Procesos y planificación

Departamento de INGENIERIA TELEMATICA
www.it.uc3m.es

Índice

- Concepto de proceso
  - Proceso
  - Modelo de estados
  - Cola de espera
  - Información de control de procesos, Bloque de Control de Procesos (PCB)
  - Memoria y tablas de E/S
- Ejecución de procesos
- Operaciones básicas
- Modelo de hilos
- Planificación de procesos

Proceso

- Proceso – un programa en ejecución; un proceso en ejecución debe progresar de manera secuencial
- Necesita recursos reservados antes o durante su ejecución
  - Tiempo de CPU, ficheros, memoria, E/S etc.
- “Unidad” de trabajo
- Los procesos SO ejecutan código del sistema
- La ejecución puede ser concurrente
- Los procesos pueden ser *de un sólo hilo* o *multihilo*

Proceso (cont.)

- El SO se responsibiliza de la administración de procesos (hilos)
  - Creación, borrado
  - Planificación
  - Sincronización
  - Comunicación
  - Manejo de bloqueos
- Términos intercanbiables: “Trabajos” – “Procesos”
- Un proceso contiene
  - (La sección de texto)
  - Contador de programa
  - Pila
  - Sección de datos
  - Registros

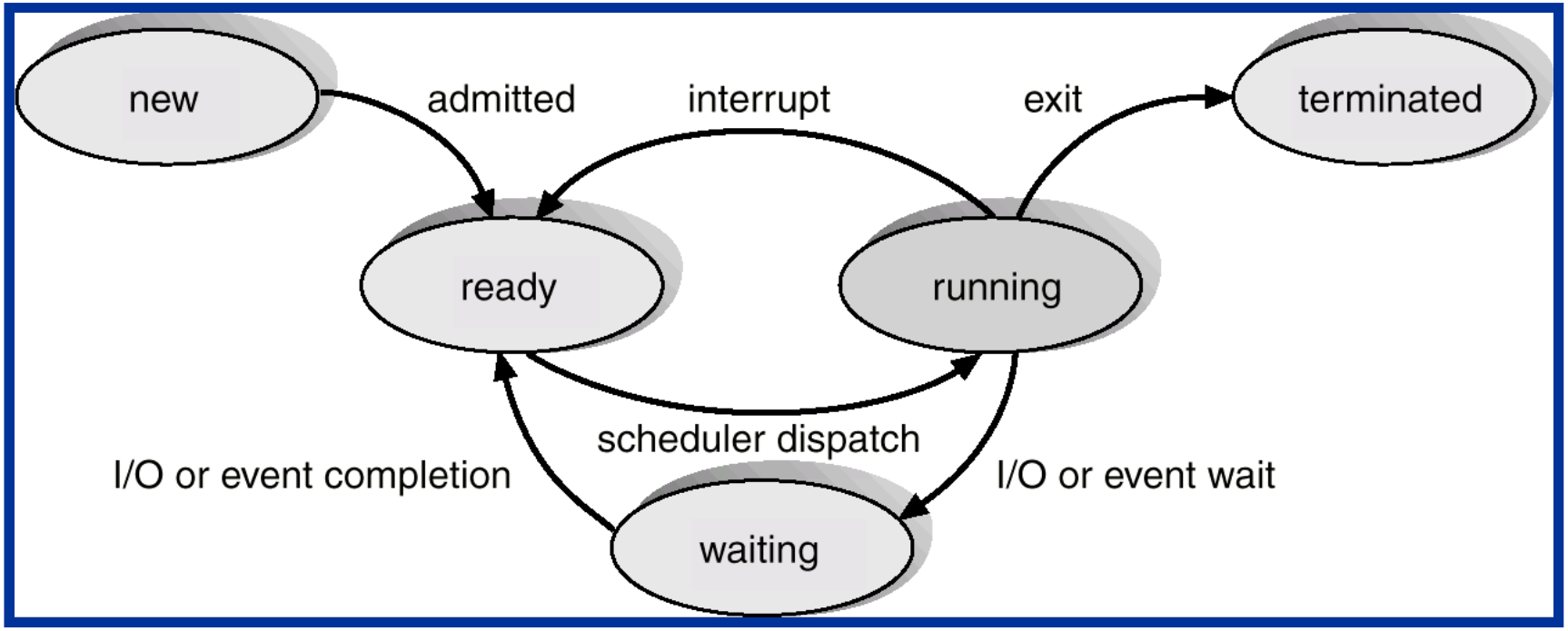
Modelo de estados

- Mientras que un proceso se ejecuta, cambia de estado
  - Nuevo: El proceso se está creando
  - Corriendo: Las instrucciones se están ejecutando
  - Esperando: El proceso está esperando que ocurra algún evento
  - Listo: El proceso está esperando que sea asignado a un procesador
  - Terminado: El proceso ha finalizado su ejecución
- La terminología y granularidad puede cambiar en diferentes implementaciones de SO

Cola de espera

¡No se discute en esta clase!

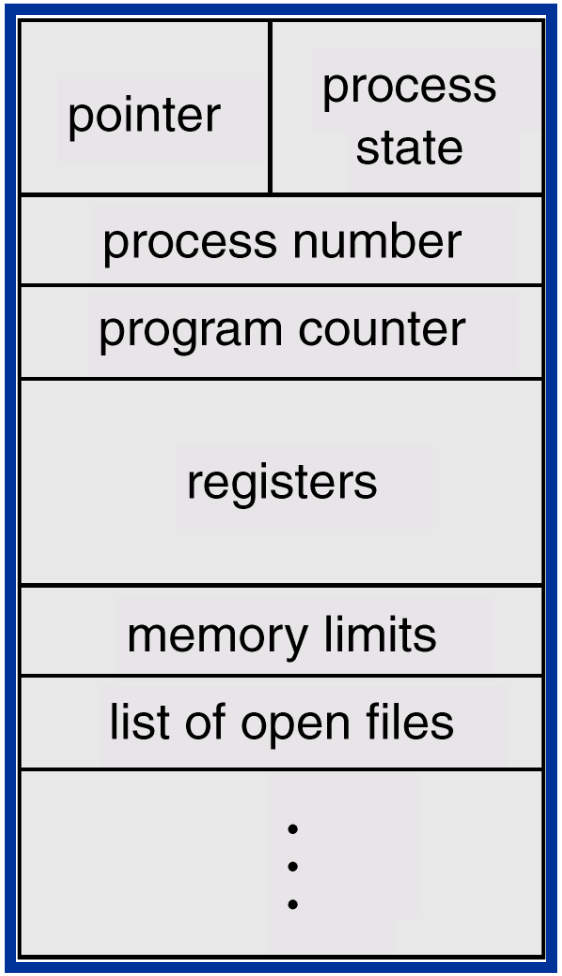
Estados de un proceso



Bloque de Control de Proceso (PCB)

Información asociada con cada proceso

- Identificación de proceso
- Estado de proceso
  - Corriendo, esperando, etc.
- Contador de programa
  - Dirección de la siguiente instrucción
- Registros de la CPU
  - Depende de la arquitectura del ordenador
- Informacion de planificación de CPU
  - Prioridad del proceso (ver discusión más adelante)
- Información de administración de memoria
  - Valor de los registros, tabla de páginas
- Información contable
  - Limites de tiempo, tiempo usado, etc.
- Información de estado E/S
  - Lista de dispositivos de E/S, ficheros abiertos, etc

