Resumen Parcial 1

LCC

2021

Índice

1. Introducción

Recopilación de definiciones de sistema operativo a lo largo del libro:

- El sistema operativo es el principal programa que se ejecuta en toda computadora de propósito general. Es el único programa que interactúa directamente con el hardware de la computadora.
- Un sistema operativo es más bien un gran programa, que ejecuta otros programas y les provee un conjunto de interfaces para que puedan aprovechar los recursos de cómputo.

1.1. Por qué estudiar sistemas operativos?

- **Desarrollo** Permite evitar errores al programar (rendimiento deficiente, pérdida de información, etc) y mejorar la calidad del producto final.

  Seguridad Tanto quien defiende como quien ataca un sistema operativo debe conocer y comprender los vectores de ataque más comunes
- **Administración** Da herramientas para poder comparar diferentes sistemas de archivos (parte de un SO): cuál conviene? conviene combinarlos? como se evita la pérdida o corrupción de la información y cómo se recupera?
- Los métodos y algoritmos que comprende un sistema operativo se pueden exptrapolar a otros campos

1.2. Funciones/objetivos de un sistema operativo

- **Abstracción**: Los programas no tienen que preocuparse de los detalles de acceso a hardware o de la configuración de una computadora.
- **Administración de recursos**: Los procesos que se ejecutan en un SO compiten por los recursos (memoria, espacio de almacenamiento, tiempo de procesamiento, etc). El sistema operativo debe gestionar estos recursos de forma efectiva y compatible con las necesidades preestablecidas.
- **Aislamiento**: Los procesos y usuarios del sistema no deben afectar ni ser afectados por otros procesos o usuarios directamente. La experiencia de cada proceso o usuario se verá afectada indirectamente ya que el sistema virtual exclusivo en el que están será menos poderoso.

  Las funciones de aislamiento suelen requerir hardware especializado.

Funciones	Ejemplos
Abstracción	**Archivos** y **directorios** en diferentes dispositivos de almacenamiento. **Procesos**: Representa la instancia de un programa que está siento ejecutado. Se puede extrapolar a **Tareas**. **Sockets**: abstraen el hardware y las capas inferiores a las de transporte **Memoria virtual**: se utiliza almacenamiento secundario como extensión virtual del primario **Malloc** / **Calloc**: ocultan cuánta memoria hay disponible y dónde se encuentra
Adm de recursos	**Memoria virtual**: se prioriza la memoria primaria por sobre la secundaria (acceso más lento) **Gestión de recursos y usuarios**: cada usuario accede a sus directorios personales sin que se produzcan corrupciones o inconsistencias **Bloqueos**: permiten que ciertos procesos se queden a la espera recursos **Señales** (alarm, CtrlC, kill): permiten indicar si se están desperdiciando recursos: bucles infinitos u otras anomalías en la ejecución de un programa
Aislamiento	**Archivos** y **directorios** separados por usuario. **Gestión de recursos y usuarios**: se aislan los recursos de forma de que diferentes usuarios puedan acceder de forma aparentemente exclusiva **Bloqueos**: garantizan el acceso exclusivo (y no compartido) de ciertos recursos **Señales** (alarm, CtrlC, kill): se utilizan para dar la ilusión de uso exclusivo

1.3. Evolución de los sistemas operativos

1.3.1. Procesos por lotes / batch processing

Se cargaban lotes (*batches*) de tarjetas perforadas a lectores, luego se cargaban en memoria, se monitoreaban y se producían resultados. El sistema cumplía un rol de monitor: asistía a operadores en la carga de programas y bibliotecas, notificaban resultados y contabilizaban los recursos empleados.

Con el tiempo se implementaron protecciones que evitaban que la corrupción de un trabajo corrompiera los siguientes y que se entrara en bucles infinitos (estableciendo alarmas). Éstas protecciones requieren modificaciones en el hardware.

El tiempo de carga y puesta a punto era parte considerable del procesamiento.

1.3.2. Sistemas en lotes con dispositivos de carga - *spool* (*Simultaneous Peripherial Operations On-Line*)

Contaban con computadoras de propósito específica (más económicas y limitadas) para leer las tarjetas, convertirlas en cinta magnética (más rápidas) y dejarlas listas para cuando se terminara el trabajo anterior en la máquina central. Análogamente para la salida, la computadora central guardaba los resultados en cinta que luego serían leídos e impresos por equipos especializados.

1.3.3. Sistemas multiprogramados

> Cuando la ejecución de un programa se dedica al cálculo numérico, se dice que está **limitada por CPU** (*CPU-Bound*) y mientras se está leyendo o escribiendo resultados desde o hacia medios externos (el límite lo imponenen éstos dispositivos) se dice que está **limitada por entrada-salida** (*I-O bound*).

Buscan maximizar el tiempo de uso efectivo del procesador ejecutando varios procesos al mismo tiempo. Esto implica cambios radicales en el hardware y obliga a garantizar la protección de recursos: un proceso no debe escribir en el espacio de memoria ni en el espacio de monitor de otro proceso. Esta protección de recursos la realiza la unidad de manejo de memoria (MMU, *Memory Management Unit*).

Además, es posible que sistemas con estas características requieran bloqueos para ofrecer acceso exclusivo a ciertos dispositivos de entrada-salida.

1.3.4. Sistemas de tiempo compartido

Los sistemas interactivos y los sistemas multiusuarios [1] cambiaron la programación y depuración de código (los cambios se pueden ver apenas se implementan) y la cantidad de tiempo que el sistema espera a que un programa esté listo: siempre hay algún proceso para ejecutar.

Además introducen un nuevo tipo de control sobre la multitarea: la multitarea apropiativa.

> En la **tarea cooperativa** (*cooperative multitasking*), implementada en los sistemas multiprogramados (1.3.3), cada proceso tiene control de la CPU hasta que hace una llamada al sistema (**multitarea semi-cooperativa**) o coopera con una llamada a *yield* (ceder el paso): **tarea cooperativa pura**.
> En la **tarea apropiativa** (*preemptive multitasking*) el reloj del sistema interrumpe periodicamente a los procesos y transfiere de manera forzosa el control al sistema operativo. Éste luego puede elegir el próximo proceso a ejecutar lo cual da lugar a diversas políticas de planificación de procesos (3).

Los sistemas de tiempo compartido también empiezan a incorporar algunas abstracciones conocidas como los archivos o directorios. El código necesario para emplearlos se fue centralizando cada vez más hasta formar parte del **núcleo del sistema operativo**.

En los sistemas de tiempo compartido en comparación con los multiprogramados, la velocidad de cambio entre una tarea y otra es rápido: el hardware emite periódicamente interrupciones (señales) que indican un cambio en el proceso activo para dar la ilusión de uso exclusivo. Además, los procesos se empiezan a clasificar por **prioridades** según qué tan críticos son para el funcionamiento del sistema, qué tanta carga de interactividad tienen, etc.

[1] En buena medida diferenciados de los anteriores por la incorporación de terminales

1.4. Dispositivos móviles

Algunas características que diferencian a los dispositivos móviles de las computadoras personales (de escritorio o portátiles) son:

- **Almacenamiento de estado sólido**: componente electrónica de almacenamiento sin partes móviles que utiliza memoria no volátil (no necesita energía para mantener la información guardada).

  No emplean memoria virtual: no pueden mantener en ejecución programas que escedan el espacio real de memoria.

- **Multitarea monocontexto**: como los dispositivos no tiene memoria virtual, se deben limitar la cantidad de procesos interactivos en ejecución. Además, como las pantallas son pequeñas comparadas con la de las PC, no tiene sentido tener más de un solo programa visible en todo momento.

- **Consumo eléctrico**: como en los dispositivos móviles los eventos son muchos y de muy distinta naturaleza, éstos operan bajo una filosofía de *siempre encendido*, es decir, están siempre pendientes al entorno.

- **Entorno cambiante**: el dispositivo móvil tiene que cambiar de perfil de energía de forma más abrupta y frecuente que las PC, cambiar la limunosidad dependiendo de la luminosidad circundante, desactivar funcionalidades si está en un nivel crítico de carga, poder aprovechar las conexiones fugaces mientras se lo desplaza, y configurar ágilmente ante la rotación de pantalla, entre otros

El *jardín amurallado* es un modelo de distribución de software en donde la empresa Apple en el caso de IOS o la empresa Google en el caso de Android se reserva el derecho de aprobar y/o eliminar una aplicación de la tienda de aplicaciones que ofrece. Este modelo rompe con el principio de *generatividad* de Jonathan Zittrain: las plataformas o ecosistemas de tecnología tienen la capacidad de producir cambios motivados por una audiencia larga, variada y no coordinada. La tecnología según este punto de vista se adapta, se supera, se perfecciona, se vuelve más accesible.

1.5. Organización de los sistemas operativos

Existen 2 formas primarias de organización interna de un sistema operativo:

- **Monolíticos**: existe un solo proceso privilegiado que opera en modo supervisor que incluye todas las rutinas que puede utilizar un sistema operativo.

