Gestión de procesos

Sistemas Operativos (SO)
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Licencia Creative Commons

Licencia Creative Commons

Índice

Conceptos previos

- Ráfaga de CPU
  - Intervalo de tiempo consecutivo que un proceso está ejecutandose en la CPU
- Ráfaga de E/S
  - Intervalo de tiempo consecutivo que un proceso está realizando una E/S

```
for ( i = 0 ; i < 5 ; i ++ )
        s  += sqrt( a[i] );

printf("Resultado:%f\n",s);

for ( i = 0 ; i < 5 ; i++ )
        a[i] = 0;
```

50 ms

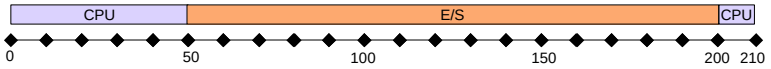
150 ms

10 ms

- 2 ráfagas de CPU de 50 y 10 ms respectivamente
- 1 ráfaga de E/S de 150 ms

Conceptos previos (II)

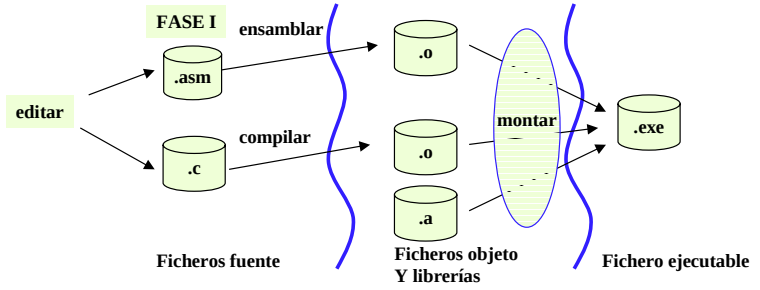
- Diagrama de gantt
  - Diagrama horizontal que muestra para cada instante que valor toma un cierto parámetro
  - P. ej. diagrama de gantt de las ráfagas del proceso anterior:



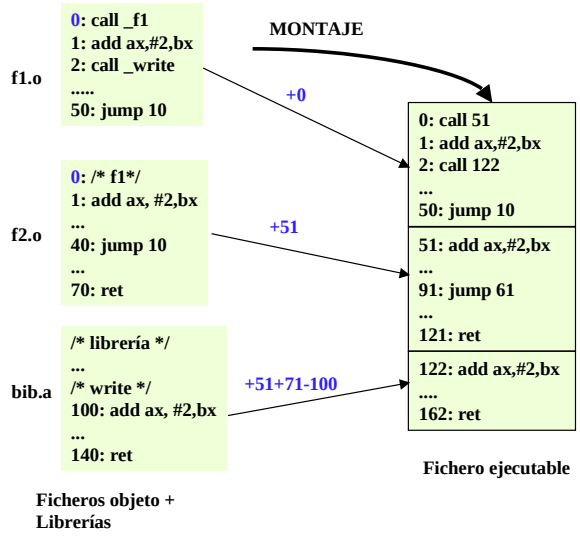
Generación de ejecutables

- Evolución de los programas:
  - Lenguaje alto nivel -> Lenguaje máquina-> Ejecución
- Fase I
  - Compilación: Traducción de lenguaje alto nivel a código objeto
  - Montaje: Creación de un fichero ejecutable a partir de 1 o varios ficheros objeto y librerías
- Fase II
  - Carga/Ejecución: Carga un fichero ejecutable en memoria física y da control a la primera instrucción del programa

Generación de ejecutables



Fase de montaje

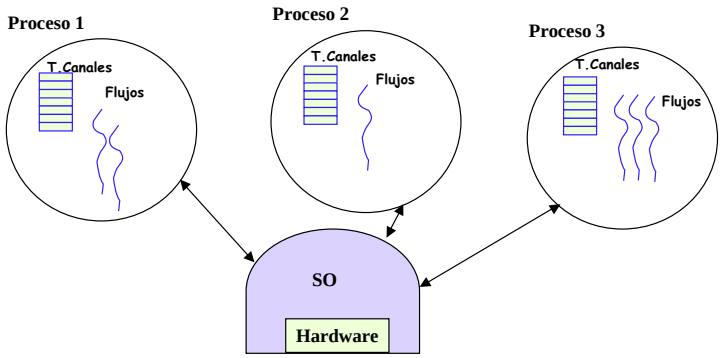


Concepto de proceso y flujo

- Un fichero ejecutable es algo estático, código almacenado en disco
- Cuando se carga en memoria y se empieza a ejecutar es un programa en ejecución o PROCESO.
- A cada parte del programa (código) que se puede ejecutar de forma independiente se le puede asociar un FLUJO
- FLUJO: instancia de ejecución de un proceso

Procesos y flujos

- Visión:



Procesos

- Proceso: Unidad de asignación de recursos que proporciona el SO (memoria, flujos, canales) :
  - Los recursos son “pedidos” por los flujos pero se asignan al proceso
  - Los flujos de un mismo proceso COMPARTEN todos los recursos del proceso
  - Flujos de procesos distintos NO COMPARTEN RECURSOS (ni memoria, ni canales)
- La cantidad de recursos de un proceso es dinámica (se asignan/se liberan)
- Los procesos se gestionan mediante llamadas a sistema
  - Crear/destruir/modificar/consultar

