Práctica 2.3. Procesos

Objetivos

En esta práctica se revisan las funciones del sistema básicas para la gestión de procesos: políticas de planificación, creación de procesos, grupos de procesos, sesiones, recursos de un proceso y gestión de señales.

Contenidos

Preparación del entorno para la práctica
Políticas de planificación
Grupos de procesos y sesiones
Ejecución de programas
Señales

Preparación del entorno para la práctica

Algunos de los ejercicios de esta práctica requieren permisos de superusuario para poder fijar algunos atributos de un proceso, ej. políticas de tiempo real. Por este motivo, es recomendable realizarla en una **máquina virtual** en lugar de las máquinas físicas del laboratorio.

Políticas de planificación

En esta sección estudiaremos los parámetros del planificador de Linux que permiten variar y consultar la prioridad de un proceso. Veremos tanto la interfaz del sistema como algunos comandos importantes.

Ejercicio 1. La política de planificación y la prioridad de un proceso puede consultarse y modificarse con el comando `chrt`. Adicionalmente, los comandos `nice` y `renice` permiten ajustar el valor de *nice* de un proceso. Consultar la página de manual de ambos comandos y comprobar su funcionamiento cambiando el valor de *nice* de la *shell* a -10 y después cambiando su política de planificación a `SCHED_FIFO` con prioridad 12.

```
$man chrt
Permite consultar y modificar los atributos de planificación de un proceso.
$man nice
Ejecutar un proceso con prioridad de planificación modificada.
$man renice
Modificar la prioridad de un proceso.

$ sudo renice -n -10 -p 23920
23920 (process ID) old priority 0, new priority -10
$ sudo chrt -f -p 12 23920
```

Ejercicio 2. Escribir un programa que muestre la política de planificación (como cadena) y la prioridad del proceso actual, además de mostrar los valores máximo y mínimo de la prioridad para la política de planificación.

```
#include <sys/time.h>
#include <sys/resource.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sched.h>

int main(){
   int schd = sched_getscheduler(0);
   if(schd == -1){
      perror("Error en getscheduler.");
      exit(1);
   }

   if(schd == SCHED_OTHER){
      printf("Politica SCHED_OTHER.\n");
   }
   else if(schd == SCHED_FIFO){

   }
   else if(schd == SCHED_RR){
      printf("Politica SCHED_RR.\n");
   }
   struct sched_param *schdprm;
   if(sched_getparam(0, schdprm)==-1){
      perror("Error en getparam.");
      exit(1);
   }
   printf("Prioridad del proceso: %i.\n", schdprm->sched_priority);
   printf("Prioridad maxima de la politica: %i.\n", sched_get_priority_max(schd));
   printf("Prioridad minima de la politica: %i.\n", sched_get_priority_min(schd));

   return 0;
}

./ej2
Politica SCHED_OTHER.
Prioridad del proceso: 0.
Prioridad maxima de la politica: 0.
Prioridad minima de la politica: 0.
```

Ejercicio 3. Ejecutar el programa anterior en una *shell* con prioridad 12 y política de planificación `SCHED_FIFO` como la del ejercicio 1. ¿Cuál es la prioridad en este caso del programa? ¿Se heredan los atributos de planificación?

```
$ sudo chrt -f -p 12 24225
$  ./ej2
Politica SCHED_FIFO.
Prioridad del proceso: 12.
```

```
Prioridad maxima de la politica: 99.
Prioridad minima de la politica: 1.

Se han heredado los atributos.
```

Grupos de procesos y sesiones

Los grupos de procesos y las sesiones simplifican la gestión que realiza la *shell*, ya que permite enviar de forma efectiva señales a un grupo de procesos (suspender, reanudar, terminar…). En esta sección veremos esta relación y estudiaremos el interfaz del sistema para controlarla.

