Resumen Introducción a los Sistemas Operativos

Sistema Operativo

Un sistema operativo es un software que actúa como intermediario entre el usuario, una computadora y su hardware

Es software

- Necesita un procesador y memora para ejecutarse

Cosas que hace:

- Gestiona el HW
- Controla la ejecución de los procesos
- Interfaz entre aplicaciones y HW
- Actúa como intermediario entre un usuario de una computadora y el HW de la misma

Dos perspectivas:

- Desde el usuario
- Desde el sistema

Perspectiva del usuario:

- Abstracción con respecto a la arquitectura
    - Arquitectura: conjunto de instrucciones, organización de memoria, E/S,, estructura del bus
- El S.O “oculta” el HW y presenta a los programas abstracciones más simples de manejar
- Los programas de aplicación son los “clientes” del S.O

- Comparación: uso de escritorio y uso de comando de texto

Perspectiva desde desde la administración de recursos:

- Administra los recursos de HW de uno o mas procesos
- Provee un conjunto de servicios a los usuarios del sistema
- Maneja la memoria secundaria y dispositivos de I/O(input/output-entrada/salida)
- Ejecución simultánea de procesos
- Multiplexación de tiempo(CPU) y espacio(memoria)

Objetivos del S.O:

- Comodidad
  - Hacer más fácil el uso del hardware
- Eficiencia
  - Hacer un uso más eficiente del sistema
- Evolución
  - Permitir la introducción de nuevas funciones al sistema sin interferir con funciones anteriores

Componentes e un S.O:

- Kernel
- Shell
- Herramientas
  - Editores, compiladores, librerías, etc.

Kernel (Núcleo):

- “Porción de código”
  - Que se encuentra en memoria principal
  - Que se encarga de la administración de los recursos
- Implementa servicios:
  - Manejo de memoria

    - Manejo de CPU
    - Administración de procesos
    - Comunicación y concurrencia
    - Gestión de E/S

Servicios del S.O:

- Administración y planificación del procesador
    - Multiplexación de la carga de trabajo
    - Imparcialidad, “justicia” en la ejecución
    - Que no haya bloqueos
    - Manejo de prioridades
- Administración de memoria
    - Administración de memoria eficientemente
    - Memoria física vs memoria virtual. Jerarquías de memora
    - Protección de programas que compiten o se ejecutan concurrentemente
- Administración del almacenamiento-Sistema de archivos
    - Acceso de medios de almacenamiento externos
- Administración de dispositivos
    - Ocultamiento de dependencias de HW
    - Administración de accesos simultáneos
- Detección de errores y respuestas
    - Errores de HW internos y Externos
        - Errores de memoria/cpu
        - Errores de dispositivos
    - Errores de SW
        - Errores Aritméticos
        - Acceso no permitido a direcciones de memoria
    - Incapacidad del S.O para conceder una solicitud de una aplicación

- Interacción del Usuario(shell)
- Contabilidad
    - Recoger estadísticas del uso
    - Monitorear parámetros de rendimiento
    - Anticipar necesidades de mejoras futuras
    - Dar elementos si es necesario facturar tiempo de procesamiento

Complejidad

- Un S.O es un software extenso y complejo
- Es desarrollado por partes
- Cada una de estas partes deben ser analizadas y desarrolladas entendiendo su función, cuáles son sus entradas y sus salidas

Funciones principales de un S.O

- Brindar abstracciones de alto nivel a los procesos de usuario
- Administrar eficientemente el uso de:
    - La CPU
    - La memoria
    - Otros dispositivos
- Brindar asistencia para la realización de Entrada-salida por parte de los procesos

Problemas que un S.O debe evitar

- Que un proceso se apropie de la CPU
- Que un proceso intente ejecutar instrucciones “importantes”
    - E/s - flags del procesador
- Que un proceso intente acceder a una posición de memoria fuera de su espacio declarado

    - Proteger los espacios de direcciones
        - Gestionando el uso de la cpu
        - Detectando intentos de ejecución de instrucciones de E/S ilegales
        - Detectar accesos ilegales a memoria
        - Proteger el vector de interrupciones

Apoyo del hardware

- Modos de ejecución: Define limitaciones en el conjunto de instrucciones que se puede ejecutar en cada modo
- Interrupción de Clock: Se debe evitar que un proceso se apropie de la CPU
- Protección de la Memoria: Se deben definir límites de memoria a los que puede acceder cada proceso(registros base y límite)

Modos de ejecución:

- El bit en la CPU indica el modo actual
- Las instrucciones en modo Supervisor o Kernel
    - Necesitan acceder a estructuras del Kernel, o ejecutar código que no es del proceso
- En modo Usuario, el proceso puede acceder sólo a su espacio de direcciones, es decir a las direcciones “propias”
- El Kernel del S.O se ejecuta en modo supervisor
- El resto del S.O y los programas de usuario se ejecutan en modo usuario

Tener en cuenta que...

- Cuando se arranque el sistema, arranca con el bit en modo supervisor
- Cada vez que comienza a ejecutarse un proceso de usuario, este bit se DEBE PONER en modo usuario
    - Mediante una instrucción especial

- Cuando hay un trap o una interrupción, el bit de modo se pone en modo Kernel
    - Única forma de pasar a modo Kernel
    - No es el proceso de usuario quien hace el cambio explícitamente

Cómo actúa...

- Cuando el proceso de usuario intenta por sí mismo ejecutar instrucciones que pueden causar problemas (llamadas instructivas privilegiadas), el HW lo detecta como una operación ilegal y produce un trap al S.O

En windows...

- En WIN2000 el modo núcleo ejecuta los servicios ejecutivos. El modo usuario ejecuta los procesos de usuario
- Cuando un programa se bloquea en modo usuario, a lo sumo se describe un suceso en el registro de sucesos. Si el bloqueo se produce estando en modo supervisor se genera la BSOD(pantalla azul)

Resumiendo...

- Modo Kernel:
    - Gestión de procesos: Creación y terminación, planificación, intercambio, sincronización y soporte para la comunicación entre procesos
    - Gestión de memoria: Reserva de espacio de direcciones para los procesos, Swapping, gestión y páginas de segmentos.
    - Gestión E/S: gestión de buffers, reserva de canales de E/S y de dispositivos de los procesos
    - Funciones de soporte: Gestión de interrupciones, auditoría, monitoreo
- Modo usuario
    - Debug de procesos, definición de protocolos de comunicación gestión de aplicaciones(compilador, editor, aplicaciones de usuario)

    - En este modo se llevan a cabo todas las tareas que no requieran accesos privilegiados
    - En este modo no se puede interactuar con el hardware
    - El proceso trabaja en su propio espacio de direcciones

Protección de la memoria

- Delimitar el espacio de direcciones del proceso
- Poner limites a las direcciones que puede utilizar un proceso

La memoria principal aloja al S.O y a los procesos de usuario.

