Gestión de Procesos

Sistemas Operativos - ITIG

Álvaro Polo Valdenebro

apoloval@gsyc.es

Abril 2009

©2009 GSyC
Algunos derechos reservados.
Este trabajo se distribuye bajo la licencia
Creative Commons Attribution Share-Alike
disponible en http://creativecommons.org/licenses/by-sa/2.1/es

Contenidos

1. Estructura
2. Operaciones sobre procesos
3. Planificación
4. Hilos de ejecución

Contenidos

1 Estructura

2 Operaciones sobre procesos

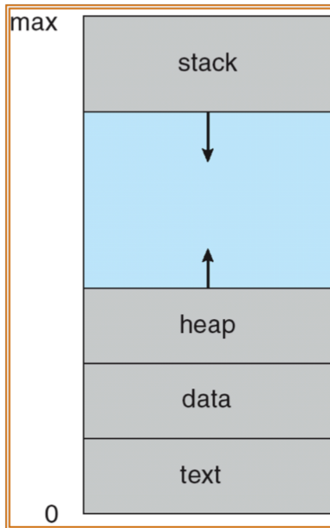
3 Planificación

4 Hilos de ejecución

Proceso

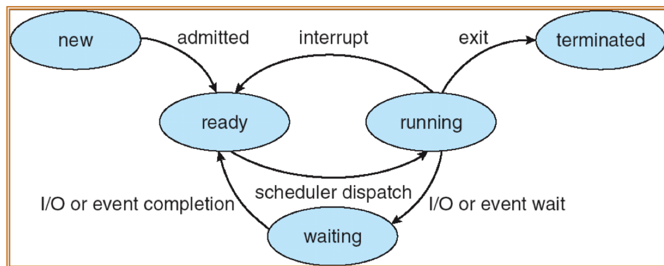
- Instancia de un programa en ejecución
  - Un mismo programa puede tener *n* réplicas en memoria
  - Cada proceso se considera independiente (aunque sea una instancia de un mismo programa)
- Es una entidad activa
  - Contador de programa
  - Registros del procesador
  - Código
  - Pila de ejecución
  - Datos
  - Heap

Memoria de un proceso



Estado de un proceso

Los procesos cambian de estado



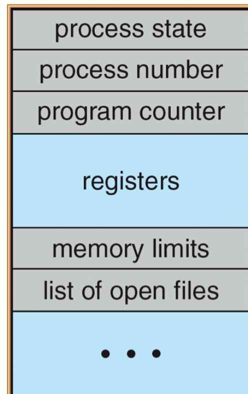
- **Nuevo**; el proceso está siendo creado
- **En ejecución** en alguna CPU
- **En espera** de datos E/S o que se produzca un evento
- **Preparado** para ejecutar
- **Terminado**, su ejecución ha finalizado

GSyC

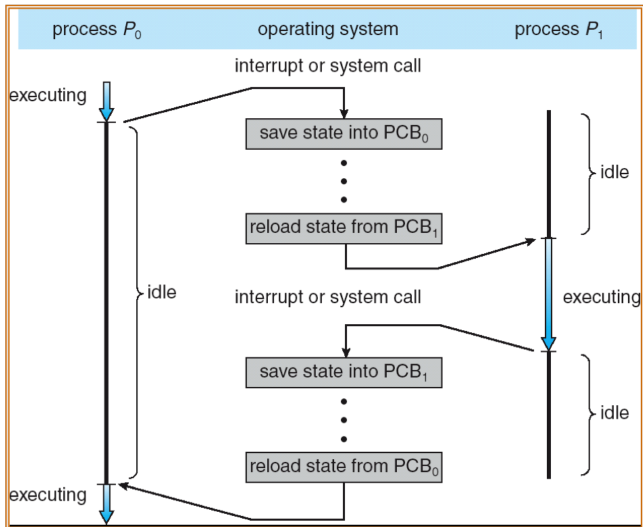
Bloque de control del proceso (PCB)

El SO emplea estructuras para gestionar los procesos

- **Estado del proceso**: en ejecución, en espera, preparado, ...; pid, uid, gid
- **Contador de programa**, indica la dirección de la siguiente instrucción a ejecutar
- **Registros de la CPU**, estado de los mismos
- **Información de planificación**, empleada para almacenar los parámetros que determinan la prioridad de ejecución
- **Información de memoria**, almacena datos acerca de qué recursos de memoria tiene asignado el proceso
- **Estadísticas de ejecución**: tiempo invertido
- **Información de recursos** asignados al proceso, archivos abiertos, etc.