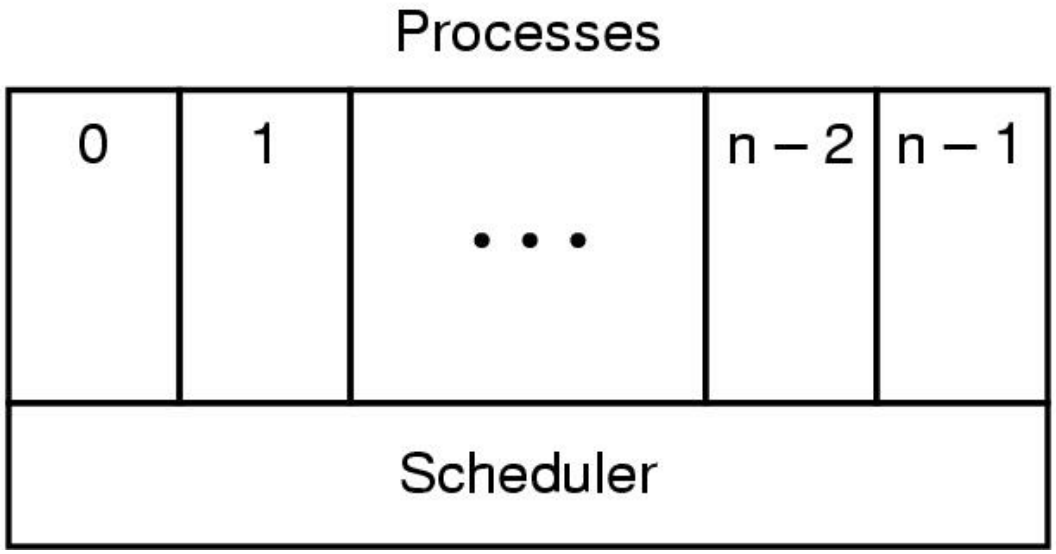


PCB: Ejemplo de comando `ps`

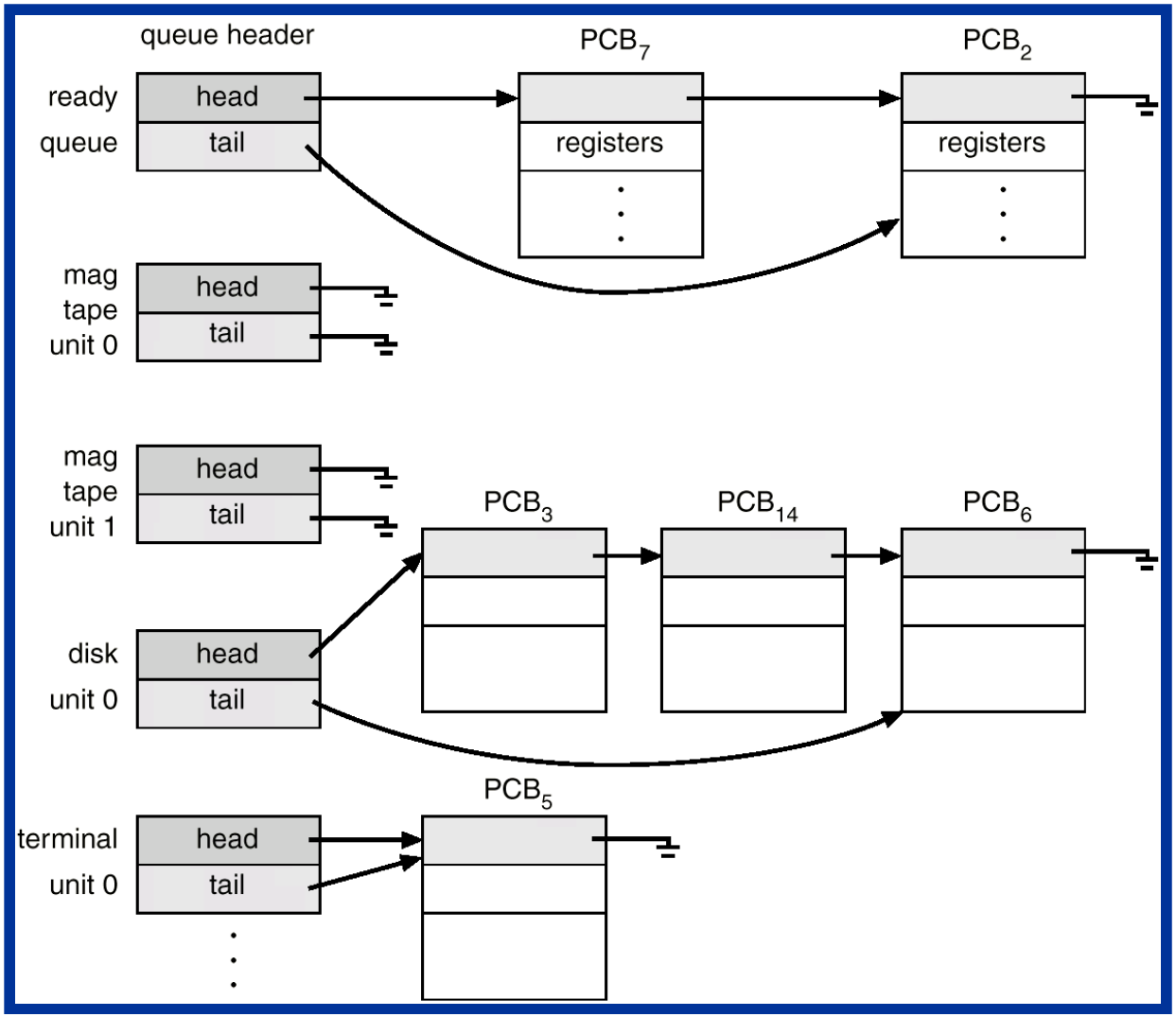
```
ralf@varpa:ralf> ps aux | less

USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   1032   428 ?        S    Apr15   3:22 init [4]
root         2  0.0  0.0      0     0 ?        SW   Apr15   0:01 [keventd]
root         3  0.0  0.0      0     0 ?        SWN  Apr15   0:06 [ksoftirqd_CPU0]
root         4  0.0  0.0      0     0 ?        SWN  Apr15   0:06 [ksoftirqd_CPU1]
root         5  0.0  0.0      0     0 ?        SW   Apr15 111:03 [kswapd]
root         6  0.0  0.0      0     0 ?        SW   Apr15   0:00 [bdflush]
root         7  0.0  0.0      0     0 ?        SW   Apr15  24:20 [kupdated]
root         9  0.0  0.0      0     0 ?        SW   Apr15   0:00 [scsi_eh_0]
root        10  0.0  0.0      0     0 ?        SW   Apr15   0:00 [scsi_eh_1]
root        13  0.0  0.0      0     0 ?        SW<  Apr15   0:00 [mdrecoveryd]
root     23779  0.0  0.0      0     0 ?        SW   Jun09   0:20 [rpciod]
root     17301  0.0  0.0      0     0 ?        SW   Jun09   0:00 [lockd]
root     10813  0.0  0.0   1440   628 ?        D    Jun09  15:25 /sbin/syslogd
root     10815  0.0  0.0   1716   368 ?        S    Jun09   0:00 /sbin/klogd
root     10819  0.0  0.1   1056  1056 ?        SL   Jun09   1:31 /usr/sbin/watchdog
daemon   10836  0.0  0.0   1140   480 ?        S    Jun09   6:34 /sbin/portmap
root     10842  0.0  0.0   1188   488 ?        S    Jun09   0:00 /usr/sbin/rpc.rquotad
root     10876  0.0  0.0   4752   708 ?        S    Jun09   0:46 /usr/sbin/slapd -h
ldap:///
root     10877  0.0  0.0   4752   708 ?        S    Jun09   0:00 /usr/sbin/slapd -h
ldap:///
```

Planificación de procesos: colas

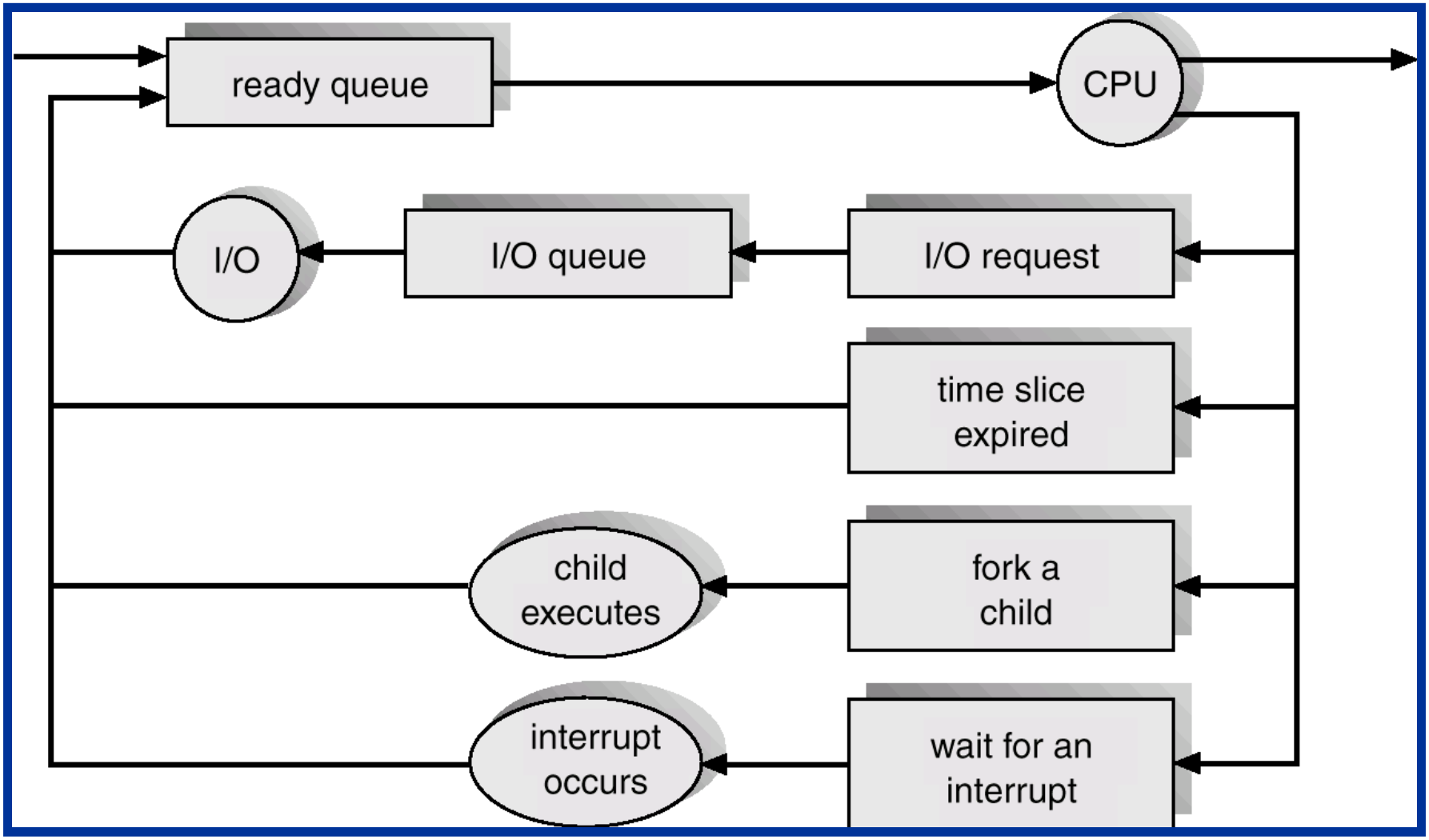
- Cola de trabajo – conjunto de todos los procesos en el sistema
- Cola de preparados residiendo en – conjunto de todos los procesos memoria principal, listos y esperando a ejecutar
- Cola de dispositivos – conjunto de todos los procesos esperando un dispositivo E/S
- Proceso de migración entre varias colas



Cola de preparados y varios



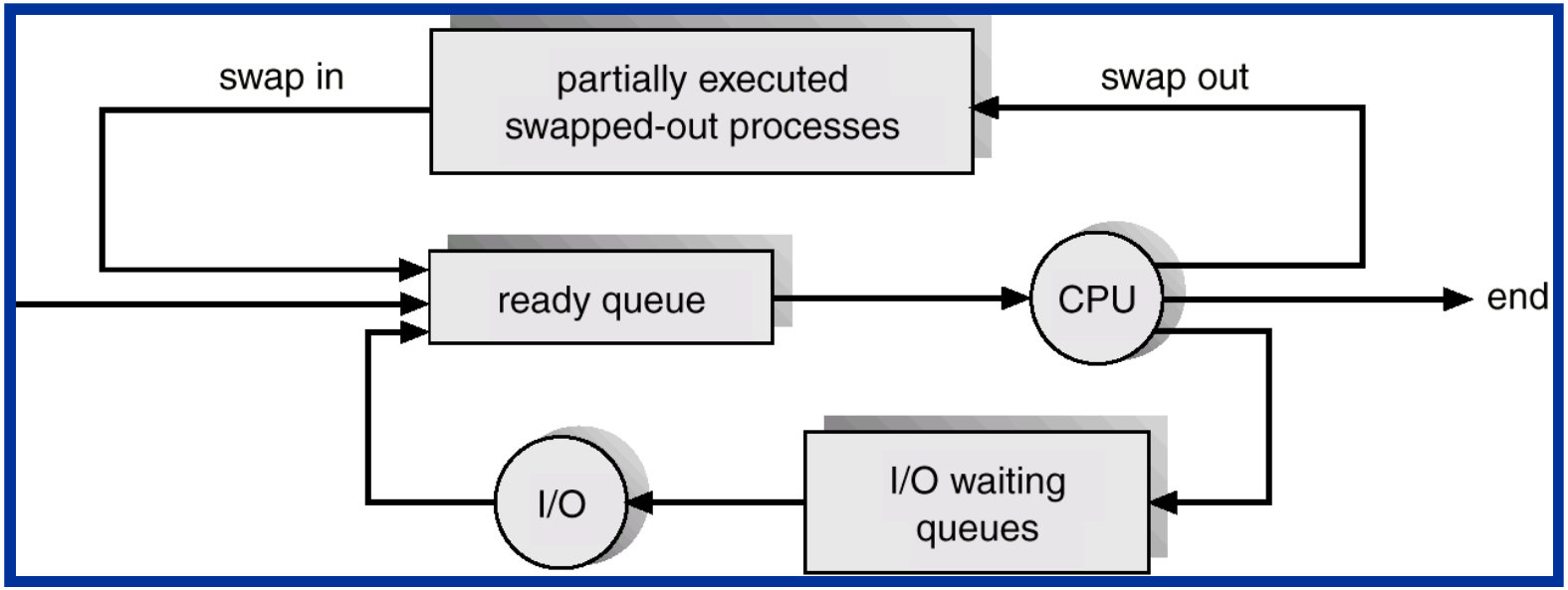
Representación de la planificación de procesos



Planificadores

- El planificador a corto plazo se invoca muy frecuentemente (milisegundos) ⇒(debe ser rápido).
- El planificador a largo plazo se invoca muy infrecuentemente (segundos, minutos) ⇒(puede ser lento).
- El planificador a largo plazo controla el *grado de multiprogramación.*
- Los procesos pueden describirse como:
  - *Proceso orientado a E/S* – pasa mucho más tiempo haciendo E/S que cálculos, muchas ráfagas de CPU.
  - *Proceso orientado a CPU* – pasa mucho tiempo haciendo cálculos; pocas ráfagas de CPU pero largas.

