Universidad Católica Argentina

Sistemas Operativos

Lic. Ing. Osvaldo Clúa

Procesos

Lineamientos para un S.O.

Dado que las funciones principales del S.O. son por un lado mediar entre el usuario del sistema de computación y su hardware, y por otro, administrar los recursos del sistema.

¿Cómo se implementaría la ejecución de un programa?

Modelo de Procesos

- El Sistema operativo debe organizar el software que corre en unidades secuenciales: los Procesos.
- Un proceso es entonces:
  - La imagen de un programa en ejecución.
    - La imagen es una copia del programa.
  - Con las estructuras del Sistema Operativo para administrarlo

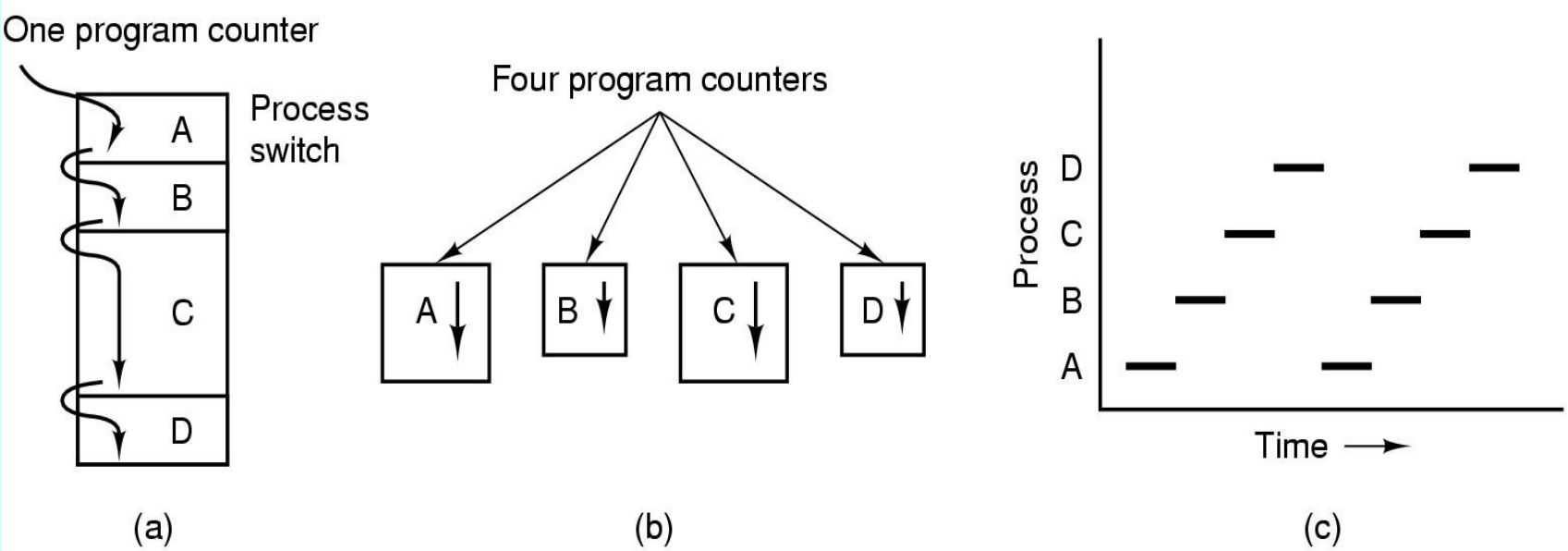
¿Qué tiene un proceso?

- La imagen del programa (una copia de su código ejecutable y de su área de datos).
- La información acerca de sus estado de ejecución:
  - Los valores del *program counter*, registros y variables.
  - Información necesaria para su administración por parte del Sistema Operativo (id, prioridad, ...).

Multiprogramación

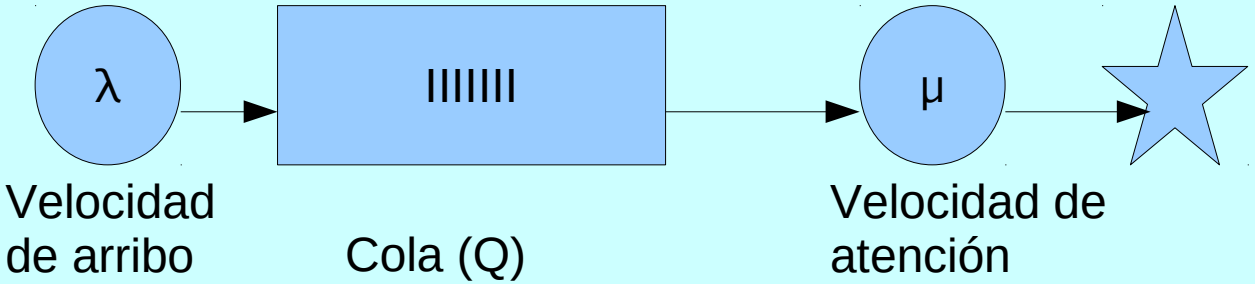
- La diferencia de velocidades CPU-I/O es del orden de 10^3 o 10^4.
- Esto significa mucho tiempo ocioso mientras se completa una operación de I/O.
- Para aprovecharlo aparece la Multiprogramación.
- Cuando hay mas de un procesador se conoce como Multiprocesamiento.

Multiprogramación (2)



- La CPU va conmutando *(switching)* de un proceso a otro.
- Es un *multiplexado* de la CPU.

¿Quién gana con la Multiprogramación?



- Modelo de una estación de servicio a clientes.
- μ y λ son magnitudes aleatorias [Clientes/Segundo].
- $\rho = \lambda / \mu$ es el factor de uso.