- El kernel debe proteger para que los procesos de usuario no puedan acceder donde no les corresponde
- El kernel debe proteger el espacio de direcciones de un proceso del acceso de otros procesos

Protección de la E/S

- Las instrucciones de E/S se definen como privilegiadas
- Deben ejecutarse en Modo Kernel
    - Se deberían gestionar en el kernel del sistema operativo
    - Los procesos de usuario realizan E/S a través de llamadas al SO(es un servicio del SO)

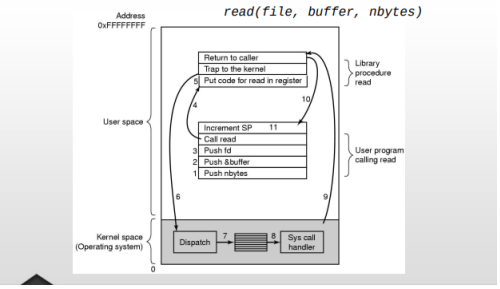
Protección de la CPU

- Uso de interrupción por clock para evitar que un proceso se apropie de la CPU
- Se implementa normalmente a través de un clock y un contador

- El kernel le da valor al contador que se decrementa con cada tick de reloj y al llegar a cero puede expulsar al proceso para ejecutar otro
- Las instrucciones que modifican el funcionamiento del reloj son privilegiadas
- Se le asigna al contador el valor que se quiere que se ejecute un proceso
- Se la usa también para el cálculo de la hora actual, basándose en cantidad de interrupciones ocurridas cada tanto tiempo y desde una fecha y hora determinada

System calls

- Es la forma en que los programas de usuario acceden a los servicios del S.O
- Los parámetros asociados a las llamadas pueden pasarse de varias maneras: por registros, bloques o tablas de memoria o la pila
- Se ejecutan en modo Kernel o superviso



- Categorías de system calls:
  - Control de procesos
  - Manejo de archivos
  - Manejo de dispositivos
  - Mantenimiento de información del sistema
  - Comunicaciones

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

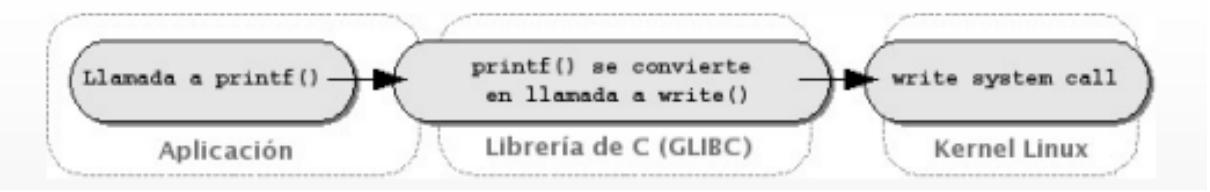
Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970



- Para iniciar la system call se indica:
  - Numero de syscall que se quiere ejecutar
  - los parametros de la misma
- Luego se emite un aviso al SO para pasar a modo kernel y gestionar la system call
- Se evalúa al system call deseada y se ejecuta

Ejemplos:

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Procesos

Definición de proceso

- Es un programa en ejecución
- Para nosotros serán sinónimos: tarea, job y proceso

Diferencias entre un programa y un proceso

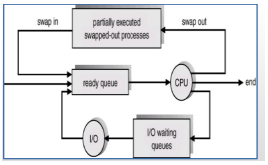
Programa

- Estático
- No tiene program counter

Proceso

- Dinámico
- Tiene program counter

- Existe desde que se edita hasta que se borra



- Su ciclo de vida comprende desde que se solicita ejecutar hasta que termina

Componentes de un proceso

Proceso: Entidad de abstracción

Un proceso(para poder ejecutarse) incluye como mínimo:

- Sección de código(texto)
- Sección de Datos(variables globales)
- Stack(s)(datos temporarios: parámetros, variables temporales y direcciones de retorno)

Stacks

- Un proceso cuenta con 1 o más stacks
  - En general: modo usuario y modo kernel
- Se crean automáticamente y su medida se ajusta en run-time
- Está formado por stack frames que son pushed(al llamar a una rutina) y popped(cuando se retorna de ella)
- El stack frame tiene los parámetros de la rutina(variables locales), y los datos necesarios para recuperar el stack frame anterior(el contador de programa y el valor del stack pointer en el momento del llamado)

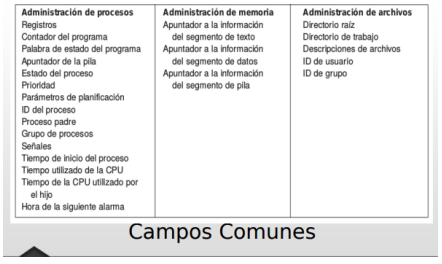
Atributos de un proceso

- Identificación del proceso, y del proceso padre
- Identificación del usuario que lo “disparó”

- Si hay estructura de grupos, grupo que lo disparó
- En ambientes multiusuario, desde que terminal y quien lo ejecuto

Process Control Block(pcb)

- Estructura de datos asociada al proceso(abstracción)
- Existe una por procesos
- Es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando termina
- Contiene la información asociada con cada proceso:
  - PID, PPID, etc
  - Valores de los registros de la CPU(PC, AC, etc)
  - Planificación(estado, prioridad, tiempo consumido, etc)
  - Ubicación(representación)
  - Accounting
  - Entrada salida(estado, pendientes, etc)



¿Qué es el espacio de direcciones de un proceso?

- Es el conjunto de direcciones de memoria que ocupa el proceso
  - Stack, text y datos

- No incluye su PCB o tablas asociadas
- Un proceso en modo usuario puede acceder sólo a su espacio de direcciones;
- En modo Kernel, se puede acceder a estructuras internas, o a espacios de direcciones de otros procesos

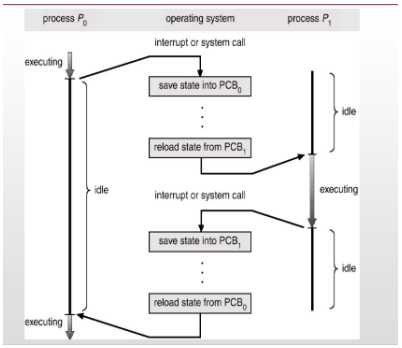
El contexto de un proceso

- Incluye toda la información que el SO necesita para administrar el proceso, y la CPU para ejecutarlo correctamente.
- Son parte del contexto, los registros de CPU, inclusive el contador de programa, prioridad del proceso, si tiene E/S pendientes, etc.

