Procesos

1.

Conceptos

Procesos
Atributos
Estados
Transición de estados

Un **proceso**

es una instancia de ejecución de un programa.

Es un concepto dinámico.

Conceptos

- **Programa**: secuencia de instrucciones en un lenguaje dado. Es un concepto estático.
- **Proceso**: instancia de ejecución de instrucciones de un programa.
- Un proceso tiene un **contexto** (contador de programa, estado, registros del procesador, pila y datos, etc.)
- En un sistema multiusuario, cada ejecución de un programa tendrá un contexto.
- El sistema operativo necesita el concepto de proceso para poder gestionar el procesador. Es la unidad de asignación de la CPU.

Ejecución de un programa

- Al ejecutar un programa sus instrucciones se copian en la memoria RAM para ser ejecutadas.
- Cuando termina, el programa es borrado de la memoria del sistema, dejándola libre.
- Cada uno de estos programas que pasan a la memoria del sistema y son ejecutados es lo que se conoce con el nombre de PROCESO.
- El tiempo que un proceso estuvo en la memoria del sistema ejecutándose, se conoce como TIEMPO DE EJECUCIÓN de un proceso.

Identificación de un proceso

- PID
- PPID
- UID y GID
- Duración y prioridad
- Directorio de trabajo activo
- Ficheros abiertos
- Tamaño de memoria asignada, fecha de inicio del proceso, terminal de ejecución
- Estado del proceso

Identificación de un proceso

Preparado (R)

- Proceso que está listo para ejecutarse. Simplemente está esperando a que el sistema operativo le asigne un tiempo de CPU.

Ejecutando (O)

- Sólo uno de los procesos preparados se está ejecutando en cada momento.

Suspendido (S)

- Un proceso se encuentra suspendido si no entra en el reparto de CPU, se encuentra esperando algún tipo de evento (ej., la recepción de una señal sw o hw). En cuanto dicho evento se produce, el proceso pasa a formar parte del conjunto de procesos preparados.

Parado (T)

- Un proceso en pausa tampoco entra en el reparto de CPU, pero no porque esté suspendido esperando un evento (ctrl + z). En este caso, sólo pasarán a estar preparados cuando reciba una señal determinada que les permita continuar.

Zombie (Z)

- Un proceso al finalizar avisa a su proceso padre, a fin que éste elimine su entrada de la tabla de procesos. Si el padre, por algún motivo, no recibe esta comunicación, no lo elimina de la tabla de procesos. Entonces, el proceso hijo queda en estado zombie, no está consumiendo CPU, pero sí continua consumiendo recursos del sistema.

Transición de estados

Tipos de procesos en Linux

- **Interactivos** (primer o segundo plano):
  - Un proceso en ***primer plano*** es aquél que podemos ver por pantalla en una ventana de terminal. Una consola concreta está bloqueada cuando un proceso está en primer plano.
  - Cuando un proceso se ejecuta en ***segundo plano*** la consola está libre y puede ser utilizada mientras se sigue ejecutando el proceso en segundo plano.
- **Demonios** (daemons): son procesos que se ejecutan en segundo plano pero no interactivos. Son programas que se encargan de gestionar y administrar el sistema, de forma transparente para el usuario

Comandos para visualizar procesos

- **ps**: muestra información sobre los procesos que se están ejecutando.
  - Ej. ps aux
- **top**: devuelve un listado dinámico de los procesos.
- **htop**
- **pstree**: visualiza en forma de árbol los procesos en ejecución.

Comandos para enviar señales a procesos

- **kill**
- El comando kill, que literalmente quiere decir matar, sirve no solo para matar o terminar procesos sino principalmente para enviar señales (signals) a los procesos. La señal por default (cuando no se indica ninguna es terminar o matar el proceso), y la sintaxis es kill PID, siendo PID el número de ID del proceso. Asi por ejemplo, es posible enviar una señal de STOP al proceso y se detendrá su ejecución, después cuando se quiera mandar una señal de CONTinuar y el proceso continuara desde donde se quedo.
- **kill -l**

Comandos para enviar señales a procesos

- **nice:** Permite cambiar la prioridad de un proceso. Por defecto, todos los procesos tienen una prioridad igual ante el CPU que es de 0. Con nice es posible iniciar un programa (proceso) con la prioridad modificada, más alta o más baja según se requiera. Las prioridades van de -20 (la más alta) a 19 la más baja. Solo root o el superusuario puede establecer prioridades negativas que son más altas. Con la opción -l de ps es posible observar la columna NI que muestra este valor.
- #**nice -n -5 comando**