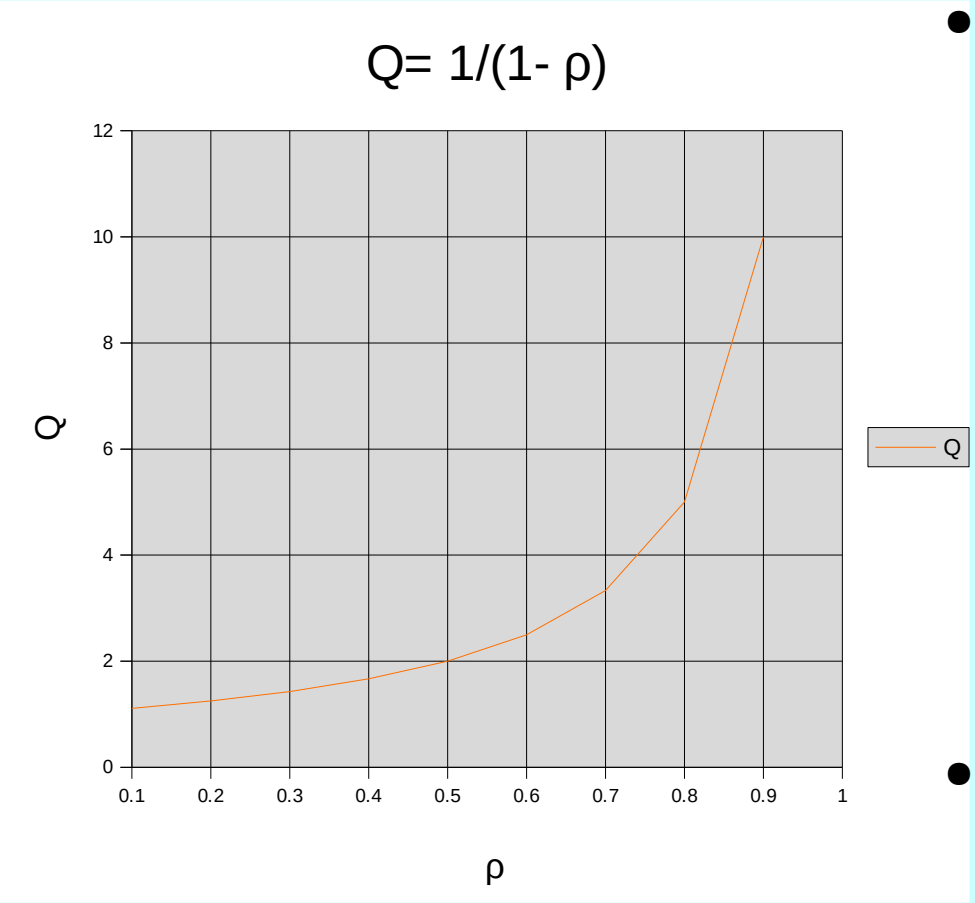
¿Quién gana con la Multiprogramación? (2)

- De la teoría de colas :
  - se trata de una configuración M/M/1.
    - M -Arriban según un proceso de Markov (probabilidad exponencial que se corresponde con Poisson).
    - M -Se atienden según un proceso de Markov.
    - 1 -Hay un solo centro de atención
- El número de elementos en el sistema es:
  - $N = \rho/(1-\rho)$

¿Quién gana con la Multiprogramación? (3)

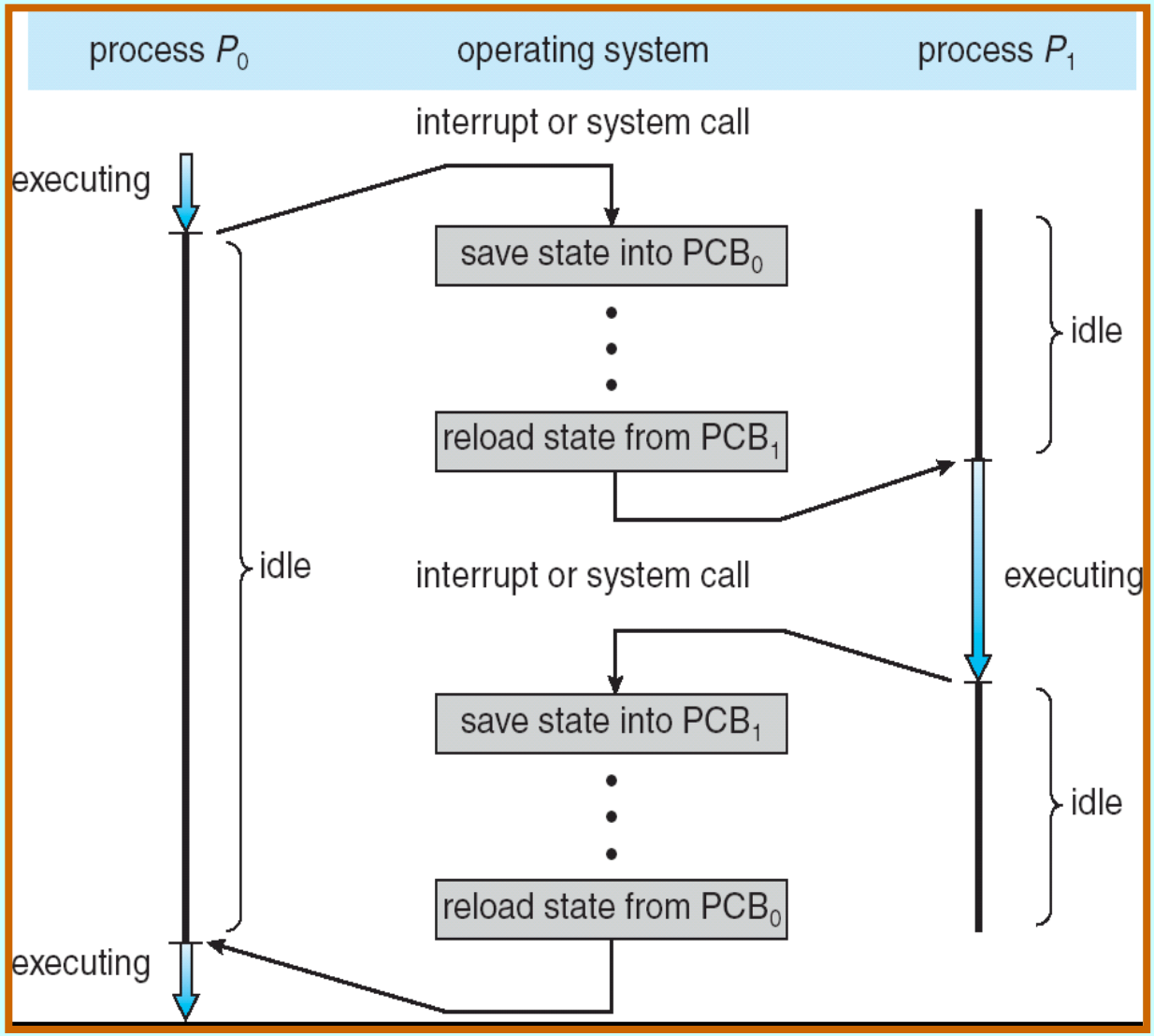
- En el sistema hay (valor esperado, esperanza matemática)
  - N= $\rho/(1-\rho)$ clientes
    - ρ en servicio + $\rho^2/(1-\rho)$ en espera.
  - El tiempo de respuesta es $N/\lambda = 1/(1-\rho) * (1/\mu)$
  - El tiempo de espera es $(N/\lambda) - (1/\mu)$
  - Como un cliente tarda $(1/\mu)$ se define:
    - Q = $1/(1-\rho)$ como multiplicador de cola. (en cuanto me afecta la existencia de una cola)