Cambio de contexto(Context Switch)

- Se produce cuando la CPU cambia de un proceso a otro
- Se debe resguardar el contexto del proceso saliente, que pasa a espera y retornará despues a la CPU
- Se debe cargar el contexto del nuevo proceso y comenzar desde la instrucción siguiente a la última ejecutada en dicho contexto
- Es tiempo no productivo de CPU
- El tiempo que consume depende del soporte de HW

Ejemplo:



Sobre el Kernel del Sistema Operativo

- Es un conjunto de módulos de software
- Se ejecuta en el procesador como cualquier proceso
- Entonces:
  - ¿El Kernel es un proceso?Y si es así¿Quién lo controla?
- Diferentes enfoques de diseño

Enfoque 1 - El Kernel como entidad independiente

- El Kernel se ejecuta fuera de todo proceso
- Es una arquitectura utilizada por los primeros SO
- Cuando un proceso es “interrumpido” o realiza una System Call, el contexto del proceso se salva y el control se pasa al Kernel del sistema operativo
- El Kernel tiene su propia región de memoria
- El Kernel tiene su propio Stack
- Finalizada su actividad, le devuelve el control al proceso (o a otro diferente)
- Importante:
  - El Kernel NO es un proceso. El concepto de proceso solo se asocia a programas de usuario
  - Se ejecuta como entidad independiente en modo privilegiado

Enfoque 2 - El Kernel “dentro” del proceso

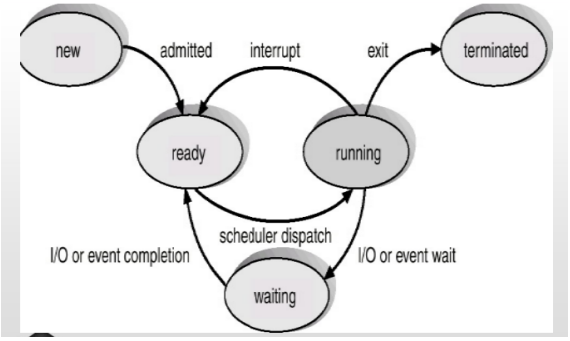
- El “código” del Kernel se encuentra dentro del espacio de direcciones de cada proceso

- El Kernel se ejecuta en el MISMO contexto que algún proceso de usuario
- El Kernel se puede ver como una colección de rutinas que el proceso utiliza
- Dentro de un proceso se encuentra el código del programa (user) y el código de los módulos de SW del SO
- Cada proceso tiene su propio stack (uno en modo usuario y otro en modo Kernel)
- El proceso es el que se Ejecuta en Modo Usuario y el Kernel del SO se ejecuta en Modo Kernel (Cambio de modo)
- El código del Kernel es compartido por todos los procesos
    - En administración de memoria veremos “como”
- Cada interrupción(incluyendo las System Call) es atendida en el contexto del proceso que se encontraba en ejecución
    - Pero en modo Kernel!!!(se pasa a este modo sin necesidad de hacer un cambio de contexto completo)
    - Si el SO determina que el proceso debe seguir ejecutándose luego de atender la interrupción, cambia a modo usuario y devuelve el control. Es más económico y performante

Estados de un proceso

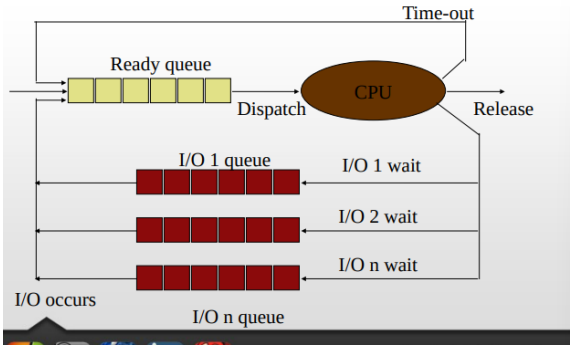
En su ciclo de vida, un proceso pasa por diferentes estados

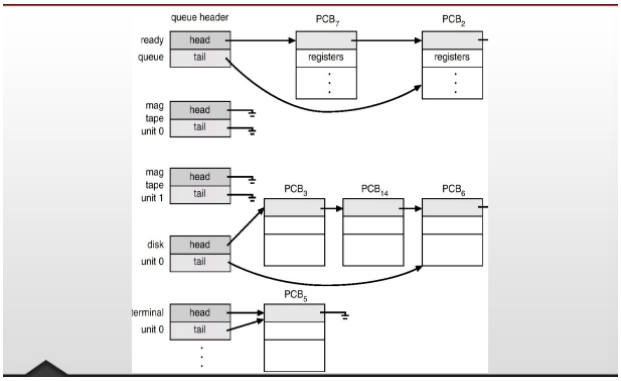
- Nuevo (new)
- Listo para ejecutar (ready)
- Ejecutándose (running)
- En espera (waiting)
- Terminado (terminated)



Colas en la planificación de procesos

- Para realizar la planificación, el SO utiliza la PCB de cada proceso como una abstracción del mismo
- Las PCB se enlazan en Colas siguiendo un orden determinado
- Ejemplos
  - Cola de trabajos o procesos
    - Contiene todas las PCB de procesos en el sistema
  - Cola de procesos listos
    - PCB de procesos residentes en memoria principal esperando para ejecutarse





Módulos de planificación

- Son módulos (SW) del Kernel que realizan distintas tareas asociadas a la planificación
- Se ejecutan ante determinados eventos que así lo requieren:
  - Creación/Terminación de procesos
  - Eventos de Sincronización o de E/S
  - Finalización de lapso de tiempo
  - Etc
- Scheduler (planificador) de long term
- Scheduler de short term
- Scheduler de medium term

  Su nombre proviene de la frecuencia de ejecución

  Existen otros módulos: Dispatcher y Loader

  Pueden no existir como módulos separados de los schedulers vistos, pero la función debe cumplirse

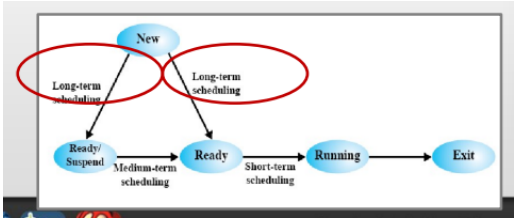
  Dispatcher: hace cambio de contexto, cambio de modo de ejecución...”despacha” el proceso elegido por el Short Term (es decir, salta a la instrucción a ejecutar)