Ejercicio 4. El comando `ps` es de especial importancia para ver los procesos del sistema y su estado. Estudiar la página de manual y:

- Mostrar todos los procesos del usuario actual en formato extendido.

```
$ sudo ps -f -u $USER
UID        PID  PPID  C STIME TTY          TIME CMD
cursore+  1378  1373  0 Nov27 ?        00:00:09 /usr/bin/openbox --startup /usr/
cursore+  1387     1  0 Nov27 ?        00:00:00 dbus-launch --sh-syntax --exit-w
cursore+  1388     1  0 Nov27 ?        00:00:00 /usr/bin/dbus-daemon --fork --pr
cursore+  1456     1  0 Nov27 ?        00:00:00 /usr/libexec/imsettings-daemon
cursore+  1460     1  0 Nov27 ?        00:00:00 /usr/libexec/gvfsd
cursore+  1465     1  0 Nov27 ?        00:00:00 /usr/libexec/gvfsd-fuse /run/use
cursore+  1556     1  0 Nov27 ?        00:00:00 /usr/bin/VBoxClient --clipboard
cursore+  1558  1556  0 Nov27 ?        00:00:00 /usr/bin/VBoxClient --clipboard
cursore+  1566     1  0 Nov27 ?        00:00:00 /usr/bin/VBoxClient --display
cursore+  1568  1566  0 Nov27 ?        00:00:00 /usr/bin/VBoxClient --display
cursore+  1574     1  0 Nov27 ?        00:00:00 /usr/bin/VBoxClient --seamless
cursore+  1575  1574  0 Nov27 ?        00:00:00 /usr/bin/VBoxClient --seamless
cursore+  1580     1  0 Nov27 ?        00:00:00 /usr/bin/VBoxClient --draganddro
cursore+  1584  1580  0 Nov27 ?        00:02:41 /usr/bin/VBoxClient --draganddro
cursore+  1590  1378  0 Nov27 ?        00:00:00 /usr/bin/ssh-agent /bin/sh -c ex
cursore+  1613     1  0 Nov27 ?        00:00:28 tint2
cursore+  1615     1  0 Nov27 ?        00:00:00 /usr/bin/python /usr/share/syste
cursore+  1616     1  0 Nov27 ?        00:00:01 nm-applet
cursore+  1618     1  0 Nov27 ?        00:00:01 abrt-applet
cursore+  1656     1  0 Nov27 ?        00:00:00 /usr/bin/pulseaudio --start --lo
cursore+  1664     1  0 Nov27 ?        00:00:00 /usr/libexec/at-spi-bus-launcher
cursore+  1683  1664  0 Nov27 ?        00:00:00 /bin/dbus-daemon --config-file=/
cursore+  1694     1  0 Nov27 ?        00:00:03 /usr/libexec/at-spi2-registryd -
cursore+  1757     1  0 Nov27 ?        00:00:00 /usr/libexec/dconf-service
cursore+  1758     1  0 Nov27 ?        00:00:00 /usr/libexec/xdg-desktop-portal
cursore+  1765     1  0 Nov27 ?        00:00:00 /usr/libexec/xdg-document-portal
cursore+  1769     1  0 Nov27 ?        00:00:00 /usr/libexec/xdg-permission-stor
cursore+  1781     1  0 Nov27 ?        00:00:00 /usr/libexec/xdg-desktop-portal-
cursore+  3041     1  0 Nov27 ?        00:00:00 /usr/libexec/gvfsd-trash --spawn
cursore+  3050     1  0 Nov27 ?        00:00:01 /usr/libexec/gvfs-udisks2-volume
cursore+  3055     1  0 Nov27 ?        00:00:00 /usr/libexec/gvfs-afc-volume-mon
cursore+  3061     1  0 Nov27 ?        00:00:00 /usr/libexec/gvfs-gphoto2-volume
cursore+  3066     1  0 Nov27 ?        00:00:00 /usr/libexec/gvfs-mtp-volume-mon
cursore+  3071     1  0 Nov27 ?        00:00:00 /usr/libexec/gvfs-goa-volume-mon
```

```
cursore+  3075    1  0 Nov27 ?       00:00:00 /usr/libexec/goa-daemon
cursore+  3083    1  0 Nov27 ?       00:00:02 /usr/libexec/goa-identity-servic
cursore+  3091    1  0 Nov27 ?       00:00:00 /usr/libexec/mission-control-5
cursore+  3119  1378  0 Nov27 ?      00:00:28 nautilus --new-window
cursore+  3622    1  0 Nov27 ?       00:00:00 /usr/libexec/gvfsd-metadata
cursore+  6798    1  0 Nov27 ?       00:01:45 /usr/libexec/gnome-terminal-serv
cursore+  6805  6798  0 Nov27 ?      00:00:00 gnome-pty-helper
cursore+  6855  6798  0 Nov27 pts/1    00:00:00 bash
cursore+ 23340  6855  0 10:29 pts/1    00:00:00 man strcat
cursore+ 23354 23340  0 10:29 pts/1    00:00:00 less -s
cursore+ 23920  6798  0 10:49 pts/2    00:00:00 bash
cursore+ 24225  6798  0 11:02 pts/0    00:00:00 bash
```

