Sistemas Operativos 1

Procesos

Los primeros sistemas de computadoras:

- Permitían sólo ejecutar un programa a la vez
- Este programa tenía control completo del sistema y sus recursos

Hoy:

- Múltiples programas pueden cargarse en memoria y ser ejecutados en forma concurrente
- Requiere un control más estricto

Procesos

Los procesos en ejecución son:

- Procesos del sistema operativo ejecutando código del sistema
- Procesos del usuario ejecutando código del usuario

Proceso – un programa en ejecución;

- Programa (ejecutable) – entidad pasiva en almacenamiento secundario
- Proceso – entidad activa

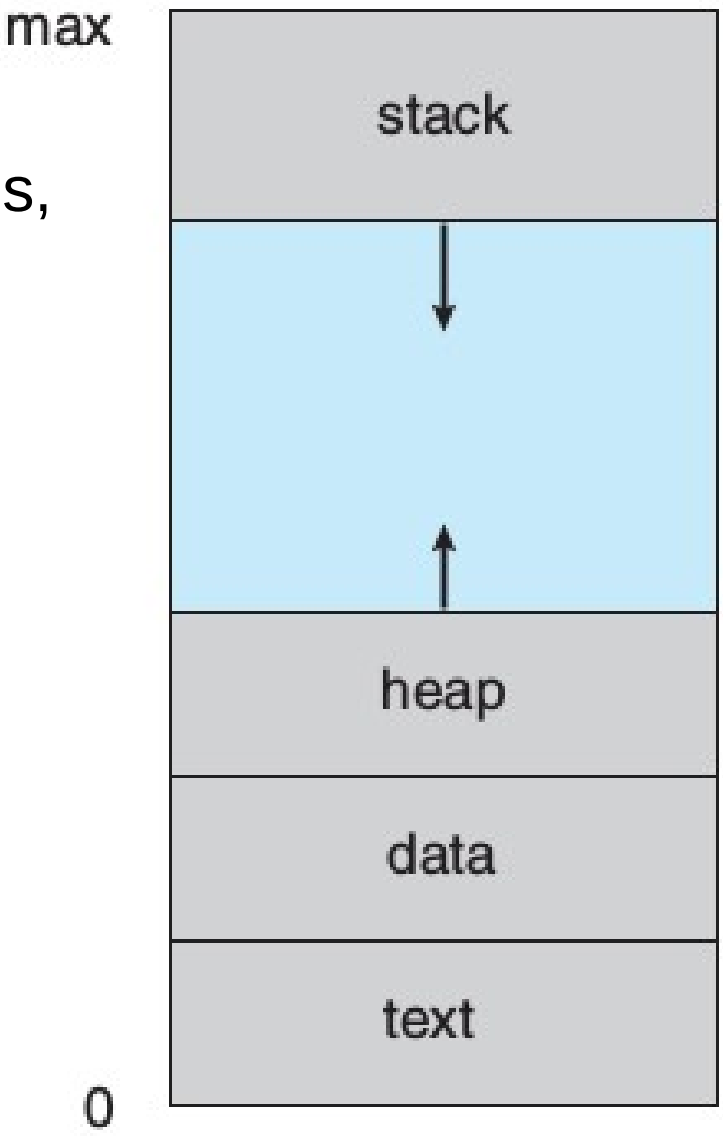
- Los procesos en ejecución deben progresar en forma secuencial

Procesos

Un proceso (o job) incluye:

- **Stack:** Datos temporales (parámetros, variables locales, valores de retorno)
- **Heap:** Memoria alocada

