eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ

7.1 Gestión de procesos.

Introducción a los Sistemas Operativos, 2023-2024

Pablo González Nalda

Depto. de Lenguajes y Sistemas Informáticos
EU de Ingeniería de Vitoria-Gasteiz,
UPV/EHU

24 de marzo de 2024

Contenidos de la presentación

Contenidos

Mecanismos para acelerar programas

En principio sólo estamos ejecutando **un programa** en el ordenador.

La bajada de rendimiento se debe a la diferencia de velocidad entre dispositivos de E/S y procesador.

a) E/S Asíncrona

- Ordenación especial de las instrucciones.
- Incrementa la complejidad de la programación.
- Necesita sincronización posterior.

b) Buffering: utilizar varios búferes donde se van a realizar E/S de manera transparente al programa.

- Probabilidad de mucha E/S, lleno o vacío
- Ocupar memoria

Mecanismos para acelerar programas

c) Spooling: uso de almacenamiento intermedio.

- Cuando se utilizan dispositivos de E/S lentos y no de forma interactiva.
- Se puede utilizar un dispositivo intermedio más rápido.
- Suele haber un hardware dedicado

Ventajas e inconvenientes

Ventajas

- solapa E/S con su propia ejecución.
- solapa E/S con la ejecución de otro trabajo.
- es fácil obtener varias copias del trabajo.

Inconvenientes

- Espacio en disco.
- Tiempo de tratamiento.
- Se sigue sin resolver el problema: Un usuario, en general no tiene ocupada la CPU todo el tiempo.

Soluciones

Posible Solución: si tenemos suficiente espacio en memoria cargamos varios programas.

Cuando un programa en ejecución necesita esperar por E/S, se le para y si hay otro que pueda ejecutarse, que lo haga.

Contexto de ejecución

Los programas deben realizar su función independientemente de si se ejecutan solos (monoprogramación) o concurrentemente con otros (multiprogramación).Para ello se debe crear un *Contexto de Ejecución* con la información sobre la ejecución de un programa.

El SO mantiene por cada programa la información necesaria para ejecutar correctamente el programa. El hecho de pasar de ejecutar un programa a ejecutar otro se llama *cambio de contexto* (*Context Switch, CS*).

PROCESO = Información de contexto + programa(inst. y datos)

PROCESO: Instancia de un programa en ejecución.

Información de contexto: Identificador, Tabla de Canales, Estado de los recursos del procesador (registros,...)

Contenidos

Control de procesos

Flujo de ejecución: código ejecutable que ocupa la CPU. Se determina con el registro PC *(Program Counter)*, que indica dónde está en la RAM la instrucción de Lenguaje Máquina que se está ejecutando.

Un proceso es un conjunto de uno o más flujos de ejecución denominados `hilos`, junto al contexto de ejecución. El contexto es el conjunto de todos los datos que permiten al SO ejecutar el programa: código y datos, pila, estado de la memoria y de la E/S (datos de gestión relativos al fichero).

Tipos de hilos

Kernel Level Thread El SO gestiona los hilos

User Level Thread La aplicación gestiona los hilos

Biblioteca `Pthreads` (*Posix Threads*): biblioteca estándar que simplifica el desarrollo de aplicaciones basadas en hilos. Además asegura la portabilidad de las aplicaciones a diferentes entornos.

