

Procesos

Gustavo Romero

Arquitectura y Tecnología de Computadores

25 de octubre de 2010

Índice

1 Definición

2 Control

3 Estado

4 IPC

Lecturas recomendadas

Jean Bacon	Operating Systems (4)
Abraham Silberschatz	Fundamentos de Sistemas Operativos (3)
William Stallings	Sistemas Operativos (3)
Andrew Tanuenbaum	Sistemas Operativos Modernos (2.1)

Definición

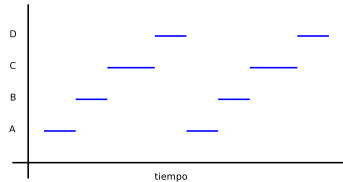
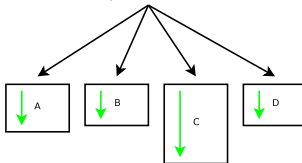
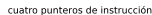
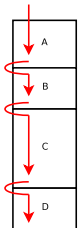
Definición

- ¿Qué es un **proceso**?
 - **programa** en ejecución.
 - entorno de **protección**.
 - algo **dinámico**.
- Componentes básicos:
 - hebras/hilos de ejecución.
 - espacio de direcciones.
- La tarea fundamental de un SO es la gestión de procesos:
 - **creación**.
 - **planificación**.
 - **comunicación** (sincronización).
 - **finalización**.
- Un **programa** es...
 - una lista de **instrucciones** (especie de receta de cocina).
 - algo **estático**.
 - **varios** procesos pueden ejecutar un mismo programa.
- Puede **lanzar**, o ser lanzado por, otros procesos.

Ejemplo: gcc

- **gcc** es un **programa**.
- Muchos usuarios pueden utilizarlo **simultáneamente**.
- Una única copia del programa es **compartida** por todos.
- Los procesos que ejecutan gcc para cada usuarios son **indenpendientes**.
 - un fallo en uno no afecta a los demás.
- gcc para cumplir con sus funciones lanza otra serie de procesos:
 - **cpp**: preprocesador.
 - **as**: ensamblador.
 - **cc**: compilador.
 - **ld**: enlazador.

- un puntero de instrucción



Modelo de procesamiento

- Clasificación en función del coste del cambio de proceso:

- procesamiento **pesado**: proceso UNIX.

- hebra de actividad y espacio de direcciones unificados.
 - el cambio de proceso implica 2 cambios de espacio de direcciones.

$ED_x \longrightarrow ED_{SO} \longrightarrow ED_y$

- procesamiento **ligero**: hebras tipo núcleo.

- hebra de actividad y espacio de direcciones desacoplados.
 - el cambio de hebra implica 1 ó 2 cambios de espacio de direcciones en función de si las hebras lo comparten o no.

$ED_x \longrightarrow ED_{SO} \longrightarrow ED_y / ED_y$

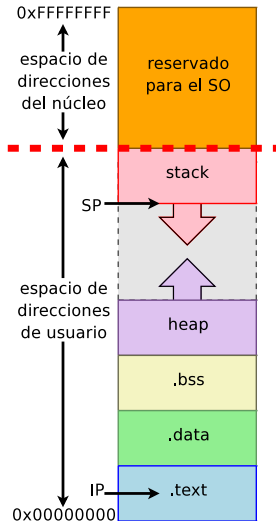
- procesamiento **superligero/pluma**: hebras tipo usuario.

- hebra de actividad y espacio de direcciones unificados.
 - el cambio de hebra no implica ningún cambio de espacio de direcciones.

$ED_x \longrightarrow ED_y$

Ejemplo: Espacio de direcciones en Linux

- Espacio de direcciones lógicas a las que puede acceder un proceso:
 - código: `.text`
 - datos:
 - inicializados: `.data`
 - sin inicializar: `.bss`
 - dinámicos: `heap`
 - pila: `stack`
- En una parte del mismo espacio de direcciones del proceso se ejecuta el núcleo de Linux.
- Divisiones típicas:
2GB/2GB y 3GB/1GB.