Conmutación entre procesos



Contenidos

1. Estructura
2. Operaciones sobre procesos
3. Planificación
4. Hilos de ejecución

Creación de procesos

- Un proceso (padre) puede crear varios procesos (hijos)
- Estos a su vez pueden crear más hijos, dando lugar a un árbol de procesos
- Cada proceso posee un identificador o pid
- Los procesos padres pueden compartir sus recursos con sus procesos hijos (ficheros, dispositivos, memoria compartida, ...)

Ejecución de padre e hijo

Tras crear un proceso hijo, el proceso padre puede

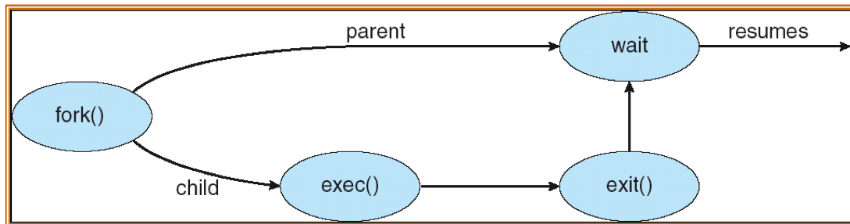
- Ejecutar otra tarea concurrentemente con su hijo
- Esperar a que sus hijos hayan terminado

El nuevo proceso, puede

- Ser un clon del proceso padre (conserva una réplica del mismo código y datos)
- Cargar un nuevo programa y comenzar a ejecutarlo

GSyC

Ejecución de padre e hijo

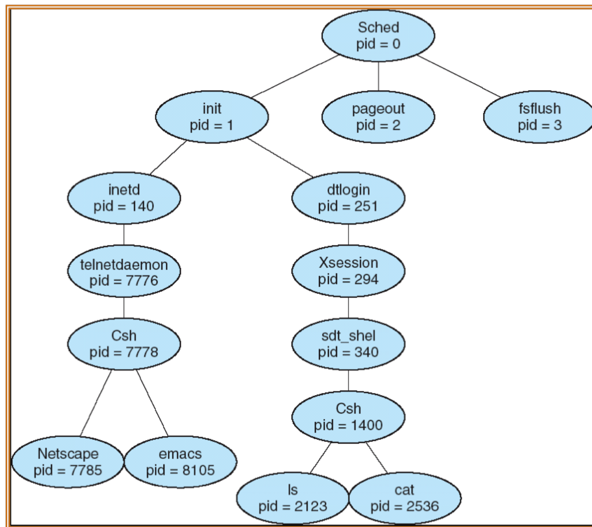


Ejecución de padre e hijo

Ejecución de padre e hijo en Unix

```
int main(int argc, char *argv[])
{
  pid_t  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);
  }
}
```

Árbol de procesos en Solaris



Terminación de procesos

- Un proceso termina
  - Invocando una llamada al sistema (`exit()`)
  - Terminando el programa principal
  - Cuando se produce un error irrecuperable o no controlado
- En la terminación, el proceso indica un código de estado al padre
  - `exit()` admite un valor entero que es recogido por `wait()`
  - El mismo valor se puede transmitir como resultado de `main()`
- El sistema operativo libera todos los recursos del proceso

GSyC

Herencia de procesos

Algunos SSOO permiten que un padre termine antes que sus hijos

- Si lo permite, el proceso hijo lo hereda otro proceso (`init` en UNIX)
- Si no lo permite, se produce una terminación en cascada

Señales entre procesos

Los procesos se pueden enviar señales a otros procesos (o grupos de procesos)

- De terminación (`SIGTERM`, `SIGKILL`)
- El propio núcleo manda señales de error a los procesos (`SIGSEGV`, `SIGABRT`, ...)
- Se comprueba la autoridad del emisor para mandar la señal

Contenidos

1 Estructura

2 Operaciones sobre procesos

3 Planificación

4 Hilos de ejecución

Planificación de procesos

El objetivo es la multiprogramación

- Admitir varios procesos ejecutando pseudo-paralelamente
- Maximizar el uso de la CPU

Para ello disponemos de un planificador de procesos...

