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.

nice [OPTION] [COMMAND [ARG]...]
run a program with modified scheduling priority
Run COMMAND with an adjusted niceness, which affects process scheduling. With no COMMAND, print the current niceness. Nicenesses range from -20 (most favorable scheduling) to 19 (least favorable).

-n, --adjustment=N
add integer N to the niceness (default 10)
--help
display this help and exit
--version
output version information and exit

renice [-n] priority [-g|-p|-u] identifier..
alter priority of running processes
-n, --priority priority
Specify the scheduling priority to be used for the process, process group, or user. Use of the option -n or --priority is optional, but when used it must be the first argument.
-g, --pgrp
Interpret the succeeding arguments as process group IDs.
-p, --pid
Interpret the succeeding arguments as process IDs (the default).
-u, --user
Interpret the succeeding arguments as usernames or UIDs.
-V, --version
Display version information and exit.
-h, --help
Display help text and exit.

Comando:
ps
Salida:
PID TTY TIME CMD
77604 pts/0 00:00:00 bash
7735 pts/0 00:00:00 ps

Comando:
renice -n -10 -p 77604

Comando:
Chrt -f -p 12 77604

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.

```c
#include <sched.h>

#include <stdlib.h>

#include <stdio.h>

int main() {

   //Obtenemos política de planificación, 0 por ser el proceso actual

  int sched_ = sched_getscheduler(0);

  if (sched_ == -1) {

   perror("getscheduler");

   exit(1);
```

```
 }

  if (sched_ == SCHED_OTHER) printf("Política de planificación: SCHED_OTHER \n");

  if (sched_ == SCHED_BATCH) printf("Política de planificación: SCHED_BATCH \n");

  if (sched_ == SCHED_IDLE) printf("Política de planificación: SCHED_IDLE \n");

  if (sched_ == SCHED_FIFO) printf("Política de planificación: SCHED_FIFO \n");

  if (sched_ == SCHED_RR) printf("Política de planificación: SCHED_RR \n");

  //Obtenemos parámetros de prioridad

  struct sched_param p;

  int param_ = sched_getparam(0, &p);

  if (param_ == -1) {

      perror("getparam");

      exit(1);

  }

  printf("Prioridad: %i \n", p.sched_priority);

  //Obtenemos prioridades máxima y mínima

  int max = sched_get_priority_max(sched_);

  int min = sched_get_priority_min(sched_);

  if (max == -1 || min == -1) {

      perror("min, max");

      exit(1);

  }

  printf("Prioridad máxima: %i \n", max);

  printf("Prioridad mínima: %i \n", min);

  return 0;

}
```

Salida:

Política de planificación: SCHED_OTHER

Prioridad: 0

Prioridad máxima: 0 Prioridad mínima: 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?

Política de planificación: SCHED_FIFO

Prioridad: 12

Prioridad máxima: 99

Prioridad mínima: 1

¿Se heredan los atributos de planificación?

Sí, se heredan 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.