  Loader: Carga en memoria el proceso elegido por el Long term

Long term Scheduler

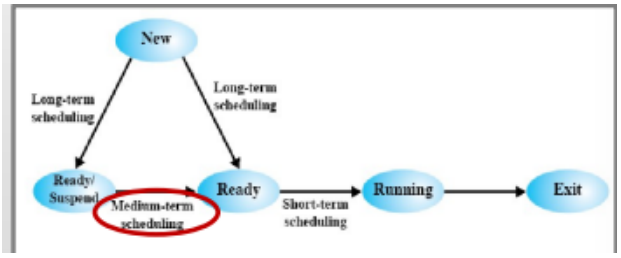
Controla el grado de multiprogramación, es decir la cantidad de procesos en memoria

Puede no existir este scheduler y absorber esta tarea del short term



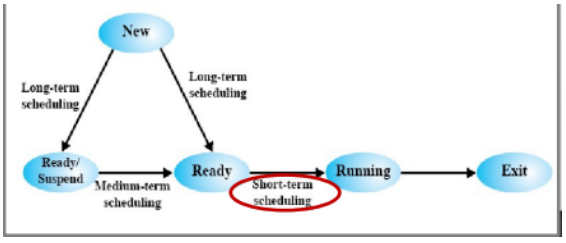
Medium Term Scheduler (swapping)

- Si es necesario, reduce el grado de multiprogramación
- Saca temporalmente de memoria los procesos que sea necesario para mantener el equilibrio del sistema
- Términos asociados: swap out (sacar de memoria), swap in (Volver a memoria)

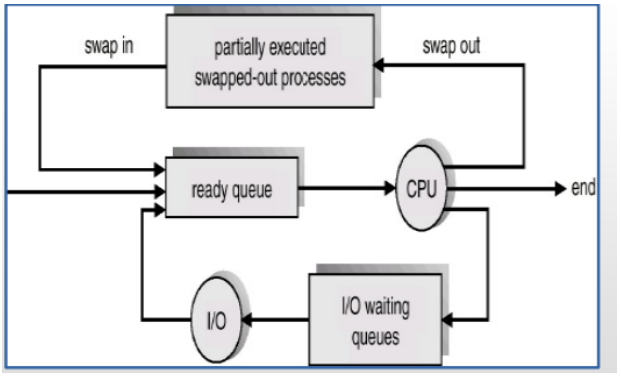


Short Term Scheduler

- Decide a cuál de los procesos en la cola de listos se elige para que use la CPU
- Términos asociados: apropiativo, no apropiativo, algoritmo de scheduling



Procesos en espera y swapeados



Sobre el estado nuevo (new)

- Un usuario “dispara” el proceso. Un proceso es creado por otro proceso: su proceso padre
- En este estado se crean las estructuras asociadas, y el proceso queda en la cola de procesos, normalmente en espera de ser cargado en memoria

Sobre el estado listo (ready)

- Luego que el scheduller de largo plazo eligió al proceso para cargarlo en memoria, el proceso queda en estado listo
- El proceso sólo necesita que se le asigne CPU
- Está en la cola de procesos listos (ready queue)

Sobre el estado en ejecución (running)

- El sheduler de corto plazo lo eligió para asignarle CPU
- Tendrá la CPU hasta que el período de tiempo asignado (quantum o time slice), termine o hasta que necesite realizar alguna operación de E/S

Sobre el estado de espera (waiting)

- El proceso necesita que se cumpla el evento esperado para continuar
- El evento puede ser la terminación de una E/S solicitada, o la llegada de una señal por parte de otro proceso
- Sigue en memoria, pero no tiene la CPU
- Al cumplirse el evento, pasará al estado listo

Transiciones

New-Ready: Por elección del scheduler de largo plazo (carga en memoria)

Ready-Running: Por elección del scheduler de corto plazo (asignación de CPU)

Running-Waiting: el proceso “se pone a dormir”, esperando por un evento.

Waiting-Ready: Terminó la espera y compite nuevamente por la CPU.

Caso especial: Running-ready

- Cuando el proceso termina su quantum (franja de tiempo) sin haber necesitado ser interrumpido por un evento, pasa al estado de ready, para competir por CPU, pues no está esperando por ningún evento...
- Se trata de un caso distinto a los anteriores, porque el proceso es expulsado de la CPU contra su voluntad

- Esta situación se da en algoritmos apropiativos

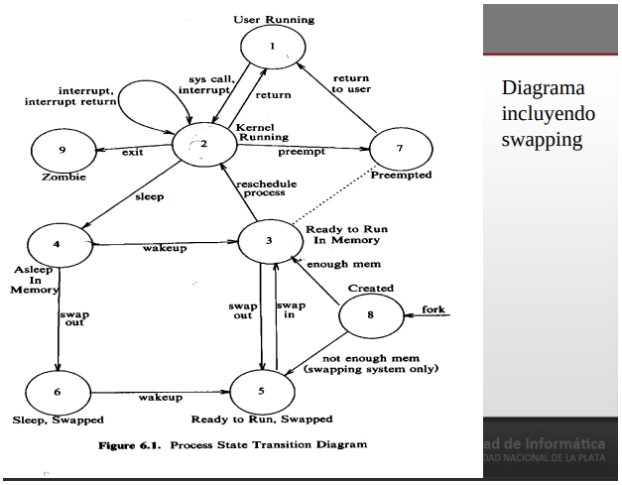
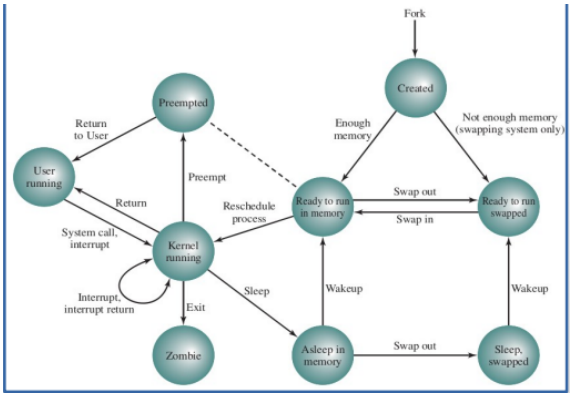