Estructuras de control del sistema operativo

- Para **gestionar** procesos y recursos el SO debe disponer de **información** sobre estos.
- El SO mantiene **tablas** sobre cada entidad que gestiona:
 - Tablas de **memoria**: principal y secundaria, protección, traducción.
 - Tablas de **E/S**: dispositivos y canales, estado de las operaciones.
 - Tablas de **ficheros**: existencia, atributos, localización,...
 - Tablas de **procesos**: localización y atributos.
- Las tablas anteriores no suelen estar separadas sino entrelazadas.
- Normalmente las tablas se inicializan al arrancar el sistema mediante autoconfiguración.
- Ejemplo: Linux, `struct task_struct` en `sched.h`.

Estructuras de control de procesos

- Representación física de un proceso:
 - **Imagen del proceso:**
 - programa a ejecutar.
 - espacio de direcciones disponible para código, datos y pila.
 - **Bloque de Control del Proceso (PCB) o descriptor de proceso:**
 - atributos para la gestión del proceso por parte del SO.
 - **estructura de datos más importante del SO.**
- Atributos de un proceso:
 - **Identificación** del proceso: identificadores del proceso, proceso padre, usuario.
 - Estado del **procesador**: registros de propósito general, de estado y control, puntero de pila.
 - Información de **control** del proceso: estado, planificación, estructuración, comunicación y sincronización, privilegios, gestión de memoria, control de recursos y utilización.

Control

Control de procesos

- Modos de ejecución.
- Creación de procesos.
- Finalización de procesos.
- Jerarquía de procesos.
- Cambio de procesos.
- Ejecución del sistema operativo

Modos de ejecución

- La mayor parte de los procesadores proporcionan al menos dos modos de ejecución:
 - **modo usuario**: permite la ejecución de instrucciones que no afectan a otros procesos.
 - **modo núcleo**: permite la ejecución de todas las instrucciones.
- Cuando existen otros modos intermedios sirven para implementar controladores de dispositivos y bibliotecas del sistema o de ciertos lenguajes.
- Un bit en la palabra de estado indica en que modo se está ejecutando el procesador.
 - El **bit** puede consultarse como el resto de la **palabra de estado**.
 - Se modifica cuando se produce una llamada al sistema o una interrupción.
 - Al retornar de la llamada al sistema o de la interrupción se devuelve el valor original de dicho bit desde la pila.

Modos de ejecución

- Funciones típicas del núcleo de un sistema operativo:
 - Gestión de procesos:
 - creación y terminación de procesos.
 - planificación y activación de procesos.
 - intercambio de procesos.
 - sincronización y comunicación entre procesos.
 - gestión de los bloques de control de procesos.
 - Gestión de memoria:
 - reserva de espacios de direcciones.
 - intercambio (*swapping*).
 - gestión de páginas y/o segmentos.
 - Gestión de E/S:
 - gestión de almacenes temporales (*buffers*).
 - reserva de canales de DMA y dispositivos.
 - Funciones de soporte:
 - gestión de interrupciones.
 - auditoría.
 - monitorización.

Creación de procesos

- Salvo sistemas extremadamente simples los SO deben tener mecanismos para la creación de nuevos procesos.
- Posibles causas de la creación de un procesos:
 - 1 Inicialización del sistema.
 - interactivos / no interactivos.
 - primer / segundo plano.
 - 2 Llamada al sistema para crear un proceso.
 - `fork()` + `exec()` / `CreateProcess()`.
 - 3 Petición de usuario.
 - lanzamiento de una nueva aplicación desde el interfaz de usuario.
 - 4 Inicio de un proceso por lotes.
 - sistemas de colas de trabajos en servidores.

Creación de procesos

Pasos en la creación de un proceso:

- ➊ Asignar un **identificador de proceso único**.
- ➋ **Reservar espacio** para el proceso:
 - estructuras de datos del SO (**PCB**).
 - imagen del proceso.
- ➌ **Inicialización** del bloque del control del proceso (**PCB**).
 - ppid, estado, ip, sp, prioridad, E/S,...
- ➍ Establecimiento de **enlaces** adecuados:
 - cola de trabajos.
- ➎ Creación o expansión de otras estructuras de datos:
 - auditoría, monitorización, análisis de rendimiento,...