Procesos

- Los procesos no comparten recursos entre si
  - No comparten memoria
  - No comparten canales
  - No comparten flujos

Características de los procesos

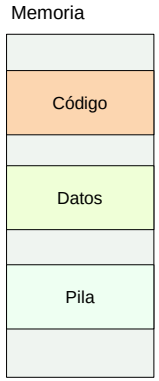
- Se identifican mediante un PID (Process IDentifier)
  - Único en el sistema
  - Constante durante toda la vida del proceso
- Están asociados a un usuario
  - Tendrá acceso a los recursos en función del usuario
  - Se puede cambiar de usuario
- Tiene un mínimo de un flujo
  - Cada flujo tiene un identificador local al proceso
- Dispone de canales para realizar la E/S
  - Todos los flujos acceden a los mismos canales

Características de los procesos

- Gestión de excepciones y eventos
- Tienen prioridades, usadas para planificar
- Esta información se almacena en el PCB (Process Control Block)
  - Los campos que contiene dependen del Sistema Operativo
  - Lo veremos más adelante

Representación de un proceso

- Imagen en memoria del proceso
  - Código
  - Datos
  - Pila

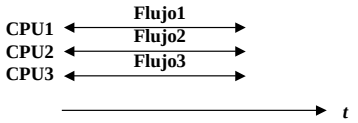


Concurrencia de procesos

- ¿Porqué nos puede interesar ejecutar multiples procesos simultaneamente?
  - Cada proceso hace una tárea diferente simultaneamente
  - Aprovechar el tiempo de E/S de un proceso
  - Si tenemos varios procesadores
    - Podemos ejecutar más rápido la misma tárea paralelamente
  - Compartir recursos entre diferentes usuarios
    - Hay que dar la ilusión de que cada uno tiene su(s) procesador(es)

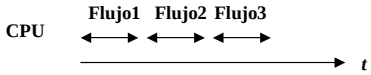
Concurrencia y paralelismo

- Los recursos físicos son limitados. El S.O. reparte los recursos.
- Como repartir/distribuir:Concurrencia,Paralelismo
- Paralelismo: 1 flujo asociado a un procesador

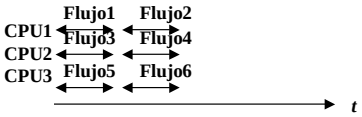


Concurrencia y paralelismo

- Concurrencia: Cuando un procesador es compartido en el tiempo por varios flujos



- Normalmente combinaciones de las dos



Concurrencia de procesos

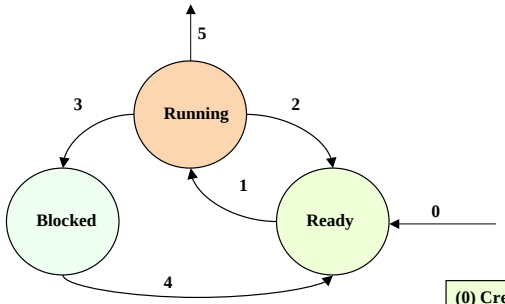
- Los procesos pueden ser de dos tipos
  - Independientes
    - Los procesos no comparten nada entre si (recursos, memoria)
  - Cooperativos
    - Los procesos comparten diferentes recursos
      - Memoria
      - Ficheros
      - ...
    - El nivel de compartición puede variar
      - Normalmente hay una pila privada al menos para cada proceso

Representación de un proceso

- Dentro del sistema
  - Cada proceso tiene un Process Control Block (PCB)
    - Identificador del proceso
    - Estado del proceso
    - Recursos del proceso (páginas de memoria, ficheros abiertos, ...)
    - Estadísticas del proceso (cpu consumida, total memoria ocupada, ...)
    - Información de planificación (prioridad, quantum, ...)
    - Contexto de ejecución
      - Dentro del PCB
      - En la pila del procesos y en el PCB la @
    - ...

Ciclo de vida de un proceso

- Grafo simplificado de estados:



(0) Creación del proceso
(1) Planificador escoge este proceso
(2) Planificador coge otro proceso
(3) Proceso bloqueado por E/S
(4) E/S disponible
(5) Fin del proceso

Estados de un proceso

- Creación de un proceso:
  1. Asignar identificador (PID)
  2. Asignar espacio para el proceso
     - Mediante un proceso cargador (Loader)
     - Por copia del proceso padre (UNIX)
  3. Asignar PCB
  4. Iniciar PCB
  5. Encolar PCB (planificación)
  6. Crear o ampliar otras estructuras de datos

Carga de un ejecutable

- Poner en memoria física la información del fichero ejecutable y dar control a la primera instrucción del programa
  - Copiar código
  - Inicializar datos
  - Expandir pila y datos no inicializados
  - Iniciar estructuras del proceso:
    - Tabla de canales…

Copia del padre (UNIX)