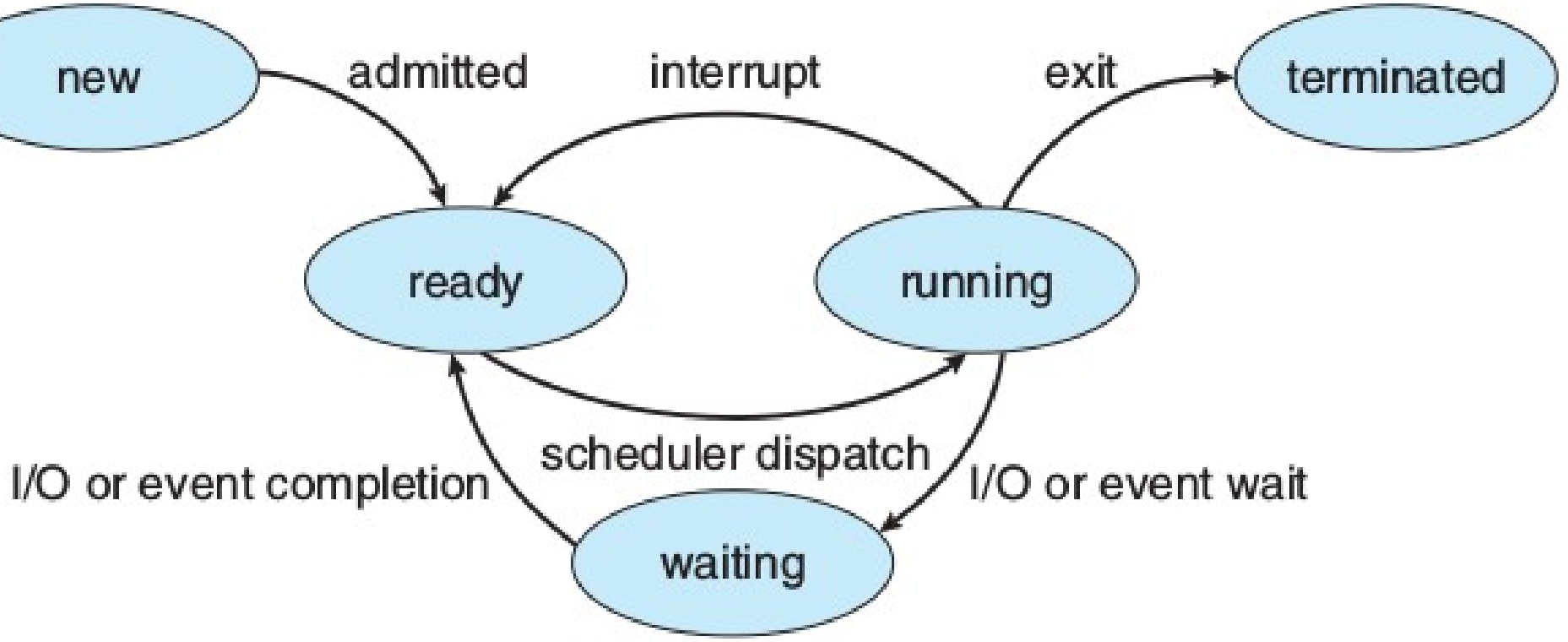
  dinamicamente durante la ejecución
- **Segmento de Datos:** Variables globales
- **Segmento de texto:** Código compilado del programa



Procesos: Estados

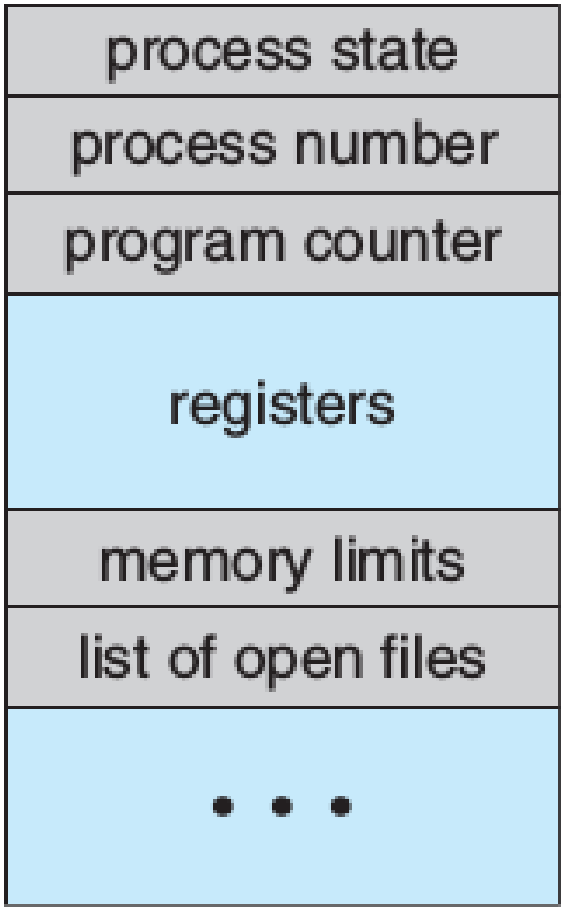
- **New:** El proceso está siendo creado.
- **Running:** Se están ejecutando las instructiones.
- **Waiting:** El proceso está esperando por un evento (ej: finalización de I/O o recepción de una señal).
- **Ready:** El proceso está esperando ser asignado al procesador.
- **Terminated:** El proceso finalizó su ejecution.