Diagrama incluyendo swapping

Explicación por estado

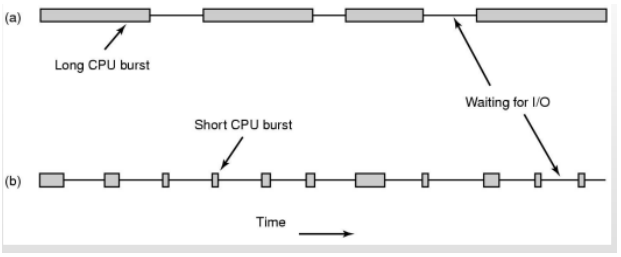
1. Ejecución en modo usuario
2. Ejecución en modo Kernel
3. El proceso está listo para ser ejecutado cuando sea elegido
4. Proceso en espera en memoria principal
5. Proceso listo, pero el swaper debe llevar al proceso a memoria principal antes que el kernel lo pueda elegir para ejecutar
6. Proceso en espera en memoria secundaria
7. Proceso retornando desde el modo Kernel al user. Pero el kernel se apropia, hace un context switch para darle a la cpu a otro proceso
8. Proceso recientemente creado en transición: existe, pero aun no está listo para ejecutar, ni está dormido
9. Proceso ejecutó la system call exit y está en estado zombie. Ya no existe más, pero se registran datos sobre su uso, código resultante del exit. Es el estado final

Diagrama de trancisiones UNIX



Comportamiento de los procesos

Procesos alternan ráfagas de CPU y de I/O



- CPU-bound
  - Mayor parte del tiempo utilizando la CPU
- I/O bound
  - Mayor parte del tiempo esperando por I/O
- La velocidad de la CPU es mucho más rápida que la de los dispositivos de E/S
  - Pensar: Necesidad de atender rápidamente procesos I/O-bound para mantener el dispositivo ocupado y aprovechar la CPU para los procesos CPU-bound

Planificación

- Planificación
    - Necesidad de determinar cual de todos los procesos que están listos para ejecutarse, se ejecutará a continuación en un ambiente multiprogramado
- Algoritmo de planificación
    - Algoritmo utilizado para realizar la planificación del sistema

Algoritmos apropiativos y No apropiativos

- En los algoritmos apropiativos existen situaciones que hacen que el proceso en ejecución sea expulsado de la CPU
- En los algoritmos no apropiativos los procesos se ejecutan hasta que el mismo (por si propia cuenta) abandone la CPU
    - Se bloquea por E/S o finaliza
    - No hay decisiones de planificación durante las interrupciones de reloj

Categorías de los Algoritmos de planificación

- Según el ambiente es posible requerir algoritmos de planificación diferentes, con diferentes metas:
    - Equidad: Otorgar una parte justa de la CPU a cada proceso
    - Balance: Mantener ocupadas todas las partes del sistema
- Ejemplos:
    - Procesos por lotes (Batch)
    - Procesos Interactivos
    - Procesos en Tiempo real

Procesos Batch

- No existen usuarios que esperen una respuesta en una terminal
- Se pueden utilizar algoritmos no apropiativos
- Metas propias de este tipo de algoritmos:
  - Rendimiento: Maximizar el número de trabajos por hora
  - Tiempo de retorno: Minimizar los tiempos entre el comienzo y la finalización
  - El tiempo en espera se puede ver afectado
  - Uso de la CPU: mantener la CPU ocupada la mayor cantidad de tiempo posible
- Ejemplos de Algoritmos
  - FCFS - First Come First Served
  - SJF - Shortest Job First

Procesos interactivos

- No solo interacción con los usuarios
  - Un servidor, necesita de varios procesos para dar respuesta a diferentes requerimientos
- Son necesarios algoritmos apropiativos para evitar que un proceso acapare la CPU
- Metas propias de este tipo de algoritmos:
  - Tiempo de respuesta: Responder a peticiones con rapidez
  - Proporcionalidad: Cumplir con expectativas de los usuarios

        - Si el usuario le pone STOP al reproductor de música, que la música deje de ser reproducida en un tiempo considerablemente corto
    - Ejemplos de algoritmos:
        - Round Robin
        - Prioridades
        - Colas multinivel
        - SRTF - Shortest remaining time first

Política Versus Mecanismo

- Existen situaciones en las que es necesario que la planificación de uno o varios procesos se comporte de manera diferente
- El algoritmo de planificación debe estar parametrizado, de esta manera los procesos/usuarios pueden indicar los parámetros para modificar la planificación
- El Kernel implementa el mecanismo
- El usuario/proceso/administrador utiliza los parámetros para determinar la Política
- Ejemplo:
    - Un algoritmo de planificación por prioridades y una System Call que permite modificar la prioridad de un procesos
    - Un proceso puede determinar las prioridades de los procesos que el crea, según la importancia de los mismos

Creación de procesos

- Un proceso es creado por otro proceso
- Un proceso padre tiene uno o más procesos hijos
- Se forma un árbol de procesos

Actividades en la creación

- Crear la PCB
- Asignar PID(process idenfication) único
- Asignarle memoria para regiones
    - Stack, text y datos
- Crear estructuras de datos asociadas
    - Fork (copiar el contexto, regiones de datos, text y stack)

Relación entre procesos Padre e hijo

Con respecto a la Ejecución:

- El padre puede continuar ejecutándose concurrentemente con su hijo
- El padre puede esperar a que el proceso hijo (o los procesos hijos) terminen para continuar la ejecución

Con respecto al espacio de direcciones:

- El hijo es un duplicado del proceso padre (caso Unix)
    - Se crea un nuevo espacio de direcciones copiando el del padre
- Se crea el proceso y se le carga adentro el programa (caso Windows)
    - Se crea un nuevo espacio de direcciones vacío

Creación de procesos(continuación)

- En UNIX: (2 System Calls)

- system call fork() crea un nuevo proceso igual al llamador
- system call excecve(), generalmente usada después del fork, carga un nuevo programa en el espacio de direcciones

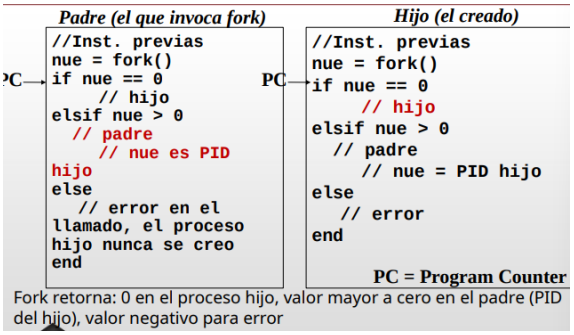
- En Windows:(1 system call)
  - system call CreateProcess() crea un nuevo proceso y carga el programa para la ejecución

¿Cómo funciona el Fork?