Comandos para enviar señales a procesos

- **renice**: asi como nice establece la prioridad de un proceso cuando se incia su ejecución, renice permite alterarla en tiempo real, sin necesidad de detener el proceso.

```
# nice -n -5 yes     (dejar ejecutando 'yes' y en otra terminal se analiza con 'ps')
# ps -el
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0 12826 12208  4  75  -5 -   708 write_ pts/2    00:00:00 yes
# renice 7 12826
12826: prioridad antigua -5, nueva prioridad 7
# ps -el
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0 12826 12208  4  87   7 -   708 write_ pts/2    00:00:15 yes
```

Comandos para enviar señales a procesos

- **nohup y &**
- Cuando se trata ejecutar procesos en background (segundo plano) se utiliza el comando nohup o el operador & . Aunque realizan una función similar, no son lo mismo.
- Si se desea liberar la terminal de un programa que se espera durará un tiempo considerable ejecutándose, entonces se usa . Esto funciona mejor cuando el resultado del proceso no es necesario mandarlo a la salida estándar (stdin), como por ejemplo cuando se ejecuta un respaldo o se abre un programa Xwindow desde la consola o terminal. Para lograr esto basta con escribir el comando en cuestión y agregar al final el símbolo & (ampersand).

Comandos para enviar señales a procesos

$> yes > /dev/null &

$> tar czf respaldo /documentos/* > /dev/null/ &

$> konqueror &

(con estos ejemplos se ejecuta el comando y se libera la terminal regresando el prompt)

Sin embargo lo anterior produce que el padre del proceso PPID que se invocó sea el proceso de la terminal en si, por lo que si cerramos la terminal o salimos de la sesión también se terminaran los procesos hijos que dependan de la terminal, no muy conveniente si se desea que el proceso continué en ejecución.

Comandos para enviar señales a procesos

Para solucionar lo anterior, entonces se usa el comando nohup que permite al igual que '&' mandar el proceso y background y que este quede inmune a los hangups (de ahí su nombre nohup) que es cuando se cuelga o termina la terminal o consola de la cual se ejecutó el proceso.

$ nohup yes > /dev/null &

$ nohup czf respaldo /documentos/* > /dev/null/

$ nohup konqueror

Asi se evita que el proceso se "cuelgue" al cerrar la consola.

Comandos para enviar señales a procesos

- **Jobs :**Si se tiene acceso a una única consola, y se tienen que ejecutar varios comandos por largo tiempo en segundo plano, con el objeto de liberar la terminal y continuar trabajando, puede ser difícil de controlar. El comando jobs lista los procesos actuales en ejecución:

```
# yes > /dev/null &
[1] 26837
# ls -laR > archivos.txt &
[2] 26854
# jobs
[1]-  Running                 yes >/dev/null &
[2]+  Running                 ls --color=tty -laR / >archivos.txt &
```

Comandos para enviar señales a procesos

- Con los comandos fg (foreground) y bg background es posible manipular procesos suspendidos temporalmente, ya sea porque se les envió una señal de suspensión como STOP (20) o porque al estarse ejecutando se presionó ctrl-Z. Para reanudar su ejecución en primer plano usamos fg:

```
# jobs
[1]-  Stopped                 yes >/dev/null &
[2]+  Stopped                 ls --color=tty -laR / >archivos.txt &
# fg %1
# jobs
[1]+  Running                 yes >/dev/null &
[2]-  Stopped                 ls --color=tty -laR / >archivos.txt &
```

Comandos para enviar señales a procesos

- Obsérvese como al traer en primer plano al 'job' o proceso 1, este adquirió el símbolo [+] que indica que esta al frente. Lo mismo sería con bg que volvería a reinicar el proceso pero en segundo plano. Y también es posible matar los procesos con kill indicando el número que devuelve jobs: kill %1, terminaría con el proceso en jobs número 1.

Procesos en segundo plano

top [Ctrl+Z]

ping 8.8.8.8 > /dev/null &

jobs

tail f /var/log/dmesg > /dev/null &

jobs

ping 8.8.4.4 > /dev/zero [Ctrl+Z]

jobs

bg 4

jobs

fg 3 [Ctrl+Z]

jobs l

fg

ps

Ejecución de procesos

Terminador Secuencial

Lo define el símbolo ";“ . Permite ejecutar órdenes de forma secuencial en una sola línea de comandos.

```
$ date ; pwd ; mkdir ~/agenda ; echo “terminado”
```

Terminador Paralelo

Lo define el símbolo "&“. Ejecuta comandos en paralelo, es decir, al mismo tiempo.

```
$ uname -r & whoami & echo “LSO” &
```

Terminador de Afirmación

Lo define el símbolo "&&“ . Se ejecuta comando2 si y sólo si comando1 se cumple sin problemas.

```
$ mkdir ~/libros && echo “Directorio creado”
```

Terminador de Negación

Lo define el símbolo "||“. Se ejecuta comando2 si y sólo si comando1 falla por cualquier motivo

```
$ mkdir /opt/aplicacion || echo “Error. No se creó el directorio”
```

Gracias