- Se crea el nuevo proceso como replica del padre:
  - 2 métodos:
    - Copia de toda la memoria del proceso padre
    - Compartición del espacio de direcciones del proceso padre
      - Clone (linux), sproc (IRIX)
  - Se tienen que replicar estructuras:
    - Tabla de canales, programación de signals
  - La replicación de estructuras se tiene que hacer de forma “segura”
    - Asegurar la integridad del sistema

Tabla de canales

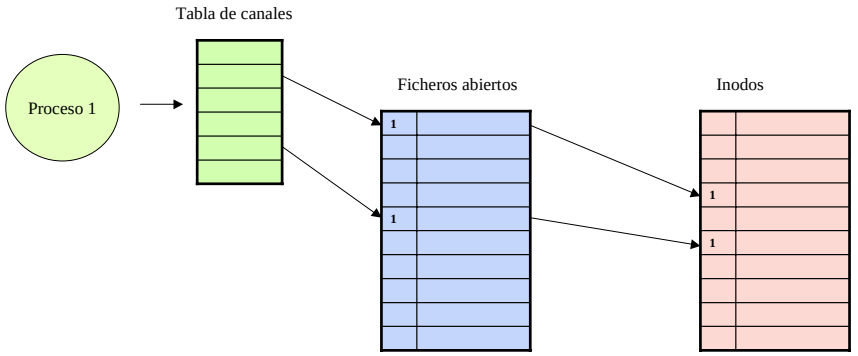
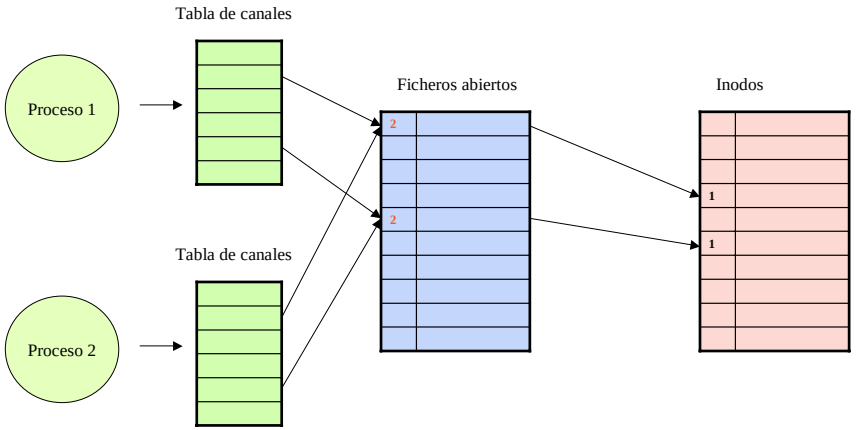


Tabla de canales

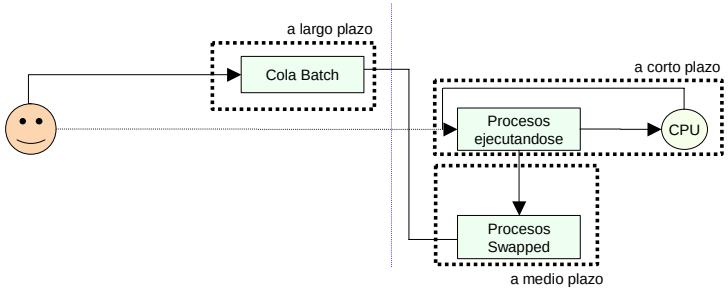


Planificación

- El proceso pasa de Ready a Run según la política de planificación del sistema
- Existe un Planificador
  - Se encarga de decidir cual es el siguiente proceso que se ejecuta
  - Cambia el estado de un proceso de Run a Ready siguiendo unas políticas

Planificadores del sistema

- Planificación
  - Tarea fundamental del Sistema Operativo
- Tres planificadores diferentes en el Sistema



Planificadores del sistema (II)

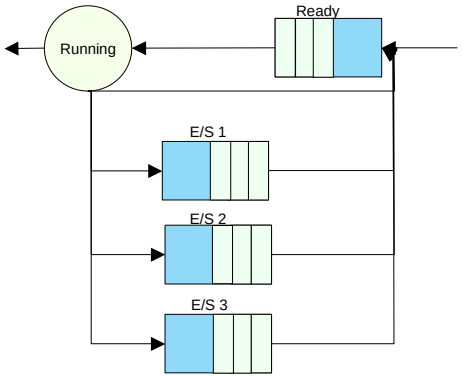
- Planificador a largo plazo (batch)
  - Controla el grado de multiprogramación en el sistema
  - Se ejecuta cuando empieza/acaba un proceso
  - Opcinal en sistemas de tiempo compartido
- Planificador a medio plazo
  - Encargado de suspender y restaurar posteriormente procesos (swap out y swap in)
  - Se ejecuta cuando hay escasez de recursos
    - p.ej. Muchos procesos ejecutándose

Planificadores del sistema

- planificador a corto plazo
  - Selecciona el proceso a ejecutar
  - Se ejecuta frecuentemente
    - Cuando se crea/acaba un proceso
    - Cada cierto tiempo (dependiente de la planificación)
    - Cuando un proceso inicia/finaliza la E/S
    - Ha de ser eficiente
  - Veremos diferentes políticas de planificación
    - FCFS
    - Prioridades
    - Round Robin
    - ...