Procesos: Estados



PCB: Process Control Block

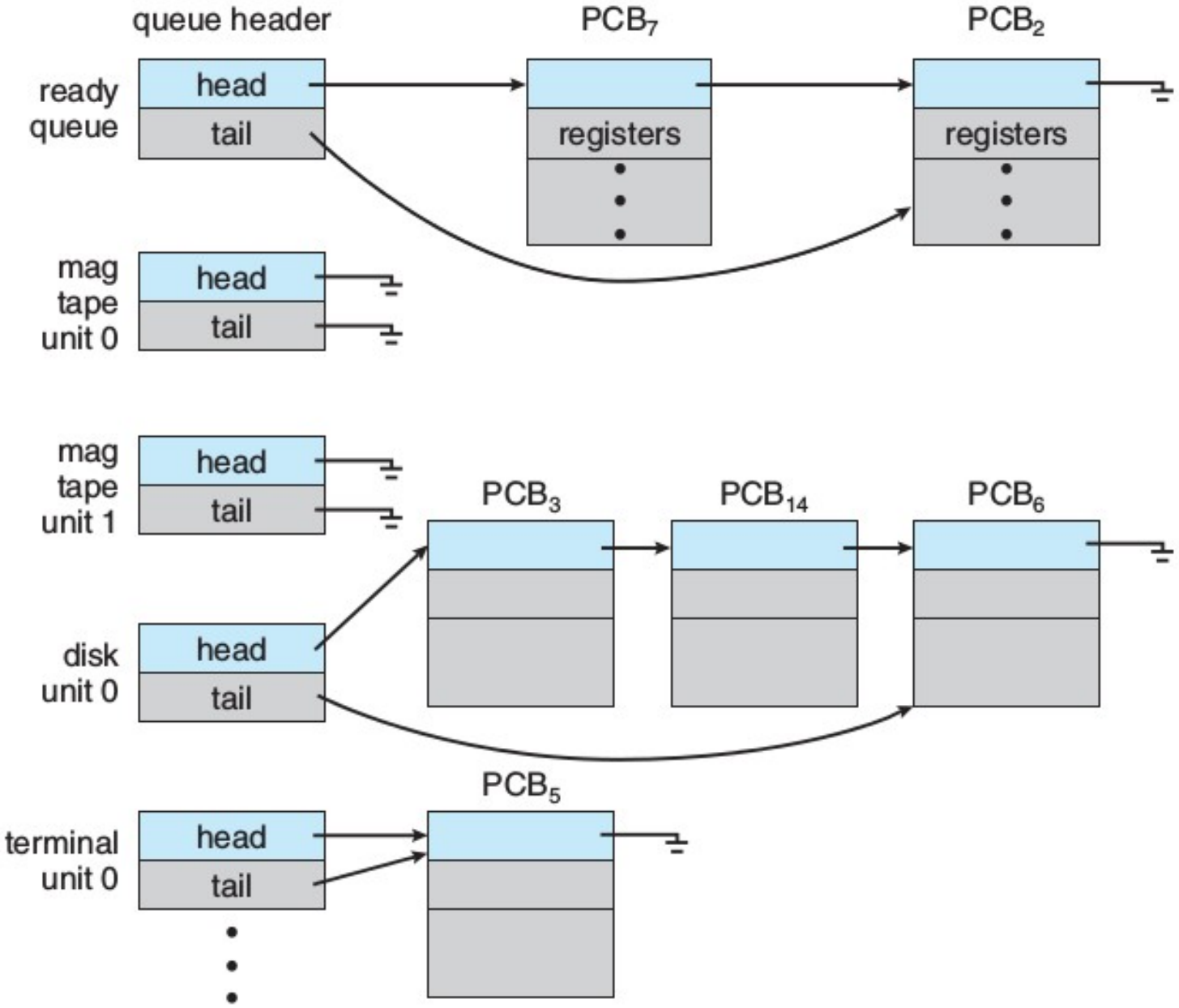
- Estado del proceso: [new | ready | running | …]
- PID: Process ID
- Contador del programa– dirección de la siguiente instrucción a ejecutar
- Registros de CPU
- Información de scheduling de CPU – ej: prioridad del proceso
- Información de administración de memoria – valores de los registros base y límite, tablas de página o segmento
- Información de auditoría – ej: tiempo de CPU
- Información de estado de E/S – lista de archivos abiertos, dispositivos de E/S asignados a procesos