Padre

//Instrucciones previas

```
nue = fork()
if nue == 0
       // hijo
elsif nue > 0
       // padre
      // nue es el PID del hijo
else
      // error
end
```

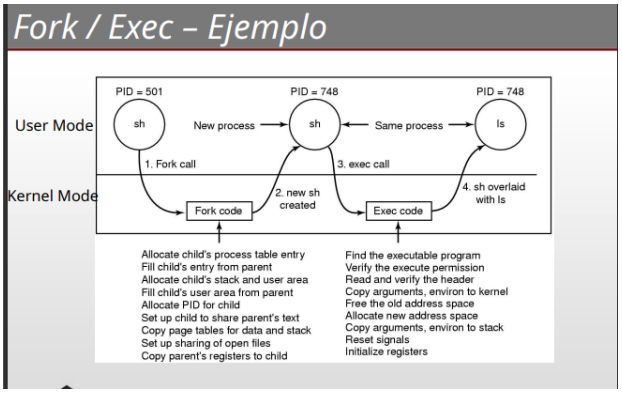


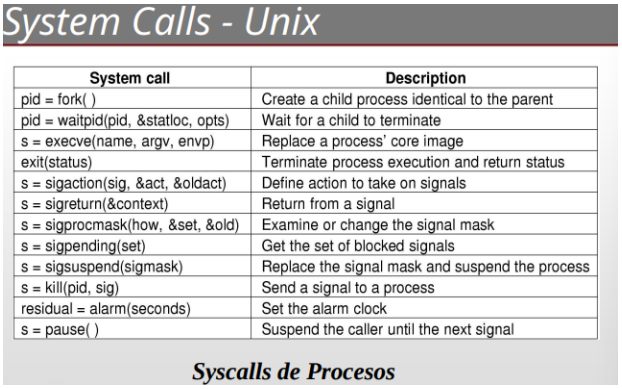
Fork retorna: 0 en el proceso hijo, valor mayor a cero en el padre (PID del hijo), valor negativo para error

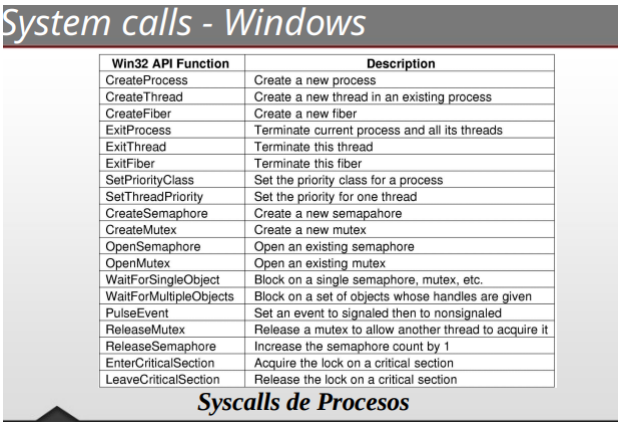
Terminación de procesos

- Ante un (exit), se retorna el control al sistema operativo
  - El proceso padre puede esperar recibir un código de retorno(via wait). Generalmente se lo usa cuando se requiere que el padre espere a los hijos
- Proceso padre puede terminar la ejecución de sus hijos(kill)

- La tarea asignada al hijo se terminó
- Cuando el padre termina su ejecución
  - Habitualmente no se permite a los hijos continuar, pero existe la opción
  - Terminación en cascada







Memoria

- La organización y administración de la “memoria principal” es uno de los factores mas importantes en el diseño de los S.O
- Los programas y datos deben estar en el almacenamiento principal para:
  - Poderlos ejecutar
  - Referenciarlos directamente
- El SO debe
  - Llevar un registro de las partes de memoria que se están utilizando y de aquellas que no
  - Asignar espacio en memoria principal a los procesos cuando estos la necesitan
  - Libera espacio de memoria asignada a procesos que han terminado
- Se espera de un S.O un uso eficiente de la memoria con el fin de alojar el mayor número de procesos
- El S.O. debe:
  - Lograr que el programador se abstraiga de la alocación de los programas
  - Brindar seguridad entre los procesos para que unos no accedan a secciones privadas de otros
  - Brindar la posibilidad de acceso compartido a determinadas secciones de la memoria (librerías, código en común, etc.)
  - Garantizar la performance del sistema

Administración de la memoria

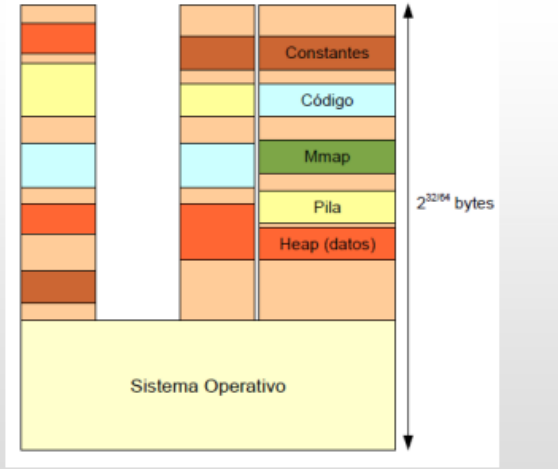
- División lógica de la memoria física para alojar múltiples procesos
    - Garantizando protección
    - Depende del mecanismo previsto por el HW
- Asignación eficiente
    - Contener el mayor número de procesos para garantizar el mayor uso de la CPU por los mismos

Requisitos

- Reubicación
    - El programador no debe ocuparse de conocer donde será colocado en la Memoria RAM
    - Mientras un proceso se ejecuta puede ser sacado y traído a la memoria (swap) y, posiblemente, colocarse en diferentes direcciones
    - Las referencias a la memoria deben “traducir” según ubicación actual del proceso
- Protección
    - Los procesos NO deben referenciar-acceder a direcciones de memoria de otros procesos
        - Salvo que tengan permiso
    - El chequeo se debe realizar durante la ejecución:
        - NO es posible anticipar todas las referencias a memoria que un proceso puede utilizar
- Compartición
    - Permitir que varios procesos accedan a la misma porción de memoria.
        - Ej: rutinas comunes, librerías, espacios explícitamente compartidos
    - Permite un mejor uso-aprovechamiento - de la memoria RAM, evitando copias innecesarias (repetidas) de instrucciones

Abstracción - Espacio de Direcciones

- Rango de direcciones (a memoria) posibles que un proceso puede utilizar para direccionar sus instrucciones y datos
- El tamaño depende de la Arquitectura del procesador
  - 32 bits: 0.. $(2^{(32)})$ -1
  - 64 bits: 0.. $(2^{(64)}–1)$
- Es independiente de la ubicación “real” del proceso en la Memoria RAM



