4 Planificación

4.1 Trabajos prácticos

4.1.1 Práctica con Linux

Como decíamos anteriormente POSIX proporciona planificación basada en prioridades. Estas prioridades pueden ser modificadas dinámicamente mediante servicios específicos.

La planificación en Linux se lleva a cabo en la función "`schedule`" del archivo `kernel/sched.c`.

Linux implementa el esquema de planificación definido en las extensiones de tiempo real de POSIX. Existen tres formas de planificar un proceso:

- SCHED_FIFO: Un proceso con este tipo de planificación tiene asociada una prioridad estática. Son más prioritarios que los procesos «normales». Una vez que han conseguido el procesador, sólo pueden ser expulsados por otro proceso más prioritario. Esto significa que no tienen un *quantum* máximo de ejecución.
- SCHED_RR: Igual que SCHED_FIFO, pero con una política de Round Robin para los procesos de igual prioridad.
- SCHED_OTHER: Es la política de planificación «clásica» de UNIX: un Round Robin con mayor *quantum* a los procesos que menos CPU han consumido recientemente.
- Existe un bit adicional, SCHED_YIELD, que afecta a las tres políticas de planificación, haciendo que un proceso ceda la CPU a cualquier otro que se encuentre preparado, únicamente para el próximo ciclo de planificación, no indefinidamente.

Podemos analizar paso a paso el código de "`__schedule()`", que presenta un aspecto poco ortodoxo tratándose de una función en C, ya que contiene numerosos saltos y etiquetas. La idea es que la línea de ejecución más frecuente no conlleve ni un solo salto. Véalo con: [1]

```
ubuntu@ubuntu:~$ less /usr/src/linux/kernel/sched.c
```

Observe que la mayoría de las funciones definen -mayormente- estructuras de datos; las que comienzan con declaraciones del tipo «struct».

En la función «`__schedule()`» la línea de ejecución más frecuente utiliza una bifurcación `goto`, a través de una etiqueta.

[1] Es probable que en vez de `linux` exista un directorio que incluya el número de versión del núcleo, por ejemplo `linux-2.4.20`, por eso cuando haya escrito la palabra `linux` presione la tecla de tabulador una o dos veces, esto invoca la función de autocompletar del intérprete `bash`.

4.1.1.1 Cambio de contexto

Linux implementa el «sistema de archivos proc» que es como una especie de «mapa» de estructuras de datos en memoria, expresados como si fueran archivos.

Podemos con el comando:

```
ubuntu@ubuntu:~$ cat /proc/stat
```

Ver el número de cambios de contexto en la línea `cntxt`, note que con la flecha de cursor hacia arriba se recuperan los comandos escritos, por lo tanto si recupera este comando y lo ejecuta repetidas veces notará cuantos cambios de contexto se efectúan en pocos segundos, aún cuando supuestamente el sistema «no está haciendo nada».

4.1.1.2 Desalojo

Un proceso puede ceder «voluntariamente» la CPU ya sea porque ha completado su tarea o porque está esperando la llegada de un evento, tal como datos provenientes del disco, que se presione una tecla o un paquete de la red.

Un proceso puede ceder «involuntariamente» la CPU. A esto se lo denomina «desalojo» y ocurre cuando un proceso de mayor prioridad quiere usar la CPU. El desalojo puede tener un impacto particularmente negativo en el rendimiento y un desalojo constante puede llevar el sistema a un estado conocido como «hiperpaginación» (*thrashing* en inglés, literalmente significa «fustigamiento»). Este problema ocurre cuando los procesos son desalojados constantemente y ninguno nunca alcanza a ejecutar completamente.

Para comprobar la ocurrencia de desalojo voluntario e involuntario en un proceso, revise el contenido del archivo «`/proc/PID/status`», en el que «`PID`» es el PID del proceso a revisar.

El comando «`grep`» busca una palabra (una cadena de caracteres) dentro de un archivo. Entonces, si combinamos lo visto hasta el momento, el comando:

```
ubuntu@ubuntu:~$ grep voluntary /proc/self/status
voluntary_ctxt_switches:     4
nonvoluntary_ctxt_switches:  0
```

Busca dentro de ese archivo la palabra «voluntary» correspondiente al estado del propio proceso «`grep`». **Experimente** con otros procesos.

Comando top El comando «`top`»[2] provee una vista continua de la actividad del procesador. Muestra -y actualiza periódicamente- un listado de los procesos (tareas o *tasks*, según los términos de Linux) mas intensivos en CPU en el sistema, y puede proveer una interfaz interactiva para la manipulación de procesos. Las columnas son:

- PID: el Identificador de Proceso de cada tarea.