¿Quién gana con la Multiprogramación? (4)

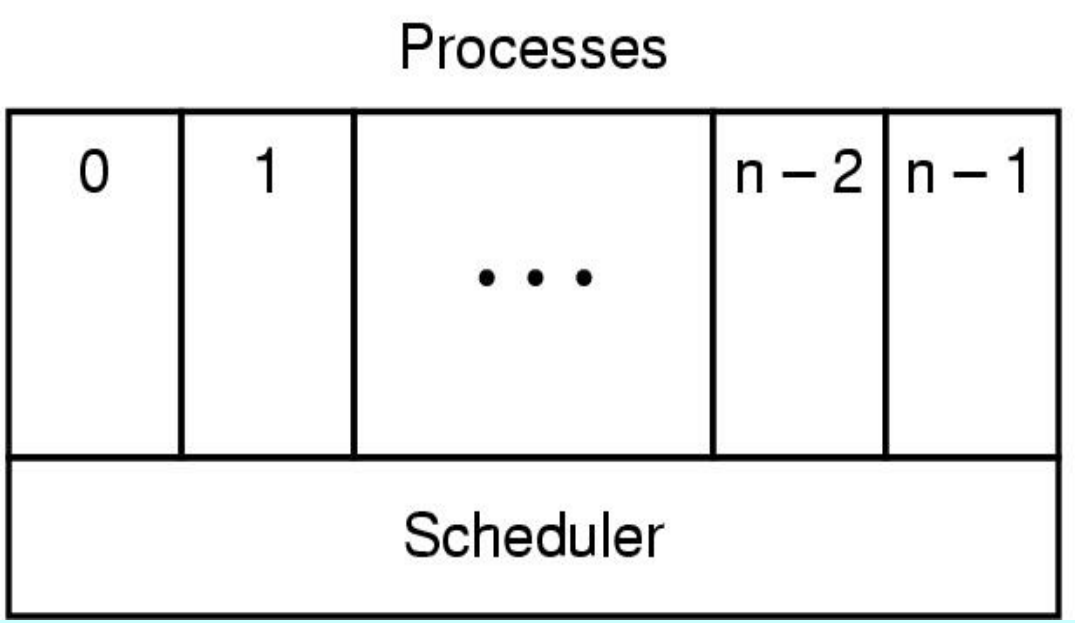


- Interprete este gráfico:
  - ¿Cuál es la carga del procesador?
  - ¿Cómo se comporta la espera?
- Y... ¿Quién gana con la Multiprogramación?

Implementación de la Multiprogramación

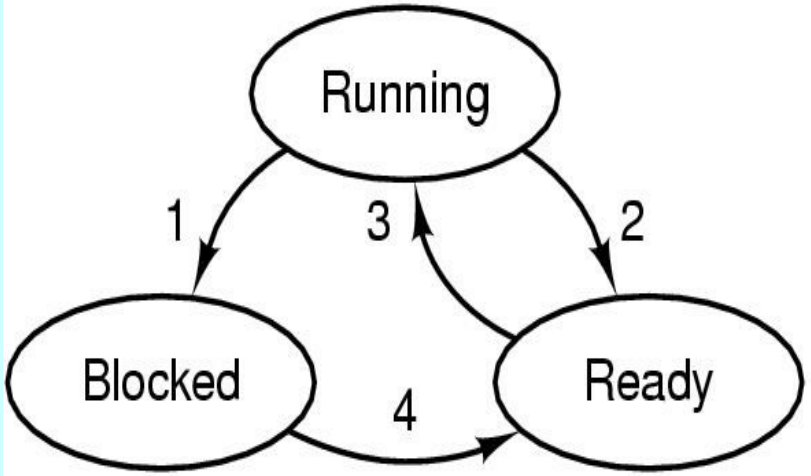


Implementación de la Multiprogramación (2)



- El *Scheduler* decide a que proceso dar el control
- El *dispatcher* realiza el cambio de estado

Estados de Un Proceso



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

- Durante su ejecución, un proceso pasa por distintos estados.
- Faltan estados para la creación y la destrucción de los procesos.