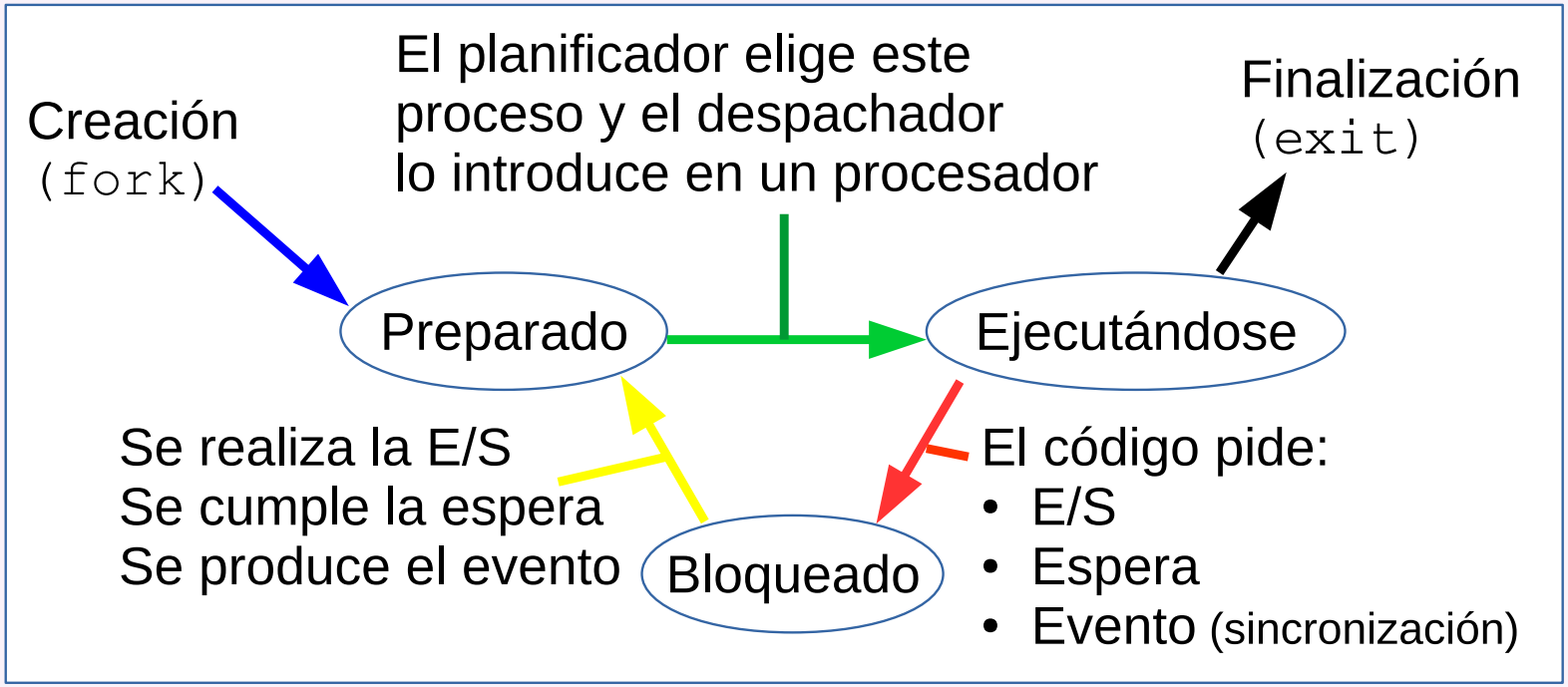
Estados y cambio de contexto

Para repartir las CPUs entre los procesos se define en qué estado se encuentran. Los principales son *ejecutándose, bloqueado y preparado*.