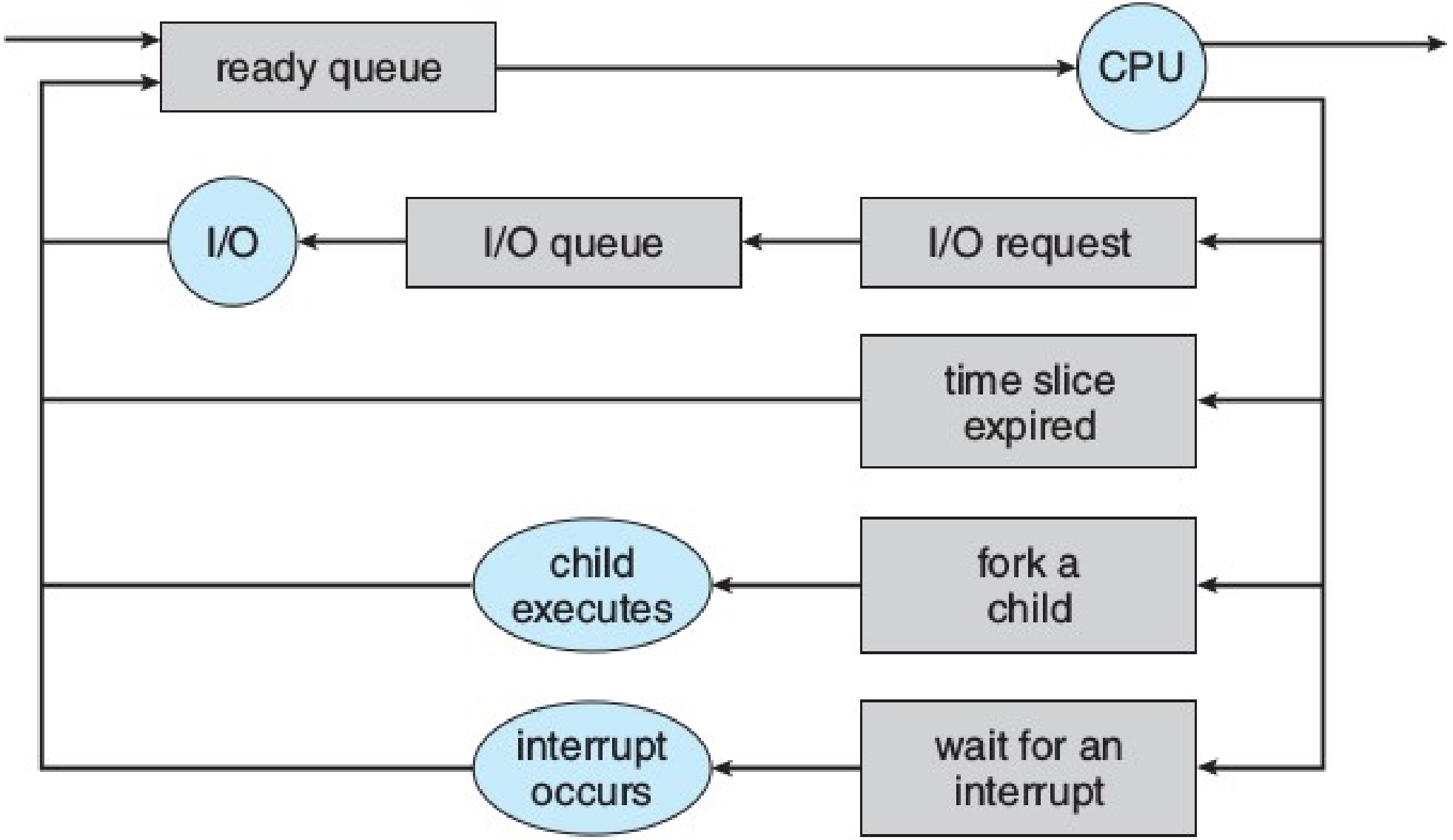
Colas de Scheduling de Procesos

- Objetivo – tener procesos corriendo todo el tiempo para maximizar el uso de CPU
- Para cumplir el objetivo se usa un scheduler de procesos
- Cola de **tareas** – conjunto de todos los procesos del sistema
- Cola de **listos** – conjunto de todos los procesos en memoria, listos y esperando para ejecutar
- Colas de **dispositivos** – conjunto de todos los procesos esperando por un dispositivo de E/S
- Los procesos migran entre las distintas colas

Ready Queue (y Devices Queues)



Scheduling Queues



Schedulers

Scheduler = Planificador

- **Scheduler de Largo Plazo (o scheduler de tareas):** selecciona que proceso puede incluirse en la cola de procesos listos (ready queue)– carga los procesos de disco a memoria