Estructuras de los estados

- El sistema utiliza diferentes colas para gestionar los estados
- Cada cola puede tener una política diferente

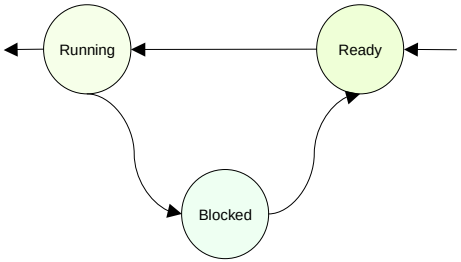


Tipos de politicas de planificación

- No Apropiativas
  - El Sistema Operativo no expulsa nunca al proceso de la cpu
  - El proceso abandona voluntariamente la CPU
    - p.ej.: mediante el inicio de una E/S
- Apropiativas
  - El Sistema Operativo puede decidir expulsar a un proceso del procesador y dárselo a otro (apropiación)
  - La apropiación puede ser:
    - Diferida
      - El Sistema Operativo hace efectiva la planificación cada cierto tiempo
    - Immediata
      - El Sistema Operativo hace efectiva la planificación tan pronto como hay un cambio

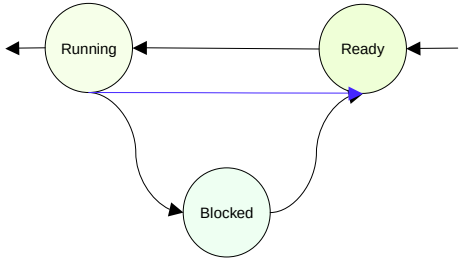
Grafo de estados

- No apropiativa



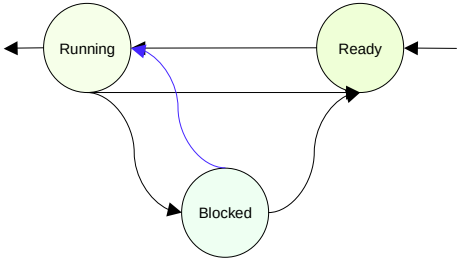
Grafo de estados (II)

- Apropiativa diferida



Grafo de estados (III)

- Apropiativa inmediata



Propiedades de los algoritmos

- ¿ Cómo sabemos que algoritmo es mejor ?
  - Cada algoritmo maximiza diferentes criterios o propiedades
  - Dependiendo de nuestros objetivos el mejor será uno u otro

Propiedades de los algoritmos (II)

- Justicia
  - Un algoritmo es justo si garantiza que todo proceso recibirá CPU en algún momento
- Eficiencia
  - Maximizar el % del tiempo que está la CPU ocupada
- Productividad (Throughput)
  - Maximizar el número de trabajos por unidad de tiempo
- Tiempo de espera
  - Minimizar el tiempo en la cola de ready

Propiedades de los algoritmos (III)

- Tiempo de respuesta
  - Minimizar el tiempo que tarda un proceso en obtener su primer resultado
- Tiempo de retorno
  - Minimizar el tiempo total que tarda en ejecutarse 1 proceso

Algoritmos de planificación

- FIFO
- Prioridades
- Round Robin
- Colas Multinivel

First Come, First Served (FCFS)

- El primer proceso en llegar a Ready es el primero en ser planificado
- No apropiativo
- Tiempo de espera elevado
  - No es apropiado para sistemas de tiempo compartido
- Provoca un efecto convoy con los procesos de E/S

Prioridades

- Cada proceso tiene asignada una prioridad
  - estática
  - dinámica
  - estática + dinámica
- El proceso más prioritario es planificado para ejecutarse
- Apropiativas o no apropiativas
- Entre procesos de igual prioridad
  - FCFS

Prioridades (II)

- Puede provocar inanición (starvation)
  - No se planifica NUNCA un proceso por no tener suficiente prioridad
  - Solución: envejecimiento (aging). Se aumenta la prioridad del proceso cada unidad de tiempo
- Ejemplos de prioridades dinámicas:
  - Shortest Job First
    - No apropiativo
    - La prioridad del proceso es el tiempo de ráfaga de CPU (predicción)
  - Shortest Remaining Time
    - Apropiativo
    - La prioridad del procesos es el tiempo ***restante*** de ráfaga