Comando:
99393

Salida:

```
UID        PID    PPID  C STIME TTY          TIME CMD
usuario+   4213      1  0 11:06 ?      00:00:00 dbus-launch --autolaunch=4fe
usuario+   4214      1  0 11:06 ?      00:00:00 /usr/bin/dbus-daemon --syslo
usuario+   4265      1  0 11:06 ?      00:00:00 /usr/libexec/dconf-service
usuario+  37734      1  0 11:07 ?      00:00:00 /lib/systemd/systemd --user
usuario+  37735  37734  0 11:07 ?      00:00:00 (sd-pam)
usuario+  37822  37734  0 11:07 ?      00:00:00 /usr/bin/pipewire
usuario+  37823  37734  0 11:07 ?      00:00:00 /usr/bin/pipewire-media-sess
usuario+  37829  37734  0 11:07 ?      00:00:00 /usr/bin/pulseaudio --daemon
usuario+  37831  37734  0 11:07 ?      00:00:00 /usr/bin/dbus-daemon --sessi
usuario+  37835      1  0 11:07 ?      00:00:00 /usr/bin/gnome-keyring-daemo
usuario+  37846  37734  0 11:07 ?      00:00:00 /usr/libexec/gvfsd
usuario+  37851  37734  0 11:07 ?      00:00:00 /usr/libexec/gvfsd-fuse /run
usuario+  37956   4130  0 11:07 ?      00:00:00 /usr/libexec/gnome-session-b
usuario+  38001  37734  0 11:07 ?      00:00:00 /usr/libexec/tracker-miner-f
usuario+  38077  37734  0 11:07 ?      00:00:00 /usr/libexec/gvfs-udisks2-vo
usuario+  38114  37956  0 11:07 ?      00:00:00 [notify-send] <defunct>
usuario+  38128  37734  0 11:07 ?      00:00:00 /usr/lib/x86_64-linux-gnu/xf
usuario+  38142  37734  0 11:07 ?      00:00:00 /usr/libexec/gvfs-afc-volume
usuario+  38146  37734  0 11:07 ?      00:00:00 /usr/lib/x86_64-linux-gnu/xf
usuario+  38159  37734  0 11:07 ?      00:00:00 /usr/libexec/gvfs-mtp-volume
```

```
usuario+  38163  37734  0 11:07 ?     00:00:00 /usr/libexec/at-spi-bus-laun
usuario+  38164  37734  0 11:07 ?     00:00:00 /usr/libexec/gvfs-gphoto2-vo
usuario+  38172  38163  0 11:07 ?     00:00:00 /usr/bin/dbus-daemon --confi
usuario+  38180  37734  0 11:07 ?     00:00:00 /usr/libexec/gvfs-goa-volume
usuario+  38184  37734  0 11:07 ?     00:00:00 /usr/libexec/goa-daemon
usuario+  38186  37734  0 11:07 ?     00:00:01 /usr/libexec/at-spi2-registr
usuario+  38212  37734  0 11:07 ?     00:00:00 /usr/libexec/goa-identity-se
usuario+  38268  37734  0 11:07 ?     00:00:00 /usr/libexec/gnome-session-c
usuario+  38286  37734  0 11:07 ?     00:00:00 /usr/libexec/gnome-session-b
usuario+  38335  37734  4 11:07 ?     00:01:52 /usr/bin/gnome-shell
usuario+  38921  37734  0 11:07 ?     00:00:00 /usr/libexec/gvfsd-metadata
usuario+  38995  37734  0 11:07 ?     00:00:00 /usr/libexec/xdg-permission-
usuario+  38997  37734  0 11:07 ?     00:00:00 /usr/libexec/gnome-shell-cal
usuario+  39011  37734  0 11:07 ?     00:00:00 /usr/libexec/evolution-sourc
usuario+  39051  37734  0 11:07 ?     00:00:00 /usr/libexec/evolution-calen
usuario+  39086  37734  0 11:07 ?     00:00:00 /usr/libexec/dconf-service
usuario+  39089  37734  0 11:07 ?     00:00:00 /usr/libexec/evolution-addre
usuario+  39177  37734  0 11:07 ?     00:00:00 /usr/bin/gjs /usr/share/gnom
usuario+  39370  37734  0 11:07 ?     00:00:00 sh -c /usr/bin/ibus-daemon -