Añadido un planificador a medio plazo



Índice

- Estructuras de un sistema operativo
- Concepto de proceso
- Ejecución de procesos
  - Concurrencia
  - Paralelismo
  - Cooperación de procesos
- Operaciones básicas
- Modelo de hilos
- Planificación de procesos

Concurrencia/paralelismo

- Concurrencia
  - Ejecución interrelacionada de procesos en sistemas de procesador único
  - Ejecución concurrente desde el punto de vista lógico pero ejecución secuencial desde el punto de vista del procesador
- Paralelismo
  - La ejecución de múltiples componentes soporta la ejecución de procesos simultáneos (p.e. dos procesadores)

Procesos cooperantes

- Un proceso *independiente* no puede afectar o ser afectado por la ejecución de otro proceso.
- Un proceso *cooperante* puede afectar o ser afectado por la ejecución de otro proceso
- Ventajas de la cooperación de procesos
  - Compartir información
  - Cálculo más rápido
  - Modularidad
  - Conveniencia

Comunicación entre procesos

- Memoria compartida
  - 2 o más procesos pueden leer y/o escribir en una zona de Memoria Principal común
  - Típicamente un proceso da acceso a una parte de su zona de MP a otros
  - El SO debe proporcionar mecanismos para permitir esto
- Paso de mensajes
  - El SO ofrece llamadas para:
    - Enviar un mensaje a un proceso
    - Recibir un mensaje de un proceso

Procesos cooperantes: problema productor–consumidor

- Productor (proceso)
  - Produce información que puede ser consumida
- Consumidor (proceso)
  - Consume información producida por el productor
- Ambos procesos corren concurrentemente
  - Con memoria compartida se utiliza un buffer que puede ser rellenado y vaciado en paralelo
  - Sincroniza ambos procesos
    - Supera la situación en que el consumidor intenta consumir información mientras el buffer está vacío

Índice

- Estructuras de un sistema operativo
- Concepto de proceso
- Ejecución de procesos
- Operaciones básicas
  - Generación de procesos
  - Jerarquía de procesos
  - Técnicas de conmutación (conmutación de modos)
  - Terminación de procesos
- Modelo de hilos
- Planificación de procesos

Creación de procesos

- Habitualmente los SOs permiten la creación de procesos durante la ejecución
- Los procesos deben borrarse dinámicamente
- El SO debe proporcionar estos mecanismos adicionales
- Los procesos padre crean procesos hijo, que, pueden crear otros procesos, formando un árbol de procesos

Recurso de proceso

- Compartición de recursos
  - Padres e hijos comparten todos los recursos
  - El padre comparte parte de sus recursos con su hijo
  - Padre e hijo no comparten recursos
- Ejecución
  - Padre e hijo ejecutan concurrentemente
  - El padre espera hasta que el hijo termina

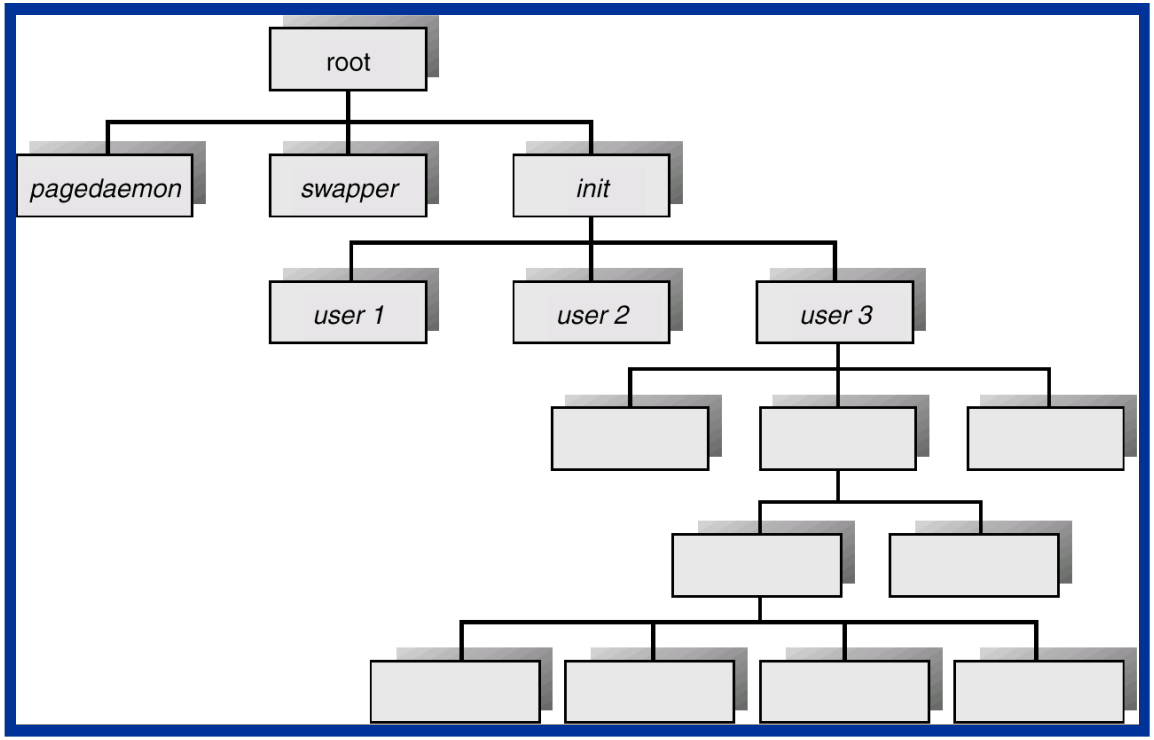
Jerarquía de procesos

- Ejecución de procesos padre-hijo
  - El padre continua para ejecutar concurrentemente (con su hijo)
  - El padre espera hasta que alguno o todos sus hijos han terminado
- Espacio de direcciones
  - El proceso hijo es un duplicado del proceso padre
  - El proceso hijo pertenece al programa cargado

Árbol de procesos

- El proceso que crea se llama proceso **padre**
- Los nuevos procesos se llaman procesos **hijo**
- Los procesos hijo pueden crear nuevos procesos hijo

Árbol de procesos:



Árbol de procesos (cont.)

- UNIX
  - La llamada del sistema **fork** crea un nuevo proceso
  - La llamada del sistema **exec** se usa después de un **fork** para reemplazar el espacio de memoria con un nuevo programa
  - Cada proceso (árbol) pertenece a una raíz inicial

- Windows
  - CreateProcess() permite crear un nuevo proceso pasándole directamente el programa que va a ejecutar

`fork` de C en proceso separado

```
void main(int argc, char *argv[])
{
int pid;
        /* fork another process */

        pid = fork();
        if (pid < 0) { /* error occurred */
                fprintf(stderr, "Fork Failed");
                exit(-1);
        }
        else if (pid == 0) { /* child process */
                execlp("/bin/ls","ls",NULL);
        }

        else { /* parent process */
                /* parent will wait for the child to complete */
                wait(NULL);
                printf("Child Complete");
                exit(0);
        }
}
```

fork (cont.)

- El proceso hijo podrá ser una copia idéntica del proceso padre
- El punto de ejecución es el mismo(en ambos procesos) en el momento de creación
- Los sistemas mantiene dos copias idénticas del ejecutable en memoria
- En caso de que la creación de procesos falle el valor que devuelve es -1
- El código de error está disponible en la variable `errno`

Exec

- Un nuevo proceso (proceso hijo) puede ejecutar un nuevo programa con la ayuda de la llamada exec
- La definición de la llamada del sistema es

```
int execl( const char *path, const char *arg, ...);
```

El `const char *arg` y las subsecuentes elipses en la función `execl` se pueden ver como `arg0, arg1, ..., argn`. Juntas describen una lista de uno o más punteros a strings terminados en null que representan la lista de argumentos disponibles para el programa ejecutado. El primer argumento, por convención, debería apuntar al nombre de fichero asociado con el nombre de fichero ejecutado. La lista de argumentos debe estar terminada en un puntero a NULL.

Exec (cont.)

- La llamada a la función `execl` no devolverá el control al punto de llamada
- El sistema substituye el programa que está corriendo por el especificado en el primer argumento
- El espacio en memoria se reserva para el código, datos y pila
- El proceso ejecutará la primera instrucción en el nuevo programa

Otras llamadas para ejecución de programas

```
EXEC(3)              Linux Programmer's Manual              EXEC(3)


NAME
       execl, execlp, execle, execv, execvp - execute a file

SYNOPSIS
       #include <unistd.h>

       extern char **environ;

       int execl( const char *path, const char *arg, ...);
       int execlp( const char *file, const char *arg, ...);
       int  execle( const char *path, const char *arg , ..., char
       * const envp[]);
       int execv( const char *path, char *const argv[]);
       int execvp( const char *file, char *const argv[]);

DESCRIPTION
       The exec family of functions replaces the current  process
       image  with  a new process image.  The functions described
       in this  manual  page  are  front-ends  for  the  function
       execve(2).   (See  the manual page for execve for detailed
       information about the replacement of the current process.)

       The  initial  argument for these functions is the pathname
       of a file which is to be executed.
```

Pasos de generación de un proceso

1. Creación de un proceso y número de identificación de proceso (PID)
2. Inserción del PID en la lista de procesos
3. Reserva de memoria
4. Inicialización del PCB
   - Se anota PID y PID del padre
   - Se resetea la información de estado
   - Se resetea la pila del contador y el contador de programa
   - Se determina el estado del proceso
   - Se inserta el proceso en la lista / cola de planificación