Round Robin

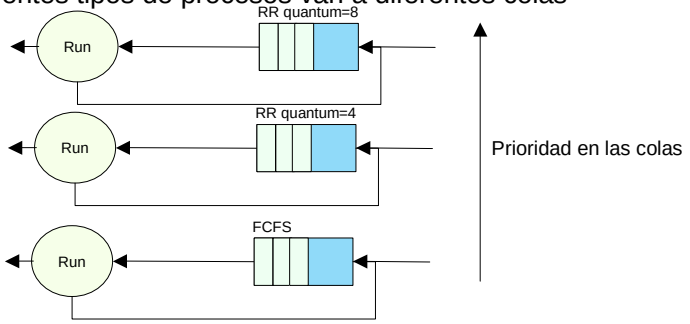
- El SO asigna un tiempo de CPU a cada proceso llamado quantum
  - puede ser constante o calcularse dinámicamente
- Un proceso abandona la CPU por dos motivos:
  - Su quantum ha finalizado y es apropiado
  - La abandona voluntariamente para hacer E/S
- El proceso a planificar se elige según FCFS
  - El Sistema asigna un nuevo quantum a ese proceso
- La elección del quantum es muy importante
  - Pequeño: demasiados cambios de contexto
  - Grande: se aproxima a FCFS

Round Robin con prioridades

- Se aplican prioridades normalmente
  - no apropiativas
  - apropiativas (inmediatas o diferidas)
- En caso de empate: Round Robin

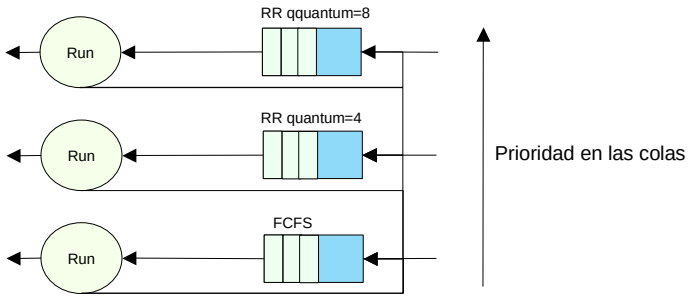
Colas Multinivel

- El sistema tiene diferentes colas de Ready
  - Cada cola tiene una prioridad
    - Se escoje el proceso de la cola más prioritaria no vacia
  - Dentro de cada cola se aplica una política diferente
  - Diferentes tipos de procesos van a diferentes colas



Colas Multinivel Realimientadas

- Igual que el anterior
  - pero además los procesos pueden avanzar/retroceder por las diversas colas



Cambio de contexto

- Cuando un proceso pasa de Run a Ready y se selecciona otro para ejecutar, se tiene que realizar un cambio de contexto
  1. Salvar el contexto del procesador: PC y registros
  2. Actualizar el PCB
  3. Mover el PCB a la cola apropiada
  4. Seleccionar otro proceso
  5. Actualizar el PCB del proceso a ejecutar
  6. Actualizar estructuras de datos de gestión de memoria
  7. Restaurar el contexto del nuevo proceso
- Se realiza en modo privilegiado

Estado blocked

- Un proceso pasa de Run a Blocked cuando realiza una E/S
  - Para ello tiene que ejecutar una llamada al sistema
  - El cambio de estado es explícito
  - El PCB del proceso se encola en la cola que hace referencia a la E/S
  - Dependiendo de la política de planificación, cuando se acaba la E/S se pasa a:
    - Ready: apropiación diferida
    - Run: apropiación inmediata

Finalización de un proceso

- Un proceso finaliza de forma:
  - Explícita:
    - Por parte del proceso: exit
    - Por parte del usuario o del administrador
  - Implícita:
    - Error en la ejecución:
      - Excepciones
      - Accesos incorrectos al espacio del proceso
      - …

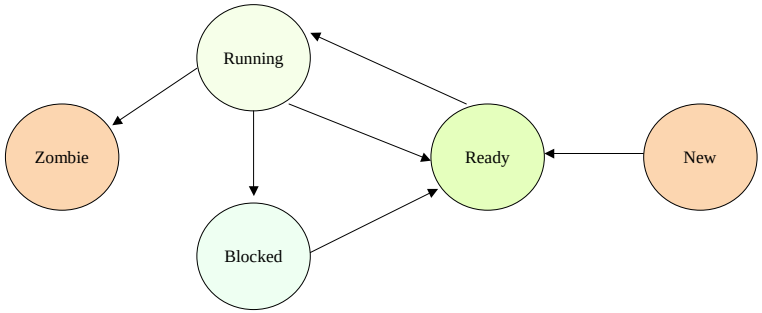
Finalización de un proceso

- Que hace el SO?
  - Actualiza el PCB con la información de estado de finalización
  - Cierra la tabla de canales
  - Decrementa todas las referencias a la tabla de ficheros abiertos
  - Libera la memoria del proceso
  - (UNIX) Si el proceso tenía hijos, estos pasan a ser hijos del proceso init (PID=1)
  - (UNIX) Pasa a estado de Zombie
- UNIX: el PCB del proceso lo libera el padre cuando consulta el estado de finalización del proceso hijo