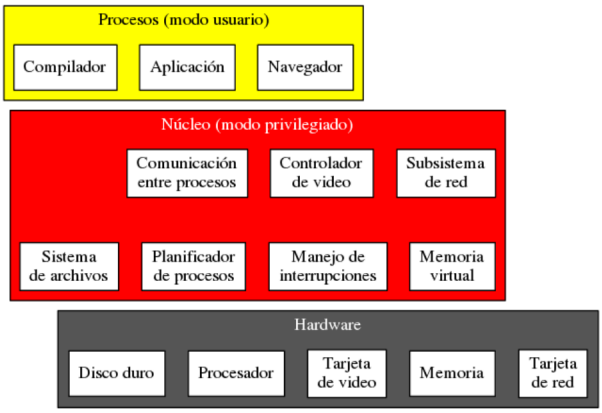


Figura 1: Esquematización de los componentes en un sistema monolítico

- **Microkernel**: el núcleo del sistema se mantiene en el mínimo posible de funcionalidad. Existen procesos especiales sin privilegios que implementan el acceso a dispositivos y las políticas de uso del sistema.

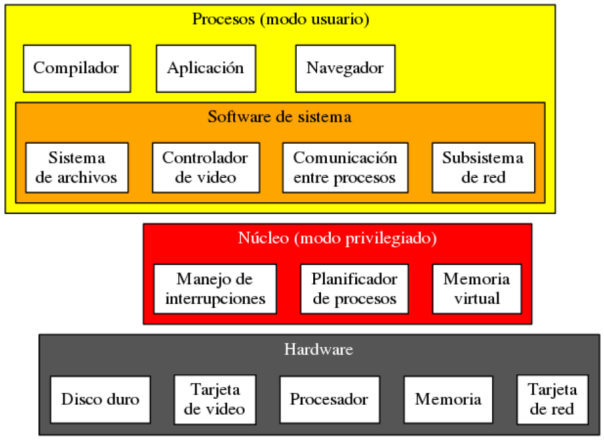


Figura 2: Esquematización de los componentes en un sistema microkernel

	Ventajas
Sistemas monolíticos	Los mecanismos de comunicación son simples →
	Mayor velocidad de ejecución (menos cambios de contexto para cualquier operación)
	El mayor acoplamiento permite más flexibilidad para adecuarse a nuevos requisitos
Sistemas microkernel	Siguen esquemas lógicos más limpios
	Permiten implementaciones más elegantes
	Se prestan al armado de sistemas distribuidos
	Faciltan la comprensión de las piezas por separado y la extensión del sistema
	Son más robustos: el núcleo puede reiniciar o reemplazar componentes y autorepararse

1.5.1. Sistemas híbridos

Los sistemas híbridos son mayormente monolíticos pero manejan procesos centrales a nivel usuario como lo hacen los microkernel. Un ejemplo de este tipo de proceso central es el sistema de archivos en espacio de usuario de Linux y otros Unixes FUSE (*Fylesystem in Userspace*).

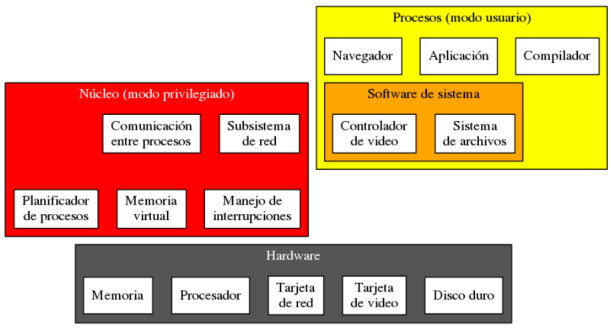


Figura 3: Esquematización de los componentes en un sistema híbrido

1.6. Relación con el hardware

1.6.1. Multiprogramación y multiprocesamiento

- Un **sistema multiprogramado** da la *ilusión* de que está ejecutando varios procesos al mismo tiempo, pero en realidad está alternando entre los diversos procesos que compiten por su atención.
- Un **sistema multiprocesador** tiene la capacidad de estar atendiendo simultáneamente a diversos procesos.

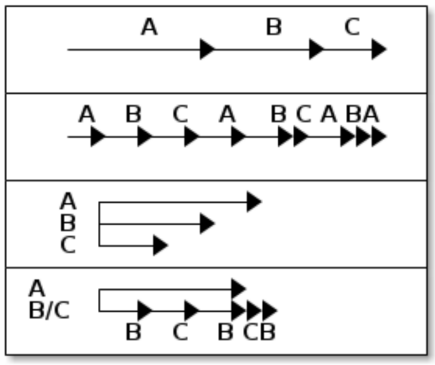


Figura 4: Esquema de ejecución de los procesos A, B y C es un sistema secuencial, multiprogramado, multiprocesado (multitarea pura) puro e híbrido

En un momento determinado sólo puede estar ejecutando sus instrucciones un número de procesos igual o menor al número de procesadores que tenga el sistema.

1.6.2. Eventos del sistema

Los eventos que ocurren alrededor del sistema son manejados por interrupciones o excepciones (o trampas).

- **Interrupción:** es generada por causas externas al sistema (un dispositivo requiere atención). Se organizan por prioridades. Ejemplos: se accionó alguna tecla, llegó un paquete a la interfaz de red, los datos solicitados al controlador de disco duro ya están disponibles.
- **Excepción:** es generada por un proceso (una condición en el proceso que requiere la atención del sistema operativo). Ejemplos: se produjo un acceso ilegal a memoria fuera del segmento (segmentation fault), el proceso en ejecución lanzó una llamada al sistema, se produjo una división sobre cero, el proceso en ejecución estuvo activo ya demasiado tiempo (es hora de un cambio de contexto).

Funciones del SO respecto a interrupciones: ver pag 76 del libro

1.6.3. Llamadas al sistema

Cuando un proceso requiere de alguna acción privilegiada, accede a ella realizando una llamada al sistema. Las hay de muchos tipos, ver par 91 del libro.

2. Administración de procesos

En un sistema multiprogramado o de tiempo compartido, un proceso es la imagen en memoria de un programa, junto con la información relacionada con el estado de su ejecución. Un programa es una entidad pasiva y un proceso una activa: empleando al programa define la actuación que tiene en el sistema.

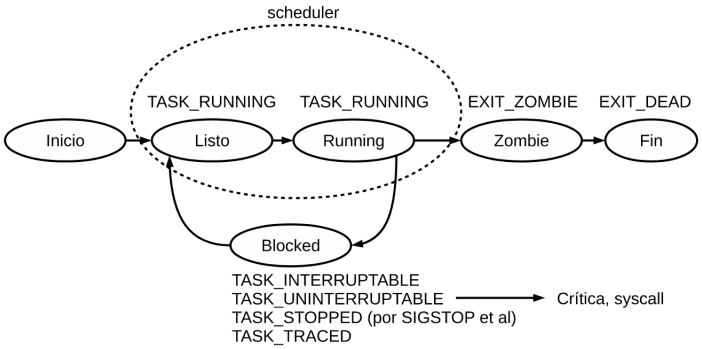
> En un sistema por lotes se habla de **tareas**. Éstas requieren menos estructura y no son interrumpidas en el transcurso de su ejecución.

2.1. Estados e información de un proceso

Los estados de ejecución de un proceso son

- Nuevo: se pidió al sistema operativo la creación del proceso. Los recursos y estructuras están siendo creadas.
- Listo: puede iniciar o continuar su ejecución pero todavía no le asignaron procesador.
- En ejecución: está siendo ejecutado. Sus instrucciones están siendo procesadas en algún procesador.
- Bloqueado: a la espera de un evento para poder continuar su ejecución.

- Zombie: finalizó su ejecución. El sistema tiene que hacer operaciones de limpieza como: notificar al proceso padre, cerrar las conexiones activas, liberar memoria, etc.
- Terminado: terminó de ejecutarse. Los recursos y estructuras están a la espera de ser limpiadas por el sistema.



(a) Grafo de estados de una tarea

(b) Grafo de estados de un proceso

Los sistemas operativos usan una estructura llamada bloque de control de proceso (PCB - *Process Control Block*) que tiene información asociada a cada proceso. Algunos campos relevantes que contiene son:

- Estado del proceso
- Contador de programa: siguiente instrucción a ser ejecutada por el proceso
- Registros de CPU: estado del CPU mientras el proceso está en ejecución
- Información de planificación (*scheduling*): prioridad del proceso, cola a donde está agregado, etc. (ver sección 3)
- Información de administración de memoria: mapeo de memoria (página o segmento), pila de llamadas (*stack*). (ver sección 4)
- Información de contabilidad: utilización de recursos como tiempo empleado (de usuario y de sistema), uso acumulado de memoria y dispositivos, etc
- Estado de E/S: listado de dispositivos y archivos que el proceso tiene abiertos en un momento dado

2.2. Procesos e hilos

Para evitar desperdiciar tiempo en asuntos administrativos (tareas burocráticas) debido a la complejidad del PCB es que surgen los hilos de ejecución o procesos ligeros (LWP, *Lightwight processes*). La diferencia entre éstos últimos y los procesos es que los hilos comparten un solo espacio de direccionamiento en memoria y los dispositivos y archivos abiertos. Además, cada hilo se ejecuta de forma aparentemente secuencial y maneja su contador de programa y pila.

