2 Procesos

2.1. Trabajos prácticos

2.1.1. Práctica con Linux

2.1.1.1. Estados de los procesos

El comando ps Volvamos a abrir una ventana de Terminal, tal como vimos en la Subsección 1.1.1.

UNIX provee el comando «ps» (*process status*) para ver el estado de los procesos. Si coloca este comando en Linux verá básicamente dos procesos: el del intérprete de comandos `bash` y el del propio proceso `ps`. Fíjese en las columnas: la primera indica el número identificador del proceso (PID, de *process identification*). Con la flecha de cursor puede recuperar este comando para colocarlo de nuevo; repita esta operación varias veces, notará que el PID del «`bash`» no cambia, pero sí lo hace el de «`ps`». Esto es así porque «`bash`» es un proceso que sigue ejecutando, en cambio «`ps`» cumple todo su ciclo de vida y cada invocación que hace es, en realidad, un nuevo proceso y, por lo tanto, el sistema operativo le asigna un número nuevo.

Cada vez que ejecuta «`ps`» el intérprete de comandos genera un hijo con las llamadas al sistema `fork()` y luego `exec()`[1], las cuales utilizaremos en un programa en lenguaje C. El comando «`ps`» sin opciones nos muestra lo que ya hemos visto. Si queremos ver todos los procesos que se están ejecutando en el sistema coloquemos el comando

```
ubuntu@ubuntu:~$ ps aux
```

La opción «`a`» significa «*all*», es decir, todos los procesos; la opción «`u`» indica «formato de usuario» y la opción «`x`» es para que muestre también a los procesos no asociados a terminal.

Si quiere ver parcialmente el árbol de procesos debería utilizar el comando

```
ubuntu@ubuntu:~$ ps fax
```

La opción «`f`» -proviene de «*forest*» o «árbol»- genera una salida en forma de árbol de procesos. También existe el comando `pstree`, que nos muestra un árbol de procesos más completo y más claro a partir del proceso `init` (o `systemd`).

Si en vez de ver todos los procesos que están ejecutando en nuestro sistema, queremos ver los procesos ejecutados por (o que pertenecen a) un determinado usuario, debemos usar la opción «`-u`» seguida del nombre de usuario:

[1] o execv().

```
ubuntu@ubuntu:~$ ps -fu ubuntu
```

Con la opción «-T» de este comando podemos ver a los hilos o *threads* de un proceso, cuyo número identificador (PID) indicaremos con la opción «-p»:

```
ubuntu@ubuntu:~$ ps -T -p 1234
```

Si quiere aprender más acerca de las opciones del comando «ps», consulte las páginas del manual con el comando

```
ubuntu@ubuntu:~$ man ps
```

El comando top Podemos utilizar además el comando «top», que nos muestra en forma decreciente los procesos que más recursos consumen, y va actualizando la lista periódicamente. Para salir de este programa presione la tecla «q».

Observe que el orden se va alterando. Es decir, que de la cola de procesos listos para ejecutar, el planificador los va eligiendo por su prioridad, pero ésta no es constante en Linux, sino que se ve alterada de acuerdo con un sistema de «créditos», que veremos más adelante.

Con «top» también podemos ver los hilos si invocamos el comando con la opción «-H».

Otro visor de procesos interactivo aún más vistoso es «htop». Este programa permite ver la actividad de un hilo.

Urgando en /proc Todas estos comandos o aplicaciones obtienen su información leyendo el directorio «/proc». Como habíamos adelantado en el apartado Subsubsección 1.1.1.10, el sistema de pseudo archivos[2] proc tiene su origen en el trabajo de Killian (1984) «*Processes as Files*» y su objetivo era el de mostrar información acerca de los procesos en ejecución. En Linux, actúa también como una interfaz a estructuras internas de datos en el núcleo, se puede usar tanto para obtener información del sistema como para cambiar ciertos parámetros del núcleo durante su ejecución. Contiene, entre otras cosas, un directorio para cada proceso en ejecución cuyo nombre es el número identificador del proceso (PID) -como se observa en la Figura 2.1- , un enlace llamado «*self*» que apunta al proceso que está accediendo al sistema de archivos proc en ese instante.