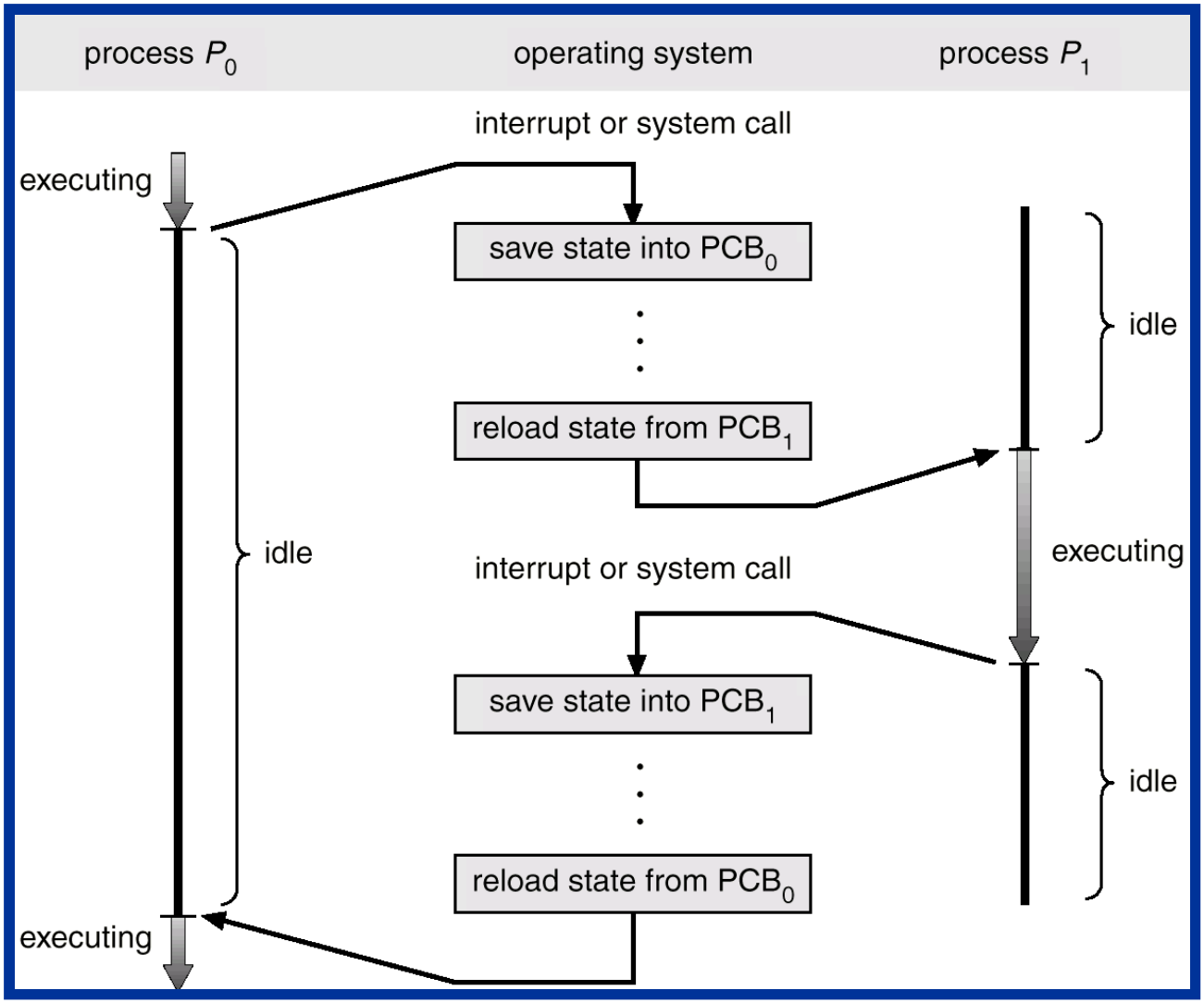
Técnicas de conmutación

- El proceso que está corriendo abandona la CPU
- Otro proceso con estado *preparado* se mueve al estado *corriendo*
- Acciones requeridas
  - Respaldar el contexto actual (contador de programa, puntero de pila, otros registros, quizás información de gestión de memoria)
  - Actualizar el PCB del proceso e insertar en la cola adecuada
  - Actualizar el PCB del proceso que va a ejecutar
  - Subir el nuevo contexto

Cambio de contexto

- Cuando la CPU conmuta a otro proceso, el sistema debe guardar el estado del proceso antiguo y cargar estado guardado para el nuevo proceso
  - El contexto del proceso guardado permite la continuación posterior del proceso en un área de memoria diferente
- El tiempo de cambio de contexto es un gasto; el sistema no realiza un trabajo útil mientras conmuta
- El tiempo de conmutación también depende del suporte hardware

Conmutación de la CPU de proceso a proceso



Proceso desocupado

- Proceso permanentemente corriendo
- Prioridad baja
- No se debe parar
- Puede ejecutar tareas de soporte
  - Chequeo consistente, etc.
- Se puede cambiar a estado de espera por otro proceso

Terminación de procesos

- El proceso ejecuta la última sentencia y pregunta al sistema operativo para borrarlo (**exit**)
  - Datos de salida del hijo al padre (via **wait**)
  - Los recursos del proceso se desalojan por el sistema operativo
- El padre puede terminar la ejecución del proceso hijo (**abort**)
  - El hijo ha excedido los recursos reservados
  - La tarea asignada al hijo no se requiere ya más
  - El padre termina
    - El sistema operativo no permite al hijo continuar su padre termina
    - Terminación en cascada

Terminación de procesos (cont.)

- Terminación normal
  - El proceso termina
  - El usuario termina el proceso
- Errores
  - Error simple; puede ser corregido
  - Error severo; acceso a memoria erróneo
- Matando procesos
  - Comando `kill()`
- Terminación en cascada
  - Matar procesos hijo (por el SO) en caso de que el padre se haya matado

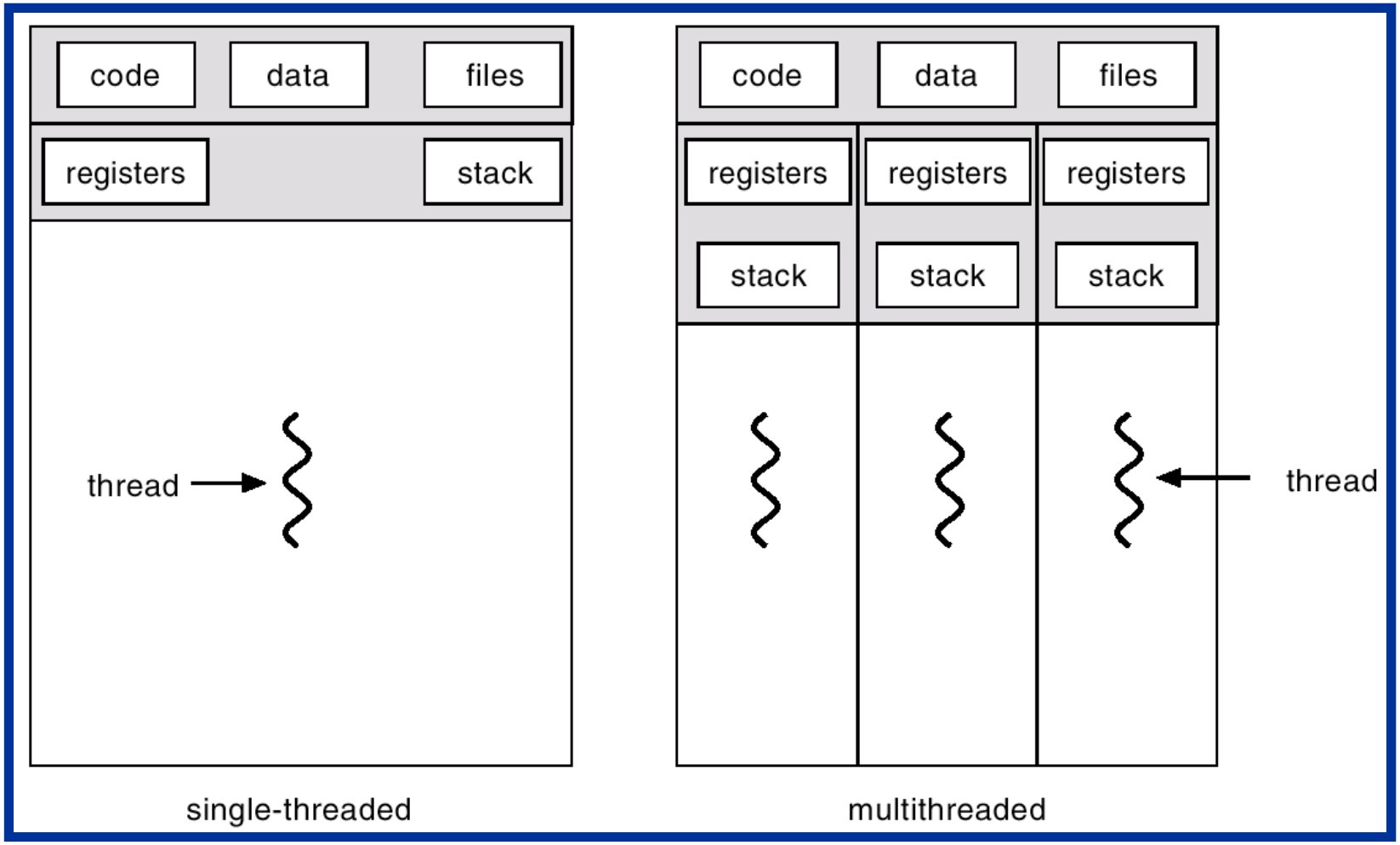
Índice

- Estructuras de un sistema operativo
- Concepto de proceso
- Ejecución de procesos
- Operaciones básicas
- Modelo de hilos
  - Definición
  - Modelo de hilo de proceso
  - Ejemplo de servidor web
  - Modelos multicolour
  - Problemas de hilado
- Planificación de procesos

Definición

- Un hilo también se conoce como proceso ligero (LWP)
  - Consta de un ID de hilo
  - Contador de programa
  - Conjunto de registros
  - Pila
- Comparte el código del mismo proceso con otros hilos y la misma
  - Sección de datos
  - Otros recursos del OS
    - Abrir ficheros, etc.
- Múltiples hilos pueden hacer más de una tarea al mismo tiempo
- Un proceso tradicional pesado tiene sólo un único hilo (de control)

Modelo de hilo de proceso



Beneficios

- Reactividad
  - Incrementa la sensibilidad en aplicaciones interactivas cuando se ejecutan operaciones prolongadas
- Compartición de recursos
  - Varios hilos en el mismo espacio de direcciones
- Economía
  - La reservar de recursos y memoria es costosa; los hilos mantienen estos costes bajos
- Utilización de Arquitecturas Multiprocesador (MP)
  - Los hilos pueden correr en diferentes procesadores al mismo tiempo

Ejemplo: un servidor web

- El servidor web sirve peticiones en un bucle infinito
- Las peticiones se pueden ejecutar sólo una tras otra (secuencialmente)
- El tiempo de respuesta puede ser muy largo (acceso a base de datos compleja, etc.); los datos se guardan en el disco y esperan una operación E/S
- Baja carga de CPU
- Realización con procesos hijo (costoso)
  - Cada petición será servida por un proceso hijo (solución estándar)
- Realización con hilos (barato)
  - Las peticiones se pueden despachar con diferentes hilos esperando o por hilos creados bajo demanda

Hilos de usuario o de kernel

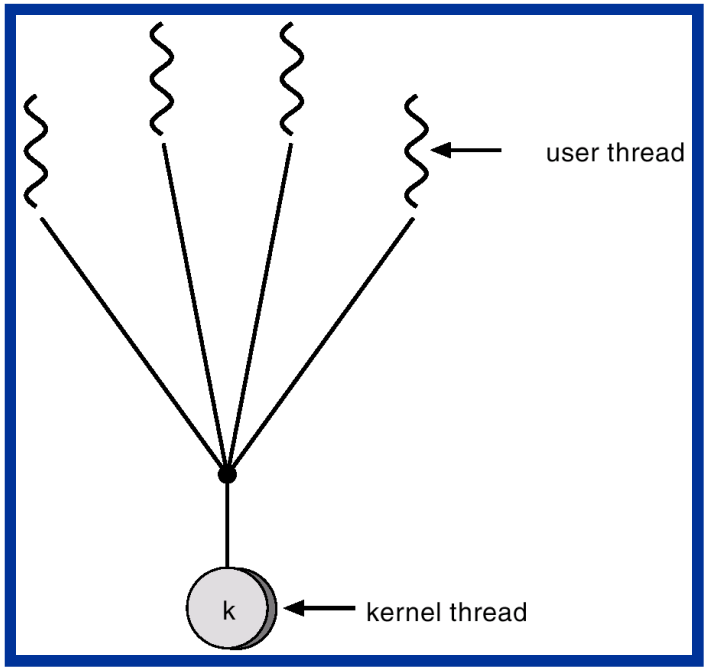
- Hilo de usuario
  - Implementado por el usuario (librería de hilos) en el nivel de usuario
  - Rápido de crear y manejar (sin conocimiento o soporte del kernel)
  - En caso de kernels de un único hilo, una llamada al sistema bloquea también otros hilos (bloqueará el proceso entero)
- Hilo de kernel
  - Soportado directamente por el SO
  - El kernel ejecuta: creación, planificación y administración
  - Administración más lenta que los hilos de usuario
  - Llamadas al sistema no bloqueantes
  - Los hilos pueden planificarse en diferentes procesadores

