4. Prácticas de gestión de procesos

Entrega. Mostrar al profesor en el aula un fichero de texto plano con las respuestas a los apartados 1–4, y la instalación realizada en los apartados 5–6.

4.1. Uso de CPU (2.5 puntos)

Se proporcionan tres programas: `test1`, `test2` y `test3` para arquitecturas de 32 y 64 bits. Los siguientes ejercicios se pueden realizar en cualquier ordenador y se pide contestar a las siguientes preguntas en un fichero de texto plano.

1. Lanza un navegador, que es proceso que requiere de bastantes recursos, y examina su ejecución con el comando `top`. Pulsa `f` y selecciona para visualizar el núcleo en el que se están ejecutando los procesos. Observa la ejecución del navegador durante varios minutos. ¿En qué núcleo se ha ejecutado?

2. De forma similar, selecciona en `top` visualizar el número de hilos que tienen los procesos y obtén cuántos usa el navegador.

3. Por último, obtén los estados por lo que pasa la ejecución del navegador.

4. Determina el número de núcleos lógicos (incluyendo el hyperthreading) que tiene nuestro procesador leyendo el archivo `/proc/cpuinfo` o ejecutando los comandos `nproc` o `lscpu`.

5. Dale a los archivos `test*` adecuados a nuestra arquitectura permiso de ejecución (`chmod +x test*`). Crea una copia del programa `test1` y llámala `test1_2`. Lanza en background tantas instancias de `test1` como núcleos tiene nuestro ordenador y una vez `test1_2` en background.

6. Inspecciona su comportamiento con `top`. ¿`test1` es un programa de uso intensivo de CPU o de memoria? Justifícalo explicando los valores de los campos que muestra top para ese proceso.

7. Utiliza el comando `renice` para cambiar la prioridad de `test1_2` hasta que, en comparación con los procesos `test1`, deje de ralentizar significativamente el sistema. ¿Qué valores muestra ahora `top` para ambos procesos?

8. Como usuario, examina los límites del sistema con `ulimit -a`. Cambia el tiempo máximo de cpu a 10 segundos. Ejecuta en esa terminal una nueva instancia de `test1`. ¿Qué ocurre? ¿Por qué?

9. Por último, mata con `kill -KILL` *`pid`* o con `killall -KILL` *`test1`* todos los procesos `test1` y también `test1_2`.

   NOTA: `ulimit` solo tiene efecto en el shell actual, por lo que bastará con cerrar la terminal para restaurar el sistema.

4.2. Análisis de procesos (2.5 puntos)

1. Tras matar los procesos anteriores, ejecuta una instancia de `test1` en foreground (primer plano) y utiliza CNTL-Z y `bg` para pasarlo a background (segundo plano). Vuélvelo a pasar a foreground con `fg` y mátalo con CNTL-C. Comprueba con `top` que no ha quedado ninguna instancia en ejecución.

2. Ejecuta en backgroud una instancia de `test1` y otra de `test2`. Compara con `top` los recursos consumidos por ambos procesos. ¿En qué se diferencia `test2`?

3. Mata ambos procesos. Ejecuta de nuevo el proceso `test2` con `strace -c` durante un minuto, más o menos. A continuación, detenlo con CTRL-C e inspecciona la salida de `strace`.

   ¿Qué *syscall* tiene un mayor número de llamadas?

   ¿Qué significa? (la operación de cada *syscall* se puede consultar con el comando `man`).

   ¿Coincide con los datos observados con `top`?

4. Ejecuta de nuevo `strace` sobre `test2` sin la opción `-c`. Considera válidas las llamadas aunque den errores al reservar memoria (el programa se ha diseñado así para no ralentizar al ordenador).

   Examina las llamadas de reserva de memoria y una vez estabilizada la situación determina la memoria en bytes que se está solicitando reservar en cada llamada ¿cuántas páginas de memoria se reservan en cada llamada?

   Nota: para conocer el tamaño de la página utiliza `getconf PAGESIZE`.

4.3. Funciones (2.5 puntos)

1. Como usuario, ejecuta `strace -c ./test3` durante unos 20 segundos. Detenlo con CTRL-C y observa la salida de *strace*. ¿Qué `syscall` tiene un mayor número de llamadas? ¿Qué significa?

2. Ejecuta de nuevo `test3` sin `strace`, pero en foreground. Inspecciona el directorio de `/proc` para ese proceso y, en función de la información que nos proporcionan los descriptores de fichero en este directorio, averigua los ficheros o seudoficheros a los que está accediendo el programa (se necesitará ver hacia a dónde apuntan los enlaces simbólicos).

3. Intenta detener `test3` con CTRL-C. Luego inténtalo con `kill -KILL pid`. ¿Qué señales se están enviando en cada caso y por qué este comportamiento? (Pista: el código fuente contiene una línea `signal(SIGINT,SIG_IGN)`).

4.4. Automatización de tareas (1.5 puntos)

Según lo explicado en el apartado 3.5 de los apuntes (automatización de tareas), realiza los siguientes ejercicios en la máquina cliente:

1. Como un usuario (no root) programa una tarea para imprimir el texto `"Hola Mundo"` en un minuto a partir de este momento. Durante la espera visualiza las tareas pendientes. ¿Cómo se recibe la salida de la tarea?

2. Como administrador crea un script (con permiso de ejecución) en el directorio `/usr/local/bin` con el siguiente contenido:

   ```
   #!/bin/bash
   echo "* Control de usuarios $(date)" >> /var/log/trabajan.log
   w >> /var/log/trabajan.log
   ```

   y programa una tarea de `cron` para que el script se ejecute periódicamente cada 5 minutos (no olvides poner un retorno de línea al final de la sentencia). Durante la espera lista las tareas pendientes. Pasado un tiempo lee el contenido del fichero `/var/log/trabajan.log`.

4.5. Backups (1 punto)

Según lo explicado en el apartado 3.6 de los apuntes (copias de seguridad), realiza los siguientes ejercicios en la máquina cliente:

1. Instala en el cliente el paquete `dump`.

2. Dentro del *home* de un usuario (no root) crea un directorio y haz un backup de nivel 0 de `/home` usando como dispositivo el fichero */root/cinta0.dump*.

3. Crea un fichero dentro del directorio anterior y haz un backup de nivel 5 de `/home` usando como dispositivo el fichero */root/cinta5.dump*.

4. Borra el directorio creado y su contenido.

5. Ve a `/root` y restaura los elementos borrados usando primero la cinta de nivel 0 y luego la de nivel 5. ¿A dónde han ido a parar los elementos restaurados?