Podemos probar cómo va cambiando si ejecutamos repetidamente el comando «ls» sobre ese enlace:

```
ubuntu@ubuntu:~$ ls -l /proc/self
lrwxrwxrwx 1 root root 0 ago 12 18:16 /proc/self -> 4856
ubuntu@ubuntu:~$ ls -l /proc/self
lrwxrwxrwx 1 root root 0 ago 12 18:16 /proc/self -> 4857
ubuntu@ubuntu:~$
```

La opción «-l» nos muestra la salida en formato «largo», es decir, información completa del archivo mostrado. Vemos cómo en la primera ejecución apunta al directorio «4856» y en la siguiente al «4857». Por tratarse de un directorio podemos ver lo que está dentro de él.

[2] Se prefiere esta expresión a la de «archivos virtuales» para no confundirla con VFS

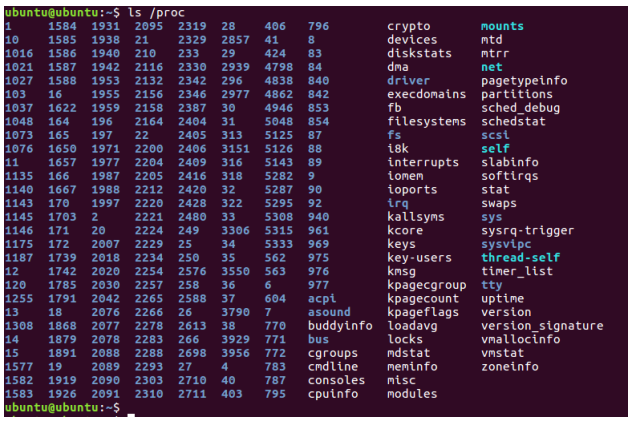

Figura 2.1: Listando /proc

Ver los cambios de contexto Dentro de los directorios de cada proceso tenemos al archivo «`status`» que, entre otras cosas, registra los cambios de contexto. A continuación utilizaremos al comando «`grep`» -que busca una cadena de caracteres dentro de un archivo- para ver los cambios de contexto que se produjeron durante la ejecución del propio «`grep`», de esta manera:

```
ubuntu@ubuntu:~$ grep ctxt /proc/self/status
voluntary_ctxt_switches:    0
nonvoluntary_ctxt_switches: 2
ubuntu@ubuntu:~$
```

En el sistema «`proc`» se muestra el uso que hace de la memoria cada proceso y sus estados. De manera similar también muestra información acerca del núcleo en ejecución, asignación de la memoria, información del procesador, sistemas de archivos, uso de los puertos de entrada y salida y de las interrupciones, controladoras de los dispositivos periféricos conectados, información de los dispositivos de red, así como también del tráfico en la misma y los distintos parámetros de configuración.

Una parte muy importante de `/proc` es el directorio `/proc/sys` el cual no sólo ofrece información muy valiosa sino también que es modificable. Al cambiar los valores contenidos dentro de los pseudo archivos se cambian los parámetros de operación del núcleo de forma inmediata, mientras permanece ejecutándose.

2.1.1.2. Obtener el PID

Ahora veremos a los principales servicios que ofrece POSIX y Win32 para la gestión de procesos, procesos ligeros y planificación.

Para ello, obtendremos el identificador del proceso mediante la llamada (*system call*) al sistema, cuyo prototipo en lenguaje C es:

```
pid_t getpid(void);
```

Desde la misma CLI vamos a ejecutar un comando que nos abrirá una ventana de edición de textos muy intuitiva. Supongamos que vamos a crear el archivo nuevo `obtenerpid.c`:

```
ubuntu@ubuntu:~$ gedit obtenerpid.c
```

en el que cargaremos el siguiente programa:

Listado de código 2.1: Programa «obtenerpid.c»

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main()
{
 pid_t id_proceso;
 pid_t id_padre;
 id_proceso=getpid();
 id_padre=getppid();
 printf("Identificador de proceso: %d\n", id_proceso);
 printf("Identificador del proceso padre: %d\n", id_padre);
 return 0;
}
```

Una vez hecho esto, guardamos y cerramos la ventana de edición y volvemos al *prompt*. Este programa se compila simplemente así:

```
ubuntu@ubuntu:~$ gcc -o obtenerpid obtenerpid.c
```

suponiendo que así ha nombrado al programa. Y se ejecuta así:

```
ubuntu@ubuntu:~$ ./obtenerpid
Identificador de proceso: 7014
Identificador del proceso padre: 3814
ubuntu@ubuntu:~$
```

es decir, «punto» «barra» «`obtenerpid`», sin espacios. **Observe** que el identificador del proceso padre (PPID o *Parent PID*) -el 3814 en este ejemplo- es el del intérprete `bash`.

Ejecútelo con «`strace`» y observe qué pasa con los números de proceso.

Las rutas o «path» Si usted alguna vez ejecutó programas en el intérprete de DOS, recordará que simplemente había que colocar el nombre del programa. Los intérpretes de comandos tienen una ruta (*path*) de búsqueda para encontrar los ejecutables. En el caso del DOS, esta ruta incluye el directorio actual (donde está usted parado ahora), pero en el caso de UNIX (Linux en este caso) no se incluye. Por este motivo, si usted crea ejecutables fuera de los directorios asignados para ejecutables (`/bin`, `/sbin`, `/usr/bin`, `/usr/sbin`, `/usr/local/bin`,

`/usr/local/sbin`), el intérprete no podrá encontrarlos y le devolverá el error de «comando desconocido». Como ahora usted ha creado un ejecutable en un directorio fuera del *path* de búsqueda de ejecutables (concretamente en `/home/ubuntu`), debe indicarle al intérprete dónde se encuentra el ejecutable, lo que puede hacer de dos maneras:

- **Absoluta**: para indicar la ruta a un archivo (ejecutable en este caso), se especifica su posición desde la «raíz» del sistema de archivos, pasando por los directorios hasta llegar al archivo. Por ejemplo «`/home/ubuntu/obtenerpid`».
- **Relativa**: la posición del archivo a partir del directorio en el cual estamos posicionados actualmente. El punto «.» significa «el directorio actual, donde estoy posicionado actualmente», es decir «`/home/ubuntu`» y, a partir de ahí, continúa con «`/ejemplo`».

2.1.1.3. Crear procesos con fork()

Éste es un ejemplo en POSIX de utilización de servicio para la creación de procesos.

Crearemos con `gedit` y luego ejecutaremos el siguiente programa, que podríamos llamar

`creaproceso.c`

Listado de código 2.2: Programa «creaproceso.c»

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main()
{
 pid_t pid;
 pid=fork();

 switch(pid){
  case -1: /* error en el fork() */
   perror("fork");
   break;
  case 0: /* proceso hijo */
   printf("Proceso %d; padre = %d\n", getpid(), getppid() );
   break;
  default: /* proceso padre */
   printf("Proceso %d; padre = %d\n", getpid(), getppid() );
 }
 return 0;
}
```

De la misma manera, guardamos y cerramos la ventana de edición y volvemos al *prompt*. Este programa se compila con:

```
ubuntu@ubuntu:~$ gcc -o creaproceso creaproceso.c
```

Y se ejecuta así:

```
ubuntu@ubuntu:~$ ./creaproceso
Proceso 7094; padre = 3814
Proceso 7095; padre = 7094
ubuntu@ubuntu:~$
```

De nuevo, el proceso número 3814 es el del `bash`. De ese se desprende «`creaproceso`» con número 7094 y, de éste, el 7095.

Observe que el proceso hijo es una copia del proceso padre en el instante en que éste solicita el servicio `fork()`. Esto significa que los datos y la pila del proceso hijo son los que tiene el padre en ese instante de ejecución.

Experimente qué pasa si ejecuta estos programas con «`strace`».

Piense y responda: ¿El proceso hijo empieza la ejecución del código en su punto de inicio, o en la sentencia que está después del `fork()`?

Observe que el hijo no es totalmente idéntico al padre, algunos de los valores del BCP han de ser distintos.

Piense y responda: ¿Cuáles deberían ser las diferencias más importantes?