Modelos multihilo

- Muchos-a-Uno
- Uno-a-Uno
- Muchos-a-Muchos

Muchos-a-Uno

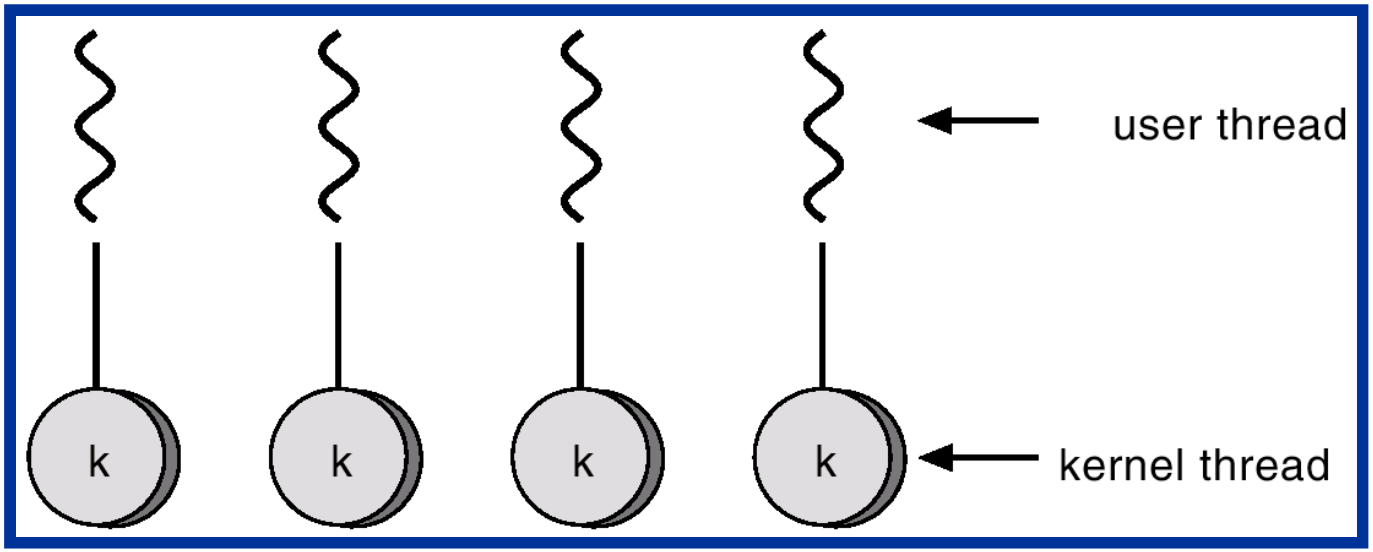
- Muchos hilos a nivel de usuario se mapean a un hilo del kernel
- Administración en espacio de usuario (eficiente)
- Usado en sistemas que no soportan hilos de kernel
- El proceso entero se bloquea en las llamadas al sistema
- Sólo un hilo de usuario puede acceder al kernel al mismo tiempo



Uno-a-Uno

- Cada hilo del nivel de usuario se mapea a un hilo del kernel
- Otros hilos pueden correr durante llamadas al sistema con bloqueo
- Corren en paralelo en multiprocesadores
- Los hilos de usuario crean automáticamente un hilo del kernel correspondiente
  - Esto puede causar una sobrecarga significativa
- Ejemplos

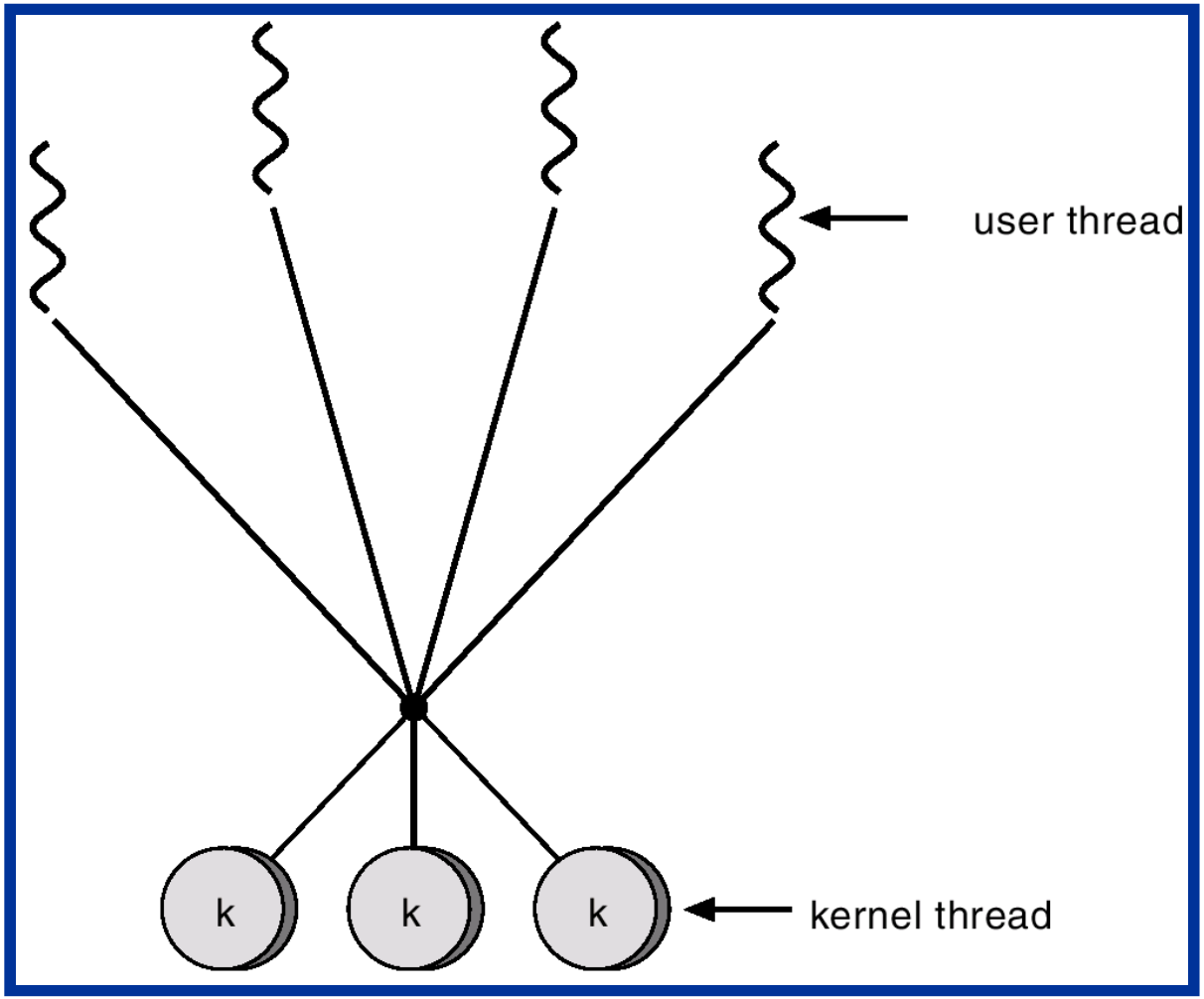
  - Windows 95/98/NT/2000, OS/2



Muchos-a-Muchos

- Permite a muchos hilos a nivel de usuario ser mapeados a muchos hilos de kernel
- Permite al sistema operativo crear un número suficiente de hilos de kernel
  - Menor o igual al número de hilos de usuario
- Llamadas al sistema no bloqueantes
- Procesamiento continuo en arquitectura MP
- Ejemplos
  - Solaris 2, Windows NT/2000 con el paquete ThreadFiber

Muchos-a-Muchos (cont.)



Problemas con los hilos

- Semántica de las llamadas al sistema `fork()` y `exec()`
  - Algunos duplican todos los hilos, algunos no
- Cancelación de hilos
  - Terminación antes de completado (este hilo se llama el *hilo objetivo*)
  - Escenarios diferentes
    - Cancelación asíncrona
      - Terminación inmediata de hilos objetivo
    - Cancelación diferida
      - Periódicamente se comprueban los hilos objetivo

Problemas con los hilos: manejo de señales

- Una señal se genera porque ocurre un evento en particular
- Una señal generada se entrega a un proceso
- Una vez que se entrega la señal debe ser manejada
  - Ejemplo: acceso ilegal a memoria
- Señales síncronas (entregadas al mismo proceso)
- Señales asíncronas (p.e. producidas por un evento externo a un proceso corriendo)
- Manejadores de señales
  - Manejador por defecto
  - Manejador definido por el usuario (manejador por defecto sobreescrito)

Problemas con hilos (cont.)

- Manejo de señales (en programas multihilo)
  - Opciones de entrega
    - Al hilo al que se le aplica la señal
    - A todos los hilos en el procesos
    - A ciertos hilos en el proceso
    - Asigna un hilo para recibir todas las señales en un proceso

Problemas con hilos (cont.)

- Reserva de hilos
  - Crea un número de hilos al principio y los pone en reserva
  - Beneficios
    - Es más rápido usar un hilo existente que crear uno nuevo a petición
    - La reserva limita el número de hilos existentes (importante cuando el número de hilos es limitado)
- Datos específicos del hilo
  - En general comparten los mismos datos con el proceso pero pueden necesitar sus propias copias (privadas)
  - La mayoría de las librerias soportan datos específicos de hilos (p.e. Java)

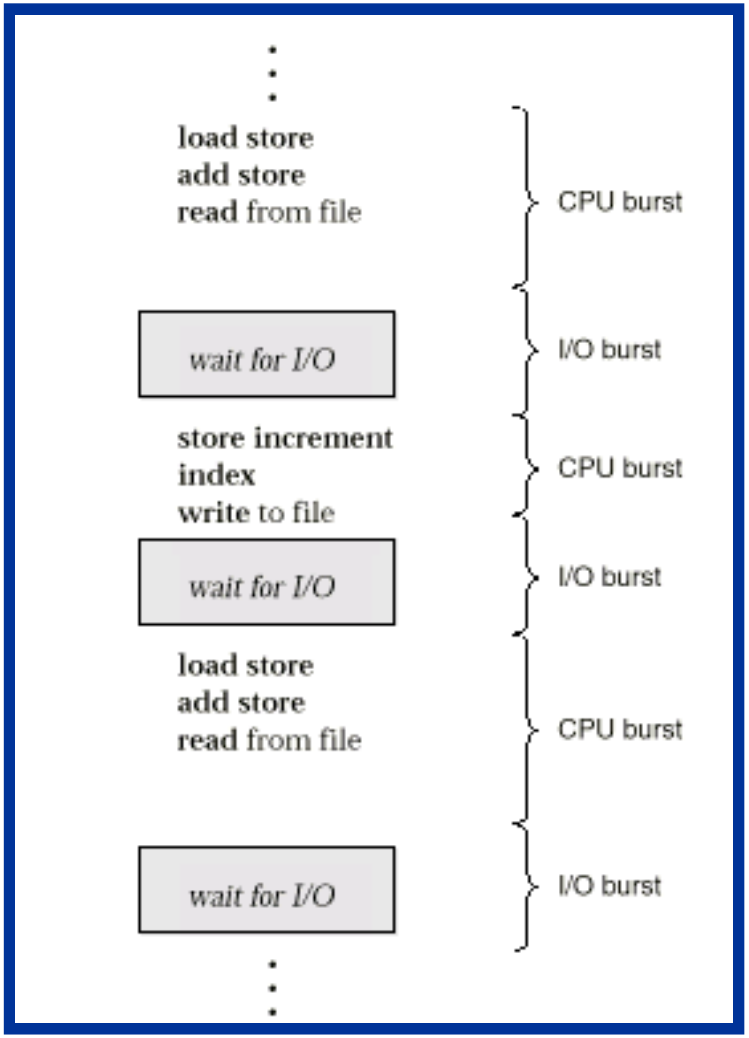
Índice

- Estructuras de un sistema operativo
- Concepto de proceso
- Ejecución de procesos
- Operaciones básicas
- Modelo de hilos
- Planificación de procesos
  - Conceptos básicos
  - Criterio de planificación
  - Planificación a largo plazo, planificación a medio plazo, planificación a corto plazo
  - Estrategias de planificación
  - Planificación de múltiples procesadores
  - Planificación de tiempo real
  - Evaluación de algoritmos