Finalización de procesos

- Una vez creados los procesos se ejecutan y, generalmente, realizan la tarea para la que se lanzarán.
- Causas de finalización de un proceso:
 - Voluntarias:
 - Terminación normal: la mayoría de los procesos realizan su trabajo y devuelven el control al SO mediante la llamada al sistema `exit()` / `ExitProcess()`.
 - Terminación por error: falta argumento,...
 - Involuntarias:
 - Error fatal: instrucción privilegiada, excepción de coma flotante, violación de segmento,...
 - Terminado por otro proceso: mediante la llamada al sistema `kill()` / `TerminateProcess()`

Jerarquía de procesos

UNIX:

- El uso de **fork()** crea una relación jerárquica entre procesos.
- **init** es primer proceso del sistema y de él dependen todos los demás.
- La relación no puede modificarse.
- Si un proceso padre finaliza antes que sus hijos estos pasan a depender del ancestro previo.
- Útil para llevar a cabo operaciones sobre grupos de procesos.

Windows:

- **CreateProcess()** no establece relación entre procesos.
- Al crear un nuevo proceso se consigue un objeto que permite su control.
 - La propiedad de este objeto puede pasarse a otro proceso.

Jerarquía de procesos

pstree

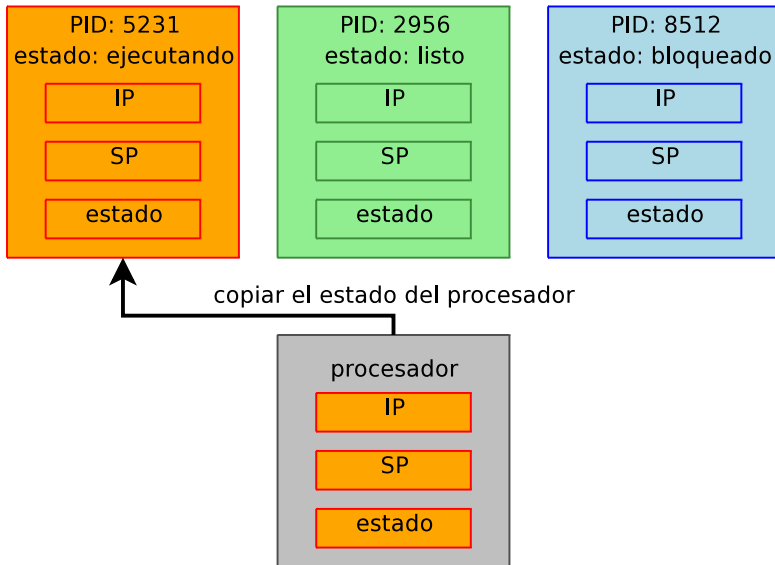
```

init--+-migration/0
      |-ksoftirqd/0
      |-watchdog/0
      |-kthread
      |-mount.ntfs-3g
      |-syslogd
      |-klogd
      |-rpc.statd
      |-acpid
      |-cupsd
      |-sshd
      |-crond
      |-dhcddbd---dhclient
      |-prefdm---gdm-binary---gdm-binary--Xorg
      |                                     |-gnome-session---ssh-agent
      |-metacity
      |-nautilus
      |-gnome-panel
      |-gnome-terminal--bash---pstree
      |                                     |-bash---su---bash
      |                                     |-bash
      |                                     |-bash---ssh
      |-firefox---run-mozilla.sh---firefox-bin--{firefox-bin}
      |                                                         |-{firefox-bin}
      |-xemacs
      |-evince---{evince}
  
```

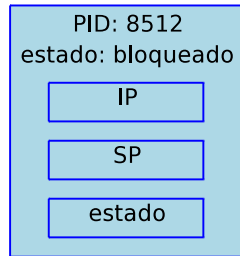
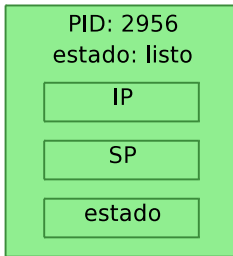
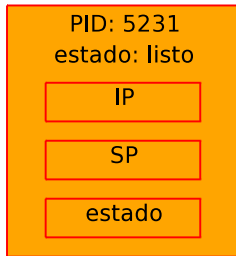
Cambio de proceso

- **Cambio de proceso:**
 - Operación costosa.
 - Linux 2.4.21: $5.4\mu\text{s}$ /13200 ciclos en un Pentium IV a 2.4GHZ.
 - Eventos que pueden provocar un **cambio de proceso**:
 - Interrupción: interrupción del reloj, finalización de operación de E/S o DMA,...
 - Excepción: fallo de página/segmento, llamada al sistema (int/syscall), operación de E/S,...
- **Cambio de modo:** cambio del modo de privilegio con el que se ejecuta el procesador.
 - Operación sencilla y poco costosa.
- **Cambio de contexto** es ambiguo... ¿a qué cambio se refiere?
¿proceso? ¿modo? ¿ambos?