[2] También puede usar los programas «htop» o «atop», si los tiene instalados

- USER: el nombre de usuario del propietario del proceso.
- PR: la prioridad del proceso.
- NI: El valor «`nice`» (mejorado) del proceso. Los valores negativos tienen prioridad más alta.
- VIRT (*Virtual Memory Size*): El tamaño total de la memoria virtual utilizada por la tarea medido en KiB[3]. Incluye todo el código, datos y bibliotecas compartidas, páginas que han sido intercambiadas y páginas que han sido mapeadas pero no usadas.
- RES (*Resident Memory Size*): Un subconjunto del espacio de direcciones virtuales (VIRT) que representa la memoria física no intercambiada que una tarea está utilizando actualmente.
- SHR (*Shared Memory Size*): Un subconjunto de memoria residente (RES) que puede ser utilizado por otros procesos. Incluirá páginas anónimas compartidas y páginas respaldadas por archivos compartidos. También incluye páginas privadas asignadas a archivos que representan imágenes de programas y bibliotecas compartidas.
- SIZE: El tamaño del código del proceso mas los datos mas el espacio de pila, en kilobytes.
- RSS: Cantidad total de memoria física usada por el proceso, en kilobytes.
- SHARE: La cantidad de memoria compartida usada por el proceso.
- STAT o S: El estado del proceso:
  - S (*Interruptible Sleep*) sueño interrumpible o durmiendo: el proceso se encuentra esperando a que se cumpla algún evento, por ejemplo, que el planificador de procesos del núcleo lo planifique para su ejecución.
  - D (*Uninterruptible sleep*) sueño ininterrumpible: el proceso se encuentra generalmente esperando una operación de entrada o salida con algún dispositivo.
  - R (*running*) corriendo: el proceso se encuentra corriendo en el procesador.
  - Z zombie: proceso terminado, pero cuyo padre aún sigue «vivo»[4] y no ha capturado el estado de terminación del proceso hijo y, por consiguiente, no lo ha eliminado de la tabla de procesos del sistema.
  - T (*traced*) trazado o detenido: un proceso que ha sido detenido mediante el envío de alguna señal generalmente. «<» procesos con valor «nice» negativo, N para procesos con valores «nice» positivos, W para procesos en área de *swap*.
- %CPU: La porción del proceso del tiempo de CPU desde la última actualización de la pantalla, expresada como porcentaje del tiempo total de CPU por procesador.
- %MEM: La porción del proceso de la memoria física.
- TIME: Tiempo total de CPU que ha utilizado la tarea desde que se inició.
- COMMAND: El nombre del comando que generó el proceso.

[3] kibibyte. Equivale a 2^{10} bytes.

[4] O bien el padre ha muerto

El comando nice El comando «nice» corre un programa con la prioridad de planificación modificada. Los rangos van desde -20 (la prioridad más alta) hasta 19 (la más baja).

El comando kill El comando «kill» envía una señal a un proceso, generalmente la de «matar» o terminar un proceso, pero no necesariamente.

El comando yes El comando «yes» imprime por pantalla una serie de «y»es en forma interminable. Para finalizar (abortar) el comando/proceso debe presionar simultáneamente CTRL–C, esto envía una «señal» desde el teclado que le indica al comando/proceso que debe finalizar.

El archivo/dispositivo «/dev/null» Podemos redirigir la salida de este comando a un archivo utilizando a la derecha del comando el carácter «>» (mayor). Existe un archivo/dispositivo especial que funciona como un «agujero negro» o una papelera de reciclaje, todo lo que va a parar ahí se pierde inmediatamente; este archivo/dispositivo especial es «/dev/null».

Foreground y background Todo comando/proceso hasta tanto no finalice, no nos entrega el «*prompt*» del intérprete de comandos, si está ejecutando desde una consola/terminal. A esto el UNIX lo denomina «*foreground*». Y en contraposición existe el «*background*», lo que está ejecutando «por detrás» o «tras bambalinas» en forma no asociada a una consola/terminal. Esto permite que un proceso disparado desde una consola corra permanentemente, desafectando a la consola/terminal para que el usuario pueda correr otros comandos/procesos.

Para enviar un comando al «*background*» debe colocar al final de la línea el carácter «&».

Ejemplo de planificación Abra una sesión como usuario «ubuntu» y ejecute el comando «top».