usuario+  39371  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-a11y-settin
usuario+  39373  39370  0 11:07 ?     00:00:00 /usr/bin/ibus-daemon --panel
usuario+  39374  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-color
usuario+  39375  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-datetime
usuario+  39377  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-housekeepin
usuario+  39379  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-keyboard
usuario+  39380  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-media-keys
usuario+  39387  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-power
usuario+  39390  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-print-notif
usuario+  39391  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-rfkill
usuario+  39392  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-screensaver
usuario+  39393  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-sharing
usuario+  39394  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-smartcard
usuario+  39395  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-sound
usuario+  39396  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-wacom
usuario+  39397  37734  0 11:07 ?     00:00:00 /usr/libexec/gsd-xsettings
usuario+  39430  38286  0 11:07 ?     00:00:11 /usr/bin/gnome-software --ga
usuario+  39435  38286  0 11:07 ?     00:00:00 /bin/sh /usr/local/bin/MiExp
usuario+  39440  39435  0 11:07 ?     00:00:01 /usr/bin/nautilus /mnt/Disco
usuario+  39461  38286  0 11:07 ?     00:00:00 /usr/bin/python3 /usr/bin/bl
usuario+  39473  38286  0 11:07 ?     00:00:00 /usr/lib/x86_64-linux-gnu/in
usuario+  39474  39373  0 11:07 ?     00:00:00 /usr/libexec/ibus-memconf
usuario+  39476  39373  0 11:07 ?     00:00:01 /usr/libexec/ibus-extension-
usuario+  39477  38286  0 11:07 ?     00:00:01 /usr/lib/x86_64-linux-gnu/li
usuario+  39479  37734  0 11:07 ?     00:00:00 /usr/libexec/ibus-x11 --kill
usuario+  39480  38286  0 11:07 ?     00:00:00 /usr/libexec/gsd-disk-utilit
usuario+  39485  38286  0 11:07 ?     00:00:00 /usr/libexec/evolution-data-
usuario+  39489  37734  0 11:07 ?     00:00:00 /usr/libexec/ibus-portal
usuario+  39552  37846  0 11:07 ?     00:00:00 /usr/libexec/gvfsd-trash --s
usuario+  39596  39373  0 11:07 ?     00:00:00 /usr/libexec/ibus-engine-sim
usuario+  39600  37734  0 11:07 ?     00:00:00 /usr/bin/gjs /usr/share/gnom
usuario+  39677  37734  0 11:07 ?     00:00:00 /usr/libexec/xdg-desktop-por
```

```
usuario+   39690   37734  0 11:07 ?        00:00:00 /usr/libexec/xdg-document-po
usuario+   39706   37734  0 11:07 ?        00:00:00 /usr/libexec/xdg-desktop-por
usuario+   39760   37734  0 11:07 ?        00:00:00 /usr/libexec/gsd-printer
usuario+   39835   37734  0 11:07 ?        00:00:00 /usr/lib/bluetooth/obexd
usuario+   39945   37734  0 11:07 ?        00:00:00 /usr/libexec/xdg-desktop-por
usuario+   40689   38335  0 11:07 ?        00:00:01 gjs /usr/share/gnome-shell/e
usuario+   40762   37846  0 11:07 ?        00:00:00 /usr/libexec/gvfsd-burn --sp