	V	F
El proceso hijo tiene su propio identificador de proceso, distinto al del padre		
El proceso hijo tiene una nueva descripción de la memoria.		
El tiempo de ejecución del proceso hijo se iguala a cero		
Todas las alarmas pendientes se desactivan en el proceso hijo		
El conjunto de señales pendientes se pone a vacío		
El valor que retorna el sistema operativo como resultado del `fork()` es distinto en el hijo y en el padre		

Observe que las modificaciones que realice el proceso padre sobre sus registros e imagen de memoria después del `fork()` no afectan al hijo, y viceversa, las del hijo no afectan al padre. **Sin embargo**, el proceso hijo tiene su propia copia de los descriptores del proceso padre. Éste hace que el hijo tenga acceso a los archivos abiertos por el proceso padre. El padre y el hijo comparten el puntero de posición de los archivos abiertos en el padre.

Piense y responda si esto podría afectar y de qué manera a lo que acaba de observar.

2.1.1.4. Creación de hilos POSIX

A continuación veremos en Listado de código 2.3 un sencillo programa que crea dos hilos, uno de ellos muestra por pantalla la palabra «hola» y el otro la palabra «mundo». Al ser independientes, las palabras pueden aparecer en cualquier orden, incluso repetidas. Para finalizar, es necesario abortar la ejecución presionando simultáneamente las teclas «Ctrl» «c».

Listado de código 2.3: Programa «holahilo.c»

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>

void * thread1()
{
        while(1){
                printf("Hola\n");
        }
}

void * thread2()
{
        while(1){
                printf("mundo\n");
        }
}

int main()
{
        int status;
        pthread_t tid1,tid2;

        pthread_create(&tid1,NULL,thread1,NULL);
        pthread_create(&tid2,NULL,thread2,NULL);
        pthread_join(tid1,NULL);
        pthread_join(tid2,NULL);
        return 0;
}
```

Una vez creado el archivo fuente, se compila así:

```
ubuntu@ubuntu:~$ gcc -o holahilo holahilo.c -lpthread
```

Note que hemos agregado al final «-lpthread». Esa una directiva para el enlazador o *linker*, indicándole que enlace el objeto con la biblioteca (o «librería», en la jerga) «pthread», que significa POSIX thread. Al ejecutarlo, verá:

```
ubuntu@ubuntu:~$ ./holahilo
Hola
Hola
Hola
Hola
mundo
mundo
mundo
mundo
...
```

Y así sin finalizar hasta tanto aborte con «Ctrl-c». La instrucción en el programa fuente C «while(1)» indica que ejecute el «printf("Hola\n");» mientras que sea verdadero; es decir, eternamente.

2.1.2. Práctica con Windows

2.1.2.1. tlist

En Russinovich y Solomon (2004) se nos propone un interesante ejercicio para visualizar el árbol de procesos en Windows. El identificador del proceso y de sus padres o creadores es un atributo particular de los procesos y que, no obstante, la mayoría de las herramientas no lo muestran. La herramienta `tlist.exe`, incluida en las Herramientas de Depuración de Windows (*Windows Debugging Tools*), nos muestra el árbol de procesos si utilizamos el modificador `/t`.

Recordemos que quedan instaladas en
`\Archivos de programa\Debugging Tools for Windows (x86)\`.

Ejemplo 1

```
c:\> cd "\Archivos de programa\Debugging Tools for Windows (x86)"
c:\Archivos de programa\Debugging Tools for Windows (x86)\> tlist /t
System Process (0)
System (4)
  smss.exe (524)
    csrss.exe (572)
    winlogon.exe (596)
      services.exe (640)
        svchost.exe (808)
        svchost.exe (856)
        svchost.exe (932)
        svchost.exe (1016)
        svchost.exe (1056)
        spoolsv.exe (1248)
        nod32krn.exe (1768) NOD32KrnSvcWindow
        nvsvc32.exe (1784) NVSVCPMMWindowClass
```