Mutación de un proceso

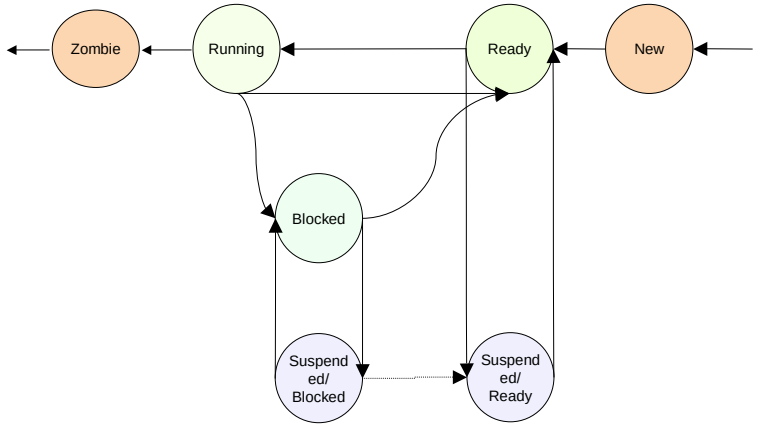
- Los procesos pueden mutar:
  - Ejecutar otro programa dentro del espacio del proceso
  - Se pierde:
    - Código
    - Datos
    - Pila
    - Programación de signals
  - No se pierde:
    - Tabla de canales
    - PCB (PID, prioridad, …)
    - Signal pendientes

Nuevo grafo de estados



Procesos suspendidos

- Con procesos suspendidos



UNIX

- Creación de procesos
  - Se heredan los recursos del padre
  - Una vez creado, se pueden modificar los recursos asignados
  - Tipos de primitivas:
    - fork
    - fork + exec
    - clone

UNIX

- Finalización de procesos
  - El proceso puede finalizar y comunicar un resultado al padre
    - exit
  - El proceso padre o el administrador pueden matar el proceso (o el mismo se puede suicidar)
    - Kill
  - El Sistema puede matar al proceso debido a un malfuncionamiento de éste (excepción, sobrepasar limites de ejecución, ...)

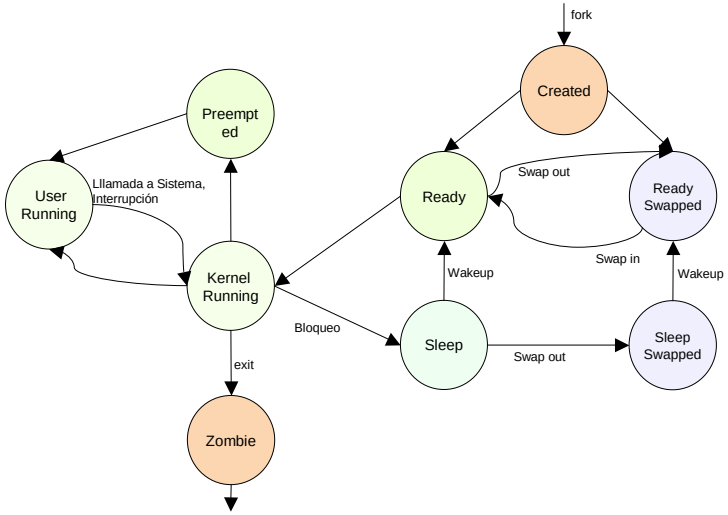
UNIX

- Sincronización entre procesos
  - Sincronización padre-hijo
    - permite recoger el resultado del hijo
      - Wait, waitpid
- Información de un proceso
  - Obtención de identificadores
    - getpid
  - Información sobre el uso de recursos
    - getrusage
  - Profiling y debugging del proceso
    - ptrace

Proceso nulo

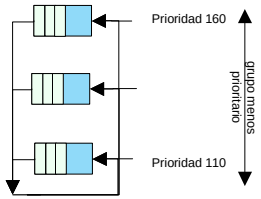
- Proceso que se ejecuta cuando no hay procesos disponibles para ejecutarse
  - El procesador debe ejecutar alguna cosa
- No cuenta como tiempo útil ni como proceso de usuario
- En general no hace nada
  - for (; ;);
  - Se puede utilizar para realizar tareas poco prioritarias del kernel cuando el procesador está libre
- Pueden existir otros procesos de sistema encargados de diferentes tareas

Unix: Estados

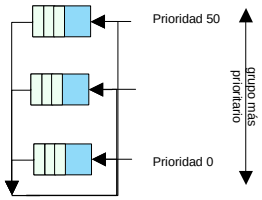


Unix: Planificación (timesharing)

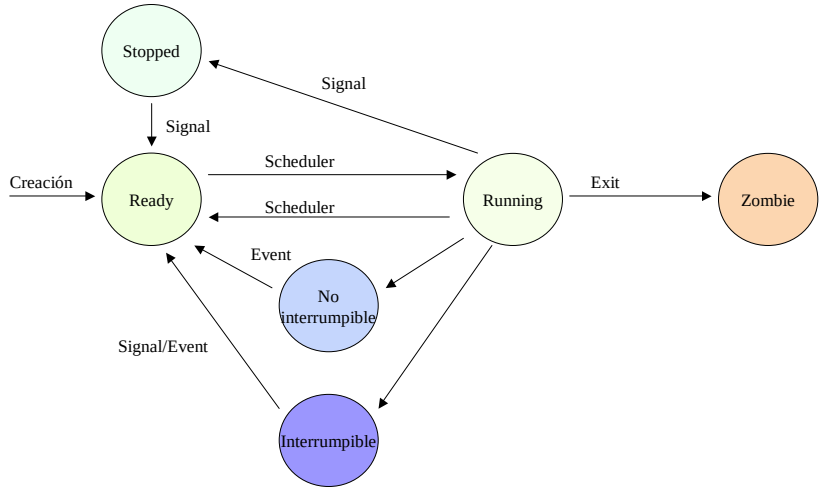
- Colas multinivel realimentadas
  - No apropiativas
  - Round Robin dentro de la cola
- Cada segundo se recalculan las prioridades
  - $P_{j,i} = Base_j + CPU_{j,i-1}/2 + nice_j$
    - Prioridades más bajas son más prioritarias
    - $CPU_{j,i} = U_{j,i} + CPU_{j,i-1}/2$
    - $Base_j$ = Prioridad base según el tipo de proceso
    - $U_{j,i}$ = Utilización del procesador por el proceso
    - $Nice_j$ = Prioridad decidida por el usuario