usuario+   51128   38335  4 11:07 ?        00:01:40 /opt/google/chrome/chrome --
usuario+   51143   51128  0 11:07 ?        00:00:00 cat
usuario+   51144   51128  0 11:07 ?        00:00:00 cat
usuario+   51181   37734  0 11:07 ?        00:00:00 /opt/google/chrome/chrome_cr
usuario+   51183   37734  0 11:07 ?        00:00:00 /opt/google/chrome/chrome_cr
usuario+   51194   51128  0 11:07 ?        00:00:00 /opt/google/chrome/chrome --
usuario+   51195   51128  0 11:07 ?        00:00:00 /opt/google/chrome/chrome --
usuario+   51211   51195  0 11:07 ?        00:00:00 /opt/google/chrome/nacl_help
usuario+   51214   51195  0 11:07 ?        00:00:00 /opt/google/chrome/chrome --
usuario+   51332   51194  5 11:07 ?        00:02:02 /opt/google/chrome/chrome --
usuario+   51339   51128  0 11:07 ?        00:00:18 /opt/google/chrome/chrome --
usuario+   51395   51214  0 11:07 ?        00:00:00 /opt/google/chrome/chrome --
usuario+   51413   51332  0 11:07 ?        00:00:00 /opt/google/chrome/chrome --
usuario+   51587   51214  0 11:07 ?        00:00:02 /opt/google/chrome/chrome --
usuario+   52704   51128  0 11:07 ?        00:00:00 /opt/google/chrome/chrome --
usuario+   71911   51214  8 11:08 ?        00:03:21 /opt/google/chrome/chrome --
usuario+   76310   38286  0 11:08 ?        00:00:00 update-notifier
usuario+   77584   37734  0 11:15 ?        00:00:05 /usr/libexec/gnome-terminal-
usuario+   77604   77584  0 11:15 pts/0    00:00:00 bash
usuario+   77616   37734  0 11:15 ?        00:00:00 /usr/lib/x86_64-linux-gnu/gc
usuario+   78250   38335  1 11:26 ?        00:00:21 /usr/share/code/code --unity
usuario+   78254   78250  0 11:26 ?        00:00:00 /usr/share/code/code --type=
usuario+   78255   78250  0 11:26 ?        00:00:00 /usr/share/code/code --type=
usuario+   78257   78255  0 11:26 ?        00:00:00 /usr/share/code/code --type=
usuario+   78275   37734  0 11:26 ?        00:00:00 /usr/share/code/chrome_crash
usuario+   78288   78254  3 11:26 ?        00:00:45 /usr/share/code/code --type=
usuario+   78297   78250  0 11:26 ?        00:00:00 /usr/share/code/code --type=
usuario+   78299   78288  0 11:26 ?        00:00:00 /usr/share/code/code --type=
usuario+   78319   78250 10 11:26 ?        00:02:09 /usr/share/code/code --type=
usuario+   78376   78250  0 11:26 ?        00:00:03 /usr/share/code/code --type=
usuario+   78403   78250  0 11:26 ?        00:00:03 /usr/share/code/code --ms-en
usuario+   78425   78376  0 11:26 ?        00:00:00 /usr/share/code/code --ms-en
usuario+   78465   78376  0 11:26 ?        00:00:00 /usr/share/code/code --ms-en
usuario+   78544   37846  0 11:26 ?        00:00:00 /usr/libexec/gvfsd-network -
usuario+   78560   37846  0 11:26 ?        00:00:00 /usr/libexec/gvfsd-dnssd --s
usuario+   79211   51214  0 11:45 ?        00:00:00 /opt/google/chrome/chrome --
usuario+   79216   51214  3 11:45 ?        00:00:02 /opt/google/chrome/chrome --
usuario+   79244   51214  0 11:45 ?        00:00:00 /opt/google/chrome/chrome --
usuario+   79294   77604  0 11:46 pts/0    00:00:00 ps -u usuario_vms -f
```

