

Procesos

Rodolfo Baader


Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina

Sistemas Operativos, segundo cuatrimestre de 2014

(2) Dónde estamos...

- Vimos
 - Qué es un SO.
 - Un administrador de recursos.
 - Una interfaz de programación.
 - Un poco de su evolución histórica.
 - Su misión fundamental.
 - Hablamos de multiprogramación.
 - Qué cosas son parte del SO y cuáles no.
- Vamos a ver...
 - qué cosas hay detrás del concepto de proceso.
 - Y qué abstracciones nos presenta el SO para lidiar con ellas.
 - Es decir, una parte de la API del SO.

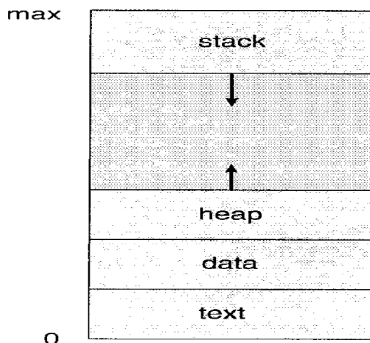
(3) Los procesos

- Un programa es una secuencia de pasos escrita en algún lenguaje.
- Ese programa eventualmente se compila en código objeto, lo que también es un programa escrito en lenguaje de máquina.
- Cuando ese programa se pone a ejecutar, lo que tenemos es un *proceso*. 
- A cada proceso se le asigna un identificador numérico único, el *pid* o *process id*.

(4) Procesos

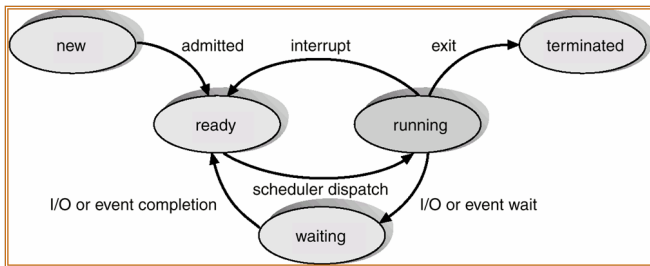
Visto desde la memoria, un proceso está compuesto por:

- El área de *texto*, que es el código de máquina del programa.
- El área de datos, que es donde se almacena el heap.
- El stack del proceso.
- ¿Dónde se almacenan las variables locales?




(5) Estado de un proceso

Estado de un proceso: a medida que un proceso se ejecuta, va cambiando de estado de acuerdo a su actividad. ⚠



- Corriendo: está usando la CPU.
- Bloqueado: no puede correr hasta que algo externo suceda (típicamente E/S lista).
- Listo: el proceso no está bloqueado, pero no tiene CPU disponible como para correr.

(6) Process Control Block

Además del proceso, el SO guarda una estructura con la información del *contexto* del proceso: la *PCB*. 

- Estado del proceso
- Contador de programa
- Registros de la CPU
- Información para el scheduler
- Información para el manejador de memoria
- Información para E/S
- etc...


(7) Procesos

- ¿Qué puede hacer un proceso?
 - Ejecutar en la CPU.
 - Hacer un *system call*.
 - Realizar entrada/salida a los dispositivos (E/S).
 - Terminar.
 - Lanzar un proceso hijo (`system()`, `fork()`, `exec()`).
- Analicemos cada una de las actividades del proceso.


(8) Actividades de un proceso: ejecutar en la CPU

- Una vez que el proceso está ejecutándose, se dedica a:
 - hacer operaciones entre registros y direcciones de memoria,
 - E/S,
 - llamadas al sistema.
- Imaginemos el programa más elemental, que sólo hace lo primero.

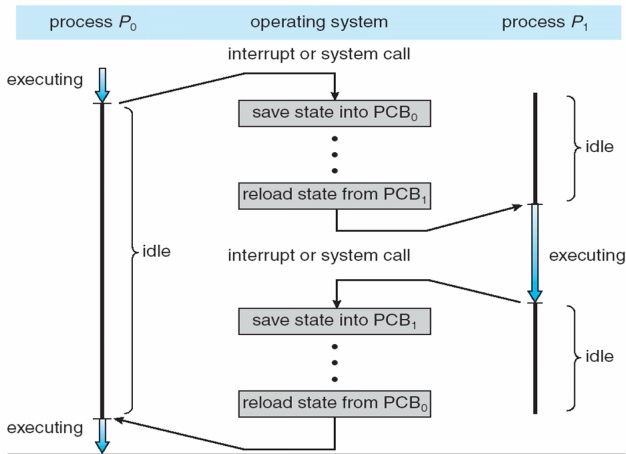
(9) Ejecutando en la CPU

- ¿Por cuánto tiempo lo dejamos ejecutar? (recordemos: sólo un proceso a la vez puede estar en la CPU).
 - Hasta que termina: Es lo mejor para el proceso, pero no para el sistema en su conjunto. Además, podría no terminar.
 - Un “ratito”. Ese “ratito” se llama *quantum*. 
- En general los SO modernos hacen *preemption*: cuando se acaba el quantum, le toca el turno al siguiente proceso.
- Surgen dos preguntas:
 - Quién y cómo decide a quién le toca.
 - Qué significa hacer que se ejecute otro proceso.