- Selecciona un proceso para ser ejecutado
- Si hay más procesos que CPUs, algunos tendrán que esperar
- Puede emplear diversas políticas para seleccionar procesos

Colas de planificación

Organizamos los procesos en colas

- Una **cola de trabajos** contiene todos los procesos del sistema
- Una **cola de procesos preparados** mantiene una relación de procesos en espera de ejecutarse
- Varias **colas de dispositivo** que contienen los procesos en espera de E/S en un periférico determinado

Los procesos migran de unas colas a otras a lo largo de su ciclo de vida

GSyC

Colas de dispositivo

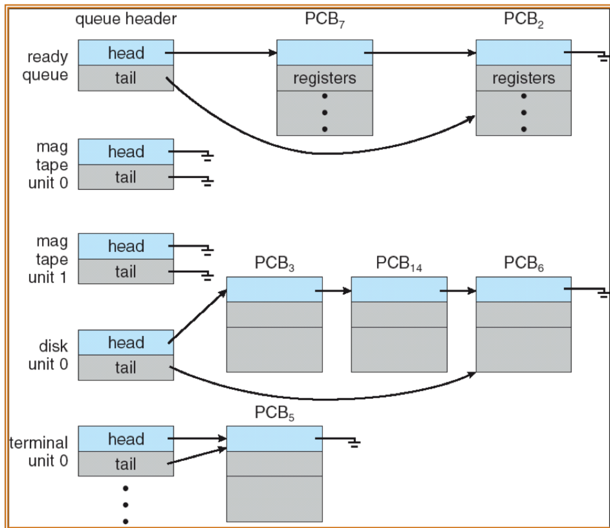
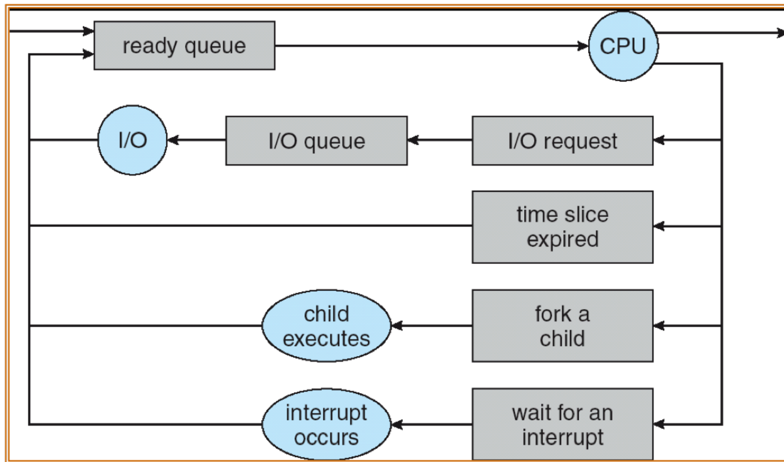


Diagrama de colas



Tipos de planificación

El planificador interviene en las siguientes circunstancias

1. Un proceso cambia del estado de ejecución al estado de espera
2. Un proceso cambia del estado de ejecución al estado preparado
3. Un proceso cambia del estado de espera al estado preparado
4. Un proceso se termina

Para 1) y 4) siempre se selecciona un nuevo proceso.
Para 2) y 3)...

