Guía de Estudio: Administración de Procesos en Linux

Esta guía de estudio está basada en la clase sobre administración de procesos en sistemas operativos, con un enfoque principal en Linux. A continuación, se presentan los conceptos clave, ejemplos prácticos y comandos importantes para comprender y gestionar procesos en Linux.

1. Introducción a los Procesos

Un **proceso** es un programa en ejecución que tiene asignado un espacio de memoria, un identificador único (PID) y recursos del sistema operativo. Los procesos en Linux siguen una **jerarquía** donde cada proceso tiene un **proceso padre** que lo crea, y puede generar **procesos hijos**.

Conceptos Clave:

- **PCB (Process Control Block)**: Estructura que almacena la información del proceso (PID, estado, memoria asignada, etc.).
- **Jerarquía de procesos**: Cada proceso es hijo de otro (excepto el proceso inicial, como `systemd`).
- **Llamadas al sistema**:

- **fork()**: Crea una copia exacta del proceso actual (proceso hijo).
- **exec()**: Reemplaza el código del proceso actual por el de otro programa ejecutable.

Ejemplo Práctico:

- **Fork**: Un programa en C que usa `fork()` imprime un mensaje, crea un proceso hijo y ambos procesos continúan ejecutándose desde el punto del `fork`. Cada proceso tiene su propio PID.
- **Exec**: Un programa (`exec1`) ejecuta `exec()` para reemplazar su código por otro programa (`exec2`), manteniendo el mismo PID pero ejecutando un código diferente.

2. Creación de Procesos

Los procesos en Linux pueden crearse desde:

- **Interfaz de línea de comandos**: Ejecutando el nombre del archivo ejecutable en una terminal (ejemplo: `./programa`).
- **Interfaz gráfica**: Haciendo clic en un ícono o menú que inicia el programa.

Proceso Interno:

1. El sistema operativo asigna un **PCB** y memoria para el programa.

2. Se carga el código y los datos en la memoria.
3. Se establece la jerarquía (proceso padre e hijo).

Ejemplo:

```
./ejemplo_fork
```

Crea un proceso que imprime su PID y genera un proceso hijo con un PID diferente.

3. Obtención de Información de Procesos

Linux ofrece herramientas para monitorear y obtener estadísticas de procesos en ejecución.

Comandos Principales:

- **`ps`** : Muestra un listado de procesos.
  - `ps` : Muestra procesos de la terminal actual.
  - `ps aux` : Muestra todos los procesos del sistema con detalles (usuario, PID, %CPU, %MEM, comando).
  - `ps -ef` : Muestra la jerarquía de procesos.
- **`top`** : Proporciona una vista dinámica de los procesos, actualizada en tiempo real, ordenada por uso de CPU.
- **`htop`** : Versión mejorada de `top` con interfaz más amigable (requiere instalación: `sudo apt install htop`).

- **`pgrep`** : Busca el PID de un proceso por su nombre (ejemplo: `pgrep firefox`).

Filtrado de Procesos:

- Usar `ps aux | grep nombre_proceso` para filtrar procesos específicos.
- Ejemplo:

  ```
  ps aux | grep firefox
  ```

  Muestra los procesos relacionados con Firefox.

Ejemplo:

```
ps aux
```

Muestra todos los procesos con detalles como PID, usuario, uso de CPU y memoria.

```
htop
```

Muestra una interfaz gráfica con barras de uso de CPU/memoria y permite ordenar por diferentes criterios (CPU, memoria, usuario, etc.).

4. Finalización de Procesos

Los procesos pueden finalizar de manera **normal** o **forzada**.

Tipos de Finalización:

1. **Normal**: El proceso completa su ejecución y llama a `exit(0)` (éxito) o `exit(n)` (error, donde `n ≠ 0`).
2. **Excepción**: Errores como división por cero o acceso inválido a memoria.
3. **Forzada**: Otro proceso o usuario termina el proceso.

Comandos para Finalizar:

- **`kill <PID>`** : Envía una señal `TERM` para solicitar al proceso que finalice ordenadamente.
- **`kill -9 <PID>`** : Envía una señal `KILL` para forzar la terminación (no ordenada, puede dejar recursos sin liberar).
- **`killall <nombre>`** : Finaliza todos los procesos con el nombre especificado (ejemplo: `killall firefox`).

Ejemplo:

```
kill 4057
```

Solicita al proceso con PID 4057 que finalice.

```
kill -9 4057
```

Fuerza la terminación del proceso con PID 4057.

5. Prioridades de Procesos

Linux permite ajustar la prioridad de ejecución de los procesos mediante el valor **nice** (nivel de cortesía).