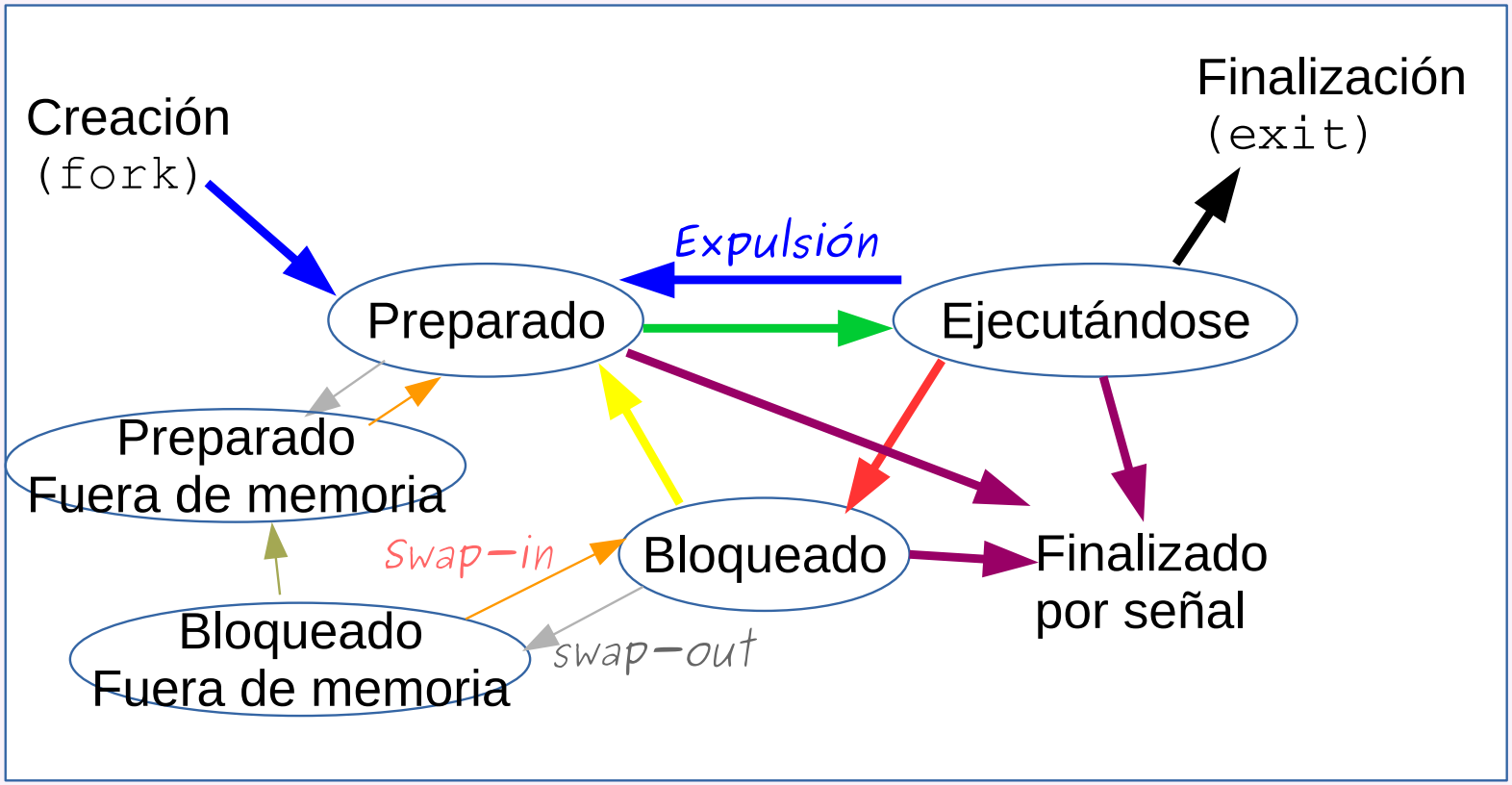
Un `cambio de contexto` *(context switch, CS)* consiste en guardar el estado intermedio de ejecución de un proceso que está en estado *ejecutándose* y sustituirlo por otro que estaba en estado *preparado*. Esta tarea la lleva a cabo el `planificador`, como último paso tras decidir qué proceso debe ser el siguiente en ejecutarse.

No es una tarea excesivamente pesada puesto que se copian punteros o apuntadores (direcciones de memoria donde comienzan las estructuras de datos).

Estados y cambio de contexto



Estados y cambio de contexto



Expulsión

Un proceso puede salir voluntariamente (debido a su código) del estado *Ejecutándose* por una E/S, una espera de tiempo o una espera a un evento. En general, por una *Llamada al Sistema* el proceso pasa a *Bloqueado* a la espera de cumplir una cierta operación.

Un proceso puede salir **forzosamente** del estado *Ejecutándose* y volver a *Preparado* cuando:

Expulsión por tiempo: se le acaba el *quantum o tiempo máximo de ejecución ininterrumpida*

Expulsión por evento: otro proceso cambia a estado *Preparado* y se quiere primar la interactividad

PCB: Bloque de control del proceso

PID	entero identificador único
estado	Posición en la cola de preparados, o bloqueado o ejecutándose (+)
datos para planif.	Prioridad, *quantum*...
contexto de ejecución	estado de CPU y pila
contabilidad	gestión de consumo, para planificación y *sysadmin*.
MMU	`Unidad de Gestión de memoria` (tablas de páginas...)
E/S	Peticiones pendientes, dispositivos y canales

No tiene por qué ser una estructura real, puede estar distribuida en diferentes tablas del `kernel`.

Proceso nulo o *idle*

Proceso nulo o *idle*: proceso vacío con operaciones `NOP` o `HLT` (*No Operación* o *HaLT*) que ocupan o paran el procesador cuando no hay otro proceso que hacer. Las operaciones `NOP` y `HLT` ponen al procesador en una situación de ahorro de energía, junto con el *Gobernador* con los estados C1 y C3.

