Resumen Procesos y API del SO

Tomás Felipe Melli

Noviembre 2024

Índice

1 Introducción **2**

2 Qué puede hacer un proceso ? **2**

3 En qué momento termina un proceso ? **2**

4 Cuánto tiempo cierto proceso hace uso de la CPU ? **2**

5 Qué es una SYSCALL **3**

6 POSIX : Portable Operating System Interface (X por UNIX) **4**

7 Qué estados tienen los procesos ? **4**

8 Cómo reciben notificación de la ocurrencia de algún evento los procesos ? **4**

1 Introducción

Un **Proceso** es un **programa en ejecución**. Este programa tiene asociado un identificador numérico llamado **PID (Process ID)**. Cada proceso se compone de un **área de texto** que es el código máquina del programa, el **área de datos** el Heap del proceso, y el **Stack** del proceso.

2 Qué puede hacer un proceso ?

- **Terminar**.
- **Ejecutarse** en la CPU.
- **Lanzar un proceso hijo** con system(), fork(), exec().
- **Dispositivos I/O**
- **Hacer un SYSCALL**

3 En qué momento termina un proceso ?

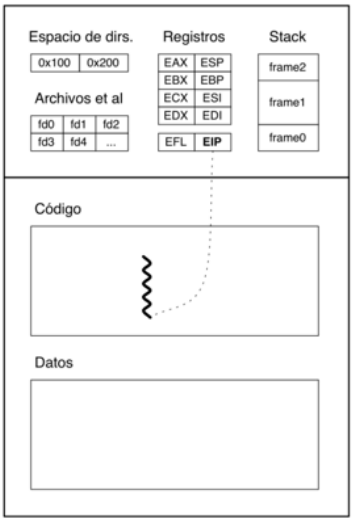
Los procesos le indican al SO meidante **EXIT()** que ya puede liberar los recursos asociados a su ejecución. Usualmente, comunican también el **estado de terminación**. Este es enviado al *PADRE*. Quién es el padre ? Debido a la forma en que se organizan los proceso, estos están jerárquicamente ordenados en forma de *árbol*. Al iniciar, el SO lanza el proceso **INIT** o **SYSTEMD**. Gracias a la syscall **FORK()**, un proceso puede crear un proceso idéntico a sí. Esta llamada al sistema devuelve un **PID (Process ID)** que es el número que identifica al proceso hijo. El padre puede hacer uso de **WAIT()** puede mantenerse en estado **SUSPENDED** de manera de esperar a que su proceso hijo termine su ejecución.

4 Cuánto tiempo cierto proceso hace uso de la CPU ?

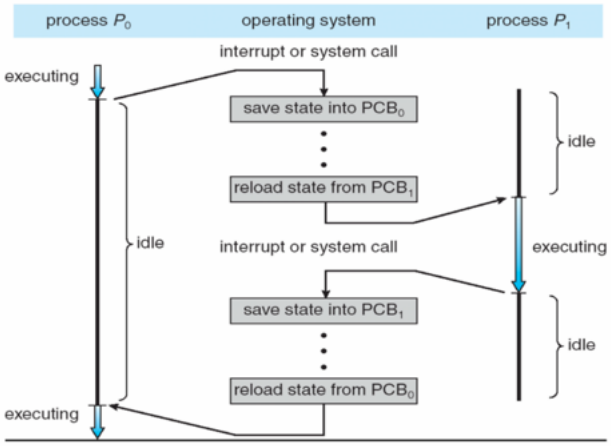
Dado que sólo un proceso a la vez puede estar en la CPU, no podemos dejarlo para siempre. Ya que no beneficia al sistema. Se define un **QUANTUM** como un intervalo de tiempo de CPU para que cierto proceso haga uso del proce. Actualmente, los sistemas operativos una vez que ese intervalo se supera, el proceso es *desalojado* del proce para darle un quantum a otro proceso. Estos sistemas se llaman **PREEMPTIVE** o por desalojo. Quién decide a quién le toca ? Existe un **módulo del Kernel** llamado **Scheduler** que es responsable de decidir a qué proceso le toca hacer uso del proce. Sigue sus reglas para decidirlo, llamada la **Política de Scheduling**, la cual es crucial para el buen rendimiento de un sistema operativo. Recordemos que para pasar de un proceso a otro, tenemos que hacer un **Context Switch** :

1. Guardar los registros
2. Guardar el IP
3. Si el programa es nuevo, cargarlo en memoria
4. Cargar los registros del nuevo programa
5. ...