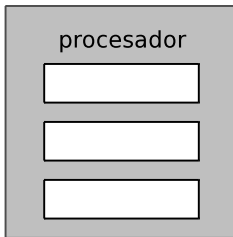
Cambio de proceso



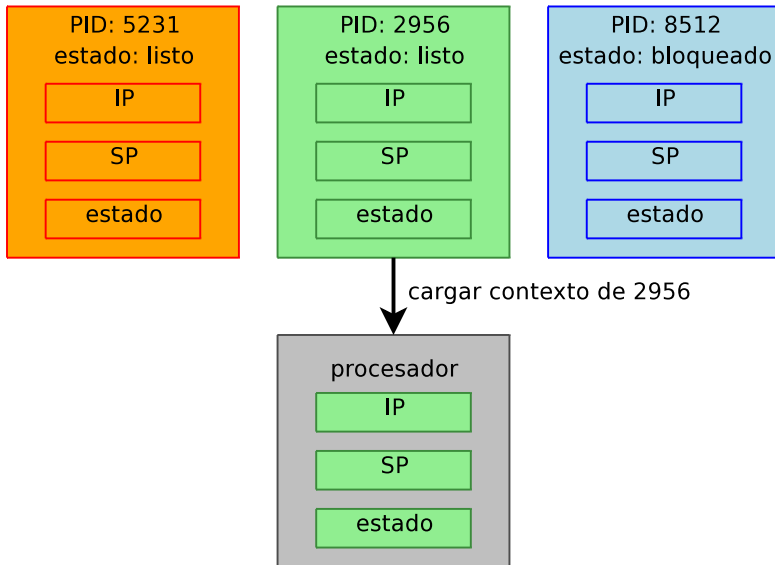
Cambio de proceso



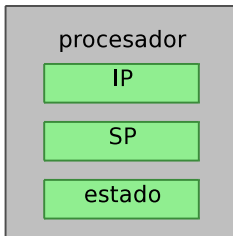
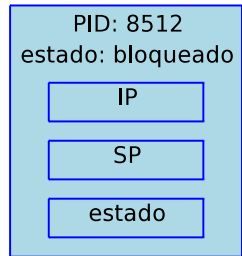
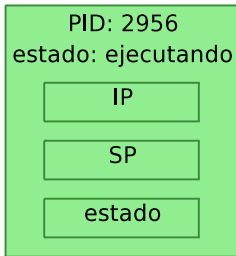
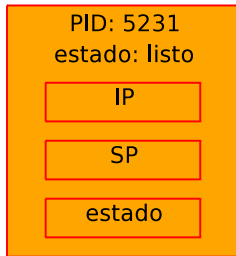
ahora se debe seleccionar un nuevo proceso



Cambio de proceso



Cambio de proceso



Cambio de proceso

Pasos a seguir para realizar el cambio de proceso:

- ➊ Salvar el estado del procesador.
- ➋ Actualizar el estado del bloque de control del proceso.
 - Como mínimo cambiar el estado (ej: ejecutando → preparado).
- ➌ Mover el PCB a la cola adecuada (ej: preparado).
- ➍ Seleccionar el nuevo proceso a ejecutar.
- ➎ Actualizar el estado del bloque de control del proceso.
 - Como mínimo cambiar el estado (ej: preparado → ejecutando).
- ➏ Actualizar las estructuras de datos de gestión de memoria.
- ➐ Restaurar el estado del proceso al que tenía en el momento de abandonar el estado ejecutando.

UNIX: fork() + exec() + wait() + exit()

```
#include <unistd.h>
#include <iostream>
using namespace std;
int main()
{
    pid_t pid = fork();

    if (pid < 0) // error
    {
        cerr << "error en fork" << endl;
    }
    else if (pid == 0) // hijo
    {
        cout << "hijo" << endl;
        execlp("/bin/ls", "ls", NULL);
    }
    else // padre
    {
        cout << "padre" << endl;
        wait(pid);
    }
}
```