`Discusión del proceso nulo en StackExchange`

> *Y*ou can see various implementations of idle tasks for example in arch/x86/kernel/process.c in the Linux kernel: the basic one just calls HLT, which stops the processor until an interrupt occurs (and enables the `C1 energy-saving mode`)

`ACPI` configura y monitoriza componentes hardware y realiza gestión de energía por ejemplo *durmiendo* componentes no usados.

Contenidos

Llamadas al sistema para identificación del proceso

Identificación de procesos

```
1 int getpid(); // Devuelve el PID del proceso
```

```
int getppid(); // identificador (PID) del proceso padre
```

```
int getuid(); // identificador del usuario al que pertenece
     del proceso (uid)
```

```
int geteuid(); // id. efectivo, al usar con SUID
```

```
int getgid(); // Devuelve el identificador del grupo del
    proceso (gid)
```

Llamadas al sistema para lanzar procesos

Clonar procesos

```
int fork();
```

Cargar nuevo ejecutable en un proceso

```
execlp("ls", "ls", "-l", NULL);
```

Llamadas al sistema para control del proceso

Finalizar procesos

```
void exit(int estado);
```

Unix guarda el código de finalización (estado) hasta que el proceso padre ejecute `wait()`

`wait()` bloquea al proceso que lo llama hasta que uno de sus hijos finalice. Si no tiene hijos devuelve -1 sin bloquear al proceso que llama. Devuelve el identificador del proceso hijo finalizado y el código de retorno devuelto por el proceso hijo.

```
int wait(int *estado);
```

Llamadas al sistema para señales

Las señales son activaciones asíncronas de funciones del proceso. Las funciones están en una tabla y por defecto tienen un comportamiento por defecto. Ver en `man 7 signal`

Se manda la señal `nombreS` al proceso pid.

```
int kill(int pid, int nombreS);

int signal (int nombreS, void fun());
```

Relaciona la función `fun` con la señal `señal`

Llamadas al sistema para control de tiempos

Llamadas para control de tiempos y bloqueo

```
void unsigned int sleep (unsigned int segundos);
// Unix bloquea el proceso durante ese tiempo o hasta que
    llegue una senal
void pause ();
// Unix bloquea el proceso hasta que llegue una senal
```

Llamadas al sistema para señales y tiempos

Gestión de la señal SIGALRM

```
unsigned alarm (unsigned segundos);
// Unix manda SIGALRM al proceso que ha hecho la llamada,
    pasados los segundos
```

```
void fnula() {
}
void esperar (unsigned seg) {
    signal(SIGALRM,fnula);
    alarm(seg);
    pause();
}
```

Llamadas al sistema para control de prioridades

Llamadas al sistema para control de prioridades

```
int nice (int valor);
```

Vale para modificar las prioridades de los procesos en el acceso a la CPU. De forma análoga, se puede cambiar la prioridad de acceso a disco y red.

Siempre es posible bajar la prioridad de los procesos, sin embargo, para subirla es necesario ser *root* o administrador.

Contenidos

Cambio de contexto

La pérdida de eficiencia por un CS es debida al *cambio de localidad* en `cachés`, `memoria virtual` y `búferes`.

¿Qué puede provocar un CS?

Llamada al sistema	Un proceso se bloquea y otro entra a ejecutarse
Fin de q	Un proceso pasa a preparado y otro pasa a ejecutarse
IRQ hw	Se produce una expulsión por evento, o el programa acaba por un error
Sincronización	Se libera una Sección Crítica y un proceso se desbloquea y pasa a ejecutarse Un proceso se bloquea en una SC y otro pasa a ejecutarse
Fin de proceso	Un proceso termina y otro pasa a ejecutarse Un proceso termina y su padre se desbloquea
Inicio proceso	Un proceso nuevo desde `fork` pasa a preparado y entra a ejecutarse

Contenidos

¿Más preguntas?

¿Más preguntas?

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ

7.1 Gestión de procesos.

Introducción a los Sistemas Operativos, 2023-2024

Pablo González Nalda

Depto. de Lenguajes y Sistemas Informáticos
EU de Ingeniería de Vitoria-Gasteiz,
UPV/EHU

24 de marzo de 2024