- **Scheduler de Corto Plazo (o scheduler CPU):** selecciona que proceso debe ejecutar a continuación (de la cola de ready) y asigna la CPU
- El scheduler de corto plazo se invoca muy frecuentemente (milisegundos) debe ser rápido, de otra forma crea un overhead sustancial.
- El scheduler de largo plazo se invoca muy infrecuentemente puede ser lento.

Schedulers (cont.)

- El scheduler de largo plazo controla el grado de multiprogramación (número de procesos en memoria)
- Los procesos se pueden clasificar como:
  - **Limitados por E/S:** pasan más tiempo haciendo E/S que computaciones, muchas ráfagas de CPU cortas
    - ej: navegando la web, copiando archivos grandes
  - **Limitados por CPU:** pasa la mayor parte haciendo computaciones; ráfagas de CPU largas
    - ej: cálculo de Pi

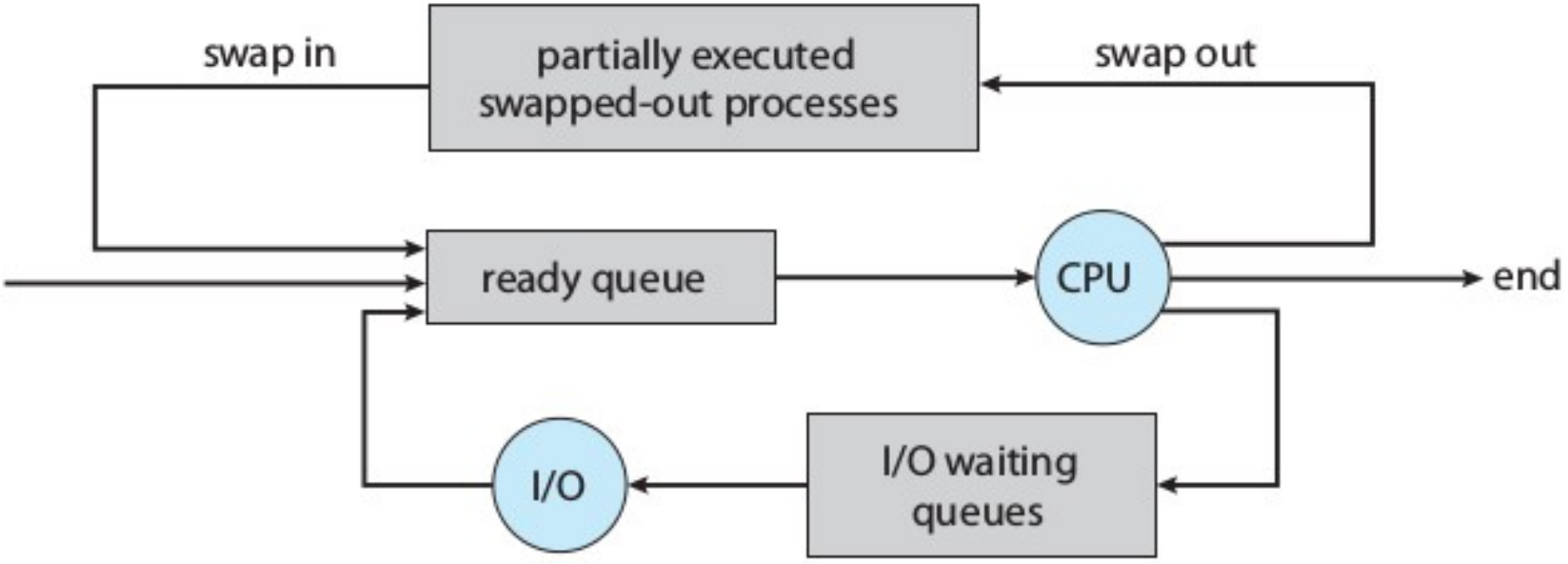
Schedulers (cont.)