```
        SMAgent.exe (1844)
        alg.exe (380)
      lsass.exe (652)
explorer.exe (1176) Program Manager
  nod32kui.exe (1444) IHW2
  SMax4PNP.exe (1452) SMax4PNP
  SMax4.exe (1460) SoundMax4
  rundll32.exe (1484) MediaCenter
  TaskSwitchXP.exe (1496) TaskSwitchXP Pro 2.0
  ctfmon.exe (1504)
  cmd.exe (1620) Command Prompt
  cmd.exe (2860) Command Shell
  cmd.exe (2904) Command Shell - tlist /t
    tlist.exe (3800)
C:\Archivos de programa\Debugging Tools for Windows (x86)>
```

Se dejan sangrías en el listado para mostrar la relación padre/hijo. Los procesos cuyos padres ya no existen están sobre la margen izquierda, como lo está `explorer.exe` en este ejemplo. Aunque exista su abuelo no hay forma de mostrar esa relación, porque Windows sólo mantiene la relación padre/hijo y no un enlace previo al creador del padre. En definitiva, vemos varias ramas pero no un árbol.

Ejemplo 2

```
C:\Archivos de programa\Debugging Tools for Windows (x86)> tlist /t
System Process (0)
System (4)
smss.exe (488)
csrss.exe (592)
wininit.exe (644)
  services.exe (688)
    svchost.exe (916)
    svchost.exe (976)
    svchost.exe (1012)
    svchost.exe (1104)
      audiodg.exe (1272)
    svchost.exe (1128)
      WUDFHost.exe (892)
      dwm.exe (2500) DWM Notification Window
    svchost.exe (1188)
      taskeng.exe (2476) MCI command handling window
      taskeng.exe (2392) TaskEng - Proceso del Motor de Programador de tareas
    SLsvc.exe (1300)
    svchost.exe (1328)
    svchost.exe (1472)
    spoolsv.exe (1632)
    svchost.exe (1660)
    ekrn.exe (1860)
    InCDsrv.exe (1968)
    mdm.exe (1988)
    PnkBstrA.exe (2032)
    svchost.exe (300)
```

```
    RichVideo.exe (364)
    svchost.exe (504)
    svchost.exe (540)
    SearchIndexer.exe (636)
    wmpnetwk.exe (3280)
  lsass.exe (704)
  lsm.exe (712)
csrss.exe (656)
winlogon.exe (788)
explorer.exe (2580) Program Manager
  RtHDVCpl.exe (2752) MMDEVAPI Device Window
  rundll32.exe (2860) MediaCenter
  GrooveMonitor.exe (2896)
  UnlockerAssistant.exe (2932) UnlockerAssistant
  PDVDServ.exe (2952) CL RC Engine3 Dummy Winidow
  InCD.exe (2988) InCD Log
  egui.exe (3012) ESET Smart Security
  wmpnscfg.exe (3104) Servicio de uso compartido de red del Reproductor de Windows Media
  Rainlendar.exe (3120) Rainlendar control window
  CursorXP.exe (4012)
  cmd.exe (3132) Símbolo del sistema - tlist  /t
    conime.exe (3424)
    tlist.exe (348)
rundll32.exe (3028)
C:\Archivos de programa\Debugging Tools for Windows (x86)>
```

2.1.2.2. Información de los procesos con el Administrador de Tareas

El Administrador de Tareas de Windows nos provee una lista rápida de los procesos que están ejecutando en el sistema. Se puede arrancar el Administrador de Tareas de tres maneras:

1. Presionando simultáneamente Ctrl+Mayúsculas+Esc
2. Haciendo clic con el botón derecho del ratón sobre la barra de tareas y seleccionando Administrador de Tareas.
3. Presionando simultánemente Ctrl+Alt+Supr y haciendo clic en el botón del Administrador de Tareas.

Una vez que ha comenzado el Administrador de Tareas, haga clic en la pestaña `Procesos` para ver la lista de procesos que se están ejecutando. Note que los procesos están identificados por el nombre de la imagen de la cual son una instancia. A diferencia de algunos objetos en Windows, a los procesos no se les puede dar nombres globales. Para ver detalles adicionales, elija `Seleccionar Columnas` desde el menú `Ver` y seleccione otras columnas para agregar.