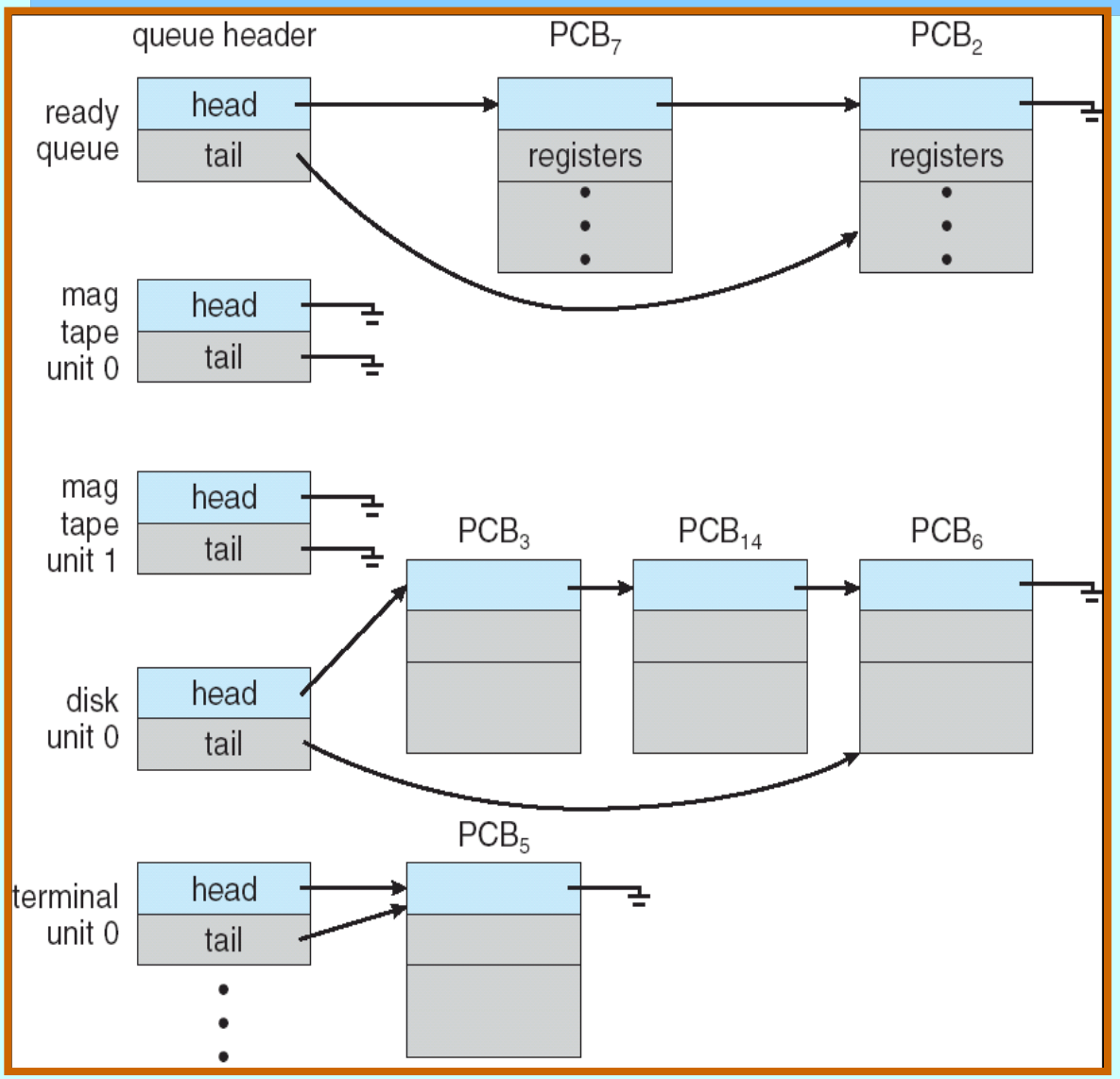
PCB (Process Control Block)

- Es la estructura de datos con la que el sistema operativo administra los procesos.
- Contiene la información acerca del proceso y su estado.
- Además la información que el S.O. precisa para manejarlo como:
  - Identificador, Estado, Recursos, Historia.

PCB

Process management	Memory management	File management
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Estados de un Proceso



- Los estados se manejan como colas.
- El dispatcher es el encargado de cambiar los PCBs entre las colas.

Dispatcher (Short Term Scheduler)

