Proceso.

Un proceso es basicamente un programa en ejecución, cada proceso esta asociado con espacio de dirección (address space), la cuál es una lista de memoria que va desde 0 hasta un máximo, a su vez cada proceso puede leer y escribir. El espacio de dirección contiene el ejecutable del programa, los datos del programa y con una pila, de igual manera un proceso esta asociado con un conjunto de recursos, comunmente incluyendo registros, una lista de OPEN FILES, alarmas pendientes, lista de procesos relacionados. Desde otro punto un proceso es un contenedor que mantiene toda la información necesaria para correr un programa.

el modelo de procesos suele tener las siguientes caracteristicas.

- Todo el software ejecutable, incluso el sistema Operativo, se organiza en varios procesos secuenciales o procesos.
- Un proceso incluye al programa en ejecución y a los valores activos del contador, registros, variables del mismo.
- Conceptualmente cada proceso tiene su cpu virtual.
- Si la CPU alterna entre los procesos, la velocidad a la que ejecuta un proceso no será uniforme.
  - No deben programarse con hipótesis implícitas acerca del tiempo
  - La mayoria de procesos no son afectados por la multiprogramación de la CPU o velocidad de procesos distintos.

- Un proceso es una actividad de cierto ~~tiempo~~ tipo que tiene un programa, entrada, salida y estado.

- Un procesador puede ser compartido entre varios procesos con cierto "algoritmo de planificación", el cual determina cuando detener el trabajo en un proceso y dar servicio a otro distinto.

PCB

Bloque de control de proceso es la estructura de datos central y más importante de un sistema operativo. Cada bloque de control de proceso contiene toda la información de un proceso que necesita un sistema operativo para su control. Estos bloques son leídos y/o modificados por casi todos los módulos de un sistema operativo incluyendo aquellos que tienen que ver con la planificación, la asignación de recursos, el tratamiento de interrupciones y el análisis y supervisión del rendimiento. El conjunto de todos los PCB se guarda en una estructura del sistema operativo llamada tabla de procesos, la cual puede implementarse como un vector o lista enlazada.

La información que se guarda en el PCB usualmente es:

El Identificador de proceso (PID): número usado por el Kernel para identificar un proceso de manera única.

Información del estado del procesador: esto nos indica si esta listo para usarse, esta en espera o esta bloqueado.

Información del control del proceso:
Valores de registro de CPU
Contador de programa

Espacio de direcciones de memoria
Lista de recursos asignados
Estadísticas del proceso.
Datos del propietario.
Permisos asignados
Señales.

Estados de los procesos.

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

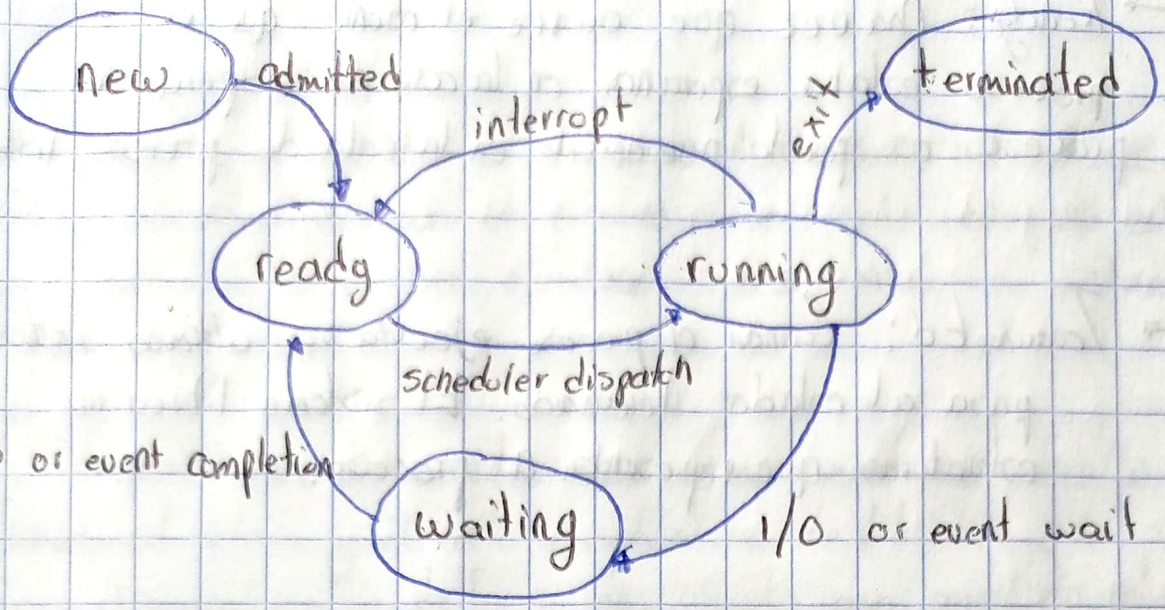
Los estados de un proceso son.

