1) - t2 (2,1)



Se garantizan las tareas para ambos casos.

12) De un ejemplo de tareas que RM no planifique y si EDF.

El ejemplo número 1 del punto 9 (y el ejemplo del punto 10) es una tarea que EDF planifica y RM no.

13) ¿Cómo se tratan las tareas esporádicas?

Las tareas esporádicas son tareas que se sabe que van a aparecer en un intervalo de tiempo determinado, pero no se sabe específicamente en qué momento. Estas tareas se tratan igual que las tareas periódicas en el peor caso.