4. Procesos

En este capítulo veremos características relevantes de los procesos en general

1. Definición de proceso.

Como ya hemos dicho, un proceso es un programa en ejecución, incluyendo el valor del *program counter*, los registros y las variables. Cada proceso tendrá un identificador (process id ó pid) que lo identifica entre los demás.

2. Procesos en memoria.

Un proceso en memoria se constituye de varias secciones:

- Código (*text*): Instrucciones del proceso.
- Datos (*data*): Variables globales del proceso.
- Memoria dinámica (*Heap*)*:* Memoria dinámica que genera el proceso (aún no lo hemos visto).
- Pila (*Stack*): Utilizado para preservar el estado en la invocación anidada de procedimientos y funciones.

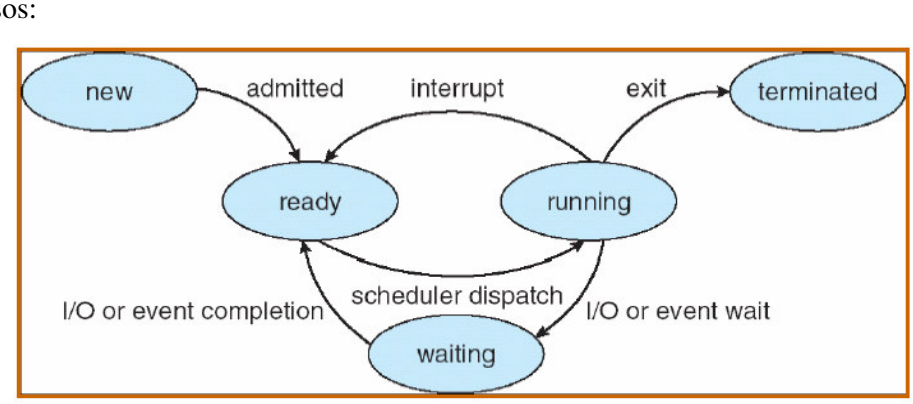
3. Estados de los procesos.

El estado de un proceso es definido por la actividad corriente en que se encuentra.
Los estados de un proceso son:

- **Nuevo** (*new*): Cuando el proceso es creado.
- **Ejecutando** (*running*): El proceso tiene asignado un procesador y está ejecutando sus instrucciones.
- **Bloqueado** (*waiting*): El proceso está esperando por un evento (que se complete un pedido de E/S o una señal).
- **Listo** (*ready*): El proceso está listo para ejecutar, solo necesita del recurso procesador.
- **Finalizado** (*terminated*): El proceso finalizó su ejecución.

4. Transiciones entre los estados.

Veamos ahora como los procesos pueden cambiar de estados a partir de determinados hechos. A continuación se muestra el diagrama de estados y transiciones de los procesos:



Nuevo –> Listo: Al crearse un proceso pasa inmediatamente al estado listo.

Listo –> Ejecutando: En el estado de listo, el proceso solo espera para que se le asigne un procesador para ejecutar. Al liberarse un procesador el planificador (*scheduler*) selecciona el próximo proceso, según algún criterio definido, a ejecutar.

Ejecutando –> Listo: Ante una interrupción que se genere, el proceso puede perder el recurso procesador y pasar al estado de listo. El planificador será el encargado de seleccionar el próximo proceso a ejecutar.

Ejecutando –> Bloqueado: A medida que el proceso ejecuta instrucciones realiza pedidos en distintos componentes (ej.: genera un pedido de E/S). El proceso es puesto en una cola de espera hasta que se complete su pedido. De esta forma, se logra utilizar en forma más eficiente el procesador.

Bloqueado –> Listo: Una vez que ocurre el evento que el proceso estaba esperando en la cola de espera, el proceso es puesto nuevamente en la cola de procesos listos.

Ejecutando -> Terminado: Cuando el proceso ejecuta su última instrucción pasa al estado terminado. El sistema libera las estructuras que representan al proceso.

5. Listas y colas de procesos.

Los procesos, según su estado, deberán esperar por determinados eventos, como ya vimos. Puede suceder, que más de un proceso esté esperando por el mismo evento, es por eso que se deben organizar en diferentes colas o listas. Veámoslas:

- **Lista de procesos del sistema (*job queue*):** Esta será una lista especial, porque los procesos que están en ella no esperan por nada en particular, sino que es la lista de todos los procesos del sistema. Al crearse un nuevo proceso se agrega el PCB a esta lista. Cuando el proceso termina su ejecución es borrado.
- **Cola de procesos listos (*ready queue*):** Esta cola se compondrá de los procesos que estén en estado listo. La estructura de esta cola dependerá de la estrategia de planificación utilizada.
- **Cola de espera de dispositivos (*device queue*):** Los procesos que esperan por un dispositivo de E/S particular son agrupados en una lista específica al dispositivo. Cada dispositivo de E/S tendrá su cola de espera, por lo que existirán varias *device queue*.

6. Bloque descriptor de proceso (*PCB*).

El proceso es representado, a nivel del sistema operativo, a través del *bloque descriptor de proceso* (***P**rocess **C**ontrol **B**lock*). En él, se guarda toda la información relevante del proceso, como estado, nombre, recursos asignados, identificador, padre del proceso (pid del proceso que lo creó), hijos del proceso (procesos que creó), entre otros datos.

7. Creación de procesos.

Como pudimos notar en el inciso anterior, los procesos de un sistema son creados a partir de otro proceso. Normalmente al creador se le denomina padre y al nuevo proceso hijo, lo cual genera una jerarquía de procesos en el sistema.
Una vez creado el nuevo proceso tendrán un hilo de ejecución propio, y el sistema generará un nuevo *PCB* para el proceso creado.

En UNIX podemos crear un nuevo proceso a través de la llamada al sistema fork.
La invocación a esta función le retorna al padre el número de *process id* del hijo recién creado y al hijo el valor 0. El hijo comienza su ejecución en el retorno del *fork*.

```
int main() {
        pid_t pid;
        pid = fork();
        if (pid < 0)      /* error occurred */
                fprintf(stderr, "Fork Failed");
        if (pid == 0)    /* child process */
                fprintf("soy el hijo!");
        else                /* parent process */
                fprintf("soy el padre!");
}
```

8. Cambio de contexto (*context switch*).

A la tarea de cambiar un proceso por otro en el procesador se le denomina cambio de contexto cambio de contexto. Esta tarea implica:

- Salvar el estado del proceso (registros, información de punteros de memoria) que está ejecutando en su PCB.
- Cambiar el estado del proceso que estaba ejecutando al que corresponda (Listo, bloqueado o terminado).
- Cargar el estado del proceso asignado a la CPU a partir de su PCB.
- Cambiar el estado del proceso nuevo a *ejecutando*.

9. Comunicación entre procesos

Procesos que se ejecutan concurrentemente pueden ser procesos independientes o cooperativos. Un proceso es cooperativo si puede afectar o verse afectado por los restantes procesos que se ejecuten en el sistema, y es independiente si no.
Evidentemente, cualquier proceso que comparta datos con otro será cooperativo.
Veamos algunas razones por las cuales es bueno tener un entorno que permita la cooperación de procesos:

- Compartir información. Dado que varios usuarios pueden estar interesados en la misma información, se debe proveer un acceso concurrente a ella.
- Acelerar cálculos. Si deseamos que una determinada operación se ejecute rápidamente, debemos dividirla en subtareas ejecutándose cada una de ellas en paralelo. Esto consigue solo si hay múltiples CPU o varios canales de E/S.

El mecanismo que provee esto es IPC (**I**nter**p**rocess **c**omunication), que permite intercambiar datos e información.