- Mostrar los procesos del sistema, incluyendo el identificador del proceso, el identificador del grupo de procesos, el identificador de sesión, el estado y el comando con todos sus argumentos.

```
[cursoredes@localhost PR3]$ ps -eo pid,gid,sid,s,comm
  PID   GID   SID S COMMAND
    1     0     1 S systemd
    2     0     0 S kthreadd
    3     0     0 S ksoftirqd/0
    5     0     0 S kworker/0:0H
    7     0     0 S migration/0
    8     0     0 S rcu_bh
    9     0     0 R rcu_sched
   10     0     0 S lru-add-drain
   11     0     0 S watchdog/0
   13     0     0 S kdevtmpfs
   14     0     0 S netns
   15     0     0 S khungtaskd
   16     0     0 S writeback
   17     0     0 S kintegrityd
   18     0     0 S bioset
   19     0     0 S bioset
   20     0     0 S bioset
   21     0     0 S kblockd
   22     0     0 S md
   23     0     0 S edac-poller
   29     0     0 S kswapd0
   30     0     0 S ksmd
   31     0     0 S khugepaged
   32     0     0 S crypto
   40     0     0 S kthrotld
   42     0     0 S kmpath_rdacd
   43     0     0 S kaluad
   44     0     0 S kpsmoused
   45     0     0 S ipv6_addrconf
   59     0     0 S deferwq
   90     0     0 S kauditd
  265     0     0 S ata_sff
  274     0     0 S scsi_eh_0
  276     0     0 S scsi_tmf_0
```

```
277    0    0 S scsi_eh_1
278    0    0 S scsi_tmf_1
279    0    0 S scsi_eh_2
281    0    0 S kworker/u2:3
282    0    0 S scsi_tmf_2
306    0    0 S kworker/0:1H
353    0    0 S kdmflush
354    0    0 S bioset
364    0    0 S kdmflush
365    0    0 S bioset
377    0    0 S bioset
378    0    0 S xfsalloc
379    0    0 S xfs_mru_cache
380    0    0 S xfs-buf/dm-0
381    0    0 S xfs-data/dm-0
382    0    0 S xfs-conv/dm-0
383    0    0 S xfs-cil/dm-0
384    0    0 S xfs-reclaim/dm-
385    0    0 S xfs-log/dm-0
386    0    0 S xfs-eofblocks/d
387    0    0 S xfsaild/dm-0
459    0  459 S systemd-journal
476    0  476 S lvmetad
484    0  484 S systemd-udevd
593    0    0 S iprt-VBoxWQueue
604    0    0 S xfs-buf/sda1
605    0    0 S ttm_swap
606    0    0 S xfs-data/sda1
607    0    0 S xfs-conv/sda1
608    0    0 S xfs-cil/sda1
611    0    0 S xfs-reclaim/sda
613    0    0 S xfs-log/sda1
614    0    0 S xfs-eofblocks/s
615    0    0 S xfsaild/sda1
744    0  744 S auditd
746    0  746 S audispd
748    0  746 S sedispatch
751    0    0 S rpciod
752    0    0 S xprtiod
773    0  773 S rngd
774    0  774 S alsactl
775    0  775 S accounts-daemon
778    0  778 S udisksd
781   32  781 S rpcbind
794    0  794 S smartd
795    0  795 S systemd-logind
800    0  800 S ModemManager
803  172  803 S rtkit-daemon
804    0  804 S gssproxy
806  993  806 S lsmd
807  998  807 S polkitd
809    0  809 S abrtd
810    0  810 S abrt-watch-log
822  991  821 S chronyd
834    0  834 S abrt-watch-log
836   81  836 S dbus-daemon
855    0  855 S mcelog
```

```
 874    0   870 S ksmtuned
1042    0  1042 S cupsd
1045    0  1045 S tuned
1047    0  1047 S rsyslogd
1048    0  1048 S sshd
1067    0  1067 S atd
1069    0  1069 S crond
1304    0  1304 S master
1307   89  1304 S qmgr
1334    0  1334 S gdm
1351    0  1349 S VBoxService
1361    0  1361 R X
1373 1000  1334 S gdm-session-wor
1378 1000  1378 S openbox
1387 1000  1378 S dbus-launch
1388 1000  1388 S dbus-daemon
1456 1000  1388 S imsettings-daem
1460 1000  1388 S gvfsd
1465 1000  1388 S gvfsd-fuse
1556 1000  1554 S VBoxClient
1558 1000  1554 S VBoxClient
1566 1000  1564 S VBoxClient
1568 1000  1564 S VBoxClient
1574 1000  1569 S VBoxClient
1575 1000  1569 S VBoxClient
1580 1000  1576 S VBoxClient
1584 1000  1576 S VBoxClient
1590 1000  1590 S ssh-agent
1613 1000  1378 S tint2
1615 1000  1378 S applet.py
1616 1000  1378 S nm-applet
1618 1000  1378 S abrt-applet
1656 1000  1651 S pulseaudio
1664 1000  1388 S at-spi-bus-laun
1683 1000  1388 S dbus-daemon
1694 1000  1388 S at-spi2-registr
1757 1000  1388 S dconf-service
1758 1000  1388 S xdg-desktop-por
1765 1000  1388 S xdg-document-po
1769 1000  1388 S xdg-permission-
1781 1000  1388 S xdg-desktop-por
3041 1000  1388 S gvfsd-trash
3050 1000  1388 S gvfs-udisks2-vo
3055 1000  1388 S gvfs-afc-volume
3061 1000  1388 S gvfs-gphoto2-vo
3066 1000  1388 S gvfs-mtp-volume
3071 1000  1388 S gvfs-goa-volume
3075 1000  1388 S goa-daemon
3083 1000  1388 S goa-identity-se
3091 1000  1388 S mission-control
3119 1000  1378 S nautilus
3162    0     0 S kworker/u2:1
3622 1000  1388 S gvfsd-metadata
6798 1000  1388 R gnome-terminal-
6805   22  1388 S gnome-pty-helpe
6855 1000  6855 S bash
23340 1000  6855 S man
```

```
23354  1000  6855 S less
23920  1000 23920 S bash
24208    89  1304 S pickup
24225  1000 24225 S bash
25407     0     0 S kworker/0:0
25452     0     0 S kworker/0:1
25496  1000 23920 S man
25507  1000 23920 S less
25550     0     0 R kworker/0:2
25566     0   870 S sleep
25573  1000 24225 R ps
```