(10) Ejecutando en la CPU (cont.)

- Para cambiar el programa que se ejecuta en la CPU, debemos:
 - Guardar los registros.
 - Guardar el IP.
 - Si se trata de un programa nuevo, cargarlo en memoria.
 - Cargar los registros del nuevo.
 - Poner el valor del IP del nuevo.
 - Otras cosas que veremos más adelante.
- A esto se lo llama *cambio de contexto* o *context switch*. 
- El IP y demás registros se guardan en la *PCB*.
- Notemos: el tiempo utilizado en cambios de contexto es tiempo muerto, no se está haciendo nada productivo. Dos consecuencias de esto:
 - Impacto en la arquitectura del HW: procesadores RISC.
 - Fundamental determinar un quantum apropiado para minimizar los cambios de contexto.
- Implementación: colgarse de la interrupción del clock.


(11) Cambio de contexto



(12) Actividades de un proceso: llamadas al sistema

- Un proceso también puede hacer llamadas al sistema.
- En todas ellas se debe llamar al kernel. A diferencia de una llamada a subrutina común y corriente, las llamadas al sistema requieren cambiar el nivel de privilegio, un cambio de contexto, a veces una interrupción, etc. 
- Eso toma tiempo.

(13) Actividades de un proceso: E/S

- La E/S es leentaaa, muuuy leentaaa.
- Quedarse bloqueado esperando es un desperdicio de tiempo...
- ...porque involucra hacer *busy waiting*, es decir, gastar ciclos de CPU.
- Hay una serie de alternativas más interesantes: 
 - Polling.
 - Interrupciones.
 - Otras que no vamos a ver por ahora.

(14) Actividades de un proceso: terminación

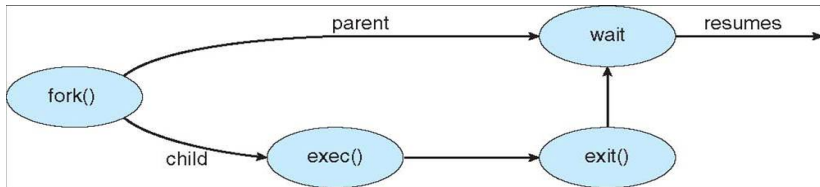
- El proceso indica al sistema operativo que ya puede liberar todos sus recursos (`exit()`).
- Además, indica su status de terminación (usualmente, un código numérico).
- Este código de status le es reportado al padre.
- ¿Qué padre?

(15) Árbol de procesos

- En realidad, todos los procesos están organizados jerárquicamente, como un árbol.
- Cuando el SO comienza, lanza un proceso que se suele llamar *root* o *init*.
- Por eso es importante la capacidad de lanzar un proceso hijo:
 - `fork()` es una llamada al sistema que crea un proceso exactamente igual al actual.
 - El resultado es el pid del proceso hijo, que es una copia exacta del padre.
 - El padre puede decidir suspenderse hasta que termine el hijo, llamando a `wait()`.
 - Cuando el hijo termina, el padre obtiene el código de status del hijo.
 - Por ende: `fork() + wait() = system()`
 - El proceso hijo puede hacer lo mismo que el padre, o algo distinto. En ese caso puede reemplazar su código binario por otro (`exec()`).

(16) Árbol de procesos (cont.)

- Cuando lanzamos un programa desde el shell, ¿qué sucede?
- El shell hace un `fork()`.
- El hijo hace un `exec()`.



(17) PCB en Linux

La PCB está definida en `task_struct`, en el archivo `sched.h`

`sched.h`

```
struct task_struct {  
    volatile long state;  
    void *stack;  
    ...  
    struct task_struct __rcu *parent;  
    ...  
    struct list_head children;  
    ...  
    struct thread_struct thread;  
    ...  
    struct files_struct *files;  
    ...  
}
```

(18) PCB en Linux

processor.h

```
struct thread_struct {  
    ...  
    unsigned long      sp;  
    unsigned long      usersp;  
    unsigned short     es;  
    unsigned short     ds;  
    unsigned short     fsindex;  
    unsigned short     gsindex;  
    ...  
    unsigned long      ip;  
    ...  
}
```