Conceptos básicos

- Base del SO multiprogramado
- Conmuta la CPU entre procesos
- Resultado: ordenador más productivo
- Ejemplo
  - Un proceso necesita esperar la entrada de datos y por tanto la CPU se mantiene desocupada
  - Un SO multiprogramado toma la CPU desde este proceso en espera y pone diferentes procesos corriendo hasta que el proceso suspendido recibe los datos de entrada

Ráfagas de ejecución



Planificador de CPU

- Planificador de CPU, también llamado planificador a corto plazo
- Los procesos se seleccionan de la cola de preparados
  - Diferentes ordenes de cola
    - First-in-first out (FIFO)
    - Cola con prioridad
    - Árbol
    - …
  - Los elementos en la cola son PCBs

Planificación de CPU

- Dependiendo del momento en el que se ejecuta la planificación, esta puede ser:
  - Planificación no-expropiativa. Se ejecuta cuando:
    - Un proceso cambia de estado "ejecutando" a estado "en espera"
    - Un proceso termina
  - Planificación expropiativa. Se ejecuta cuando:
    - Un proceso cambia de estado "ejecutando" a estado "en espera"
    - Un proceso cambia de estado "ejecutando" a estado "preparado"
    - Un proceso cambia de estado "en espera" a estado "preparado"
    - Un proceso termina

Despachador

- El modulo despachador (“dispatcher”) da el control de la CPU al proceso seleccionado por el planificador a corto plazo; esto conlleva:
  - Cambio de contexto
  - Cambio a modo de usuario
  - Saltar al lugar apropiado en el programa de usuario para continuar su ejecución

Criterio de planificación

- Utilización de CPU – mantiene la CPU tan ocupada como sea posible
- Rendimiento – nº de procesos que completan su ejecución por unidad de tiempo
- Tiempo de procesamiento – cantidad de tiempo para ejecutar un proceso en particular
- Tiempo de espera – cantidad de tiempo que un proceso ha estado esperando en la cola de “preparados”
- Tiempo de respuesta – tiempo entre la finalización de una operación de entrada/salida y la obtención de la CPU (para entornos de tiempo compartido)

Planificación

- Planificación a corto plazo
  - Administración de eventos (interrupciones)
  - Conmutación directa de procesos
- Planificación a medio plazo
  - Depende de los requerimientos de memoria de los procesos
  - Intercambio de procesos
- Planificación a largo plazo
  - Activada en caso de que el uso de la CPU esté por debajo de un cierto umbral
  - Criterio de selección
    - Prioridad
    - Tiempo estimado de ejecución y comportamiento del proceso
    - Recursos requeridos

Estrategias de planificación

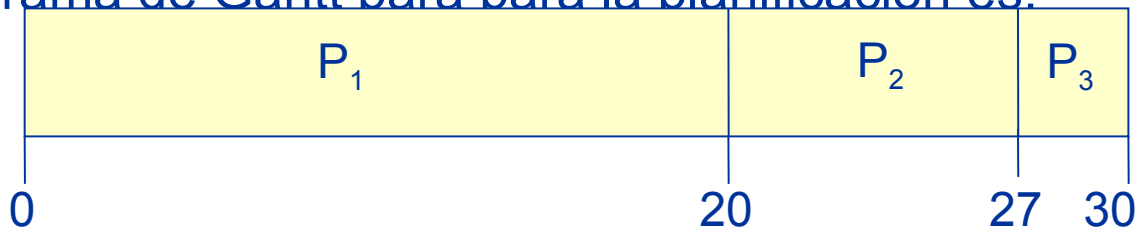
- First-come-first-served (FCFS)
  - El servicio depende de la secuencia de llegada
- Last come first served (LCFS)
  - El proceso que llega suspende un proceso que se está ejecutando, invierte la cola de llegada
- Orden Creciente de Duración (“Shortest Job First”, SJF)
  - El proceso con menor tiempo de servicio se toma primero (y puede ejecutarse hasta que termine)
- Shortest remaining time next (SRTN)
  - El proceso con menor tiempo restante hasta su completado será servido primero
- Prioridad
  - La entrada de nuevos procesos en la lista de “preparados” depende de la prioridad
  - La prioridad de los procesos servicios a menudo se reduce y evita que otros procesos sean suspendidos mucho tiempo

First-come-first-served (FCFS)

- Algoritmo sencillo
- Se puede manejar por un cola FIFO
- Ejemplo

Proceso	Fase de CPU
P_1	20
P_2	7
P_3	3

- Supongamos que los procesos llegan en orden: P_1 , P_2 , P_3
  El diagrama de Gantt para para la planificación es:



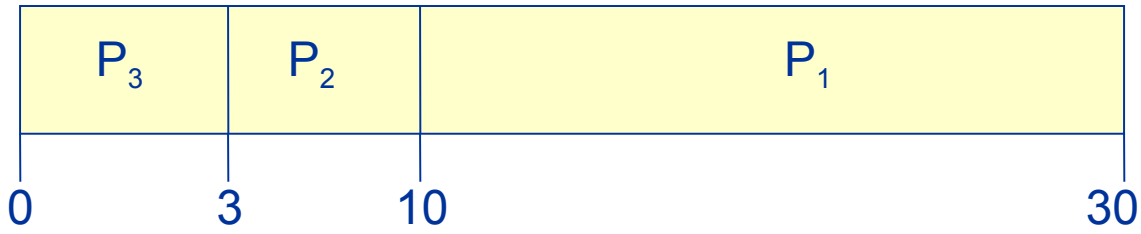
- Tiempo de espera para P_1 = 0; P_2 = 20; P_3 = 27
- Tiempo medio de espera: (0 + 20 + 27)/3 ~ 15

First-come-first-served (FCFS)

Supongamos que los procesos llegan en orden:

P_3 , P_2 , P_1

- El diagrama de Gantt para para la planificación es:



- Tiempo de espera para $P_1 = 10$; $P_2 = 3$; $P_3 = 0$
- Tiempo de espera medio: (10 + 3 + 0)/3 ~ 4
- Mucho mejor que el caso anterior
- *Efecto convoy:* el proceso corto va detras del proceso largo

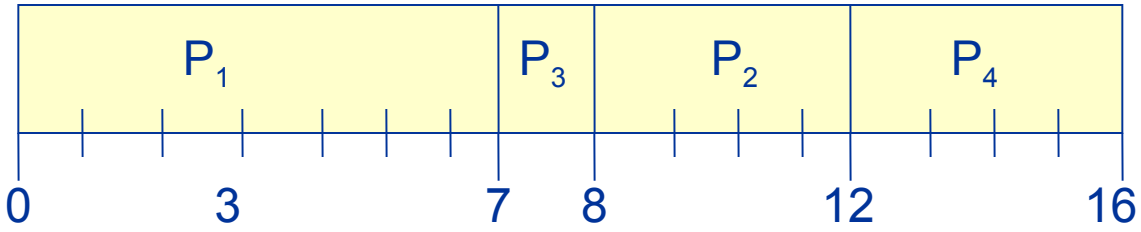
Planificación Shortest-job-first (SJF)

- Asocia a cada proceso la duración de su próxima ráfaga de CPU. Usa estas duraciones para planificar el proceso con el menor tiempo.
- Dos esquemas:
  - No expropiativo – una vez que la CPU se dá a un proceso, no puede ser expulsado hasta que completa su ráfaga de CPU
  - Expropiativo – si un nuevo proceso llega con una duración de ráfaga de CPU menor al tiempo restante del proceso que está ejecutándose actualmente, se expulsa. Este esquema se conoce como Shortest-Remaining-Time-First (SRTF)
- SJF es óptimo – tiene un tiempo medio mínimo de espera para un conjunto de procesos dado.

Ejemplo de SJF no expropiativo

Proceso	Tiempo de llegada	Fase de CPU
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (no expropiativo)

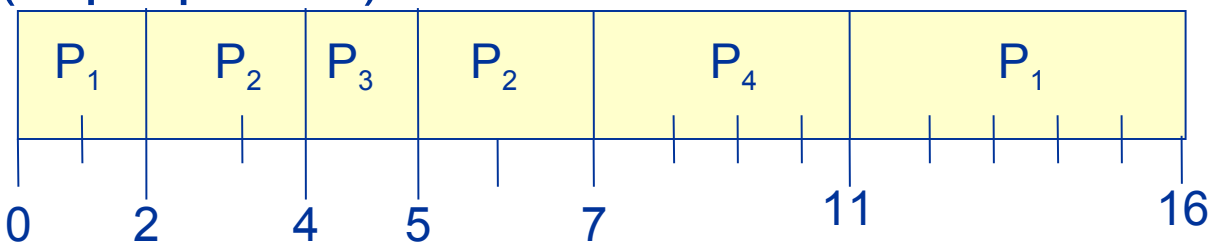


- Tiempo de espera medio = (0 + 6 + 3 + 7)/4 = 4

Ejemplo de SJF expropiativo

Proceso	Tiempo de llegada	Fase de CPU
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (expropiativo)