Tipos de planificación

El planificador interviene en las siguientes circunstancias

1. Un proceso cambia del estado de ejecución al estado de espera
2. Un proceso cambia del estado de ejecución al estado preparado
3. Un proceso cambia del estado de espera al estado preparado
4. Un proceso se termina

Para 1) y 4) siempre se selecciona un nuevo proceso.
Para 2) y 3)... podría seleccionarse o no

Planificación sin desalojo o cooperativa

- Sólo se selecciona un nuevo proceso en los casos 1) y 4)
- El proceso en ejecución no se expulsa de la CPU hasta que
  - Pasa al estado de espera
  - Termina
- Modelo forzado por restricciones del hardware (ausencia de temporizador)

Planificación con desalojo o apropiativa

- En cualquier caso puede seleccionar un nuevo proceso
- El proceso en ejecución puede expulsase de la CPU cuando
  - Pasa al estado de espera
  - Se le acaba su rodaja de CPU
  - Algún otro proceso prioritario pasa a estar listo para ejecutarse
  - Termina
- Modelo más empleado

Planificación apropiativa y concurrencia

Deben tener en cuenta los problemas de concurrencia

- Los procesos: el desalojo introduce el problema de la sección crítica
- El núcleo: un proceso en mitad de una llamada al sistema podría ser desalojado, corrompiendo datos del núcleo

Cambios de contexto

Cuando el planificador escoge un nuevo proceso para ejecutar se produce un cambio de contexto, orquestado por el *dispatcher*

1. El proceso anterior se suspende
2. Su contexto se guarda en su PCB
3. Se carga el contexto del proceso seleccionado
4. El proceso seleccionado pasa a ejecutarse

Problemas

- El tiempo invertido en el cambio de contexto se desperdicia
- ¿Cuánto? Depende de la arquitetura

Criterios de planificación

A la hora de escoger procesos para ejecutar, se tiene en cuenta

- **Utilización de la CPU**, teniéndola tan ocupada como sea posible
- **Tasa de procesamiento**, completando el mayor número de procesos por unidad de tiempo
- **Tiempo de ejecución**, reduciendo el tiempo que tarda un proceso en completarse
- **Tiempo de espera**, reduciendo el tiempo que pasa un proceso en las colas de espera y en los cambios de contexto
- **Tiempo de respuesta**, reduciendo la latencia en la comunicación de datos E/S con un proceso

Algoritmo FCFS

First-come, first-served

- Se asigna la CPU al primer proceso que la solicita
- La cola de procesos en espera es FIFO
- Empleado en planificadores cooperativos

Pros

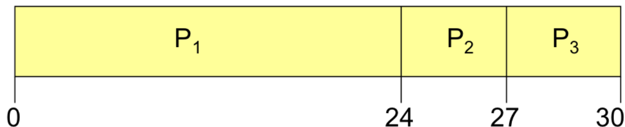
- Simple de implementar y comprender
- Bueno para trabajos por lotes

Contras

- Tiempos de espera muy elevados
- Efecto convoy
- Nefasto para sistemas interactivos

Algoritmo FCFS: ejemplo

Proceso	Rodaja de tiempo
P_1	24ms
P_2	3ms
P_3	3ms



Tiempo de espera: $P_1 = 0$; $P_2 = 24$; $P_3 = 27$

Tiempo medio $(0 + 24 + 27)/3 = 17$

Algoritmo por turnos (*round robin*)

Evolución de FCFS

- Modelo apropiativo
- La CPU se divide en cuantos de tiempo (soporte de temporizadores)
- Si se vence el cuanto, el proceso es desalojado

Pros

- Simple de implementar y comprender

Contras

- El rendimiento depende de la longitud del cuanto

Algoritmo SJF

Shortest-job first

- Se asigna la CPU al proceso con la ráfaga de CPU más corta
- Normalmente, no se conoce con exactitud: se predice (promedio exponencial de las anteriores)
- Modo apropiativo: si un proceso con ráfaga más corta entra en espera, este podría expulsar al proceso en ejecución si fuera conveniente

Pros

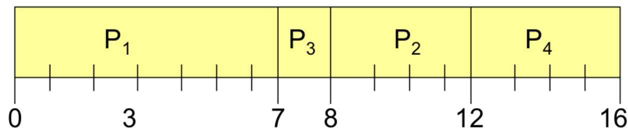
- Minimiza el tiempo medio de espera

Contras

- Las predicciones pueden ser erroneas

Algoritmo SJF: ejemplo cooperativo

Proceso	Tiempo de llegada	Rodaja de tiempo
P_1	0.0	7ms
P_2	2.0	4ms
P_3	4.0	1ms
P_4	5.0	4ms