- Observar el identificador de proceso, grupo de procesos y sesión de los procesos. ¿Qué identificadores comparten la *shell* y los programas que se ejecutan en ella? ¿Cuál es el identificador de grupo de procesos cuando se crea un nuevo proceso?

Comparten el GID 1000 y el SID 6855. El GID de un nuevo proceso es 1000.

Ejercicio 5. Escribir un programa que muestre los identificadores del proceso (PID, PPID, PGID y SID), el número máximo de ficheros que puede abrir y su directorio de trabajo actual.

```
#include <linux/limits.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sched.h>
#include <sys/types.h>

int main(){
   pid_t pid = getpid();
   pid_t ppid = getppid();
   pid_t gid = getpgid(pid);
   pid_t sid = getsid(pid);
   printf("PID: %i\nParent PID: %i\nGID: %i\nSID: %i\n",pid,ppid,gid,sid);

   struct rlimit limit;
   if (getrlimit(RLIMIT_NOFILE, &limit) == -1) {
      perror("Unable to get the resource limits");
      exit(1);
   }
   printf("Numero maximo de ficheros: %li\n", limit.rlim_max);

   char *path = malloc(PATH_MAX);
   if(getcwd(path, PATH_MAX)==NULL){
      perror("Error en getcwd.");
      exit(1);
   }
   printf("Directorio de trabajo: %s\n", path);
```

```
   free (path);

   return 0;
}


$ ./ej5
PID: 25851
Parent PID: 24225
GID: 25851
SID: 24225
Numero maximo de ficheros: 4096
Directorio de trabajo: /home/cursoredes/Documents/PR3
```