- Tiempo de espera medio = (9 + 1 + 0 +2)/4 = 3

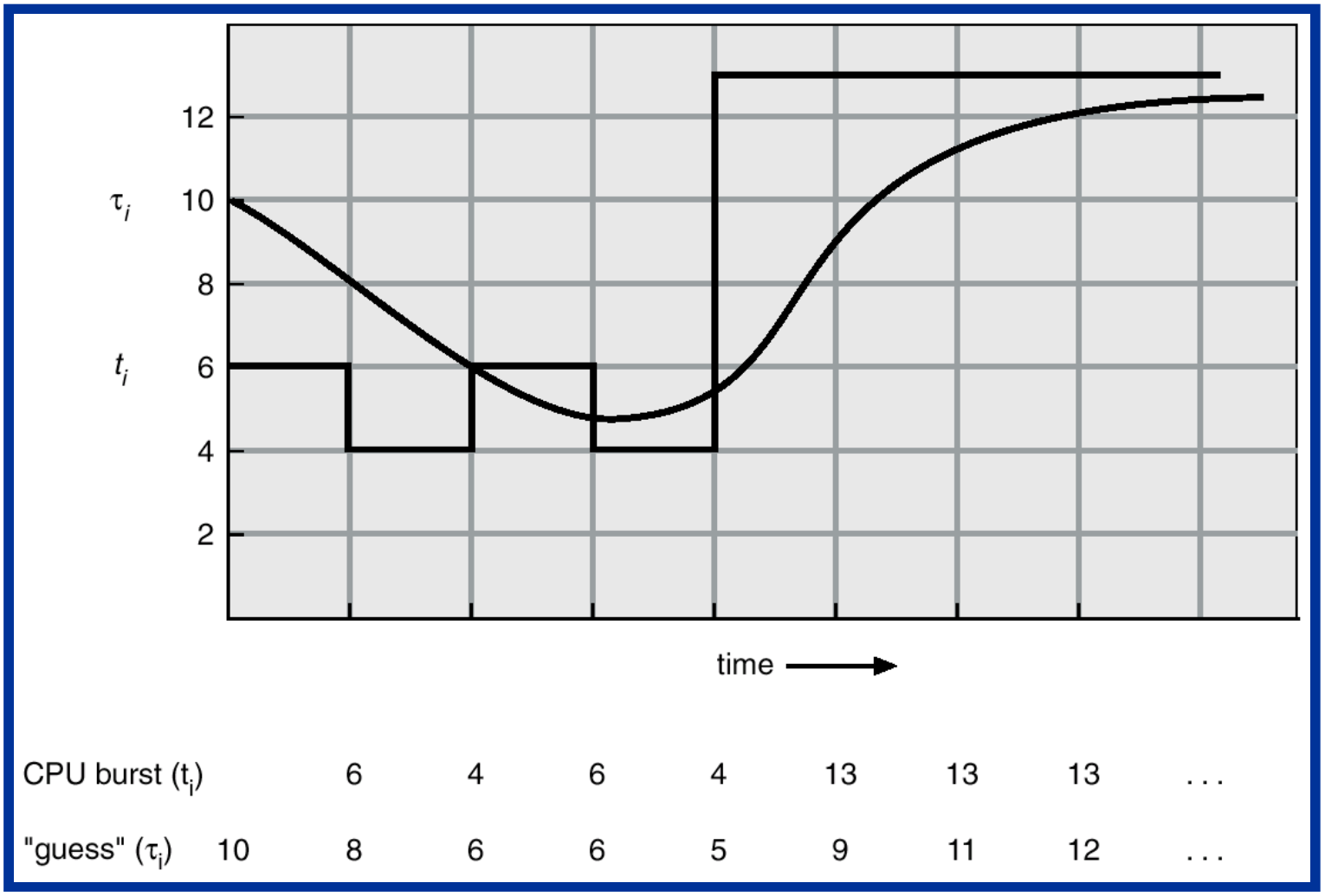
Determinar la duración de la próxima ráfaga de CPU

- No es posible determinar la duración pero sí predecirlo
- Se puede hacer usando la duración las ráfagas anteriores de CPU, usando una media exponencial

1. t_n = duración real de la ráfaga de $n-ésima$
2. τ_{n+1} = valor predecido de la próxima ráfaga de CPU
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define:

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n.$$

Predicción de la duración de la próxima ráfaga de CPU



Ejemplos de media exponencial

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - La historia reciente no cuenta
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Sólo cuenta la última ráfaga de CPU
- Si expandimos la fórmula, obtenemos:

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\,\alpha\, t_{n-1} + \ldots$$
$$+ (1-\alpha)^j \alpha\, t_{n-1} + \ldots$$
$$+ (1-\alpha)^{n+1} t_n \tau_0$$

- Como tanto α como $(1-\alpha)$ son menores o igual a 1, cada término sucesivo tiene menos peso que su predecesor

Planificación con prioridad

- Cada tarea se asocia con una prioridad
- Igual prioridad servida en orden FCFS
- Las prioridades pueden ser influenciadas por requerimientos internos también (p.e. Número de ficheros abiertos, media de ráfagas de CPU)
- Un ordenador muy cargado puede que nunca ponga en la CPU procesos de baja prioridad (bloqueo/inanición indefinido)
  - El envejecimiento puede ayudar a prevenir el bloqueo indefenido incrementando la prioridad también con respecto al tiempo de espera

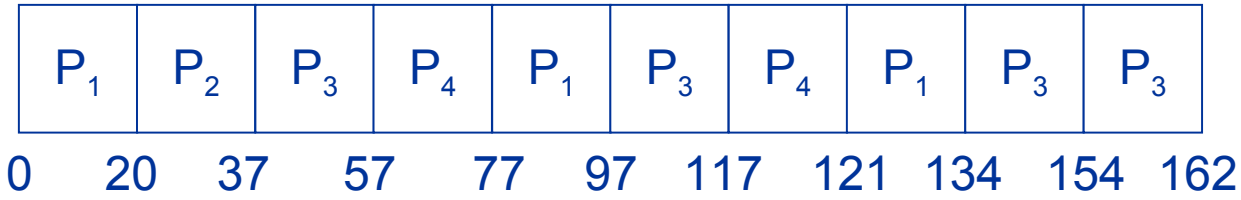
Round Robin (RR)

- Diseñado para sistemas de tiempo compartido
- Similar a FCFS pero la expropiación se añade
- Se define un tiempo de "quantum" *q* (típicamente entre 10-100ms)
- La CPU da a cada entrada en la lista un tiempo de quantum
- La cola de "preparados" se organiza como una FIFO
- Los nuevos procesos se añaden al final de la cola
- Ejecución
  - Con un tiempo pequeño de quantum, los *n* procesos funcionan como un procesador con una potencia de procesado *1/n*
  - Con un tiempo grande de quantum, los *n* procesos funcionan de forma similar a FCFS
  - Ningún proceso espera más de *(n-1)*q*
  - El esfuerzo para el cambio de contexto se debe considerar

Ejemplo de RR con quantum = 20

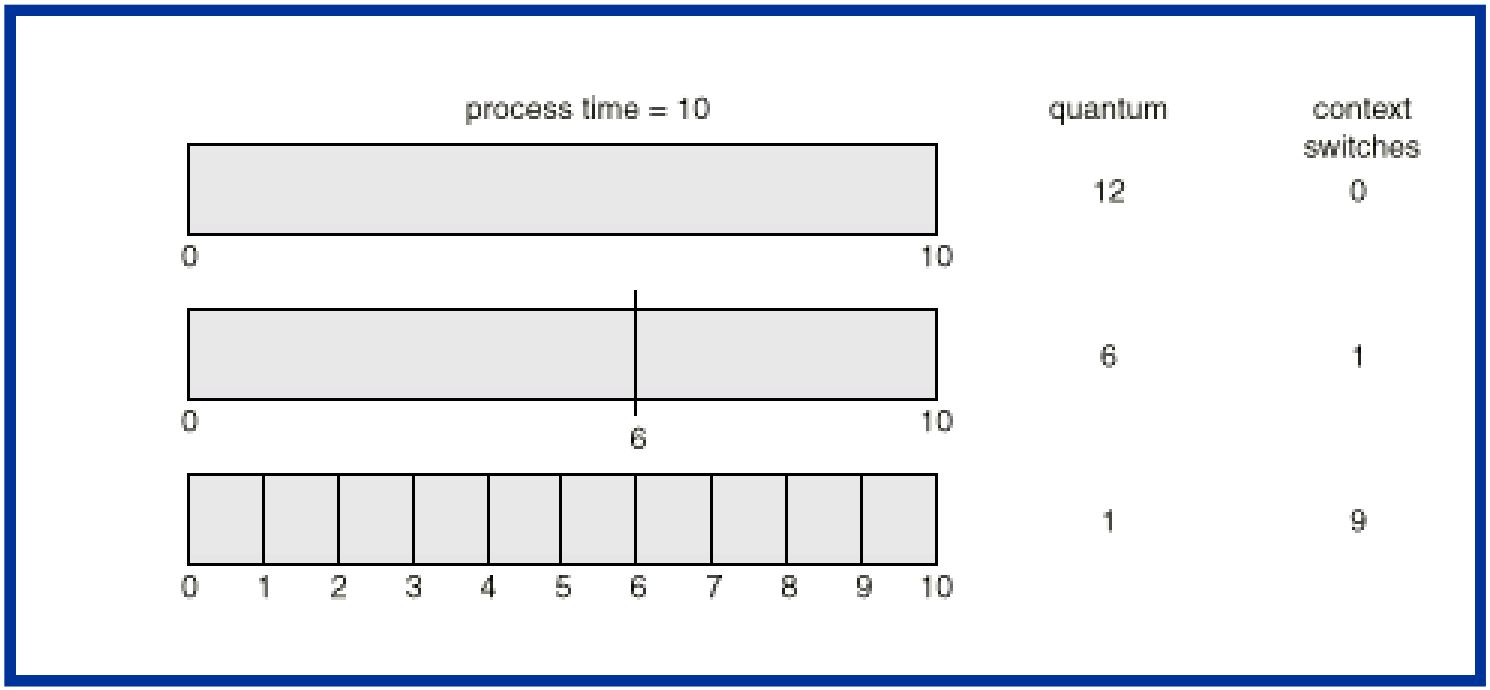
Proceso	Fase de CPU
P_1	53
P_2	17
P_3	68
P_4	24

- El diagrama de Gantt es:

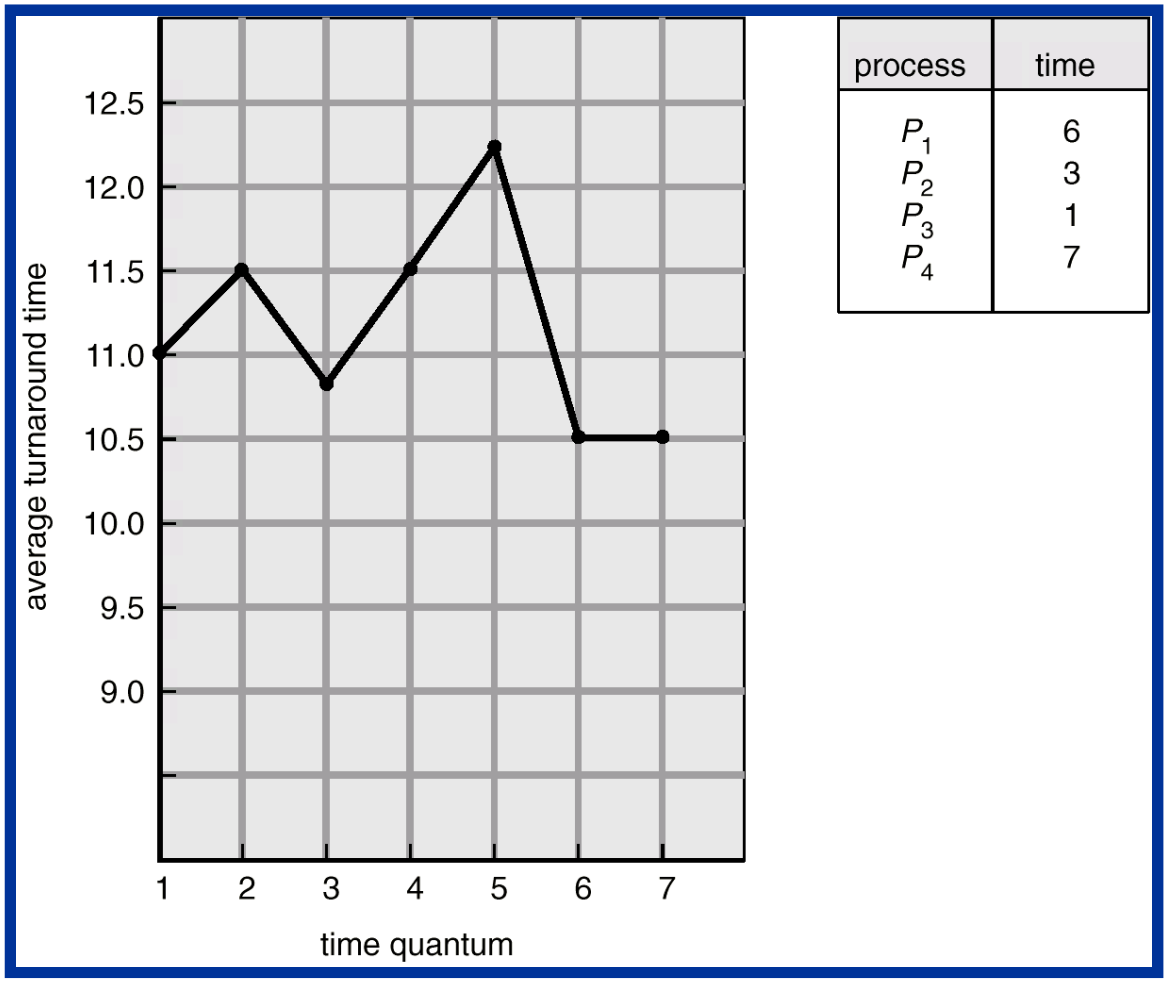


- Típicamente tiene una media más alta de tiempo que SJF pero obtiene una mejor respuesta