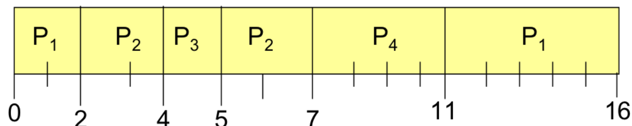


Tiempo de espera: $P_1 = 0$; $P_2 = 6$; $P_3 = 3$; $P_4 = 7$

Tiempo medio $(0 + 6 + 3 + 7)/4 = 4$

Algoritmo SJF: ejemplo apropiativo

Proceso	Tiempo de llegada	Rodaja de tiempo
P_1	0.0	7ms
P_2	2.0	4ms
P_3	4.0	1ms
P_4	5.0	4ms



Tiempo de espera: $P_1 = 9$; $P_2 = 1$; $P_3 = 0$; $P_4 = 2$

Tiempo medio $(9 + 1 + 0 + 2)/4 = 3$

Algoritmo de planificación por prioridades

- Los procesos tienen asignada una prioridad
  - Interna: asignada por el núcleo: límites de tiempo, requisitos de memoria, estimaciones de ráfagas, etc.
  - Externa: asignada por el usuario: importancia, coste económico, etc.
- La CPU se le asigna al proceso con prioridad más alta
  - Apropiativa: un proceso que entra en espera puede expulsar al proceso en ejecución si su prioridad es la mayor
  - Cooperativa: un proceso que entra en espera se coloca al comienzo de la cola si su prioridad es la mayor

Pros

- Minimiza el tiempo medio de espera

Contras

- Muerte por inanición (solución con envejecimiento)

GSyC

Sistemas multiprocesador

Dos esquemas

- **Multiprocesamiento asimétrico**. El núcleo se ejecuta en un procesador, el resto se dedica a código de usuario (ejecutar procesos)
- **Multiprocesamiento simétrico (SMP)**. Cualquier tarea (núcleo o proceso) puede ejecutarse en cualquier procesador.

GSyC

Procesamiento simétrico (SMP)

Consideraciones de implementación

- **Gestión de colas**.
  - Todas las CPUs comparten las mismas colas de procesos
  - Cada CPU tiene su propia cola de procesos
- **Concurrencia**. Las colas de procesos se comparten
- **Afinidad al procesador**. No es buena idea migrar procesos de una CPU a otra: se pierden las caches.
  - Afinidad suave: se procura no migrar, pero no se garantiza
  - Afinidad dura: el proceso no migrará de CPU
- **Equilibrio de carga**. Para colas privadas, algunas CPUs pueden quedarse sin trabajo mientras otras tienen las colas llenas.
  - *Push migration*, un agente monitor trata de equilibrar periódicamente
  - *Pull migration*, una CPU ociosa "roba" procesos en espera a otra CPU

Contenidos

1 Estructura

2 Operaciones sobre procesos

3 Planificación

4 Hilos de ejecución

Motivación

Puede ser útil escribir programas que realicen tareas en paralelo

- Aprovechan la multiprogramación
- Aumentan la capacidad de respuesta
- Ejemplo: navegador web, servidor web, procesador de textos, ...

Para ello podemos

- Podemos usar procesos
- Problemas...

GSyC

Motivación

Puede ser útil escribir programas que realicen tareas en paralelo

- Aprovechan la multiprogramación
- Aumentan la capacidad de respuesta
- Ejemplo: navegador web, servidor web, procesador de textos, ...

Para ello podemos

- Podemos usar procesos
- Problemas...
  - Costosos de crear y gestionar

Motivación

Puede ser útil escribir programas que realicen tareas en paralelo

- Aprovechan la multiprogramación
- Aumentan la capacidad de respuesta
- Ejemplo: navegador web, servidor web, procesador de textos, ...