padre

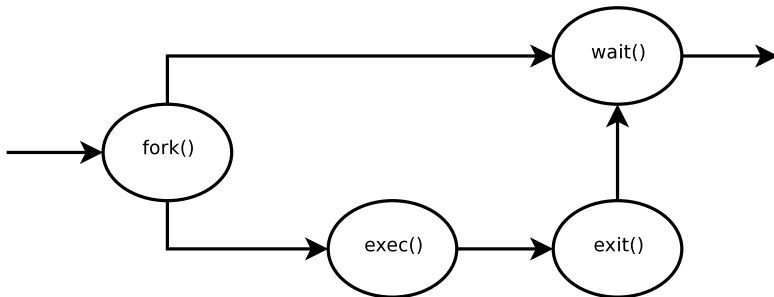
```
cout << "padre" << endl;
wait(pid);
exit(0);
```

hijo

```
cout << "hijo" << endl;
execlp("/bin/ls", "ls", NULL);
exit(0);
```

UNIX: `fork()` + `exec()` + `wait()` + `exit()`

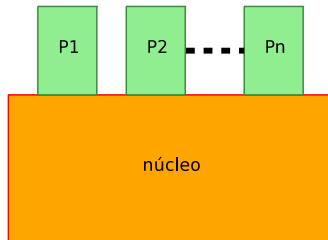
- **`fork()`**: crea un nuevo proceso.
- **`exec()`**: cambia la imagen de un proceso.
- **`wait()`**: permite al padre esperar al hijo.
- **`exit()`**: finaliza el proceso.



Ejecución del sistema operativo

Núcleo **independiente**

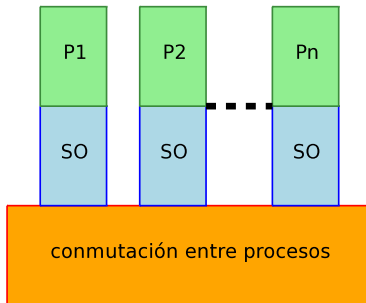
- Método más antiguo
- El SO no es un proceso (aunque se comporte como uno).
- El SO dispone de áreas de memoria y pila propias.
- El concepto de proceso es aplicable sólo a programas de usuario.
- Inconveniente: cada evento cuesta un cambio de proceso y modo.



Ejecución del sistema operativo

Ejecución **dentro** del los procesos de usuario

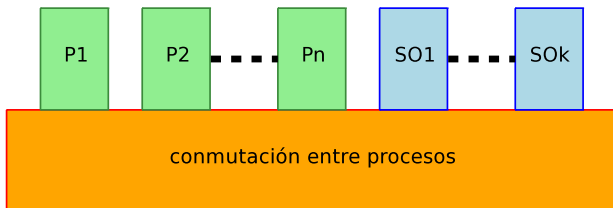
- EL SO es como un conjunto de rutinas que el usuario puede invocar y que están situadas dentro de su entorno.
- A la imagen de cada proceso se une la del SO.
- Ventaja: cada evento cuesta sólo un cambio de modo.
- Inconveniente: restamos espacio al proceso de usuario.



Ejecución del sistema operativo

Sistemas operativos basados en **procesos**.

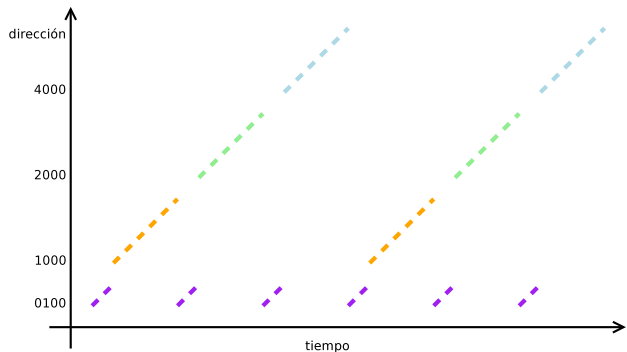
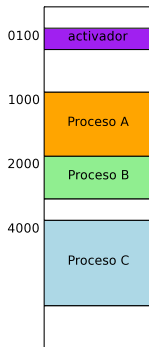
- El SO se implementa como una colección de procesos.
- Ventajas:
 - modularidad y facilidad de programación.
 - mejora del rendimiento en sistemas multiprocesador.
- Inconvenientes:
 - cada evento cuesta varios cambios de proceso y modo.