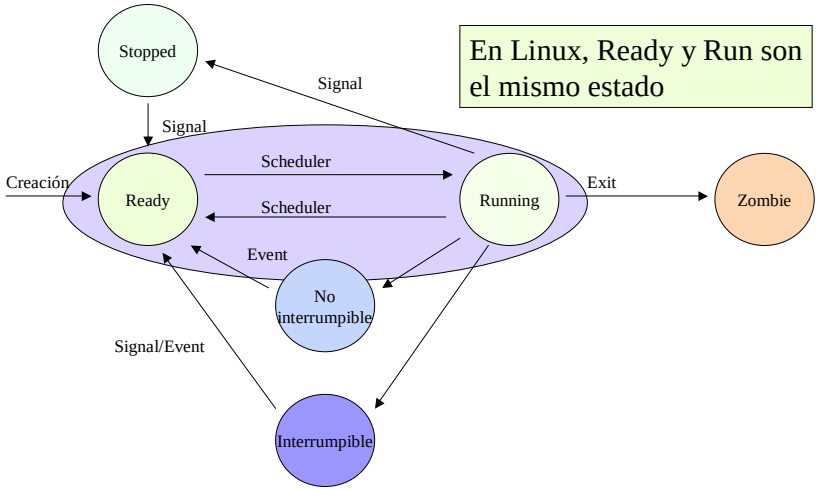
. . .



Linux: Estados



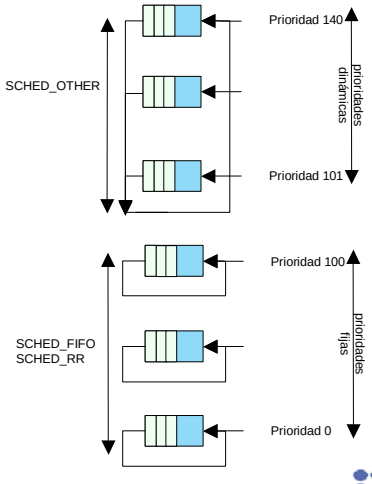
Linux: Estados



En Linux, Ready y Run son el mismo estado

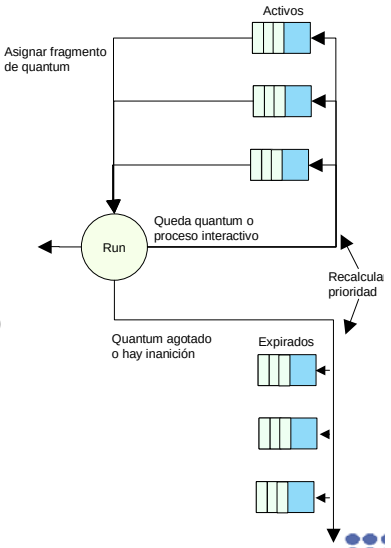
Linux: Planificación

- Cada proceso puede tener una planificación diferente
  - SCHED_FIFO: FCFS
  - SCHED_RR: Round Robin
    - Quantum variable en función de la prioridad
  - SCHED_OTHER
    - Puede haber SCHED_FIFO y SCHED_RR en la misma cola



Linux: SCHED_OTHER

- Dos conjuntos de procesos
  - Activos y Expirados
  - Cuando no quedan activos: intercambiar
- Prioridades dinámicas
  - Basandose en la E/S
- Quantum
  - No se pierde al abandonar la CPU
  - en función de la prioridad (10ms a 200ms)
  - Cuando se acaba pasa a Expirados
  - Round Robin con fragmentos del quantum
- Bonus a los procesos interactivos
  - Mayor prioridad (+1...5)
  - Se intenta mantenerlos activos



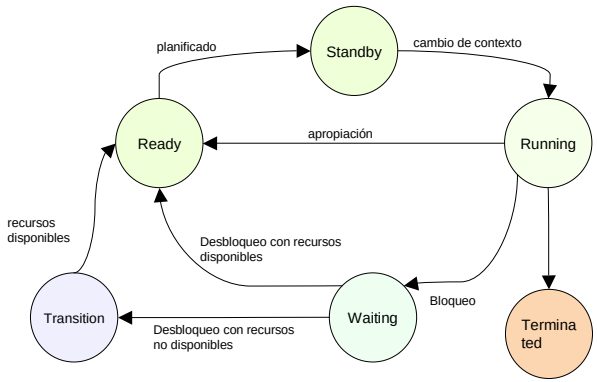
Linux

- En Linux, un proceso se llama task
- La descripción del PCB está en struct task_struct:
  - Estado del proceso
  - Punteros (memoria del proceso)
  - Tamaño del proceso
  - Identificadores (usuario real, usuario efectivo)
  - Identificadores del proceso
  - Descriptor de suceso
  - Prioridad
  - ...