Ejercicio 6. Un demonio es un proceso que se ejecuta en segundo plano para proporcionar un servicio. Normalmente, un demonio está en su propia sesión y grupo. Para garantizar que es posible crear la sesión y el grupo, el demonio crea un nuevo proceso para crear la nueva sesión y ejecutar la lógica del servicio. Escribir una plantilla de demonio (creación del nuevo proceso y de la sesión) en el que únicamente se muestren los atributos del proceso (como en el ejercicio anterior). Además, fijar el directorio de trabajo del demonio a `/tmp`.

```
#include <linux/limits.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sched.h>
#include <sys/types.h>

void printAtributos(){
   pid_t pid = getpid();
   pid_t ppid = getppid();
   pid_t gid = getpgid(pid);
   pid_t sid = getsid(pid);
   printf("PID: %i\nParent PID: %i\nGID: %i\nSID: %i\n",pid,ppid,gid,sid);

   struct rlimit limit;
   if (getrlimit(RLIMIT_NOFILE, &limit) == -1) {
      perror("Unable to get the resource limits");
      exit(1);
   }
   printf("Numero maximo de ficheros: %li\n", limit.rlim_max);

   char *path = malloc(PATH_MAX);
   if(getcwd(path, PATH_MAX)==NULL){
      perror("Error en getcwd.");
      exit(1);
   }
   printf("Directorio de trabajo: %s\n", path);
   free (path);
```

```
}

int main() {

   pid_t pid = fork();

   switch (pid) {
   case -1:
`     perror("fork");
      exit(-1);
      break;
   case 0: ;
            pid_t mi_sid = setsid();
      if(chdir("/tmp")==-1){
         perror("Error en chdir.");
         exit(1);
      }

      printAtributos();
      break;

   default:
      sleep(3);
      break;
   }

   return 0;
}


$ ./ej6
PID: 26264
Parent PID: 26263
GID: 26264
SID: 26264
Numero maximo de ficheros: 4096
Directorio de trabajo: /tmp
```

¿Qué sucede si el proceso padre termina antes que el hijo (observar el PPID del proceso hijo)? ¿Y si el proceso que termina antes es el hijo (observar el estado del proceso hijo con `ps`)?

Si el padre termina antes que el hijo, este último queda huérfano y se le asigna el PPID 1.
Si el hijo termina antes, aparece el PID del proceso padre que lo creó.

Nota: Usar `sleep(3)` o `pause(3)` para forzar el orden de finalización deseado.

Ejecución de programas

Ejercicio 7. Escribir dos versiones, una con `system(3)` y otra con `execvp(3)`, de un programa que