Tiempo de quantum y tiempo de cambio de contexto



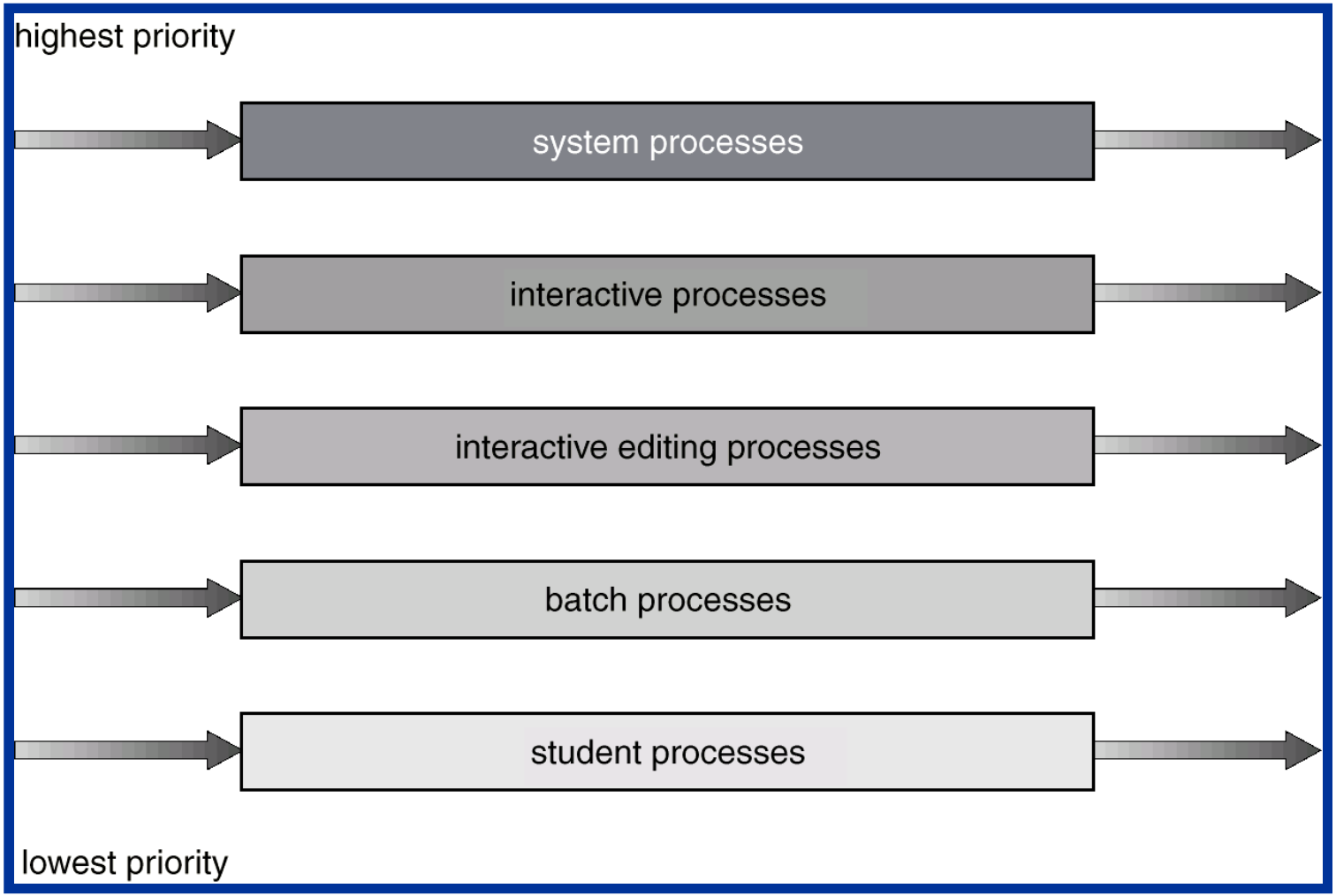
Influencia: tiempo de procesamiento y tiempo de quantum



Colas multinivel

- Se clasifican los procesos en grupos
  - Procesos en primer plano (interactivos)
    - Tiempo de respuesta rápido (p.e. Algoritmo RR)
  - Procesos en segundo plano (“background”) (lotes)
    - Tiempo de respuesta más largo (p.e. Algoritmo FCFS)
- La planificación entre colas es necesaria también
  - Arregla la planificación de prioridad; (p.e., sirve a todos los de primer plano y después a todos los de “background”). Posibilidad de inanición.
  - Intervalo de tiempo – cada cola obtiene un cierto tiempo de CPU que se puede planificar entre procesos; p.e., 80% para los procesos en primer plano en RR
  - 20% para “background” en FCFS

Colas multinivel: Ejemplo



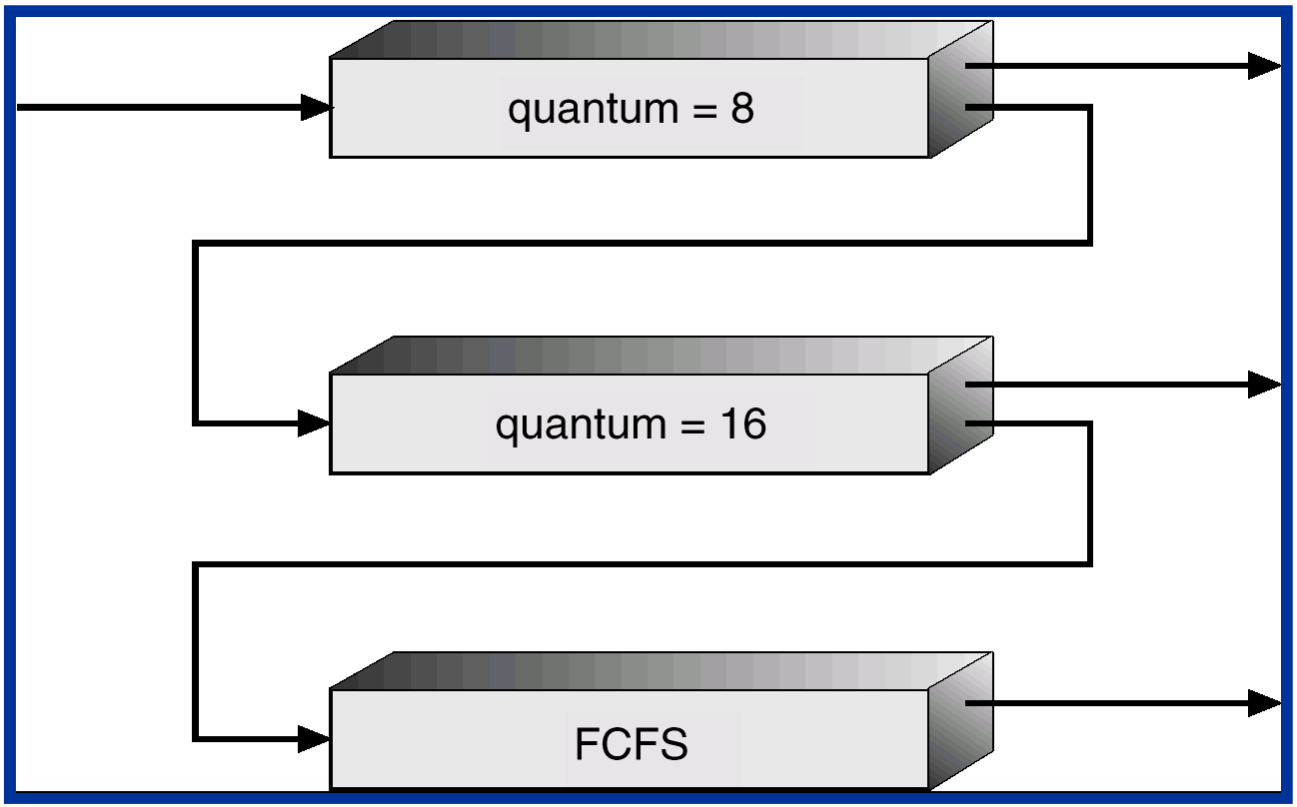
Colas multinivel realimentadas

- Se permite que los procesos se muevan entre colas (más flexible comparado con la última técnica mencionada)
- Idea
  - Procesos separados con diferentes características de ráfaga de CPU
- Parámetros de planificación
  - Número de colas
  - Algoritmos de planificación para cada cola
  - Método usado para determinar cuando ascender un proceso
  - Método usado para determinar cuando degradar un proceso
  - Método usado para determinar en qué cola entrará un proceso cuando el proceso necesite servicio

Colas múltiples realimentadas : Ejemplo

- Tres colas:
  - Q_0 – Tiempo de quantum 8 milisegundos
  - Q_1 – Tiempo de quantum 16 milisegundos
  - Q_2 – FCFS
- Planificación
  - Entra una nueva tarea en la cola Q_0 que es servida por FCFS. Cuando gana la CPU, la tarea recibe 8 milisegundos. Si no termina en 8 milisegundos, la tarea se cambia a la cola Q_1.
  - En la cola Q_1 se sirve otra vez la tarea por FCFS y recibe 16 milisegundos adicionales. Si todavía no ha terminado, es expropiada y cambiada a la cola Q_2.

Colas múltiples realimentadas: Ejemplo (cont.)



Planificación en tiempo real

- Sistemas de *tiempo real duro* – requieren terminar una tarea crítica dentro de una cantidad de tiempo garantizada
- Sistemas de *tiempo real blando* – requieren que los procesos críticos reciban una prioridad por encima de los menos afortunados

Planificación en tiempo real

Sistema en tiempo real panificable

- Dados
  - *m* eventos periódicos
  - El evento *i* ocurre con período P_i y requiere C_i segundos
- Entonces la carga sólo puede ser manejada si

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

Evaluación de algoritmos

- Modelo determinístico – toma una carga de trabajo particular predeterminada y define la ejecución de cada algoritmo para esa carga de trabajo
- Modelos de colas
  - Fórmula de Little
    - Sea *n* la longitud media de la cola
    - *W* el tiempo de espera medio
    - Y λ el tiempo de llegada medio para los nuevos procesos

$$n = \lambda \times W$$

- Simulación
- Implementación