- Es importante para el scheduler de largo plazo seleccionar una buena mezcla de procesos limitados por E/S y limitados por CPU
- Qué sucede con las colas de listos y E/S:
  - Si todos los procesos son limitados por E/S?
  - Si todos los procesos son limitados por CPU?
- Que sucede si no tenemos planificador de largo plazo?

Some time-sharing systems such as UNIX and Microsoft Windows systems often have no long-term scheduler but simply put every new process in memory for the short-term schedule

Mid-term scheduler (Swapping)

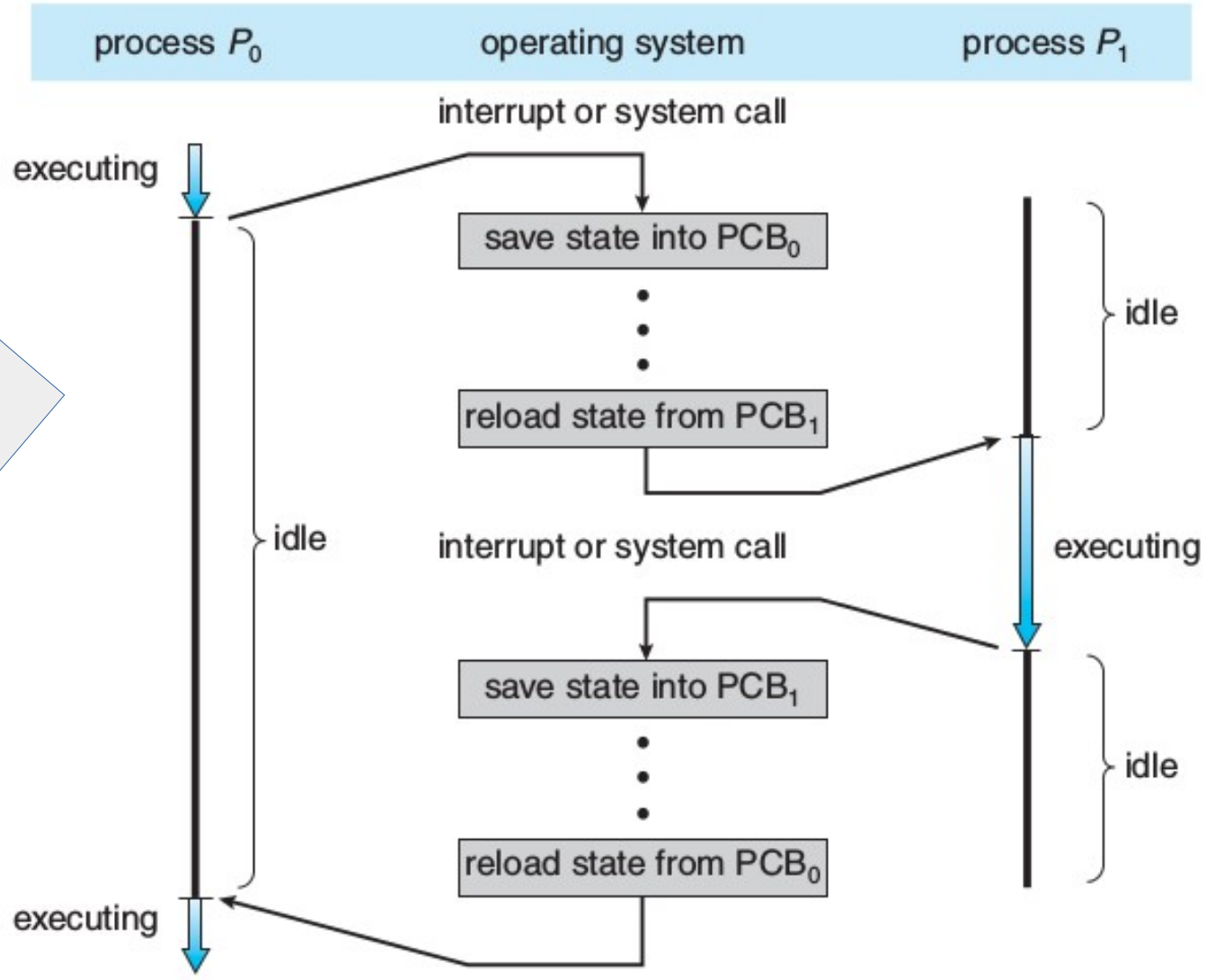
- Remueve procesos de memoria (reduce el grado de multiprogramming y libera memoria).
- Ayuda a tener un mix de procesos y soportar cambios de requerimientos de memoria.