Los **hilos de usuario** (*user threads*) o hilos verdes (*green threads*) según algunos lenguajes de programación evitan involucrar al sistema operativo: al compartir memoria no necesitan IPC (*InterProcess Communication*) pero una llamada bloqueante de un hilo bloquea a todo el proceso. En algunos SO mínimos que no manejan multiprocesamiento, los hilos de usuario funcionan como procesos con multitarea interna. En algunos casos la multitarea es cooperativa.

Los **hilos de kernel** (*kernel threads*) a diferencia de los de usuario, sí informan al sistema operativo de su existencia. Lo hacen a través de bibliotecas de sistema (pthreads para POSIX o Win32_Thread para Windows). Éstos hilos pueden aprovechar la comunicación con el sistema operativo y beneficiarse de una ejecución realmente paralela o una ejecución más comparable con un multiproceso estándar.

2.2.1. Patrones de trabajo con hilos

Existen 3 patrones de trabajo puros con hilo pero éstos se pueden combinar o anidar.

- Jefe/Trabajador: el hilo jefe genera o recopila tareas a realizar, las separa y las asigna a los hilos trabajadores. Típicamente los hilos trabajadores finalizan su operación, notiican al hilo jefe y finalizan su ejecución. Este patrón es usado por servidores y aplicaciones gráficas (GUI).

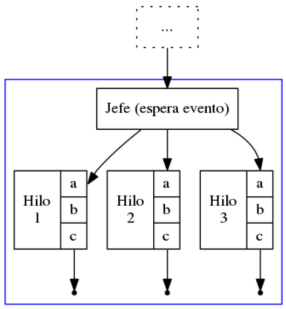


Figura 6: Patrón Jefe/Trabajador

- Equipo de trabajo: se crean muchos hilos idénticos que realizan las mismas tareas sobre diferentes datos. Los hilos son sincronizados y los resultados son agregados o totalizados para continuar con su ejecución. Este patrón se suele usar para cálculos matemáticos (criptografía, render, álgebra lineal).

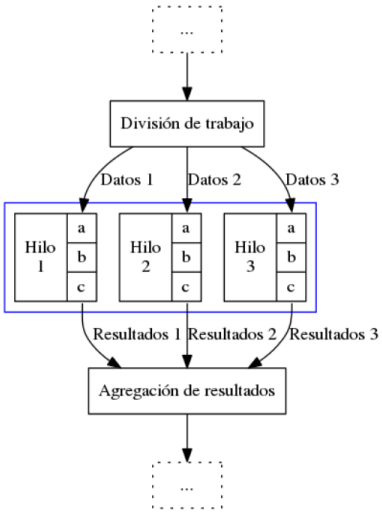


Figura 7: Patrón Equipo de trabajo

- Linea de ensamblado: cuando las tareas se pueden dividir en pasos, cada hilo puede realizar un paso y transferirle los resultado al siguiente hilo. Esto no significa que la ejecución es secuencial: los hilos se pueden estar ejecutando en paralelo sobre bloques consecutivos de información. Este patrón ayuda a mantener rutinas simples de comprender y permite que el procesamiento continue aún cuando haya bloqueos.

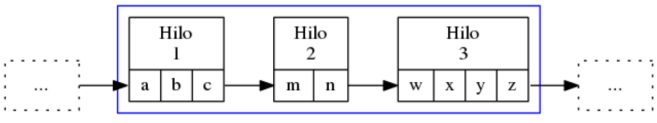


Figura 8: Patrón Linea de ensamblado

3. Planificación de procesos

La **planificación de procesos** se refiere a cómo determina el sistema operativo el orden en el que cederá el uso del procesador y a las políticas que se emplearán para que el uso no sea excesivo respecto al esperado del sistema.

- A largo plazo se decien qué procesos serán los siguientes en ser iniciados (admisión de un proceso, de nuevo a listo). Es frecuente en sistemas por lotes (principalmente con *spool*) y multiprogramados en lotes pero en la mayoría de los SO actuales esta planificación no se efectúa. Los procesos pre-declaran sus requisitos.
- A mediano plazo se decide cuáles procesos conviene bloquear y desbloquear (de en ejecución a bloqueado y de bloqueado a listo) ya sea por escasez/saturación de recursos o porque una solicitud no puede satisfacerse. Al planificador de mediano plazo se lo suele llamar agendador o **scheduler**.
- A corto plazo se decide cómo compartir el equipo entre los procesos que requieren de sus recursos (de listo a en ejecución). Debe ser código muy simple y rápido debido a la frecuencia con la que lleva a cabo. Al planificador a corto plazo se lo suele llamar despachador o **dispatcher**.

Tipos de proceso

- Procesos largos: están mucho tiempo (según la política del sistema) en estado listo o en ejecución. Lo que es lo mismo decir, procesos que está mayormente limitados por CPU (*CPU-Bound*)
- Procesos cortos: están en una ráfaga limitada por E/S (E/S-Bound) o tienden a estar bloqueados esperando a eventos como los procesos interactivos.

  Se le suele dar tratamiento preferente a los procesos interactivos para no degradar la respuesta al usuario

Medidas y métricas Para evitar que las medidas temporades dependar del hardware, se utilizan ticks y quantums.

- Tick: fracción de tiempo en la cual se puede usar el CPU sin interrupción (senalización) del temporizador (*timer*). La frecuencia con la que el temporizador senaliza se establece al inicio del sistema.
- Quantum: fracción de tiempo mínima que se permitirá a un proceso el uso del procesador. Se suele definir en término de tick y varía según el sistema operativo.

Una vez ya definidas las medidas patrón, se pueden definir las métricas que resultarán útiles para comparar políticas de planificación.

- Tiempo de respuesta (T): tiempo en espera (inactivo en la cola de procesos listos) + tiempo en ejecución
- Tiempo en espera (E = T - t) / tiempo perdido: se desea que este tiempo tienda a 0
- Proporción de penalización (P = T / t): relación entre tiempo de respuesta y tiempo en ejecución
- Proporción de respuesta (R = t / T): fracción del tiempo de respuesta en la cual el proceso se ejecutó

Se consideran también otros períodos de tiempo relevantes

- Tiempo de núcleo o *kernel*: tiempo que toman las tareas administrativas (decidir la política de planificación, hacer los cambios de contexto, atender a las interrupciones o llamadas al sistema). Este tiempo no se contabiliza cuando se calcula el tiempo del CPU utilizado por un proceso.
- Tiempo de sistema: tiempo en el que se ejecutan las instrucciones que forman parte explícita del programa
- Tiempo de uso del procesador: tiempo en el que el procesador ejecuta instrucciones asociadas a un proceso (ya sean en modo usuario o en modo núcleo)
- Tiempo desocupado (*idle*): tiempo en el que la cola de procesos listos está vacía, no hay nada que hacer
- Utilización de CPU: procentaje del tiempo en el que la CPU realiza trabajo útil. Suele estar estre el 40 % y el 90 %.

Comando time El comando time toma un programa o comando a ejecutar y devuelve el tiempo real que tardó en ejecutarse, el tiempo de usuario y el tiempo de sistema. A partir de estos valores es posible calcular

- El tiempo de uso del procesador: tiempo de usuario + tiempo de sistema
- El porcentaje de utilización del CPU: tiempo de uso del procesador / tiempo total
- El tiempo de núcleo y el tiempo desocupado: ambos están en el rango [0, tiempo total - tiempo de uso del procesador] y suman tiempo total - tiempo de uso del procesador

Y finalmente se define el valor de saturación ρ como $\rho = \frac{\alpha}{\beta}$ donde α es la frecuencia de llegada de procesos promedio y β es el tiempo de servicio requerido promedio.