Direcciones

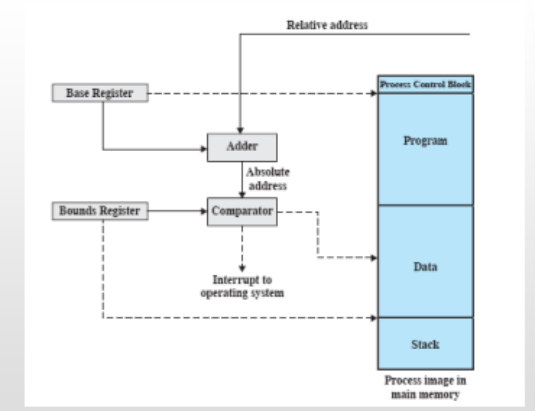
- Lógicas
  - Referencia a una localidad de memoria independiente de la asignación actual de los datos en la memoria
  - Representa una dirección en el “Espacio de Direcciones del proceso”
- Físicas
  - Referencia una localidad en la Memoria Física (RAM)
    - Dirección absoluta

  En caso de usar direcciones lógicas, es necesaria algún tipo de conversión a direcciones Físicas

Conversión de Direcciones

Una forma simple de hacer esto es utilizando registros auxiliares

- Registro Base
  - Dirección de comienzo del Espacio de Direcciones del proceso en la RAM
- Registro límite
  - Dirección final del proceso o medida del proceso - Tamaño de su Espacio de direcciones
- Ambos valores se fijan cuando el espacio de direcciones del proceso es cargado a memoria
- Varían entre procesos (Context Switch)



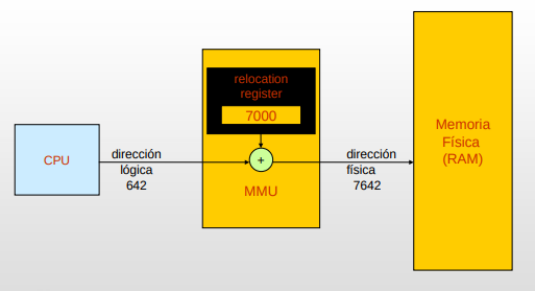
Dir. Lógicas vs. Físicas

- Si la CPU trabaja con direcciones lógicas, para acceder a memoria principal, se deben transformar en direcciones físicas
  - Resolución de direcciones (address-binding):
    - transformar la dirección lógica en la física correspondiente
- Resolución en momento de compilación (Archivos .com de DOS) y en tiempo de carga

    - Direcciones lógicas son idénticas
    - Para reubicar un proceso es necesario recompilarlo o recargarlo
- Resolución en tiempo de ejecución
    - Direcciones lógicas y físicas son diferentes
    - Direcciones lógicas son llamadas “Direcciones virtuales”
    - La reubicación se puede realizar fácilmente
    - El mapeo entre “Virtuales” y “Físicas” es realizado por hardware
        - Memory Management Unit (MMU)

Memory Management Unit (MMU)

- Dispositivo de Hardware que mapea direcciones virtuales a físicas
    - Es parte del Procesador
    - Re-programar el MMU es una operación privilegiada
        - solo puede ser realizada en Kernel Mode
- El valor en el registro “registro de realocación” es sumado a cada dirección generada por el proceso de usuario al momento de acceder a la memoria
    - Los procesos usan direcciones físicas



Mecanismo de asignación de memoria

- Particiones fijas: El primer esquema implementado
    - La memoria se divide en particiones o regiones de tamaño Fijo (Pueden ser todas del mismo tamaño o no)
    - Alojan un proceso en cada una
    - Cada proceso se coloca de acuerdo a algún criterio (Primer ajuste, Mejor ajuste, peor ajuste, Siguiente ajuste)
- Particiones dinámicas: La evolución del esquema anterior
    - Las particiones varían en tamaño y en número
    - Alojan un proceso en cada una
    - Cada partición se genera en forma dinámica del tamaño justo que necesita el proceso

Fragmentación

- La fragmentación se produce cuando una localidad de memoria no puede ser utilizada por no encontrarse de forma contigua
- **Fragmentación Interna:**
    - Se produce en el esquema de particiones fijas
    - Es la porción de la partición que queda sin utilizar
- **Fragmentación Externa:**
    - Se produce en el esquema de particiones dinámicas
    - Son huecos que van quedando en la memoria a medida que los procesos finalizan
    - Al no encontrarse en forma contigua puede darse el caso de que tengamos memoria libre para alocar un proceso, pero que no la podamos utilizar
    - Para solucionar el problema se puede acudir a la compactación, pero es muy costosa

Problemas del esquema

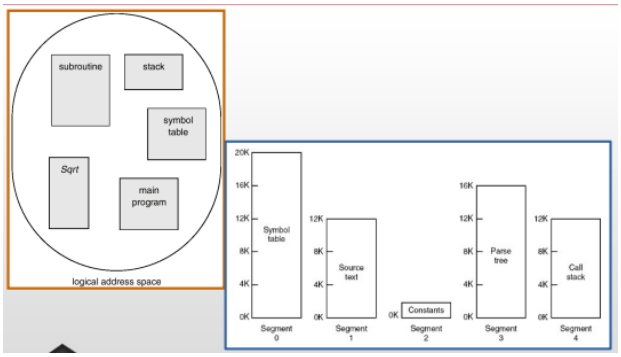
- El esquema de Registro Base + Límite presenta problemas:
  - Necesidad de almacenar el Espacio de Direcciones de forma continua en la memoria física
  - Los primeros SO definían particiones fijas de memoria, luego evolucionaron a particiones dinámicas
  - Fragmentación
  - Mantener “partes” del proceso que no son necesarias
  - Los esquemas de particiones fijas y dinámicas no se usan hoy en día
- Solución:
  - Segmentación
  - Paginación

Segmentación

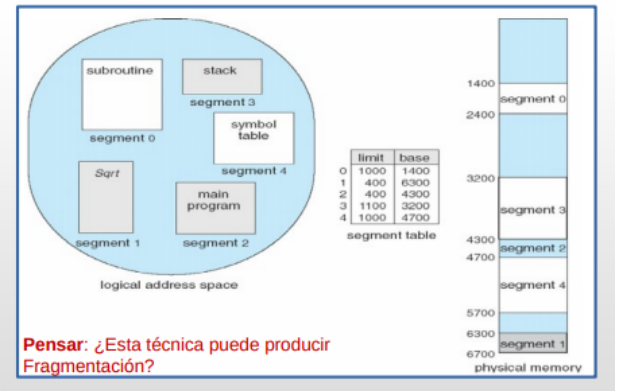
- Esquema que se asemeja a la “visión del usuario”. El programa se divide en partes/secciones
- Un programa es una colección de segmentos. Un segmento es una unidad lógica como:
  - Programa principal, procedimientos y funciones, variables locales y globales, stack, etc.