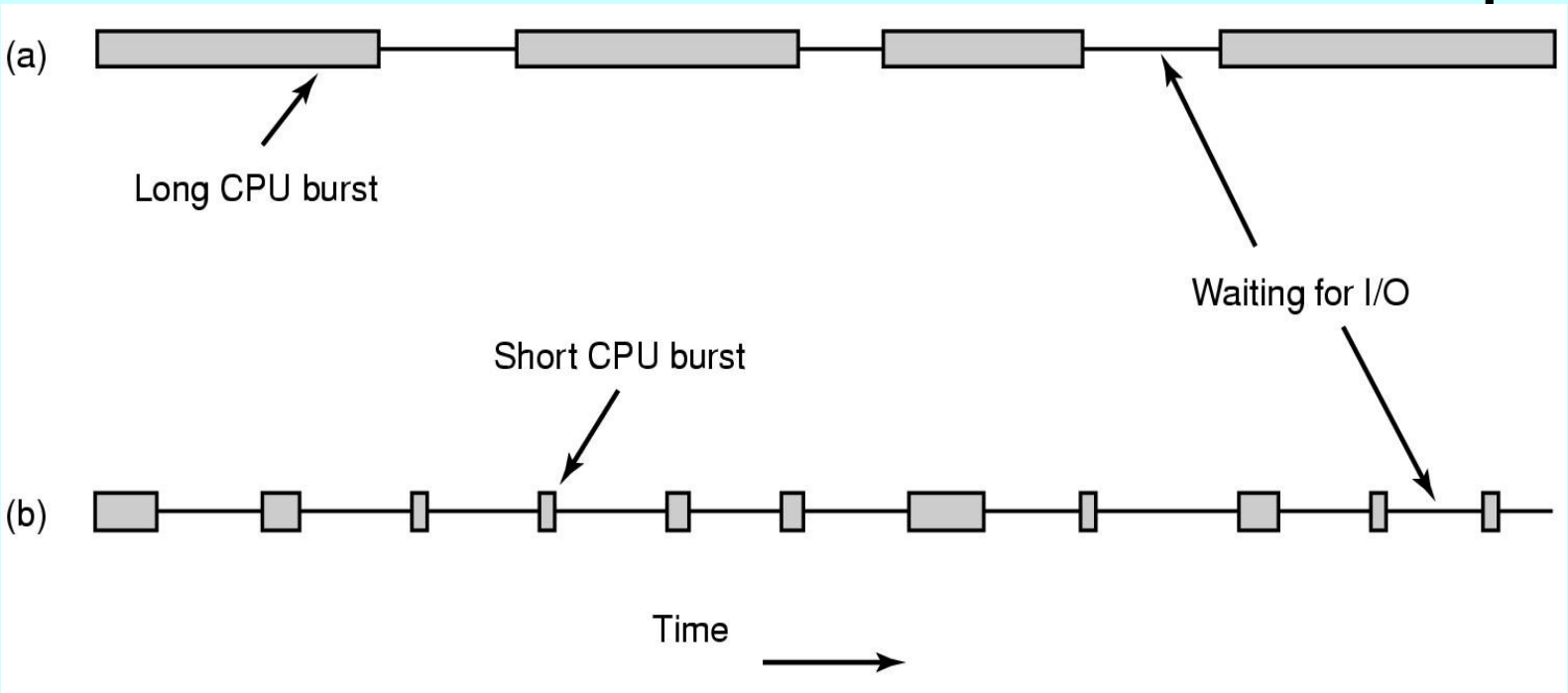
- Al pasar de Running a Blocked.
  - El manejador de interrupciones lo invoca para cambiar de estado al proceso:
    - Salva los datos necesarios en el PCB.
    - Cambia el PCB de cola.
  - Luego se decide a que proceso dar control (tarea del Scheduler).

Dispatcher (Short Term Scheduler) (2)

- Al pasar de Ready a Running
  - El Scheduler lo invoca cuando ya decidió a que proceso activar.
  - Carga el estado de la CPU con los datos del PCB.
  - Continua la ejecución del proceso.
- ¿Qué tareas tiene en el resto de las transiciones?

Scheduler (Long term)

- Decide a cuál de los procesos en ready hay que darle el control.
- Tiene en cuenta las características del proceso



Objetivos del Scheduler

- Dar una participación adecuada del reparto de tiempo de CPU (Fairness).
- Equilibrar el uso de recursos (Load Balancing).
- Aplicar las políticas generales del Sistema (prioridades, afinidad, seguridad).
- El resto depende del tipo de Sistema.

Objetivos del Scheduler (2)

- Batch (por lotes):
  - maximizar el throughput
  - Mantener la CPU ocupada.
  - Minimizar el turnaround time.
- Interactivo:
  - buen tiempo de respuesta.
  - Expectativas del usuario
- Real time:
  - Cumplir con los deadlines.
  - Desempeño predecible.

Llamados al Scheduler

- Las decisiones de scheduling se pueden tomar cuando un proceso:
  1. Pasa de running a blocked/waiting.
  2. Pasa de running a ready.
  3. Pasa de blocked/waiting a ready.
  4. Termina.
- Las transiciones 1 y 4 son no-apropiativas (*nonpreemptive).*
- El resto son apropiativas (*preemptive).*

Ejemplos de algoritmos de scheduling

- First come-First served
- Shortest Job Next
- Round Robin (aparece el concepto de time-slice y quantum)
- Múltiples colas con Prioridad

- Ver cap 5 del libro de Silberschatz (Operating Systems Concepts).
- Ver cap 2 del libro de Tanenbaum (Modern Operating Systems).

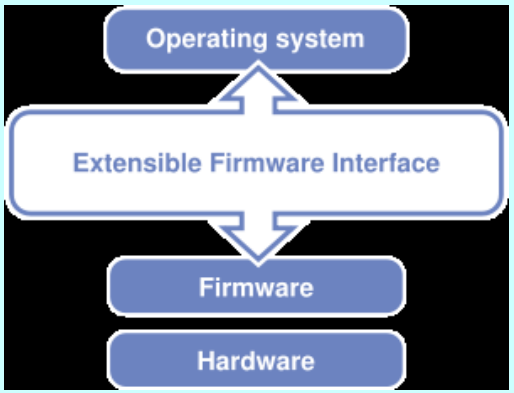
Creación/Terminación de Procesos

- Creación de Procesos:
  - Al iniciar el sistema.
  - Por pedido del usuario (uso de una System Call).
- Terminación de Procesos
  - Salida normal (voluntaria).
  - Salida por error (voluntaria).
  - Error "fatal" (involuntaria).
  - "Muerte" por otro proceso.