- Si $\rho = 0$ nunca llegan procesos nuevos, el sistema estará eventualmente idle
- Si $\rho = 1$ los procesos son despachados al ritmo al que llegan
- Si $\rho > 1$ la cola de procesos listos tiende a crecer (la calidad de servicio y la proporción de respuesta R decrementarán)

3.1. Objetivos de la planificación

- Ser justo: se trata de igual manera a procesos de características similares y nunca se posterga indefinidamente a ninguno de ellos
- Maximizar el rendimiento: se da servicio a la mayor cantidad de procesos por unidad de tiempo
- Ser predecible: un mismo proceso toma aproximadamente la misma cantidad de tiempo independientemente de la carga del sistema
- Minimizar la sobcarga: la burocracia se mantiene al mínimo
- Equilibrar los recursos: se favorece a los procesos que emplean recursos subutilizados y se penaliza a los que compiten por los sobreutilizados
- Evitar la postergación indefinida: se aumenta la prioridad de los procesos más viejos
- Favorecer el uso esperado del sistema: se maximiza la prioridad de los procesos que sirvan a solicitudes iniciadas por usuarios interactivos
- Dar preferencia a los procesos que puedan causar bloqueos: se favorece a un proceso que esté usando recursos que otros procesos necesitan aún cuando la prioridad del primero sea baja
- Favorecer los procesos con comportamiento deseable: se degrada a los procesos que causan muchas demoras
- Degradarse suavemente: se busca responder con la menor penalización posible

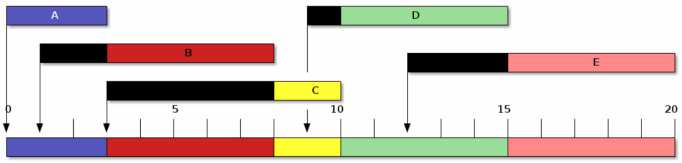
3.2. Algoritmos de planificación

Proceso	Llegada	t
A	0	3
B	1	5
C	3	2
D	D	5
E	12	5

Tabla 1: Traza patrón

3.2.1. FCFS - *First come, first serve*

Mecanismo cooperativo con mínima lógica posible: cada proceso se ejecuta en el orden en el que llega y hasta que suelta el control. El despachador es una cola FIFO.



- Ventajas

  Reduce al mínimo la carga administrativa (elige el siguiente proceso en la cola y no hay cambios de contexto)

  No requiere hardware de apoyo (temporizador)

- Desvetajas

  Los últimos procesos en llegar o los procesos cortos inoportunos son severamente penalizados

  Entre más procesos se acumulen, mayor será la inanición

3.2.2. Ronda (RR) - *Round Robin*

El algoritmo round robin emplea tarea apropiativa: cada proceso en la lista de procesos listos puede ejecutarse por solo un quantum q (el cual puede ser ajustado). Luego de un quantum la ejecución del proceso se interrumpe y en el caso de que no haya terminado se manda al final de la lista.

Mientras más aumenta q, más se parece a FCFS
Mientras más decrementa q, mejor se logra la ilusión de un proceso por procesador

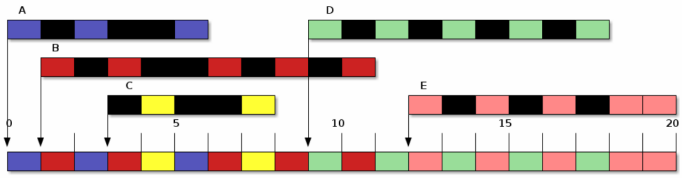


Figura 9: q = 1

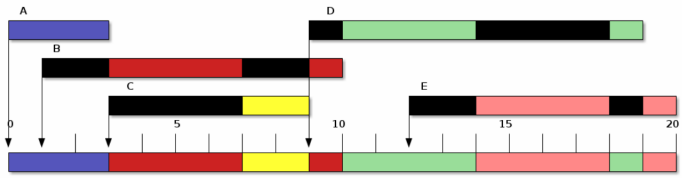


Figura 10: q = 4

- Ventajas

  Da una relación de respuesta buena entre procesos cortos y largos

  La ejecución de los procesos avanza uniformemente

- Desvetajas

  Si hay una gran cantidad de procesos, ninguno de los procesos avanza considerablemente respecto a su penalización

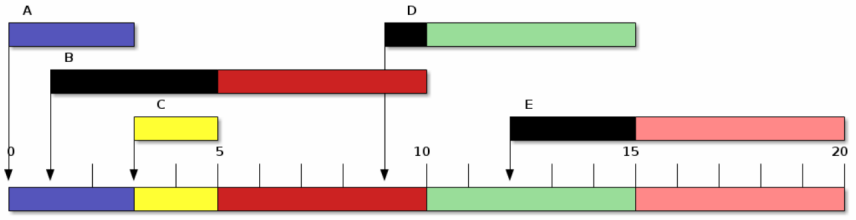
3.2.3. SPN - *Shortest process next*

SPN cooperativo Como por lo general no se tiene de antemano cuanto va a durar la ejecución de un proceso, se calcula un estimado en base a su historial.

$$e'_p = fe_p + (1 - f)q \tag{1}$$

donde f es el factor atenuante (suele ser cercano a 0.9) y q es la cantidad de quantums que empleó el proceso en su última invocación. Se puede tomar de semilla e_p un número aleatorio o el promedio de e_p de los procesos actualmente en ejecución.

Según ese estimado es que se elige el siguiente proceso a ejecutar (el proceso más corto).



- Ventajas

  Los procesos cortos se ven favorecidos

- Desvetajas

  Los procesos más largos que el promedio están predispuestos a sufrir inanición

PSPN - *Preemptive SPN* Se combina el esquema de multitarea apropiativa con la estrategia SPN. Este algoritmo no penaliza más que lo que penalizaría Round robin.

HPRN - *Highest penalty ratio next* Intenta balancear FCFS (favorece a procesos largos) y SPN (favorece a procesos cortos), y empíricamente lo hace. Ahora cada proceso inicia con un valor de penalización 1 y cada vez que tiene que esperar para ser ejecutadado s segundos se actualiza su P en base a s. El proceso que se elige para ejecutar es el de mayor P. Una desventaja de este algoritmo es que el P para cada proceso tiene que recalcularse en cada cambio de contexto lo cual es un problema cuando la cola de procesos listos crece.

3.2.4. SRR - *Selfish round robin*

Fuciona con 2 colas de procesos: procesos nuevos y procesos aceptados. Los procesos listos se agregan a la cola de nuevos y se ejecutan únicamente los procesos en la cola de aceptados según el algoritmo Round robin. Cuando la prioridad de un proceso nuevo alcanza la prioridad de algún proceso aceptado, se vuelve aceptado; si la cola de procesos aceptados queda vacía, se acepta el proceso nuevo de mayor prioridad.

El parámetro a indica el ritmo con el que se incrementa la prioridad de los procesos nuevos y el parámetro b es análogo pero para la cola de aceptados.

- Si $\frac{b}{a} < 1$ los procesos nuevos eventualmente pasarán a la cola de aceptados
- Si $\frac{b}{a} \geq 1$ la cola de procesos aceptados se acabará y se aceptarán nuevos
- Si $b = 0$ los procesos nuevos se aceptan inmediatamente y el algoritmo se convierte en una ronda

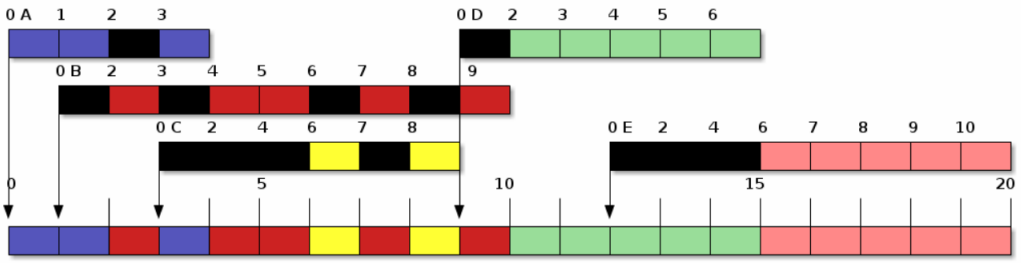


Figura 11: $a = 2$, $b = 1$

3.2.5. FB - *Multilevel feedback*

Múltiples colas de prioridad El planificador elige el primer proceso de la cola de mayor prioridad no vacía. Después de una cierta cantidad de ejecuciones, los procesos se degradan a la cola inmediata inferior.