- 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.

Comando:

ps -e -o pid, gid, sid, state, command

Salida:

PID GID SID S COMMAND

1 0 1 S /sbin/init

2 0 0 S [kthreadd]

3 0 0 I [rcu_gp]

4 0 0 I [rcu_par_gp]

5 0 0 I [netns]

7 0 0 I [kworker/0:0H-events_highpri]

8 0 0 I [kworker/0:1-events]

10 0 0 I [mm_percpu_wq]

11 0 0 S [rcu_tasks_rude_]

12 0 0 S [rcu_tasks_trace]

13 0 0 S [ksoftirqd/0]

14 0 0 I [rcu_sched]

15 0 0 S [migration/0]

16 0 0 S [idle_inject/0]

17 0 0 S [cpuhp/0]

…sigue hasta 79385 (aunque no todos)

- 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 identificador de grupo (100) y el de sesión (77604). EL identificador de grupo para un nuevo proceso es 100.

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.

```c
#include <sched.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/time.h>

#include <sys/resource.h>

#include <limits.h>

*/int main() {

   pid_t pid_ = getpid();

   if (pid_ == -1) {

       perror("getpid");

       exit(1);

   }

   printf("pid: %i \n", pid_);

   int ppid_ = getppid();

   if (ppid_ == -1){

       perror("getppid");

       exit(1);

   }

   printf("ppid: %i \n", ppid_);

   int pgid_ = getpgid(pid_);

   if (pgid_ == -1){

       perror("getpgid");

       exit(1);

   }

   printf("pgid: %i \n", pgid_);

   int sid_ = getsid(pid_);

   if (sid_ == -1){

       perror("getpsid");
```

```
        exit(1);
    }
    printf("sid: %i \n", sid_);
    struct rlimit rlim;
    if(getrlimit(RLIMIT_NOFILE, &rlim) == -1){
        perror("getrlimit");
        exit(1);
    }
    printf("Maximo ficheros. Limite actual: %li\n" , rlim.rlim_cur);
    printf("Maximo ficheros. Valor máximo: %li\n" , rlim.rlim_max);
    char *buf = malloc(PATH_MAX);
    if (getcwd(buf, PATH_MAX)==NULL){
        perror("getcwd");
        exit(1);
    }
    printf("Directorio: %s\n",buf);
    free(buf);
    return 0;
}
```

Salida:

pid: 80272

ppid: 77604

pgid: 80272

sid: 77604

Maximo ficheros. Limite actual: 1024

Maximo ficheros. Valor máximo: 1048576

Directorio: /home/usuario_vms/Documentos

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`.

```c
#include <sched.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/time.h>

#include <sys/resource.h>

#include <limits.h>

int mostrarDatos() {

   pid_t pid_ = getpid();

   if (pid_ == -1) {

       perror("getpid");

       return -1;

   }

   printf("pid: %i \n", pid_);

   int ppid_ = getppid();

   if (ppid_ == -1){

       perror("getppid");

       return -1;

   }

   printf("ppid: %i \n", ppid_);

   int pgid_ = getpgid(pid_);

   if (pgid_ == -1){

       perror("getpgid");

       return -1;

   }
```

```
   printf("pgid: %i \n", pgid_);
   int sid_ = getsid(pid_);
   if (sid_ == -1){
       perror("getpsid");
       return -1;
   }
   printf("sid: %i \n", sid_);
   struct rlimit rlim;
   if(getrlimit(RLIMIT_NOFILE, &rlim) == -1){
       perror("getrlimit");
       return -1;
   }
   printf("Maximo ficheros. Limite actual: %li\n" , rlim.rlim_cur);
   printf("Maximo ficheros. Valor máximo: %li\n" , rlim.rlim_max);
   char *buf = malloc(PATH_MAX);
   if (getcwd(buf, PATH_MAX)==NULL){
       perror("getcwd");
       return -1;
   }
   printf("Directorio: %s\n",buf);
   free(buf);
   return 0;
}

int main(){
   int pid= fork();
   if(pid == -1){
       perror("fork");
```

```
        exit(1);
    }
    else if(pid == 0){ //proceso hijo
        pid_t s = setsid();
        if(s == -1){
            perror("setsid");
            exit(1);
        }
        int chdir_ = chdir("/tmp");
        if(chdir_ == -1){
            perror("chdir");
            exit(1);
        }
        if(mostrarDatos() == -1){
            exit(1);
        }
        exit(0);
    }
    else{
        sleep(5);
    }
    return 0;
}
```

Salida:

pid: 80534

ppid: 80533

pgid: 80534

sid: 80534

Maximo ficheros. Limite actual: 1024

Maximo ficheros. Valor máximo: 1048576

Directorio: /tmp

¿Qué sucede si el proceso padre termina antes que el hijo (observar el PPID del proceso hijo)?

El hijo se queda sin padre y el ppid pasa a ser otro.

¿Y si el proceso que termina antes es el hijo (observar el estado del proceso hijo con ps)?

El ppid es el del proceso padre.