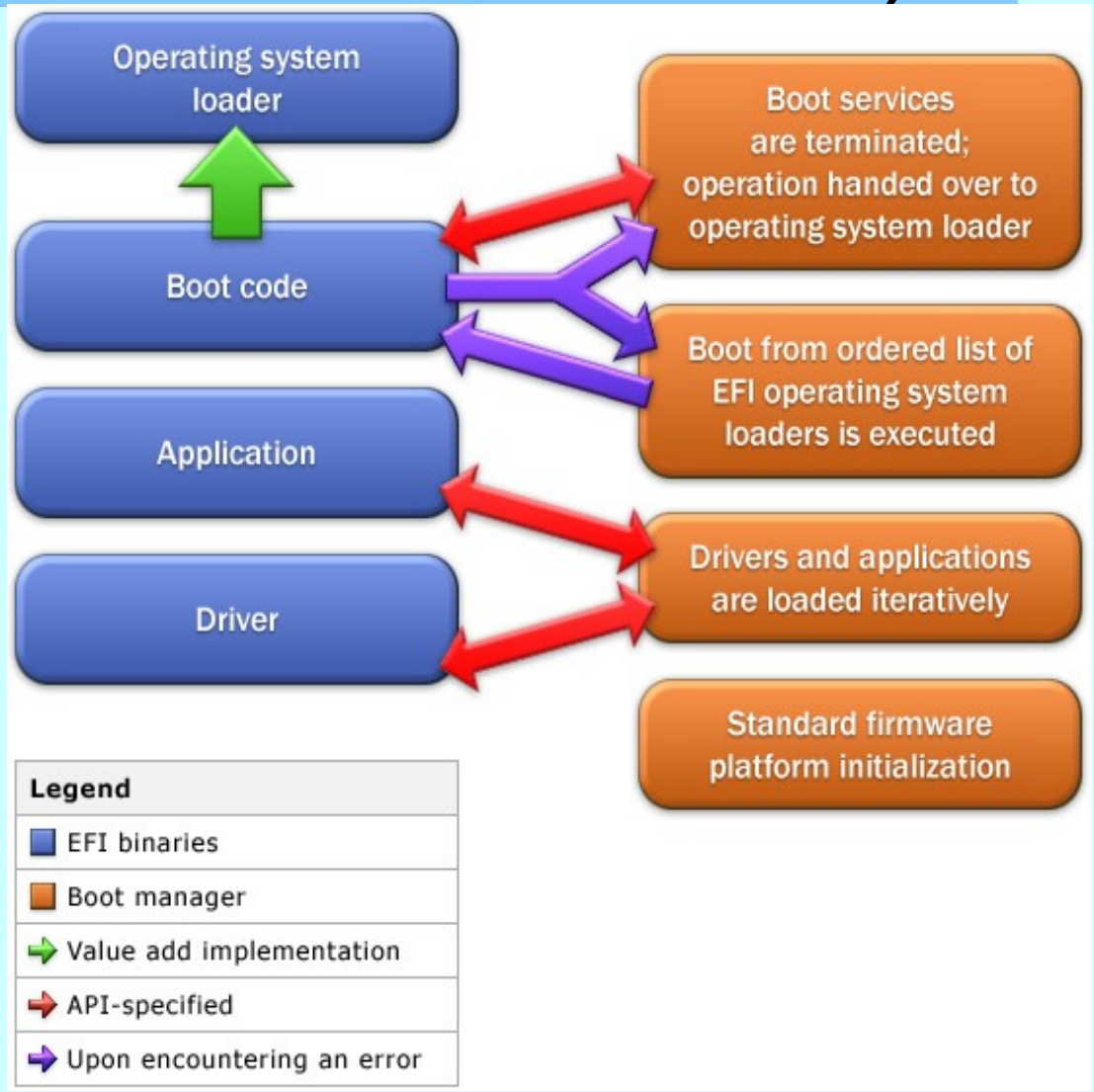
Booting

- Bootstrap loader.
  - Cargar en memoria un software que pueda lanzar un Sistema Operativo.
    - Switches en el panel.
    - Flash boot loader.
    - MBR Master Boot Record) program.
    - EFI (Extended Firmware Interface).
  - Termina cargando el first stage boot loader.

EFI
(Extensible Firmware Interface)



- Desarrollada en el Unified EFI Forum
- Usada en Itanium, Itanium 2 (HP), MacTel , embedded núcleos (core) XScale



EFI (Intel)

- Boot Services:
  - Soporte de consola.
  - Soporte gráfico.
- Runtime Services:
  - Device Drivers
  - Fecha y Hora
- Carga de código desde Internet (Tianocore)
- Device Drivers:
  - una implementación en EFI Byte Code (Driver Execution Environment).
  - el Boot Loader forma parte de EFI.
  - Soporta dispositivos con GUID Partition Table (GPT).

UEFI

Define un "boot manager" (a firmware policy engine) que carga el loader del SO y los drivers que se necesiten.

La configuración del booteo se almacena en variables NVRAM (path de loaders)

Los loaders del SO son "clases" de aplicaciones UEFI, como clases que son, se almacenan en el file system (EFI System partition) que es independiente del medio (HD, Optical Disk, etc).

Especifica un Shell para ejecutar aplicaciones (eje boot loaders), modificar variables, etc

Mantiene compatibilidad reversa con BIOS (Compatibility Support Module (CSM)): Se ignora la info de GPT y se utiliza MBR. De hecho en varias implementaciones tiene precedencia MBR sobre GPT.

UEFI

Tiene una especificación de protocolo llamado Secure Boot que impide la carga de drivers y SOs que no tengan una firma digital aceptable.

Basándose en esta funcionalidad, MS anunció que las máquinas certificadas para Windows 8 deben tener encendido el Secure Boot.

Por otro lado los fabricantes debieran tener y distribuir en sus equipos las claves de los distintos Unixs …

Existen algunas controversias a este respecto.

Proceso de boot - Linux

- Se carga el First Stage Boot Loader.
  - Puede ser uno de varios.
    - Típicamente LILO o GRUB.
    - Termina de cargarse el Boot Loader en memoria.
- Un prompt al usuario obtiene los datos de la partición y del kernel a bootear.
- El kernel se carga como una imagen ("initrd") y luego se le pasa el control con los parámetros apropiados.