Los procesos cortos se ven favorecidos (no se llegan a degradar)

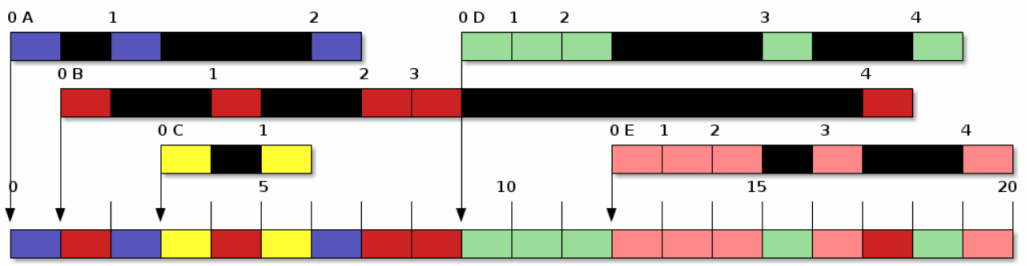


Figura 12: Los procesos se degradan después de 1 ejecución

Este algoritmo tiene dos variables ajustables:

- Cantidad de veces que se ejecuta un proceso antes de degradarse
- Duración del quantum por cada cola (creciente hacia prioridades bajas). Algunos incrementos comunes: $2^n q$, nq, $q \log(n)$ donde n es el identificador de cola y q el quantum base

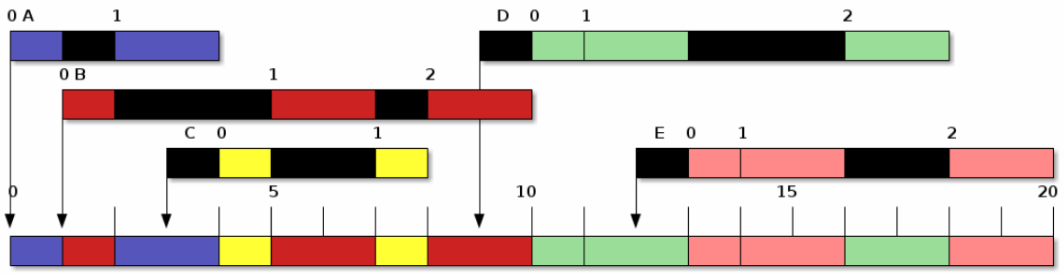


Figura 13: Los procesos se degradan después de 1 ejecución, cada cola n tiene un quantum de $2^n q$

3.2.6. Loteria

Cada proceso tiene un número de boletos y cada boleto representa una oportunidad de jugar a la loteria. El siguiente proceso a ejecutarse es elegido al azar. Una vez que un proceso se elige para ser ejecutado, no sale de juego (la probabilidad de que un proceso se ejecute es la misma en invocaciones sucesivas al planificador). Las prioridades están representadas por la cantidad de boletos que tiene un proceso.

- Ventajas

  Los procesos pueden cooperar: si un proceso de mayor prioridad tiene que esperar el resultado de uno de menor prioridad el primero puede transferirle boletos

  Es justo con procesos cortos y largos

  Presenta una degradación suave incluso en entornos de saturación

- Desvetajas

  Al ser aleatorio no se puede comparar con los anteriores

3.2.7. Resumen

	No considera historial de procesos	Considera historial de procesos
Pensado para tarea cooperativa	FCFS	SPN HPRN
Pensado para tarea apropiativa	RR Lotería	PSPN SRR FB

3.2.8. Híbridos

Algoritmo por cola de FB

- Parte de las colas pueden ser procesadas siguiendo una variación de PSPN en donde se empuje a los procesos más largos a colas con menos interrupciones.
- Las colas de más baja prioridad pueden funcionar con SRR para que sin repercutir a procesos de más alta prioridad, puedan terminar lo antes posible

Métodos dependientes del estado del sistema

- Si los procesos no son muy largos en promedio y el valor de saturación es bajo, se puede optar por algoritmos de baja carga administrativa como FCFS (RR con quantum largo para multitarea apropiativa) o SPN.
- Usar el esquema de ronda e ir ajustando el quantum periódicamente en base a la cantidad de procesos en espera y un valor mínimo para no generar una sobrecarga administrativa.

  Si hay pocos procesos en espera, el quantum se aumenta para reducir la cantidad de cambios de contexto

  Si hay muchos procesos en espera, el quantum disminuye para reducir la penalización
- Usar RR pero asignarle a cada proceso un quantum proporcional a una prioridad externa (fijada por el usuario)
- Peor servicio a continuación (WSN, *worst service next*): se considera el número de veces que se interrumpió y la cantidad de veces que tuvo que esperar por recursos. El siguiente proceso a ejecutarse es el del peor servicio y en caso de empate se sigue una ronda.

  El tener que calcular tantos valores repercute en el tiempo global de ejecución. Por esta razón se suele utilizar periódicamente para reordenar colas y luego se utilizan algoritmos más simples.

3.3. Planificación de hilos

Los hilos son mapeados a los procesos de diferentes formas y ésto da lugar a diferentes políticas de planificación de hilos.

- Mapeo muchos a uno: muchos hilos son agrupados en un solo proceso. El proceso es el encargado de repartir el tiempo de ejecución entre los hilos ya que el sistema no los conoce

  Ventajas: el código es más portable entre sistemas operativos

  Desventajas: Los hilos no aprovechan el paralelismo y todos se bloquean cuando un hilo hace una llamada bloqueante al sistema.
- Mapeo uno a uno: cada hilo es ejecutado como un proceso ligero.

  Ventajas: los hilos siguen compartiendo memoria y dispositivos abiertos. Cada hilo se puede ejecutar en un procesador distinto.

  Desventajas: el SO debe poder implementar hilos ligeros
- Mapeo muchos a muchos: combinación de los modelos anteriores

3.4. Planificación de multiprocesadores

Para trabajar en multiprocesadores, puede mantenerse una sola lista de procesos e ir despachándolos a cada uno de los procesadores como unidades de ejecución equivalentes e idénticas, o pueden mantenerse listas separadas de procesos.

3.4.1. Afinidad a procesador

Cuando un proceso se ejecuta en un procesador, gran parte de los datos estarán copiados en el caché de ese procesador. Si el proceso cambia

- a otro nucleo del procesador, sólo se deben invalidar los datos del nivel 1 (L2 se comparte)
- a otro CPU físicamente separado, se deben invalidar también los datos en el nivel 2

La afinidad **suave** entre un proceso y un procesador (o un conjunto de procesadores) indica preferencia de ejecución mientras que la afinidad **dura** da garantías de que ese proceso se va a ejecutar en ese procesador. La afinidad suave da lugar a que el despatchador pueda elegir procesador según carga ignorando la afinidad.

3.4.2. Blanceo de cargas

En un sistema multiprocesador lo ideal es que todos los procesadores estén despachando trabajos al 100 % de su capacidad. Como ésto no es posible, se trata de que la diferencia de carga de los procesadores sea la mínima posible. Se emplean esquemas de balanceo de cargas: algoritmos que analizan el estado de las colas de procesos y, de ser el caso, transfieren procesos entre las colas para homogeneizarlas. Este objetivo actua en el sentido opuesto que la afinidad de procesador.

Estrategias de balanceo:

- Migración por empuje (*push migration*): tarea que se ejecuta como parte del núcleo que revisa periódicamente el estado de los procesadores. En caso de encontrar un desbalance mayor a cierto umbral empuja a uno o más procesos del procesador más ocupado al más libre.
- Migración por jalón (*pull migration*): cuando un procesador está ocioso, ejecuta un proceso especial desocupado (*idle*). Eso significa que puede ejeutar tareas de nucleo, averiguar si otros procesadores tienen procesos en espera (y jalarlos), etc.

3.4.3. Colas de procesos listos

En los SO de uso amplio se cuenta con más de una cola de procesos listos. Mantener una sola cola dificultaría mantener la afinidad al procesador y restaría flexibilidad al sistema.

3.4.4. Procesadores con soporte a hilos hardware - *hiperthreading*

El flujo de una sola instrucción a través de un procesador puede ser dividido en partes creando una estructura conocida como *pipeline*. Idealmente en todo momento hay una instrucción del pipeline ejecutándose en cada una de las secciones del procesador, lo cual se puede pensar como un paralelismo interno al procesador.

Se comprobó empíricamente que hay patrones que intercalan los procesos que subutilizan el procesador. Para solucionar esto, cada núcleo de procesador trabaja con 2 o más hilos de hardware (*hyperthread*) y algún patrón que utilice mejor el procesador (alternar ciclos de cómputo y de espera de memoria por ejemplo).

3.5. Tiempo real

Existen procesos que requieren garantías de tiempo. Éstos procesos deben reservar los recursos de antemano y la atención en tiempo real puede manejarse periódicamente (se requiere el procesador cada 30ms) o aperiódicamente (el proceso que toma 600ms se tiene que terminar de ejecutar en menos de 2 segundos).

Un esquema de tiempo real **suave** puede implementarse mediante un esquema similar al de la retroalimentación multinivel, con las siguientes particularidades

- La cola de tiempo real recibe prioridad sobre todas las demás colas
- La prioridad de un proceso de tiempo real no se degrada conforme se ejecuta repetidamente
- La prioridad de los demás procesos nunca llegan a subir al nivel de tiempo real por un proceso automático (aunque sí puede hacerse por una llamada explícita)
- La latencia de despacho debe ser mínima

3.5.1. El problema de inversión de prioridades

Un efecto colateral de que las estructuras del núcleo estén protegidas por mecanismos de sincronización es el problema de inversión de prioridades. Un caso simple de inversión de prioridades es el siguiente

- Un proceso A de baja prioridad hace una llamada al sistema y es interrumpido a la mitad de la llamada
- Un proceso B de prioridad tiempo real hace una llamada al sistema que requiere de la misma estructura que tiene bloqueada el proceso A

Un proceso de alta prioridad no puede avanzar hasta que uno de baja prioridad libere el recurso. Una solución a este problema se resuelve con herencia de prioridades: todos los procesos que estén bloqueando recursos requeridos por procesos de mayor prioridad son tratados como procesos de la prioridad más alta hasta que liberen el recurso.