Detalles:

- **Rango de nice**: De `-20` (máxima prioridad) a `19` (mínima prioridad). Por defecto, los procesos inician con `0`.
- Solo usuarios con permisos de superusuario (`root`) pueden aumentar la prioridad (valores negativos).

Comandos:

- **`nice -n <valor> <comando>`**: Inicia un proceso con un valor nice específico.
  - Ejemplo: `nice -n 15 ./ejemplo_prioridad` (inicia con prioridad baja).
- **`renice <valor> <PID>`**: Cambia la prioridad de un proceso en ejecución.
  - Ejemplo: `renice 15 5062` (baja la prioridad del proceso con PID 5062).

Ejemplo:

```
nice -n 15 ./ejemplo_prioridad
```

Ejecuta el programa con prioridad baja (`nice 15`).

```
sudo renice -15 5062
```

Aumenta la prioridad del proceso con PID 5062 (requiere permisos de `root`).

6. Temporización de Procesos

Medir Tiempo de Ejecución:

- **`time <comando>`** : Mide el tiempo total, tiempo en modo usuario y tiempo en modo núcleo de un proceso.
  - Ejemplo:

    ```
    time ps
    ```

    Devuelve tiempos como: `real` (total), `user` (modo usuario), `sys` (modo núcleo).

Programar Ejecución:

- **`cron`** y **`crontab`** : Permiten programar tareas para ejecutarse a intervalos regulares.
  - Formato de `crontab` : `minuto hora día_mes mes día_semana comando` .
  - Ejemplo:

    ```
    0 12 * * * /ruta/script.sh
    ```

    Ejecuta `script.sh` todos los días a las 12:00.

7. Suspensión y Reanudación de Procesos

Los procesos pueden ejecutarse en **primer plano** (foreground) o **segundo plano** (background).

Comandos:

- **`comando &`** : Ejecuta un proceso en segundo plano.
  - Ejemplo: `gedit &` (inicia el editor en background).
- **`jobs`** : Lista los procesos en segundo plano.
- **`fg %<número_job>`** : Trae un proceso al primer plano.
- **`bg %<número_job>`** : Reanuda un proceso en segundo plano.
- **`Ctrl+Z`** : Suspende un proceso en ejecución (lo detiene y lo envía al background).

Ejemplo:

```
./ejemplo_ej &
```

Ejecuta `ejemplo_ej` en segundo plano.

```
jobs
```

Muestra el proceso en background con su número de job.

```
fg %1
```

Trae el proceso con job 1 al primer plano.

8. Jerarquía de Procesos

El comando **`pstree`** muestra la jerarquía de procesos en forma de árbol, destacando las relaciones padre-hijo.

Ejemplo:

```
pstree
```

Muestra que `systemd` es el proceso inicial, con procesos hijos como los de la interfaz gráfica o los iniciados por el usuario.

9. Actividades Prácticas

1. **Listar Procesos**:
   - Ejecuta `ps aux` y filtra procesos de un programa específico (ejemplo: `firefox`).
   - Usa `htop` para monitorear procesos en tiempo real.
2. **Finalizar Procesos**:
   - Identifica el PID de un proceso con `pgrep` y termínalo con `kill`.
3. **Cambiar Prioridades**:
   - Inicia un programa con `nice -n 10` y verifica su prioridad con `ps`.
   - Cambia la prioridad de un proceso en ejecución con `renice`.
4. **Programar Tareas**:
   - Crea una entrada en `crontab` para ejecutar un script cada día a las 14:00.
5. **Suspender y Reanudar**:
   - Ejecuta un programa en segundo plano, suspéndelo con `Ctrl+Z`, y reanúdalo con `bg` o `fg`.

10. Recursos Adicionales

- **Material de Windows**: Revisa el video adicional en el campus virtual para aprender sobre administración de procesos en Windows.
- **Documentación**:
  - Manual de comandos: `man ps`, `man top`, `man kill`, `man nice`, `man crontab`.
  - Tutoriales en línea sobre `htop` y `cron`.

11. Preguntas de Repaso

1. ¿Qué diferencia hay entre `fork()` y `exec()`?
2. ¿Cómo puedes listar todos los procesos de un usuario específico?
3. ¿Qué hace el comando `kill -9` y por qué debe usarse con cuidado?
4. ¿Cómo cambias la prioridad de un proceso en ejecución?
5. ¿Qué significa que un proceso esté en "background" y cómo lo traes al "foreground"?
6. ¿Cómo programas una tarea para ejecutarse cada hora en punto?

Esta guía cubre los aspectos fundamentales de la administración de procesos en Linux, con ejemplos prácticos y comandos esenciales.

Practica los ejemplos en una terminal para consolidar los conocimientos.