ejecute otro programa que se pasará como argumento por línea de comandos. En cada caso, se debe imprimir la cadena "`El comando terminó de ejecutarse`" después de la ejecución. ¿En qué casos se imprime la cadena? ¿Por qué?

```
#include <linux/limits.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sched.h>
#include <sys/types.h>

int main(int argc, char **argv){
   if(argc < 2){
      printf("Faltan argumentos.\n");
      exit(1);
   }
   if(system(argv[1])==-1){
      perror("Error en system.\n");
      exit(1);
   }
   printf("El comando terminó de ejecutarse.\n");

   return 0;
}

$ ./ej7 "ls -l"
total 64
-rwxrwxr-x 1 cursoredes cursoredes 8856 Nov 28 11:34 ej2
-rw-r--r-- 1 root      root       918 Nov 28 11:33 ej2.c
-rwxrwxr-x 1 cursoredes cursoredes 8960 Nov 28 12:10 ej5
-rw-r--r-- 1 root      root       833 Nov 28 12:10 ej5.c
-rwxrwxr-x 1 cursoredes cursoredes 9208 Nov 28 12:33 ej6
-rw-r--r-- 1 root      root      1223 Nov 28 12:33 ej6.c
-rwxrwxr-x 1 cursoredes cursoredes 8592 Nov 28 17:01 ej7
-rw-r--r-- 1 root      root       474 Nov 28 17:01 ej7.c
El comando terminó de ejecutarse.
```

```
#include <linux/limits.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sched.h>
#include <sys/types.h>

int main(int argc, char **argv){
   if(argc < 2){
      printf("Faltan argumentos.\n");
```

```
    exit(1);
  }
  if(execl("/bin/sh", "sh", "-c", argv[1], (char *) 0)==-1){
    perror("Error en execl.\n");
    exit(1);
  }
  printf("El comando terminó de ejecutarse.\n");

  return 0;
}


$ ./ej7b "ls -l"
total 80
-rwxrwxr-x 1 cursoredes cursoredes 8856 Nov 28 11:34 ej2
-rw-r--r-- 1 root      root       918 Nov 28 11:33 ej2.c
-rwxrwxr-x 1 cursoredes cursoredes 8960 Nov 28 12:10 ej5
-rw-r--r-- 1 root      root       833 Nov 28 12:10 ej5.c
-rwxrwxr-x 1 cursoredes cursoredes 9208 Nov 28 12:33 ej6
-rw-r--r-- 1 root      root      1223 Nov 28 12:33 ej6.c
-rwxrwxr-x 1 cursoredes cursoredes 8592 Nov 28 17:01 ej7
-rwxrwxr-x 1 cursoredes cursoredes 8592 Nov 28 17:20 ej7b
-rw-r--r-- 1 root      root       503 Nov 28 17:19 ej7b.c
-rw-r--r-- 1 root      root       474 Nov 28 17:01 ej7.c
```

No aparece el mensaje de "El comando terminó de ejecutarse". Se cambia de proceso al llamar a execl y no se regresa al anterior.

Nota: Considerar cómo deben pasarse los argumentos en cada caso para que sea sencilla la implementación. Por ejemplo: ¿qué diferencia hay entre `./ej7 ps -el` y `./ej7 "ps -el"`?

Ejercicio 8. Usando la versión con `execvp(3)` del ejercicio 7 y la plantilla de demonio del ejercicio 6, escribir un programa que ejecute cualquier programa como si fuera un demonio. Además, redirigir los flujos estándar asociados al terminal usando `dup2(2)`:

- La salida estándar al fichero `/tmp/daemon.out`.
- La salida de error estándar al fichero `/tmp/daemon.err`.
- La entrada estándar a `/dev/null`.

Comprobar que el proceso sigue en ejecución tras cerrar la *shell*.

```
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/limits.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sched.h>
#include <sys/types.h>

int main(int argc, char **argv) {
```

```
  if(argc<2){
    printf("Faltan argumentos.\n");
    exit(1);
  }

  pid_t pid = fork();

  switch (pid) {
  case -1:
    perror("fork");
    exit(-1);
    break;
  case 0:;
    pid_t mi_sid = setsid();
    int fd_out = open("/tmp/daemon.out", O_CREAT|O_RDWR, 0777);
    int fd_err = open("/tmp/daemon.err",O_CREAT|O_RDWR, 0777);
    int fd_null = open("/dev/null",O_CREAT|O_RDWR, 0777);

    if(fd_out==-1||fd_err==-1||fd_null==-1){
      perror("Error en open.");
      exit(1);
    }

    int d_out = dup2(fd_out,1);
    int d_err = dup2(fd_err,2);
    int d_null = dup2(fd_null,0);
    if(d_out==-1||d_err==-1||d_null==-1){
      perror("Error en dup");
      exit(1);
    }

    if(execl("/bin/sh", "sh", "-c", argv[1], (char *) 0)==-1){
      perror("Error en exec.");
      exit(1);
    }

        break;

  default:
        sleep(3);
        break;
  }

  return 0;
}
```

Señales

Ejercicio 9. El comando `kill(1)` permite enviar señales a un proceso o grupo de procesos por su identificador (`pkill(1)` permite hacerlo por nombre de proceso). Estudiar la página de manual del

comando y las señales que se pueden enviar a un proceso.