Context Switch

- Cuando la CPU cambia de un proceso a otro, el sistema debe guardar el estado del proceso viejo y cargar el estado guardado del nuevo proceso vía un cambio de contexto
- El contexto de un proceso es representado por el PCB
- El tiempo de cambio de contexto es un overhead; el sistema no está realizando ninguna tarea útil durante el cambio
- Dependiente del soporte de hardware
  - P. ej., la Sun UltraSPARC provee múltiples juegos de registros

Multitasking: Context Switch



Overhead

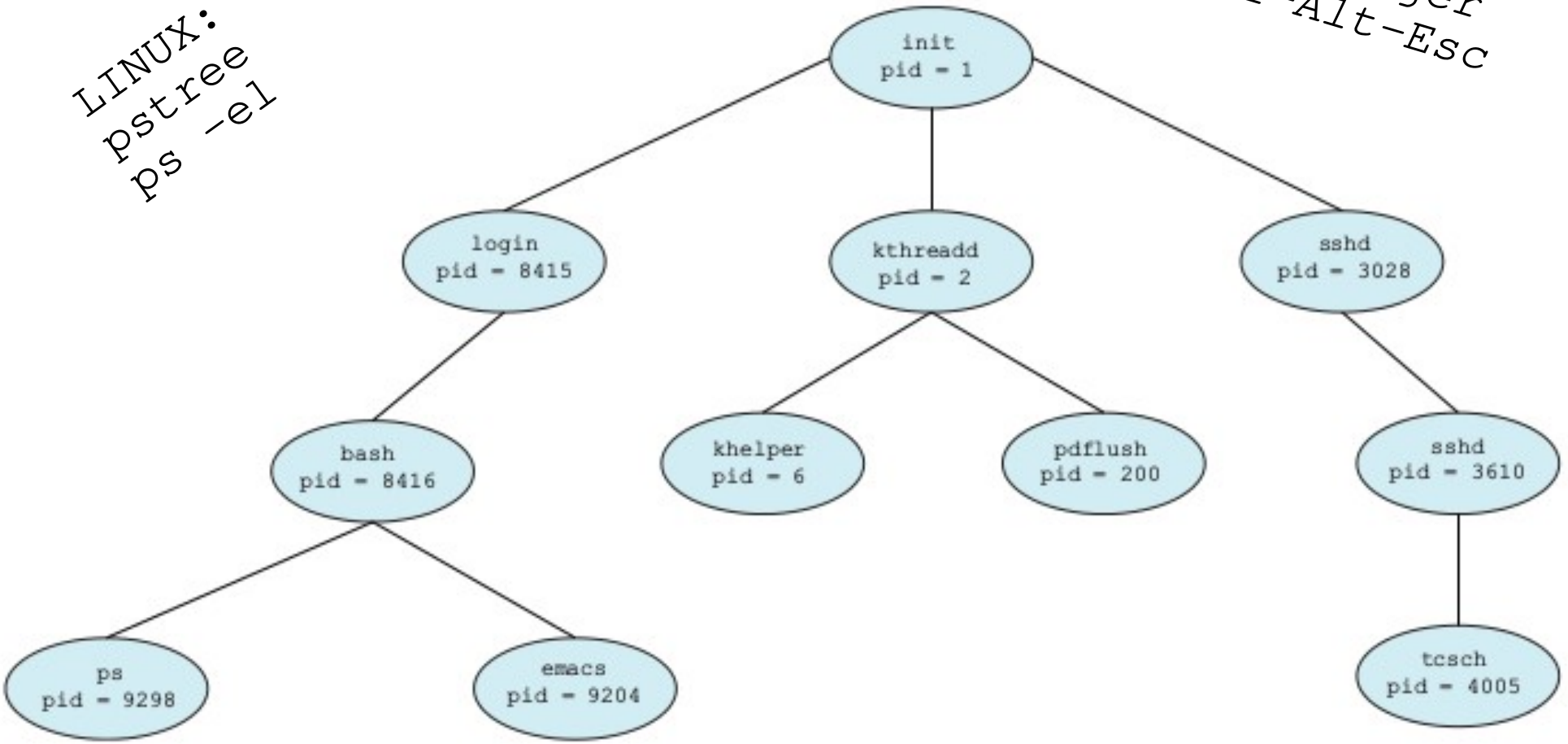
Process Creation

- Creación de procesos – se utiliza la llamada al sistema de creación de procesos
- El proceso padre crea procesos hijos, los cuales, a su vez crean otros proceso, formando un árbol de procesos
- Generalmente, los procesos son identificados y manejados a través de un identificador de proceso (pid)

Process Tree

LINUX:
pstree
ps -el

WINDOWS:
TaskManager
Crtl-Alt-Esc



Process Creation

- Recursos compartidos (tiempo de CPU, memoria, dispositivos de E/S)
  - Padre e hijo comparten todos los recursos
  - El hijo comparte un subconjunto de los recursos del padre
  - Padre e hijo no comparten recursos
    - Un proceso puede sobrecargar el sistema creando muchos procesos
- La información de inicialización se pasa de padre a hijo en la creación

Process Creation (cont)

- Ejemplos en Unix