3.6. Linux scheduler - CFS

Ver del apunte, está bastante resumido :).

4. Administración de memoria

El procesador puede utilizar directamente sus registros, su memoria caché y la memoria principal. Como todo programa debe ser cargado en memoria para ser utilizado, el sistema operativo tiene que administrarla de forma de que varios programas puedan compartirla.

Espacio de direccionamiento La memoria se estructura como un arreglo de bytes. Un procesador que soporta un espacio de direccionamiento de n bits puede referirse directamente a 2^n bytes. Sin embargo a través de un mecanismo llamado PAE (*physical address extension*) es posible referirse indirectamente a más.

Unidad de manejo de memoria - MMU En los sistemas multitarea se debe resolver cómo ubicar los programas en la memoria física disponible (principal y secundaria, ver 4.4) y esto no se puede hacer sin soporte de hardware (MMU). Por otro lado, la MMU también se encarga de verificar que un proceso pueda acceder o modificar datos de otro.

Memoria caché Memoria de alta velocidad situada entre la memoria principal y el procesador que guarda copia de las páginas (ver 4.3) a las que se acceden siguiendo los principios de localidad temporal (un recurso empleado recientemente, pronto se volverá a acceder) y localidad espacial (un recurso muy probablemente se accederá si recursos cercanos se accedieron).

Espacio en memoria de un proceso Un proceso se vuelca en memoria dividido en partes

- Sección de texto: imagen en memoria de las instrucciones a ejecutarse
- Sección de datos: espacio para variables globales y datos inicializados. Se fija en tiempo de compilación y no cambia
- Espacio de libres: espacio para asignación dinámica de memoria durante la ejecución del proceso (*Heap*)
- Pila de llamadas: espacio para la secuencia de funciones llamadas dentro del proceso e información asociada (*Stack*)

Resolución de direcciones En la compilación se sustituyen los nombres de funciones o variables por sus direcciones en memoria. En la copia en memoria de la sección de texto se deben resolver o traducir las direcciones a una forma que sea relativa al inicio del proceso en memoria.

- En tiempo de compilación: el texto del programa tiene la dirección absoluta de las variables y funciones
- En tiempo de carga: previo a la ejecución se actualizan las referencias de memoria dentro del texto para que apunten al lugar correcto
- En tiempo de ejecución: el programa hace referencia a una base y un desplazamiento lo cual permite que el proceso pueda ser reubicado en memoria mientras se ejecuta sin sufrir cambios. Requiere hardware específico

4.1. Asignación de memoria contigua

Partición de la memoria Se asigna a cada programa un bloque contiguo de memoria de tamaño fijo. El sistema operativo utiliza la región baja de la memoria y luego se puede asignar memoria a cada proceso con un registro base y uno límite. Desde el punto de vista desde el SO cada espacio asignado a un proceso es una partición.

4.1.1. Fragmentación

Cuando un proceso termina de ejecutarse y se libera su memoria se produce fragmentación: comienzan a aparecer regiones de memoria disponible interrumpidas por procesos activos.

Para asignarle memoria a los nuevos procesos y cubrir las áreas fragmentadas existen 3 estrategias:

- Primer ajuste: elegir el primer bloque en el que el proceso quepa. Es el mecanismo más simple y de más rápida ejecución pero no es óptimo.
- Mejor ajuste: elegir el espacio que mejor se ajuste al requerido. Los bloques que quedan después de esta elección están mejor ajustados (son lo más chico que se pueden hacer). Implica una revisión completa de todos los bloques lo cual es más costoso.
- Peor ajuste: elegir el bloque más grande disponible. Se busca que los bloques que queden después de esta elección sean tan grandes como sea posible. Usando un heap esta operación puede ser más rápida que mejor ajuste.

La **fragmentación externa** se produce cuando hay muchos bloques libres entre bloques asignados a procesos; la **fragmentación interna** se refiere a la cantidad de memoria dentro de un bloque que no se va a usar.

4.1.2. Compactación

El espacio total libre puede ser mucho más que lo que requiere un proceso pero al estar fragmentado no existe partición continua en la que quepa. Los procesos que emplean resolución de direcciones en tiempo de ejecución pueden lanzar una operación de compresión o compactación cuando noten un alto índice de fragmentación.

La compactación consiste en mover el contenido de los bloques para que ocupen espacios contiguos. Es una operación costosa ya que implica mover prácticamente la totalidad de la memoria y probablemente más de una vez cada bloque.

4.1.3. Intercambio con el almacenamiento secundario - *swap*

El sistema operativo puede comprometer más memoria de la físicamente disponible. Cuando la memoria se termina, el sistema suspende un proceso (usualmente uno bloqueado) y almacena una copia de su imagen en almacenamiento secundario que será luego restaurado.

Si un proceso que se quiere llevar al área de intercambio tiene operaciones de E/S pendientes, los resultados de éstas se pueden guardar en buffers (almacenamiento temporal) para luego trasladarlos al espacio correspondiente en el área de intercambio.

Esta técnica cayó en desuso por lo costoso de copiar un proceso completo en almacenamiento entero para luego restaurarlo en memoria principal. Ver 4.2.1 para una mejora

4.2. Segmentación

Los segmentos que conforman un programa se organizan en secciones (código compilado, tabla de símbolos, etc) y cuando el sistema operativo crea un proceso a partir del programa, carga algunas de estas secciones en memoria (como mínimo la sección de texto y variables globales). Para garantizar la protección de estas secciones el sistema puede asignar cada sección a segmentos diferentes los cuales pueden tener distintos juegos de permisos (texto: lectura y ejecución, datos libres y pila: lectura y escritura).

Otra ventaja de la segmentación es que incrementa la modularidad de un programa: las bibliotecas ligadas dinámicamente están representadas en segmentos independientes.

Un código compilado para procesadores que implementa segmentación contiene *direciones lógicas* que luego el sistema operativo podrá asociar a *direcciones físicas*. La traducción de direcciones lógicas a físicas puede fallar porque

- (*)El segmento no cuenta con los permisos: violación de seguridad
- (*)El tamano del segmento es menor al desplazamiento pedido: acceso de lectura arroja desplazamiento fuera de rango mientras que acceso de escritura arroja violación de segmento

- El segmento está marcado como no presente (ver 4.2.1 - segmentos de texto): segmento faltante
- (*)El segmento no existe: segmento inválido
- Una combinación de las anteriores. El sistema reacciona a la más severa

Las fallas indicadas con (*) suelen implicar la terminación de proceso.

4.2.1. Intercambio parcial

Permitir que solo ciertas regiones de un programa sean llevadas al area de intercambio. Si un programa tiene porciones de código que nunca se ejecutarán al mismo tiempo entonces tiene sentido separar su texto y datos en diferentes segmentos. En esta división se generarán segmentos que no se emplearán por un largo período de tiempo y que podrán ser llevados al swap por solicitud del proceso o por iniciativa del sistema operativo.

Segmentos de texto Una agregado que mejora el desempeno del intercambio parcial tiene que ver con copiar una única vez al disco segmentos de texto de solo lectura. Como no será modificado durante la ejecución, basta con marcarlo como no presente en las tablas de segmentos en memoria. Cualquier acceso (falla de segmento faltante) hará que se suspenda el proceso y que el SO traiga el segmento del área de intercambio a la memoria principal.

Bibliotecas dinámicas (código objeto independiente de su ubicación) Si la biblioteca reside en disco como una imágen de su representación en memoria, el sistema solamente tiene que saber su locación en el disco (no es necesario cargarla en memoria para luego llevarla al área de intercambio).

4.3. Paginación

Cada proceso está divido en bloques de tamano fijo (menor que el tamano de un segmento) llamados páginas. Los programas ya no requieren asignación de espacio contiguo de memoria (resuelve la paginación externa). La paginación requiere mayor soporte de hardware y aumenta la cantidad de información asociada a cada proceso (mapeo entre la ubicación real y lógica del programa).

La memoria física se divide en marcos (*frames*) del mismo tamano y el espacio para los procesos se divide en páginas que coinciden en tamano con los marcos. La MMU se encarga de hacer el mapeo entre marcos y páginas mediante tablas de páginas. Ahora cada dirección en memoria (potencias de 2) se puede dividir en un identificador de página (m bits más significativos) y un desplazamiento (n bits menos significativos).