Linux: Mutación de procesos

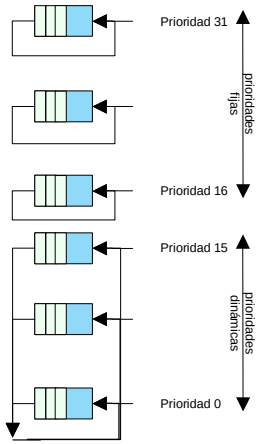
- Cargar
  - 6 llamadas a sistema: execl, execv, execle, execve, execlp. execvp (se diferencian en los parámetros)
- Que devuelve:
  - En principio nada a menos que haya un error con el fichero a ejecutar
- IMPORTANTE: No es como una llamada a rutina, al acabar, NO se vuelve al código anterior

W2K: Estados



Win2K: planificación

- Dos grupos de prioridades: fijas y dinámicas
  - Round Robin en cada prioridad
    - quantum variable según velocidad y número de procesadores
  - En caso de tener prioridades dinámicas:
    - Cuando un proceso agota su quantum baja de prioridad
    - Cuando un proceso hace E/S antes de finalizar su quantum sube de prioridad
      - Si la E/S era sobre un dispositivo *interactivo (teclado, pantalla) sube más*



Win2K: procesos

- La relación padre-hijo no es tan directa como en UNIX
  - Es fácil saber que proceso es hijo de otro
  - Es difícil saber cual es el padre de un proceso
- Un proceso hijo NUNCA es la copia del padre:
  - Siempre se tiene que especificar el ejecutable que se va a cargar
  - Algunos atributos del padre se pueden heredar
  - Aunque se tiene que decir de forma explícita cuales de los atributos heredables, se van a heredar

Win2K: planificación

- Al crear un proceso se indica a qué clase de prioridad pertenece
  - Si no se indica, el SO da una por defecto
- Se puede abandonar la CPU en cualquier momento:
  - SwitchToThread
  - El planificador busca otro thread para ejecutarse

Win2K: procesos

- Creación de un proceso:
  - CreateProcess
  - Semejante a Fork+Exec de linux
  - Se le pasa como parámetros:
    - Ejecutable
    - Línea de comandos
    - Atributos de seguridad del proceso y del thread
    - Herencia
    - Flags de prioridad y de control de creación
    - La lista de variables de entorno con su valor
    - Directorio actual
    - Características de la ventana donde se va a ejecutar
  - Devuelve información sobre el proceso y thread creado

Win2K: procesos

- Finalización del proceso:
  - ExitProcess: más expeditiva. No comprueba las relaciones entre el proceso que termina y otros procesos
    - Se pueden producir deadlocks
  - TerminateProcess: más segura. Comprueba todas las relaciones entre el proceso que termina y otros procesos
  - En ambas se pasa como parámetro el estado de finalización del proceso
  - El padre puede leer este estado cuando quiera aunque el PCB del proceso desaparece cuando ningún otro proceso lo referencia

Win2K: procesos

- En el PCB, a parte de la información necesaria para trabajar con el proceso, se incluyen contadores de rendimiento:
  - Número de veces que lee/escribe
  - Bytes leídos/escritos
  - Tiempo de CPU acumulado
  - Tiempo de ejecución en estado de usuario y de sistema
  - …
- Útiles para obtener estadísticas del rendimiento del proceso/sistema

Win2K: procesos

- Información del proceso:
  - GetCommandLine
  - GetCurrentProcessId
  - GetProcessId
  - GetCurrentProcessorNumber
  - GetPriorityClass
  - Process32First (semejante a getppid)
  - GetProcessIOCounters
  - GetProcessTimes
  - GetProcessIdOfThread
  - …

Win2K: procesos

- Se puede pedir información sobre cualquier proceso del sistema
  - Siempre que se tengan permisos sobre ese proceso
  - Existen funciones del Win32API que nos devuelven HANDLEs a procesos existentes
  - Se puede consultar (nunca modificar) información de otros procesos (aunque no sean hijos) desde el nivel de privilegios de usuario

Win2K: procesos

- Se pueden crear conjuntos de procesos:
  - Job Objects
  - Contienen uno o varios procesos que se pueden añadir de forma dinámica
  - Al cambiar alguna característica de un proceso, se cambia automáticamente en todos los procesos que forman parte del job
  - Un proceso perteneciente a un job no puede cambiar ninguna de sus características
  - Cuando se acaba el job, se matan todos los procesos
  - Muy útil para ejecutar procesos batch (transacciones por lotes)