- Puede causar Fragmentación

Programa desde la visión del usuario



Ejemplo de segmentación



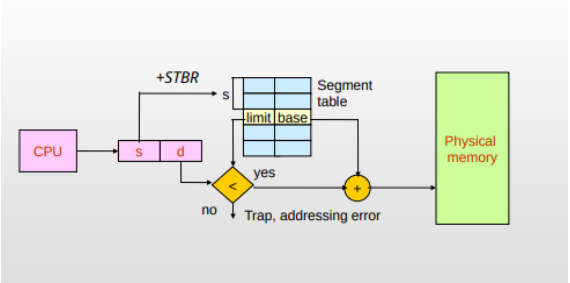
Continuación Segmentación

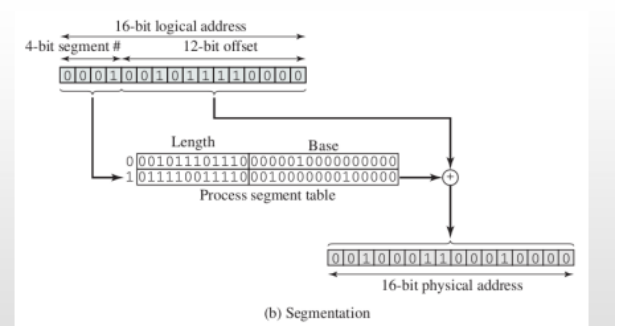
- Todos los segmentos de un programa pueden no tener el mismo tamaño (código, datos, rutinas)
- Las direcciones lógicas consisten en 2 partes:
  - Selección de segmento
  - Desplazamiento dentro del segmento

Arquitectura de la segmentación

- Tabla de segmentos

    - Permite mapear la dirección lógica en física. Cada entrada contiene:
        - Base: Dirección física de comienzo del segmento
        - Limit: Longitud del Segmento
- Segment-table base register (STBR): apunta a la ubicación de la tabla de segmentos
- Segment-table length register (STLR): cantidad de segmentos de un programa



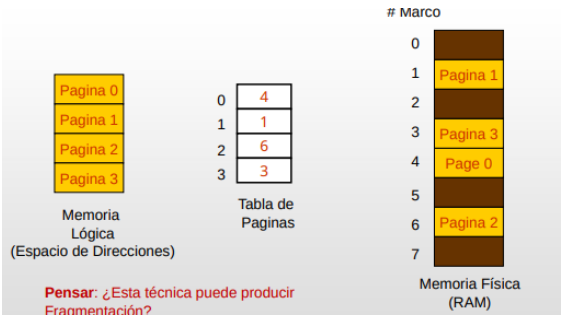


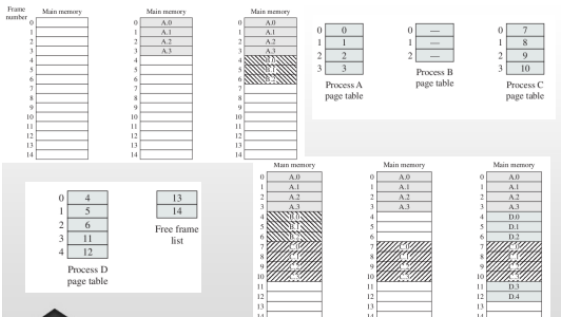
Paginación

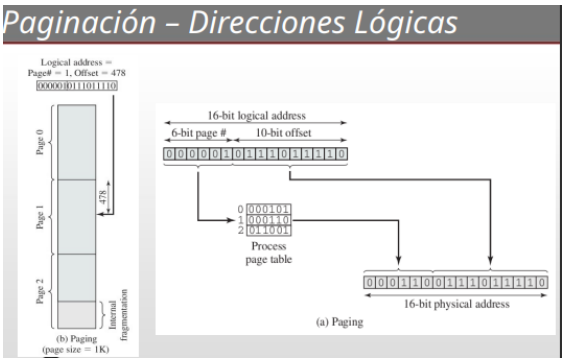
- Memoria física es dividida lógicamente en pequeños trozos de igual tamaño (Marcos)
- Memoria Lógica (espacio de direcciones) es dividido en trozos de igual tamaño que los marcos (Páginas)
- El SO debe mantener una tabla de páginas por cada proceso, donde cada entrada contiene (entre otras) el Marco en la que se coloca cada página
- La dirección lógica se interpreta como:

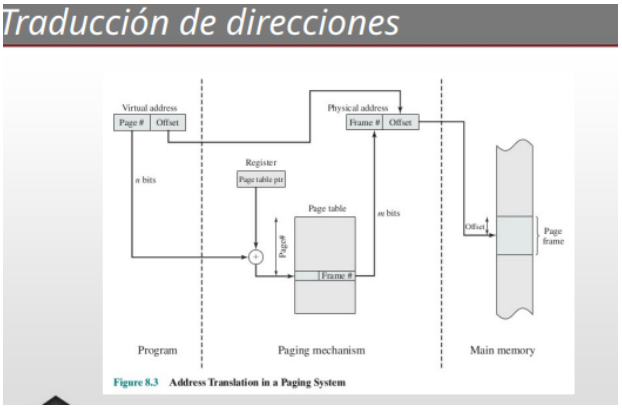
- Un número de página y un desplazamiento dentro del a misma

Ejemplos:









Ventajas sobre Segmentación

- Compartir
- Proteger

Segmentación paginada

- La paginación
  - Transparente al programador
  - Elimina Fragmentación externa
- Segmentación
  - Es visible al programador
  - Facilita modularidad, estructuras de datos grandes y da mejor soporte a la compartición y protección
- Segmentación paginada: Cada segmento es dividido en paginas de tamaño fijo

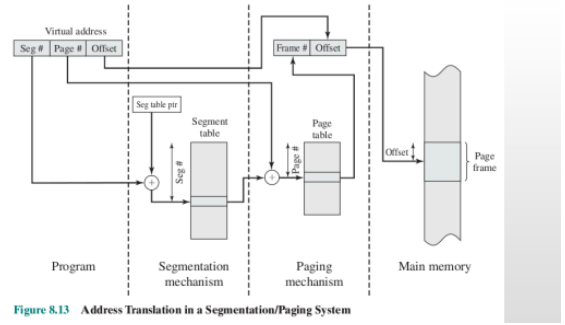


Figure 8.13 Address Translation in a Segmentation/Paging System