Estado

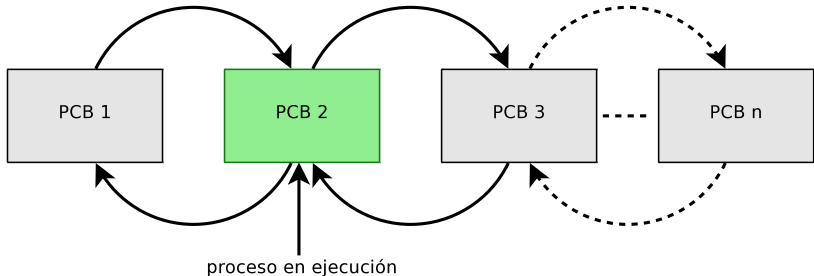
Traza de un proceso

- Comportamiento de un proceso = lista de instrucciones que ejecuta $\Rightarrow$ **traza**.
- Activador:** programa encargado de cambiar entre los PCBs de los procesos para ejecutar un proceso u otro.

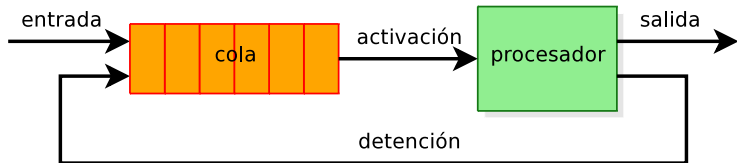
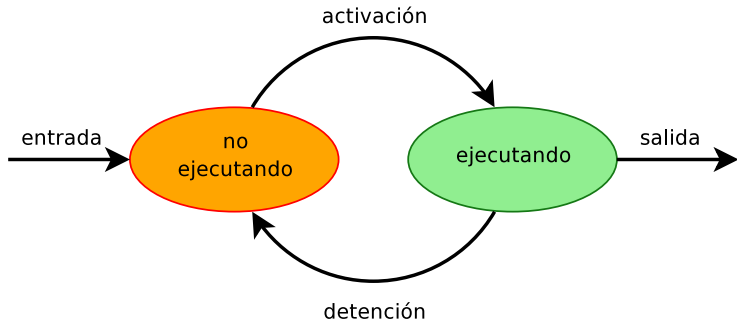


Cola de procesos

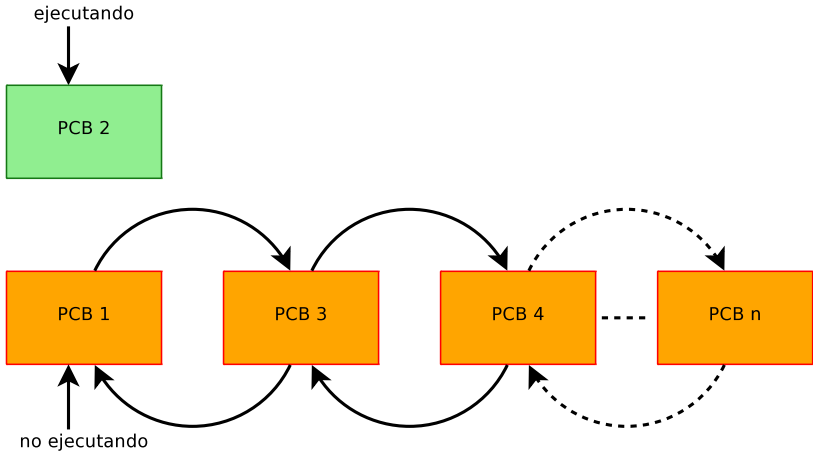
- Creación de un proceso = crear PCB + cargar imagen.
- Lista de procesos = lista de PCBs.
- **Planificador** (*"scheduler"*):
 - Parte del SO que escoge el siguiente proceso a ejecutar.
 - Gestor de las colas de planificación.
- **Activador** (*"dispatcher"*): parte del planificador que realiza el intercambio de procesos.
- Ejecución = encolar + activar.