Inicialización de Linux

- Lilo.
  - No lee file systems.
  - Según el sistema a bootear:
    - Carga un MBR de 512 bytes
    - Carga un kernel de Linux.
  - La ubicación de este Kernel se guarda en un mapa de carga.
- GRUB
  - puede leer ext2 y ext3.
  - Carga el kernel y le pasa el control.
  - Provee de una interfaz de comandos de pre-boot.
  - Tiene un menú y un editor de configuración.

Kernel Phase (1)

- El Kernel se carga como una imagen (zImage o bzImage).
- Se descomprime.
- Se hace una inicialización de sus estructuras:
  - activando algunos dispositivos y
  - guardando sus datos en las estructuras del Kernel.
- Se transfiere control al proceso 0.
- Ver man boot(1) .

Kernel Phase (2) Proceso 0

- El proceso 0 detecta el tipo de CPU, hace alguna inicialización dependiente de esta.
- Lanza la funcionalidad independiente de la arquitectura (noarch) llamando a start_kernel().
  - Inicializa la tabla de IRQ.
  - Monta el root file system.
  - Lanza a init (1).

Kernel Phase (3) init

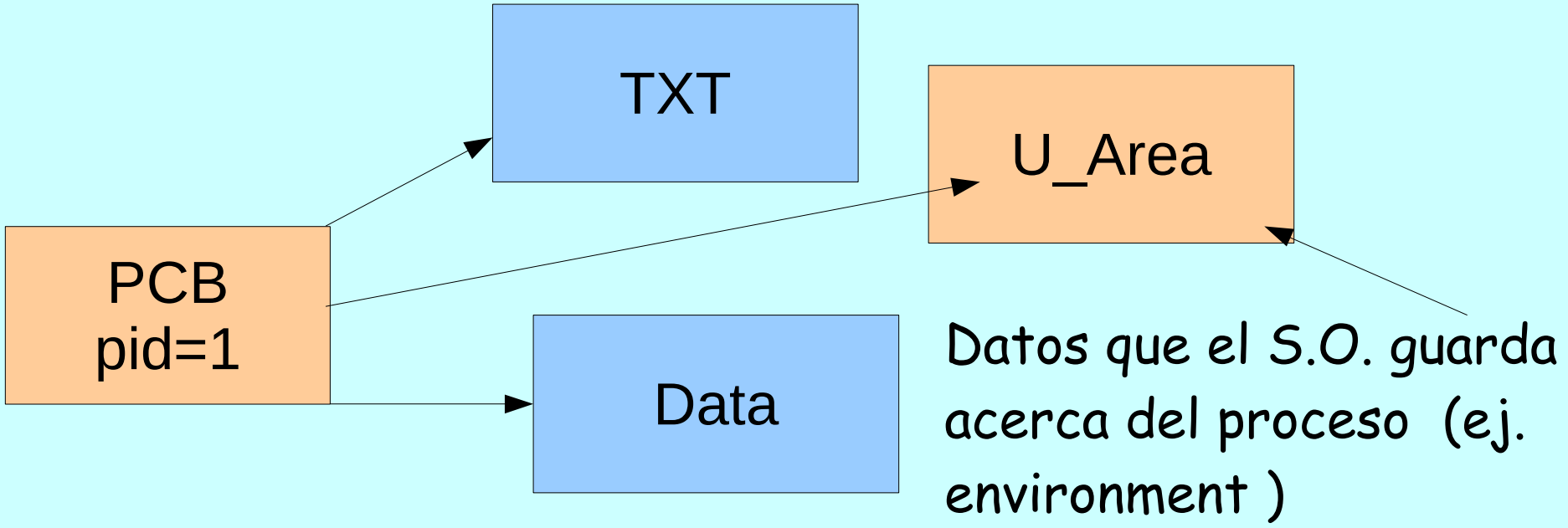
- Init es el proceso 1 su parámetro es un número (runlevel). Ver man init(1).
- Crea los procesos según /etc/inittab.
  - Ubuntu usa Upstart (or. a eventos)
  - Tiene planeado migrar a Systemd (arch de configuración en lugar de scripts)
- Chequea y monta los file-systems según /etc/fstab.

Kernel Phase (3) init

- Lanza los servicios necesarios para llegar al runlevel desde los scripts de /etc/rc.d...
- Espera un login para lanzar un shell de usuario.