Entre otras cosas, lo que quiero decir es, *toma tiempo* y por ello hay que definir un *quantum* sabiamente. Todo esto se guarda en una estructura llamada **PCB (Process Control Block)** :

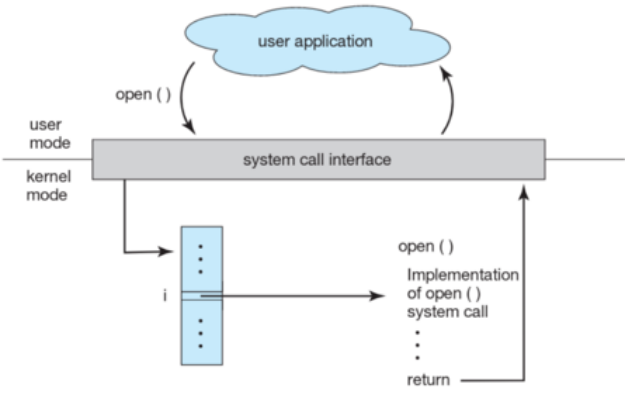


Es decir, el cambio se vería algo así:



5 Qué es una SYSCALL

Una **Syscall** o llamada al sistema es una acción que puede realizar un proceso en donde este puede hacer uso de los servicios de sistema operativo. Ya sea como vimos con el *fork()* como también con escrituras en pantalla con *write()*. El tema de la syscall es que necesitan un *context switch*, un *cambio de nivel de privilegio* y en algunas ocaciones, una *interrupción*... todo esto es **tiempo**. Las syscalls proveen una interfaz a los servicios que nos brinda el sistema operativo, la **API del sistema operativo**. La forma de implementarlas es como una *interrupción* para pasar a *modo privilegiado* y los parámetros, usando registros o una tabla. Existen las llamadas *Wrapper Functions* en C que son funciones que nos permiten interactuar con el SO con mayor facilidad. Supongamos que invocamos a **open()** :



6 POSIX : Portable Operating System Interface (X por UNIX)

POSIX es un **standard de diseño de programas**. Es decir, los programas escritos con esta regla podrán ser ejecutados en los SO que sigan este standard. Define desde funciones que permiten el manejo de archivos hasta comandos.

7 Qué estados tienen los procesos ?

Las tareas pueden encontrarse en una serie de estados :

- **Nuevo** es cuando todavía el sistema operativo no lo admitió. Será admitido cuando se agregue a la *tabla de procesos* que es una *lista de PCB's*.
- **Listo** es el estado en el que se encuentran los proceso cuando no tienen disponible la CPU, pero en cuanto el scheduler le asigne, arranca.
- **Corriendo** es que están haciendo uso de la CPU.
- **Bloqueado** ocurre cuando se queda esperando que algo suceda, como una espera por I/O.
- **Terminado** es el momento en que ya termino la ejecución completa del proceso.

8 Cómo reciben notificación de la ocurrencia de algún evento los procesos ?

Los procesos son capaces de recibir **Señales**. Esto es un mecanismo que incorporan los SO POSIX que les permite saber si algo sucedió. Al momento de recibirla, se interrumpe, y el **handler** es ejecutado. Por defecto, cada señal puede tener asociado un handler que puede ser notificado con la syscall **signal()**. Estas señales tienen un número asociado que identifica su tipo. Mediante la syscall **kill()** *un proceso le puede enciar a otro una señal.*