Modelo de 2 estados



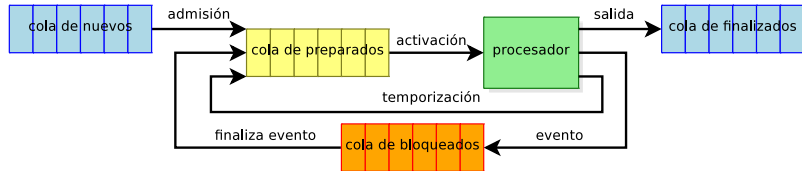
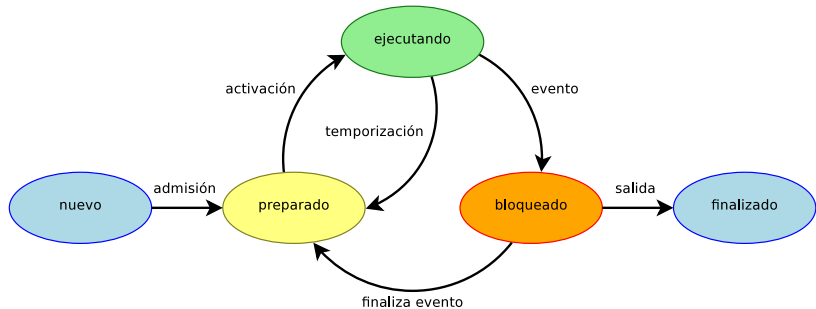
Modelo de 2 estados



Modelo de 2 estados

- Estados:
 - **Ejecutando**: proceso en ejecución.
 - **No ejecutando**: proceso que no se está ejecutando.
- Transiciones:
 - **Ejecutando** → **no ejecutando**: evento o temporización.
 - **No ejecutando** → **ejecutando**: temporización o fin de evento.
- Inconvenientes:
 - No permite discriminar fácilmente la razón por la que un proceso no se encuentra en ejecución.
 - Solución: subdividir el estado “no ejecutando” para reflejar el motivo.

Modelo de 5 estados



Modelo de 5 estados

- Estados:
 - **Nuevo:** el proceso ha sido creado.
 - **Preparado:** proceso a la espera de que se le asigne un procesador.
 - **Ejecutando:** proceso actualmente en ejecución.
 - **Bloqueado:** proceso que no puede continuar hasta que finalice un evento.
 - **Finalizado:** proceso finalizado.

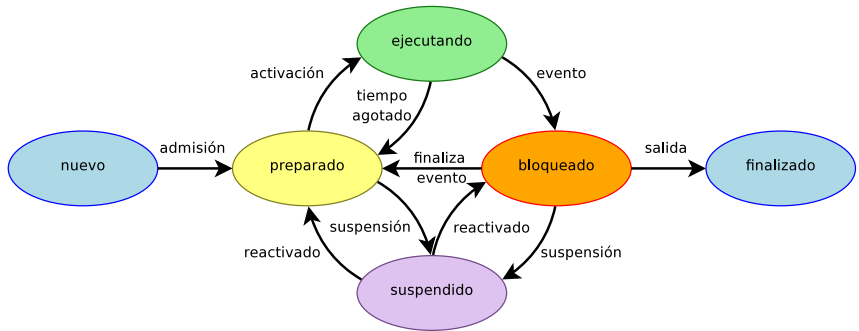
Modelo de 5 estados

- Transiciones:
 - **Nuevo** → **preparado**: se admite un nuevo proceso en el sistema.
 - **Preparado** → **ejecutando**: el planificador selecciona el proceso para su ejecución.
 - **Preparado** → **finalizado**: padre termina hijo.
 - **Ejecutando** → **finalizado**: proceso finalizado.
 - **Ejecutando** → **preparado**: tiempo de procesador agotado.
 - **Ejecutando** → **bloqueado**: se produce un evento.
 - **Bloqueado** → **preparado**: finalización de evento.
 - **Bloqueado** → **finalizado**: padre termina hijo.

Modelos de +5 estados

- Existe una buena razón para añadir al menos un nuevo estado: **suspendido**.
- Objetivo de la multiprogramación: aprovechar al máximo el procesador debido a la lentitud de las operaciones de E/S.
- ¿Resuelve el problema el modelo de 5 estados? $\implies$ no.
 - Causa: diferencia de velocidad procesador/dispositivos de E/S.
 - Todos los procesos podrían llegar a estar bloqueados en espera de E/S, un cierto recurso o la finalización de un subproceso.
 - Solución: añadir más procesos.
 - Problema: falta de memoria.
- Circulo vicioso de difícil solución:
 - Solución cara: añadir más memoria
 - Solución barata: memoria de intercambio (*"swap"*).
- **Intercambio** (*"swapping"*): proceso de expulsión de un proceso de memoria principal a secundaria y viceversa.
 - Nuevo problema: el intercambio requiere E/S.