Tamaño de la página Para evitar la fragmentación interna se puede emplear un tamano de página tan chico como sea posible, pero esto implica una carga administrativa alta:

- Las transferencias entre unidades de disco y memoria con más eficientes si pueden mantenerse recorridos continuos. El controlador de disco puede responder a solicitudes de acceso directo a memoria (DMA) si los fragmentos son continuos.
- El bloque de control de proceso (PCB) se agranda a medida que aumenta la cantidad de páginas de un proceso y esto incide en la rapidez con la que se puede hacer un cambio de contexto

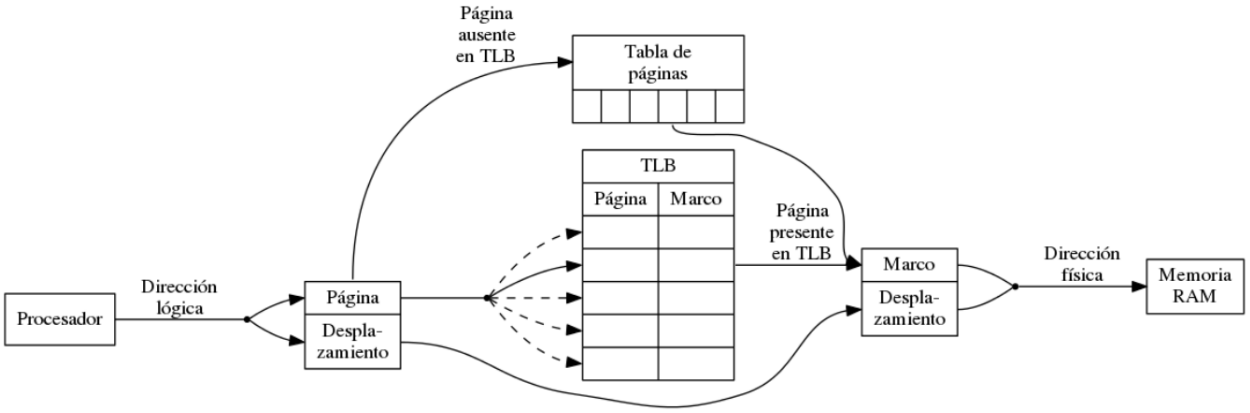
Por estas razones se trata de mantener el tamano de página tan grande como sea posible. El tamano habitual de una página suele ser 4 u 8 KB

4.3.1. Almacenamiento de la tabla de páginas

La tabla de páginas se guarda en memoria y se hace referencia a su inicio y longitud con registros especiales PTBR (*page table base register*) y PTLR (*page table length register*) respectivamente. En cada cambio de contexto, estos dos registros se modifican.

Con este mecanismo cada acceso a memoria se penaliza con un acceso adicional (el acceso a memoria se duplica): para traducir una dirección lógica a una física es necesario consultar la tabla de páginas en memoria.

Buffer de traducción adelantada (TLB - *translation lookaside buffer*) Tabla asociativa (*hash*) en memoria de alta velocidad dentro de la MMU que funciona como caché donde las claves son páginas y los valores son los marcos correspondientes en memoria física. Cuando el procesador solicita un acceso a memoria, si la traducción está en la TLB, la MMU tiene la dirección física inmediatamente, en otro caso, la MMU lanza un fallo de página (*page fault*) y se hace una consulta ordinaria en la memoria principal. Esta consulta luego se agrega a la TLB.



Como la TLB es limitada, se tiene que explicitar una política (ver 4.4.1) que elija que página/s víctima remover.

Subdividir la tabla de páginas El espacio empleado por las páginas es muy grande incluso empleando TLB. Aprovechando que la mayor parte del espacio de direccionamiento está típicamente vacío (stack y heap) se puede subdividir el identificador de página en 2 o más niveles (tablas externas y tablas intermedias). Este esquema funciona para computadoras con direccionamiento de hasta 32 bits: la paginación jerárquica implica que un fallo de página triplica el tiempo de acceso a memoria. Para un sistema de 64 bits, para que la tabla de páginas sea manejable se necesita septuplicar el tiempo de acceso.

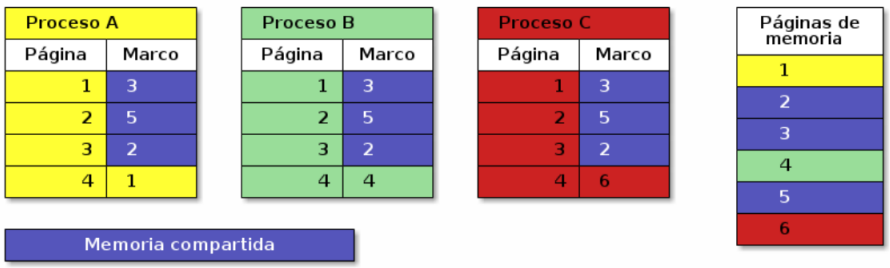
Otra alternativa es emplear funciones digestoras (*hash functions*) para mapear cada página a un espacio muestral mucho más chico (lista de direcciones físicas). Lo importante de estas funciones es que deben ser más ágiles para evitar que el tiempo que toma calcular la posición en la tabla no sea significativo frente a otras alternativas.

4.3.2. Memoria compartida

En la comunicación entre procesos (IPC) o cuando se ejecuta más de una vez un programa (sin código automodificable) no tiene sentido que las estructuras compartidas en el caso de los procesos o las páginas asociadas a cada instancia en el caso de muchas instancias de un mismo programa ocupen un marco independiente.

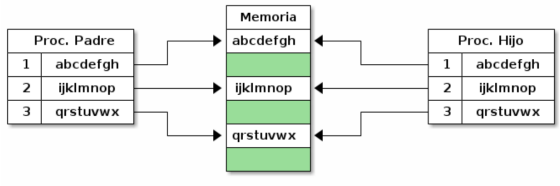
La idea también aplica a las bibliotecas de sistema (libc, openssl, zlib, libpng): las bilbiotecas suelen ser empleadas por una gran cantidad de programas.

En general, cualquier programa que esté desarrollado y compilado de forma de que todo su código sea de solo lectura permite que otros procesos entren en su espacio de memoria sin tener que sincronizarse con otros procesos que lo estén empleando.

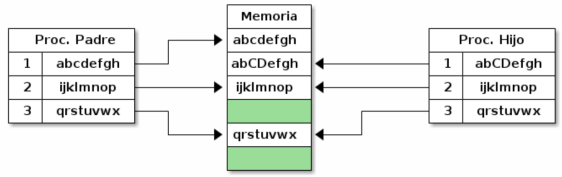


CoW - *Copy on write* En los sistemas Unix, cuando un proceso llama a fork() se crea un proceso hijo idéntico al padre con el cual se comparten todas las páginas en memoria. Para garantizar que la memoria no se modifique si no es por los canales explícitos de comunicación entre procesos, se usa el mecanismo copiar al escribir (CoW): mientras las páginas sean de solo lectura, las páginas de padre e hijo son las mismas pero ni bien cualquiera de los procesos modifica alguna de

las páginas, las modificaciones se copian a un nuevo marco y este marco deja de ser compartido. El sistema operativo, en vez de terminar el proceso debido al fallo de página que se produce al querer escribir en un bloque de solo lectura, copia la página en la que se encuentra la dirección de memoria que causó el fallo y marca esta nueva página como de lectura / escritura.



(a) Memoria luego del fork



(b) Memoria luego de la modificación de la primer página

4.4. Memoria virtual

En un sistema que emplea paginación, un proceso no conoce su dirección en memoria relativa a otros procesos, sino que trabajan con una idealización de la memoria, en la cual ocupan el espacio completo de direccionamiento, desde el cero hasta el límite lógico de la arquitectura, independientemente del tamaño físico de la memoria disponible.

Para ofrecer a los procesos mayor espacio en memoria del que se cuenta físicamente, el sistema emplea espacio en almacenamiento secundario (típicamente, disco duro), mediante un esquema de intercambio (*swap*) guardando y trayendo páginas enteras. La memoria es gestionada de forma **automática** y **transparente** por el sistema operativo (los procesos no saben de páginas). El encargado de esto es el *paginador*, no es intercambiador (*swapper*).

Paginación sobre demanda - *demand loading* Existe una gran cantidad de código durmiente o inalcanzable: código asociado a excepciones, a exportar archivos en ciertos formatos o a verificar que no haya tareas pendientes al cerrar un programa. Y un programa puede empezar a ejecutarse sin que esté completamente en memoria: basta cargar en memoria las instucciones que permiten continuar con la ejecución actual.

La paginación por demanda significa que al comienzo de la ejecución de un proceso se carga en memoria **solamente** la porción necesaria de código para comenzar y a lo largo de la ejecución solo se cargan en memoria las páginas que van a ser utilizadas (las páginas que no se necesiten no se cargarán).

La estructura empleada por la MMU para implementar el paginador flojo (*lazy*) es parecida a la que utiliza la TLB: la tabla de páginas incluye un bit de validez que indica si cada página del proceso está presente en memoria o no. Si el proceso trata de emplear un página marcada como no válida, se genera un fallo de página, el proceso se suspende, se trae a memoria la página solicitada y se actualiza el PCB del proceso y la TLB (si las páginas son válidas).

Un caso extremo de demand loading que se conoce como *pure demand loading* consiste en que todas las páginas que llegan a un proceso, llegan a través de un fallo de página. El proceso comienza sin ninguna página cargada y avanza con fallos de página sucesivos.