  - La llamada al sistema fork() crea un nuevo proceso
    - El espacio de memoria del padre es duplicado
    - Ambos procesos continúan la ejecución en la instrucción posterior al fork()
  - La llamada al sistema exec() es usada luego de un fork() para reemplazar el espacio de memoria del proceso con un nuevo programa
  - wait() saca un proceso de la cola de listos hasta que el hijo termine

Process Creation (cont)

- Ejecución
  - Padre e hijo ejecutan concurrentemente
  - El padre espera que el hijo termine
- Espacio de memoria
  - El hijo duplica el espacio de memoria del padre mismo código y datos
  - El hijo carga un nuevo programa

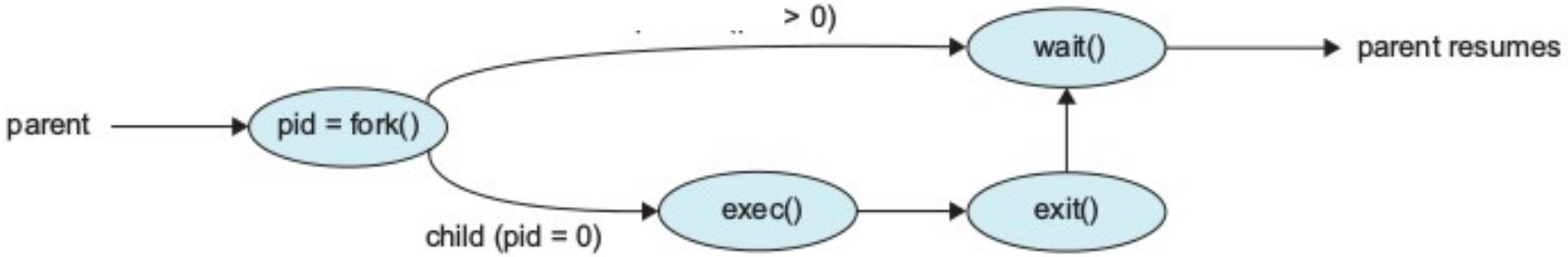
```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 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");
    }

    return 0;
}
```



Process Termination

- Un proceso ejecuta su última instrucción y pide al sistema operativo que lo elimine (exit)
  - Información de salida del hijo al padre (via wait)
  - Los recursos de un proceso son des-asignados por el SO
- Un padre puede terminar la ejecución de un proceso hijo (abort)
  - El hijo se excedió en los recursos asignados
  - La tarea del hijo no se requiere más (ej: un browser)
  - Si el padre termina
- Algunos sistemas operativos no permiten que un hijo continúe si el padre termina.
  - Todos los hijos son terminados (cascading termination)