Modelo de 6 estados



Modelo de 7 estados



Modelo de 7 estados

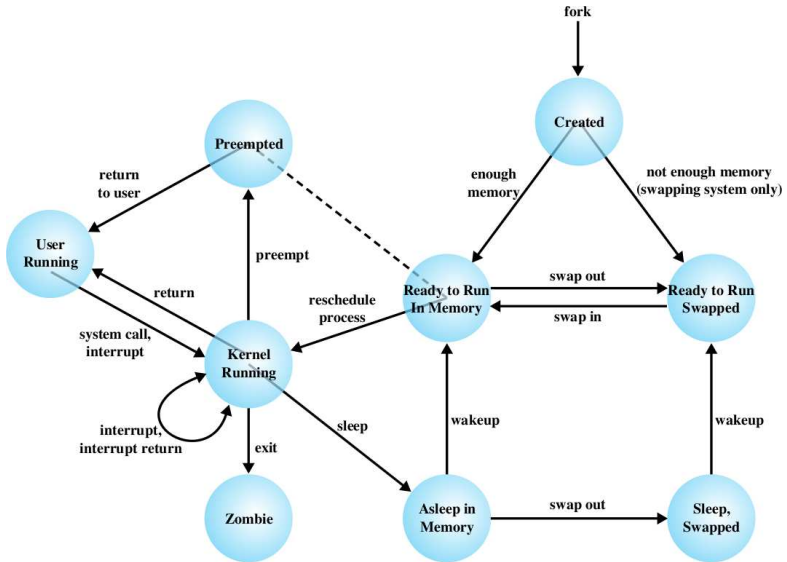
- Motivación: el reactivar un proceso sólo para descubrir que está bloqueado es muy costoso.
- Nuevos estados:
 - **Suspendido (bloqueado)**: proceso en área de intercambio y esperando un evento.
 - **Suspendido (preparado)**: proceso en área de intercambio y a la espera de espacio en memoria principal.

Modelo de 7 estados

- Nuevas transiciones:

- **bloqueado** → **suspendido (bloqueado)**: no hay procesos preparados o estos consumen demasiada memoria.
- **suspendido (bloqueado)** → **suspendido (preparado)**: sucede el evento por el que estaba bloqueado.
- **suspendido (preparado)** → **preparado**: no quedan procesos preparados en el sistema o tiene mayor prioridad que los preparados.
- **preparado** → **suspendido (preparado)**: liberar memoria o dejar sitio para un proceso bloqueado de mayor prioridad.
- **nuevo** → **suspendido (preparado)**: control de carga del sistema.
- **suspendido (bloqueado)** → **bloqueado**: queda memoria libre o proceso de alta prioridad.
- **ejecutando** → **suspendido (preparado)**: un proceso agota su tiempo y hay que liberar memoria para un proceso suspendido de mayor prioridad.
- **X** → **finalizado**: un proceso elimina a otro.

Diagrama de transiciones entre estados en UNIX



Planificación

- Los procesos pueden cambiar varias veces de cola de planificación a lo largo de su vida.
- La parte del SO encargada de realizar estos cambios es el planificador.
- Tipos de planificadores:
 - Corto plazo: selecciona entre los procesos preparados uno para ejecutar.
 - Ejecución muy frecuentemente, ej: cada 10..100ms.
 - Medio plazo: decide que procesos pasar al área de intercambio y así controla el grado de multiprogramación.
 - Largo plazo: selecciona que procesos poner en ejecución, ej: sistema por lotes.
 - ejecución en función de la carga del sistema, cada varios minutos o cuando finaliza un proceso.

IPC

Comunicación entre procesos

- Los procesos que se ejecutan concurrentemente pueden ser...
 - **Independientes:** no afecta ni es afectado por otros procesos (no comparten datos).
 - **Cooperantes:** puede afectar y ser afectado por otros procesos (si comparten datos).
- El SO debe proporcionar mecanismos para crear, comunicar y terminar procesos.
- Motivos para cooperar:
 - Compartir información: capacidad de acceso y economía de recursos.
 - Acelerar los cálculos: realizar las tareas más rápidamente.
 - Modularidad: facilidad de creación de programas.
 - Conveniencia: multitarea.

Comunicación entre procesos

- Métodos de comunicación:
 - Memoria compartida:
 - los procesos comparten un área de memoria.
 - comunicación responsabilidad de los procesos.
 - Paso de mensajes.
 - los procesos intercambian mensajes.
 - comunicación responsabilidad del sistema operativo.