Una ventaja de demand loading es que al no requerir que se tengan en memoria todas las páginas de un proceso, permite que haya más procesos activos en el mismo espacio de memoria aumentando el grado de multiprogramación del equipo. Una desventaja de demand loading es que el tiempo efectivo de acceso a memoria se ve afectado. Este mecanismo no puede satisfacer las necesidades de procesos con requerimiento de tiempo.

Acomodo de las páginas en disco El acomodo de las páginas en disco no es óptimo. Además, si el espacio asignado a la memoria virtual es compartido con los archivos en disco, el rendimiento de la memoria virtual sufrirá. En los sistemas de la familia Windows se asigna espacio de almacenamientro en el espacio libre del sistema de archivos. Los de la familia Unix, en contraste, reservan una partición de disco exclusiva para paginación.

4.4.1. Reemplazo de páginas

Al sobre-comprometer memoria los procesos que están en ejecución eventualmente van a requerir cargar en memoria física más páginas de las que caben. Si todos los marcos están ocupados, el sistema deberá encontrar una página que pueda liberar (una página víctima) y llevarla al espacio de intercambio en el disco. Luego, se puede emplear el espacio recién liberado para traer de vuelta la página requerida, y continuar con la ejecución del proceso. Este mecanismo implica una doble transferencia.

Con apoyo de la MMU se puede incluir un bit de modificación o bit de página sucia (dirty bit) a la tabla de páginas. que se marca como apagado cuando se carga una página en memoria y se enciende cuando se escribe la página. Si el bit de página sucia de la página víctima elegida está apagado, la información es idéntica a la copia en memoria y se ahorra la

mitad de tiempo en transferencia. Este mecanismo disminuye la probabilidad de tener que realizar la doble transferencia del párrafo anterior.

Anomalía de Belady En general, a mayor número de marcos menor cantidad de fallos de página. Pero en algunas cadenas particulares (1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 - 3 y 4 marcos) se produce una degradación por aumentar la cantidad de marcos.

Algoritmos de reemplazo de páginas

- FIFO: se elige de víctima a la página que haya sido cargada hace más tiempo

Pagina	1	2	3	4	1	2	5	1	2	3	4	5
Marcos	1	1	1	1	1	1	5	5	5	5	4	4
		2	2	2	2	2	2	1	1	1	1	5
			3	3	3	3	3	3	2	2	2	2
				4	4	4	4	4	4	3	3	3
M/H	M	M	M	M	H	H	M	M	M	M	M	M

Tabla 2: Política FIFO en un sistema de 4 páginas físicas

Ventaja: fácil de implementar y comprender

Desventajas: 1) no tiene en cuenta el historial de solicitudes, todas las páginas tienen la misma probabilidad de ser reemplazadas, 2) es vulnerable a la anomalía de Belady

- OPT / MIN: se elige de víctima la página que no vaya a ser utilizada por el máximo tiempo. Este algoritmo es de interés teórico ya que no es posible predecir el orden en el que se van a requerir las páginas pero da una cota mínima para los otros algoritmos

Traza	1	2	3	4	1	2	5	1	2	3	4	5
P1	1	1	1	1	1	1	1	1	1	1	4	4
P2	-	2	2	2	2	2	2	2	2	2	2	2
P3	-	-	3	3	3	3	3	3	3	3	3	3
P4	-	-	-	4	4	4	5	5	5	5	5	5
M/H	M	M	M	M	H	H	M	H	H	H	M	H

Tabla 3: Política FIFO en un sistema de 4 páginas físicas

- LRU: se elige de víctima la página menos usada recientemente (la que no se usó hace más tiempo). El resultado de evaluar una cadena S empleando LRU es equivalente a emplear OPT sobre su inversa.

Traza	1	2	3	4	1	2	5	1	2	3	4	5
P1	1	1	1	1	1	1	1	1	1	1	1	5
P2	-	2	2	2	2	2	2	2	2	2	2	2
P3	-	-	3	3	3	3	5	5	5	5	4	4
P4	-	-	-	4	4	4	4	4	4	3	3	3
M/H	M	M	M	M	H	H	M	H	H	M	M	M

Tabla 4: Política FIFO en un sistema de 4 páginas físicas

Ventajas: 1) es una buena aproximación a OPT, 2) está libre de la anomalía de Belady

Desventajas: requiere apoyo de hardware (es mucho más complejo que FIFO[2])

[2]Tiene que recorrer todas las páginas si se usa un contador por página y encontrar la máxima; o actualizar reiteradas veces una lista doblemente enlazada para acceder a la víctima en tiempo constante

5. Senales de Unix

Las senales son una forma de comunicación limitada entre procesos usada en Unixes. Son una notificación asíncrona enviada a un proceso o a un hilo específico dentro del mismo proceso para notificarlo de un evento que ocurrió.

Cuando se envía una senal, el sistema operativo interrumpe el flujo de ejecución normal del proceso destinatario para enviar la senal. La ejecución puede ser interrumpida durante cualquier instrucción no atómica. Si el proceso registró previamente un manejador (*signal handler*) entonces se ejecuta esa rutina, sino se ejecuta elmanejador por defecto.

5.1. Mecanismo de senales

Señal	Acción	Evento
SIGABRT	A	Abortar
SIGALRM	T	Alarma
SIGBUS	A	Acceso a memoria no definida
SIGCHLD	I	Un proceso hijo terminó o se detuvo
SIGCONT	C	Continuar ejecución
SIGFPE	A	Operación aritmética errónea
SIGHUP	T	Colgó (la línea serie)
SIGILL	A	Instrucción ilegal
SIGINT	T	Interrupción desde la terminal
SIGKILL	T	Kill
SIGPIPE	T	Intento de escribir a pipe sin lector
SIGQUIT	A	Salir (desde la terminal)
SIGSEGV	A	Referencia a memoria inválida
SIGSTOP	S	Stop
SIGTERM	T	Terminar
SIGTSTP	S	Stop (desde la terminal)
SIGTTIN	S	Intento de lectura en background
SIGTTOU	S	Intento de escritura en background
SIGUSR1	T	Para usuario
SIGUSR2	T	Para usuario
SIGPOLL	T	Evento sondeable (pollable)
SIGPROF	T	Timer de rendimiento expirado
SIGSYS	A	Llamada a sistema inválida
SIGTRAP	A	Trace/breakpoint trap (debuggers)
SIGURG	I	Datos urgentes (sockets)
SIGVTALRM	T	Timer virtual expirado
SIGXCPU	A	Exceso de tiempo de CPU
SIGXFSZ	A	Exceso de tamaño de archivo

Acción	Significado
T	Terminar
A	Terminación anormal (puede generar volcado de memoria)
I	Ignorar
S	Stop (detener)
C	Continuar (si estaba detenido)

Listado de senales y características

- Las excepciones que producen una división por 0 o una violación de segmento (SIGFPE y SIGSEGV) causan un volcado de memoria (*core dump*) y la terminación del programa.
- Las senales SIGKILL y SIGSTOP no se pueden interceptar. Cuando se envia SIGKILL a un proceso, el proceso termina y queda en estado zombie, en el caso de SIGSTOP el proceso pasa a una cola de estados dormidos/bloqueados y no se despierta automáticamente hasta que no es despertado por SIGCONT.

Enviando senales

- La llamada **kill** envía la senal especificada a un proceso (siempre que cuente con los permisos necesarios). Toma un identificador de proceso y una senal identificada con un entero y devuelve 0 o -1 (si hubo un error, código en errno). Si se omite el código de la senal se envía SIGTERM.

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

- La llamada **raise** envía la senal especificada al proceso actual. Toma un senal identificada y devuelve un entero

```
int raise(int sig);
```

Algunas combinaciones de teclas que causan envío de senales

- Ctrl-C envía SIGINT
- Ctrl-Z envía SIGSTP
- CTRL- envía SIGQUIT
- CTRL-T envía SIGINFO

Manejando senales Los manejadores de senales se instalan con la llamada signal() que toma el número de senal y el nuevo handler para esa senal y devuelve el handler anterior (para poder restaurarlo si es necesario). El proceso puede especificar dos comportamientos sin crear una nueva función manejadora: ignorar la senal (SIG_IGN) y usar el manejador por defecto (SIG_DFL).

```
typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
```

Problemas de sincronismo Se deben evitar los efectos secundarios: modificar errno, cambios en la máscaras de senales o su disposición y otros atributos globales. El uso de funciones no re-entrantes como malloc o printf en un handler tambien es inseguro. Hay formas complicadas de encolar las senales para procesarlas después y evitar algunos de los problemas relacionados.

Relación con excepciones de hardware La ejecución de un proceso puede resultar en una excepción de hardware (división por cero, fallo de TLB) lo cual generalmente intercepta un manejador de excepciones en el nucleo. En algunos casos, el núcleo tiene información suficiente para resolver el evento y continuar con la ejecución del proceso (fallo de TLB o fallo de paginación porque la página no está en memoria), en otros no (división por cero) y se utiliza un mecanismo de senales.

5.1.1. Relación con los estados de un proceso