Abra otra sesión en otra consola/terminal y ejecute el comando:

```
ubuntu@ubuntu:~$ yes > /dev/null &
```

Verá a continuación una respuesta del sistema como ésta:

```
[1] 829
```

El "829" es el número PID otorgado al proceso en ejecución, el «[1]» indica el número de tarea otorgado por el intérprete de comandos bash a la tarea que está ejecutando en *background*. El comando ha quedado ejecutando enviando las «y»es a «/dev/null» en donde se pierden sin remedio. A pesar de que el proceso aún no finaliza nos ha devuelto el *prompt* para seguir colocando comandos (es decir, quedó corriendo en *background*).

Vuelva a la sesión del usuario «ubuntu» (donde estaba top) y verá corriendo el proceso «yes» más o menos entre los primeros lugares.

La columna PRI, que hace referencia a prioridad, por defecto es 20. Ese 20 hace referencia a una prioridad media por defecto. Las prioridades de procesos de usuario se numeran desde 0 a 39. Mientras menor es el número, más alta será la prioridad, por lo que una prioridad cercana a 0 indicará mayor prioridad, y el proceso que la posea tendrá más oportunidades de ser planificado en el procesador, que uno que tenga una prioridad más baja, cercana al 39.

La columna NI(ce) hace referencia a una **prioridad relativa** que por lo general mapea con la prioridad absoluta PRI, y que depende del estado y carga de procesamiento, pero que va desde -20 a +19, siendo -20 la mayor prioridad de proceso de usuario (equivalente a un PRI de 0) y +19 es la menor prioridad de usuario (equivalente a un PRI de 39), siendo el NI(ce) de 0 la prioridad media, equivalente aun PRI de 20.

Observe que el orden en la lista de procesos NO es permanente ¿Puede explicar por qué?

Vuelva a la sesión anterior y coloque ahora el comando

```
ubuntu@ubuntu:~$ nice -+10 yes > /dev/null &
```

Es decir, el mismo comando que recién pero ahora «mejorado» incrementando la prioridad. El sistema le contestará por ejemplo:

```
[2] 830
```

El significado es similar al primer caso. Vuelva a la sesión del usuario «ubuntu» y observe ahora el comportamiento de «top».

Podemos cambiar la prioridad de un proceso (que ya está ejecutando, valga la redundancia) mediante el comando «renice». Supongamos que quisiéramos alterar en 1 punto, utilizando el PID del proceso, de esta forma:

```
ubuntu@ubuntu:~$ renice +1 830
```

En algunos casos es necesario trabajar como superusuario «root» para alterar las prioridades.

Observe y compare los valores de las columnas PID, PRI, NI, %CPU, STAT.

Observe los cambios de contexto y los desalojos, según vimos anteriormente:

```
ubuntu@ubuntu:~$ grep voluntary /proc/829/status
ubuntu@ubuntu:~$ grep voluntary /proc/830/status
```

Obviamente en su sistema los números PID serán otros.

¿Qué conclusiones saca? Tome notas de todo lo observado

Vuelva ahora a la sesión del usuario «ubuntu» y coloque el comando:

```
ubuntu@ubuntu:~$ kill %2
```

Apretando la tecla «enter» dos veces, el sistema le comunicará que el proceso ha sido terminado. Puede recuperar los anteriores comandos con las flechas de dirección hacia arriba (o hacia abajo) y modificar su contenido con las flechas de dirección a izquierda y derecha.

También puede matar una tarea dentro del «top» si toca la tecla «k». El programa primero le preguntará el PID del proceso que quiere finalizar que por defecto es el que está en la primera línea, presionando «Enter» acepta la sugerencia. Luego preguntará el tipo de señal a enviar (la número 15 indica finalizar), presionando «Enter» acepta la sugerencia.

Experimente colocando distintos valores de «`nice`», en vez de «-+10» pruebe con «15» por ejemplo.

Observe y compare los resultados.

4.1.1.3 Una analogía

Podemos hacer una analogía si nos imaginamos lo que ocurre en un Banco frente a la ventanilla de una caja. El cajero es el procesador, los clientes son los procesos que llegan para ser atendidos. Algunos clientes traen pocas facturas para pagar y rápidamente son despachados, otros traen muchas y tardan mucho más. El Banco podría tener la política de atender al cliente hasta que haya terminado por más que se demore mucho, o bien podría tener la política de no aceptar más de cierta cantidad de boletas o cierto tiempo de atención, en cuyo caso el cliente es «expulsado» para dar paso al siguiente. Este cliente puede volver a ponerse en la cola y repetir el proceso hasta que haya hecho todos sus trámites. Para esto puede haber un policía encargado de «desalojar» al cliente de la ventanilla. El Banco también puede tener la política de darle prioridad a los ancianos y a las embarazadas, de manera que cuando llegan estos «procesos» a la cola se los atiende primero. Puede haber una sola cola para varios cajeros o una cola para cada ventanilla.