New: es cuando el proceso es creado.
Running: el proceso tiene asignado un procesador y está ejecutando sus instrucciones.

Waiting: aquí el proceso está esperando por un evento (que se complete un pedido de E/S o una señal).

Ready: El proceso esté listo para ejecutar, solo necesita del recurso procesador.

Terminated: El proceso finalizo su ejecución.



New: al crearse un proceso pasa inmediatamente al estado Ready.

Running: el proceso solo espera para que se le asigne un procesador ejecutado para ejecutar. Al liberarse un procesador el planificador (scheduler) selecciona el próximo proceso.

Running → Ready: ante una interrupción que se genere, el proceso puede perder el recurso procesador y pasar al estado Ready. El planificador será el encargado de seleccionar el próximo proceso a ejecutar.

Running → Blocked: a medida que el proceso ejecuta instrucciones realiza pedidos en distintos componentes. El pedido puede tardar y además si está en un sistema multiprogramado, el proceso es puesto en una cola de espera hasta que se complete su pedido.

- Blocked → Ready: 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.

Running → Terminated: Cuando el proceso ejecuta sus últimas instrucciones pasa al estado terminado. El sistema libera las estructuras que representan al proceso.

Fork()

Fork crea un nuevo proceso, exactamente igual al proceso original, indagando los descriptores de ficheros, registros, ambiente, etc.

Una vez ejecutado el FORK, padre e hijo se están ejecutando simultáneamente, en un principio las variables tienen los mismos valores, después, cambios en uno no afectan al otro.

La llamada Fork devuelve un valor después de ejecutarse, este valor es:

- Cero → en el proceso hijo.
- pid → (identificador del proceso hijo → en el padre.

Mediante este valor los procesos pueden saber cuál es el padre y cual es el hijo.

Para hacer un fork o utilizarlo es necesario llamar a la biblioteca. #include <unistd.h>

Exec

Es usado para ejecutar un comando desde bash hasta execc.
Este comando no crea un proceso nuevo, solo reemplaza el bash con el comando para ser ejecutado, si es exitoso no retorna la llamada al proceso

La llamada exec tiene variantes. Generalmente una de ellas es una llamada al sistema y el resto son funciones de biblioteca que permiten pasar los parámetros de forma más cómoda pero que internamente emplean la llamada al sistema.

ejemplo

execve(name, argv, envp)

name = nombre del fichero que contiene el programa a ejecutar.
argv = vector de argumentos
envp = vector ambiente.

Biblioteca a utilizar #include <unistd.h>

WAIT

Si un proceso ha creado mediante la llamada fork uno o varios hijos, la llamada wait suspende el proceso padre hasta que alguno de sus hijos termine su ejecución y devuelve dicho valor.

Durante el tiempo entre que un proceso muere y que su padre recoge ese valor de retorno el proceso hijo se considera un proceso zombi.

Las bibliotecas que utiliza son:

```
#include <sys/types.h>
#include <sys/wait.h>
```

wait (int * status); *status = código del estado de finalización del proceso hijo.

Espera a que un hijo termine y agarra su condición de salida, devuelve el pid del hijo que termina, llamada antigua

WAITPID.

Espera a que un hijo determinado termine, devolviendo el pid de proceso que termina y agarra su condición de salida.

Para esperar a que el hijo termine, el padre debe ejecutar waitpid.

Si el parametro pid vale -1, el padre espera por el primer hijo que termina

El segundo parámetro statloc es la dirección de una variable (estado de salida) que es escrita por el proceso hijo para indicar terminación normal o anormal.

waitpid (pid, &statloc, opts).

Bibliotecas `#include <sys/types.h>`
`#include <sys/wait.h>`

waitpid (pid_t, int *status, int option)

pid_t = Valor de pid
status = almacena la información de estado en la memoria apuntado.