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7
42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13
52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8
57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3
62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Ejercicio 10. En un terminal, arrancar un proceso de larga duración (ej. `sleep 600`). En otra terminal, enviar diferentes señales al proceso, comprobar el comportamiento. Observar el código de salida del proceso. ¿Qué relación hay con la señal enviada?

```
Terminal 1:kill -s 4 29465

$ sleep 30
    - Después de SIGHUP
Hangup
    - Después de SIGILL

Illegal instruction (core dumped)

Terminal 2:

kill -s 1 29381
kill -s 4 29465
```

Ejercicio 11. Escribir un programa que bloquee las señales `SIGINT` y `SIGTSTP`. Después de bloquearlas el programa debe suspender su ejecución con `sleep(3)` un número de segundos que se obtendrán de la variable de entorno `SLEEP_SECS`. Al despertar, el proceso debe informar de si recibió la señal `SIGINT` y/o `SIGTSTP`. En este último caso, debe desbloquearla con lo que el proceso se detendrá y podrá ser reanudado en la *shell* (imprimir una cadena antes de finalizar el programa para comprobar este comportamiento).

```
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/limits.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sched.h>
#include <sys/types.h>
#include <signal.h>

int main(){
   sigset_t set;
   sigemptyset(&set);
   sigaddset(&set, SIGINT);
   sigaddset(&set, SIGTSTP);
   char *sleep_secs = getenv("SLEEP_SECS");
   sigprocmask(SIG_BLOCK, &set, NULL);
   sleep(atoi(sleep_secs));
   sigset_t pendientes;
   if(sigpending(&pendientes)==-1){
      perror("Error en sigpending.");
      exit(1);
   }
   if(sigismember(&pendientes, SIGINT)){
      printf("Señal SIGINT recibida.\n");
   }
   if(sigismember(&pendientes, SIGTSTP)){
      printf("Señal SIGTSTP recibida.\n");
      sigset_t ublock_set;
      sigemptyset(&ublock_set);
      sigaddset(&ublock_set, SIGTSTP);
      sigprocmask(SIG_UNBLOCK, &ublock_set, NULL);
   }

   printf("El programa ha finalizado.\n");

   return 0;
}
```

Ejercicio 12. Escribir un programa que instale un manejador para las señales `SIGINT` y `SIGTSTP`. El manejador debe contar las veces que ha recibido cada señal. El programa principal permanecerá en un bucle que se detendrá cuando se hayan recibido 10 señales. El número de señales de cada tipo se mostrará al finalizar el programa.

```
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/limits.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sched.h>
#include <sys/types.h>
#include <signal.h>

volatile int n_sigint = 0;
volatile int n_sigtspt = 0;

void handler_sigint(){
   printf("Recibida senal SIGINT.\n");
   n_sigint++;
}
void handler_sigtspt(){
   printf("Recibida senal SIGTSPT.\n");
   n_sigtspt++;
}

int main(){
   struct sigaction sigact_int;
   sigact_int.sa_handler = handler_sigint;
   struct sigaction sigact_stp;
   sigact_stp.sa_handler = handler_sigtspt;
   sigaction(2, &sigact_int, NULL);
   sigaction(20, &sigact_stp, NULL);
   while(n_sigint+n_sigtspt<10){}

   printf("Numero de senales SIGINT recibidas: %i.\nNumero de senales SIGTSTP recibidas: %i.\n",
n_sig$

   return 0;
}
```

Ejercicio 13. Escribir un programa que realice el borrado programado del propio ejecutable. El programa tendrá como argumento el número de segundos que esperará antes de borrar el fichero. El borrado del fichero se podrá detener si se recibe la señal `SIGUSR1`.

```
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/limits.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
```

```
#include <stdlib.h>
#include <sched.h>
#include <sys/types.h>
#include <signal.h>

volatile int kill_prog = 1;
void handler_kill(){
   kill_prog = 0;
}

int main(int argc, char **argv){
   if(argc<2){
      printf("Faltan argumentos.\n");
      exit(1);
   }
   struct sigaction act_kill;
   act_kill.sa_handler = handler_kill;
   sigaction(10, &act_kill, NULL);
   sleep(atoi(argv[1]));
   if(kill_prog == 1){
      unlink(argv[0]);
   }
   return 0;
}
```

Nota: Usar `sigsuspend(2)` para suspender el proceso y la llamada al sistema apropiada para borrar el fichero.