Para ello podemos

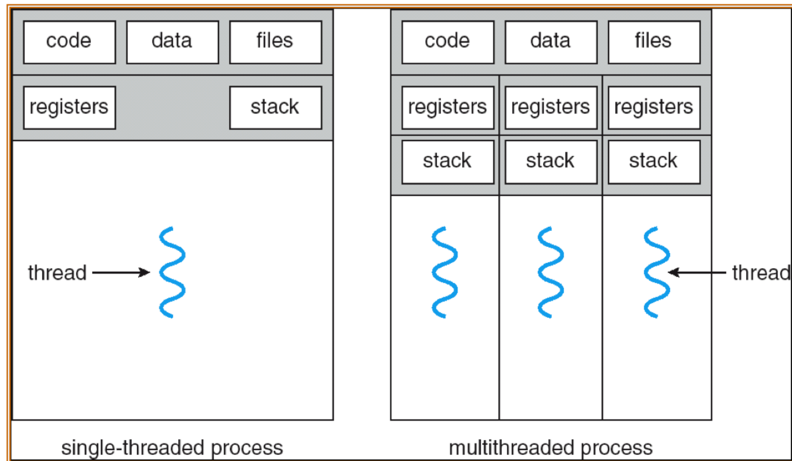
- Podemos usar procesos
- Problemas...
  - Costosos de crear y gestionar
  - Difíciles de programar (¿memoria compartida?)

Los hilos de ejecución

Cada proceso puede tener varios hilos o hebras de ejecución (*threads*)

- Cada hebra tiene mantiene su propio
  - Contador de programa
  - Estado de registros
  - Pila de ejecución
- Las hebra de un mismo proceso comparten
  - Segmento de código
  - Segmento de datos
  - Segmento *heap*
  - Recursos (ficheros, dispositivos, ...)

Monohebra vs multihebra

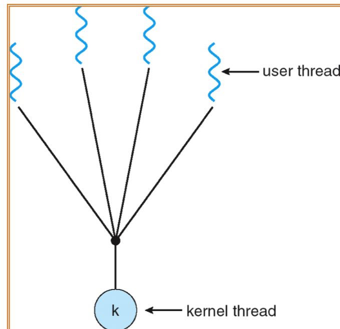


Modelos multihebra

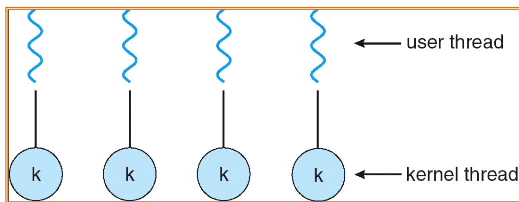
- Las **hebras de usuario** se proporcionan mediante librerías
  - Planificador implementado en espacio de usuario: el SO sólo ve una hebra
  - No pueden ejecutarse al mismo tiempo
- Las **hebras del kernel** las proporciona el núcleo del SO
  - Usan el planificador del SO
  - Pueden ejecutarse al mismo tiempo

Modelo muchos-a-uno

- Las hebras de usuario se implementan con una única hebra de kernel
- Las llamadas al sistema pueden bloquear al resto de hebras
- No es posible ejecutar distintas hebras en distintas CPUs al mismo tiempo
- La creación de hebras es barata
- Ejemplo: GNU Portable Threads



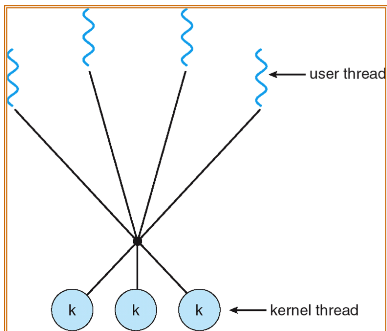
Modelo uno-a-uno



- Cada hebra de usuario se implementa con una hebra de kernel
- Las llamadas al sistema no bloquean otras hebras
- Es posible ejecutar distintas hebras en varias CPUs al mismo tiempo
- La creación de hebras es costosa
- Ejemplo: Windows, Linux

Modelo muchos-a-muchos

- M hebras de usuario se multiplexan en N hebras de kernel ($M \geq N$)
- Las llamadas al sistema no bloquean algunas hebras
- Es posible ejecutar algunas hebras en varias CPUs al mismo tiempo
- La creación de hebras es más barata que uno-a-uno y más costosa que muchos-a-uno
- Ejemplo: Solaris 8, HP-UX



GSyC