Creación de Procesos por el usuario

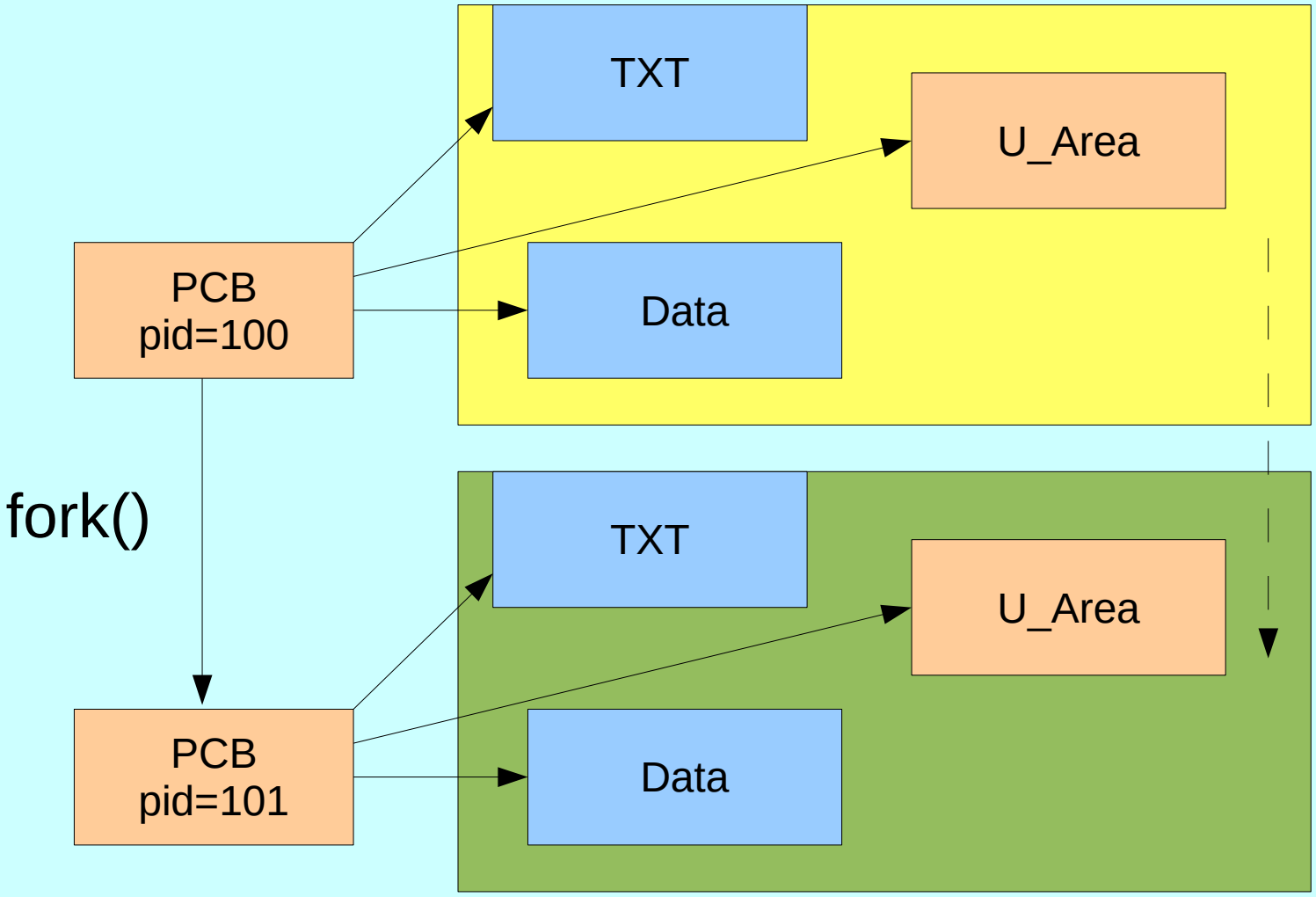
- Linux.
  - fork (2) y exec(3).
  - Ver el Lab en la página de la materia.



fork(2)

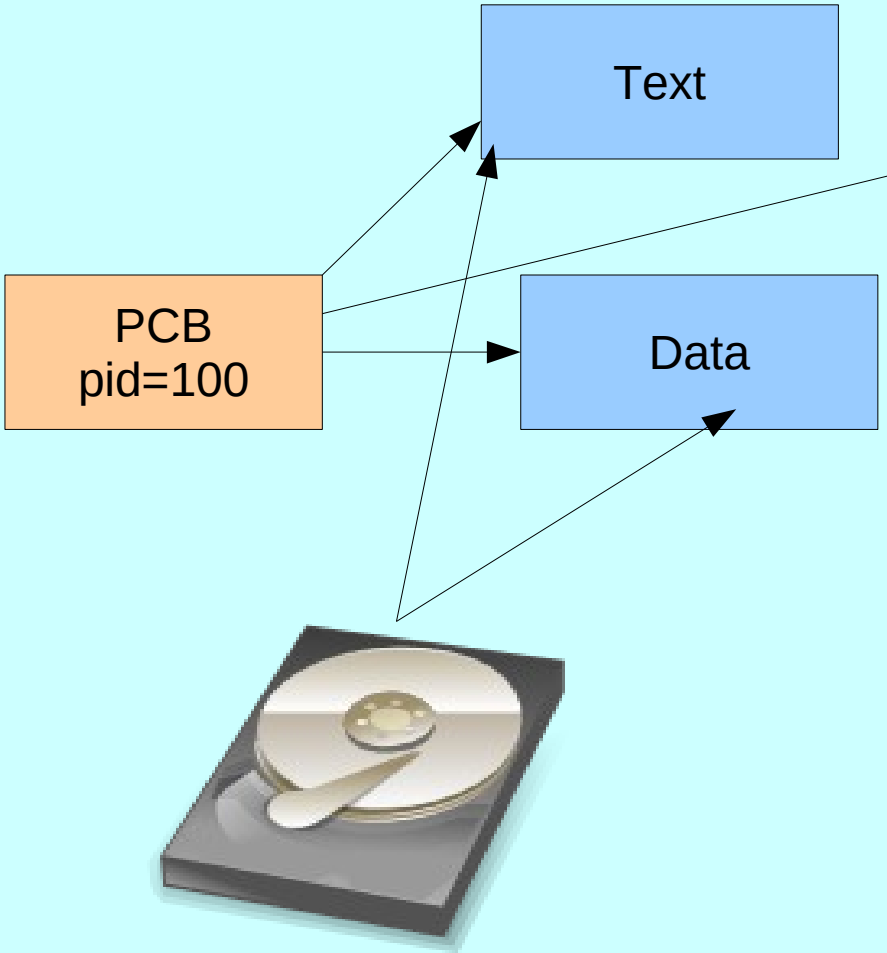
```
if ( (pidhijo = fork () ) == 0)
      cout<<endl<< "---> Es el HIJO con pid = "<<getpid()<<endl;
      exit(0);
      }
 else
      cout <<endl<< "Es el PADRE con pid = "<<getpid()<<
   " y su hijo es pid = "<<pidhijo<<endl;
      exit(0);
      }
```

fork (2)



La copia de estos elementos se hace "on demand"

exec (3)



exec(...) carga una imagen de programa desde un archivo

Terminación de procesos

- La terminación normal da paso a las rutinas registradas con atexit(3).
  - Si termina por _exit(2) la terminación es inmediata.
- Se deben limpiar las estructuras de datos usadas por el proceso.
  - Mientras tanto, el proceso está en estado Zombie.