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?

```c
#include <stdlib.h>

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

    if (argc < 2){

        printf("Número de argumentos insuficiente \n");

        exit(1);

    }

    int call = system(argv[1]);

    if (call == -1) {

        printf("Error al crear el hijo \n");

        exit(1);

    }

    printf("El comando terminó de ejecutarse \n");

    return 0;

}
```

Comando:

./a.out ls

Salida:

a.out practica.c

El comando terminó de ejecutarse

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc, char** argv) {
    if (argc < 2){
        printf("Número de argumentos insuficiente \n");
        exit(1);
    }
    int call = execl("/bin/sh", "sh", "-c", argv[1], (char *) 0);
    if (call == -1) {
        printf("Error al crear el hijo \n");
        exit(1);
    }
    printf("El comando terminó de ejecutarse \n");
    return 0;
}
```

Comando:

./a.out ls

Salida:

a.out practica.c

¿En qué casos se imprime la cadena? ¿Por qué?

En el caso que usamos execl se sustituye la imagen del proceso actual por una nueva que ejecuta el comando y por tanto no se vuelve al programa principal y la cadena no se imprime. En el caso system sí se escribe porque crea un proceso hijo que ejecuta el comando de la shell y sí que retorna.

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*.

```c
#include <stdlib.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(int argc, char** argv) {
    if(argc < 2){
        printf("Argumentos insuficientes");
        exit(1);
    }
    int pid = fork();

    if(pid == -1){
        perror("fork");
        exit(1);
    }
    else if (pid == 0){ //proceso hijo
        pid_t s = setsid();

        //redirigir flujo de la salida estándar a /tmp/daemon.out
        int f1 = open("/tmp/daemon.out", O_CREAT|O_RDWR, 0666);
        if(f1 == -1){
            perror("open1");
            exit(1);
        }
        if(dup2(f1,1)== -1){
            perror("dup2,1");
            exit(1);
        }

        //redirigir flujo de la salida de error estándar a /tmp/daemon.err
        int f2 = open("/tmp/daemon.err", O_CREAT|O_RDWR, 0666);
        if(f2 == -1){
            perror("open2");
            exit(1);
        }
        if(dup2(f2,2)== -1){
            perror("dup2,2");
            exit(1);
        }
```

```
        //redirigir flujo de la entrada estándar a /dev/null
        int f3 = open("/dev/null", O_CREAT|O_RDWR, 0666);
        if(f3 == -1){
            perror("open3");
            exit(1);
        }
        if(dup2(f3,0)== -1){
            perror("dup2,3");
            exit(1);
        }

        if(execvp(argv[1], argv + 1)==-1){
            perror("execvp");
            exit(1);
        }
        exit(0);
    }
    else{
    }
    return 0;
}
```

Comando:

./a8.out ./a5.out

Salida (en /tmp/daemon.out):

pid: 24707
ppid: 1
pgid: 24707
sid: 24707
Maximo ficheros. Limite actual: 1024
Maximo ficheros. Valor máximo: 1048576
Directorio: /home/usuarioso/Documentos

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 - terminate a process
kill [-signal | -s signal | -p] [-q value] [-a] [--timeout milliseconds signal] [--] pid | name...
kill -l [number] | -L

Comando:
kill -l

Salida:

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+136) SIGRTMIN+237) SIGRTMIN+3
38) SIGRTMIN+439) SIGRTMIN+540) SIGRTMIN+641) SIGRTMIN+742) SIGRTMIN+8
43) SIGRTMIN+944) 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-956) SIGRTMAX-857) SIGRTMAX-7
58) SIGRTMAX-659) SIGRTMAX-560) SIGRTMAX-461) SIGRTMAX-362) SIGRTMAX-2
63) SIGRTMAX-164) 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:

sleep 600

Terminal 2:

kill -s 19 99393

[1]+ Detenido sleep 600

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).

```c
#include <stdlib.h>

#include <signal.h>

int main() {

    //creamos un conjunto de señales y añadimos las señales a bloquear

    sigset_t set1;

    sigemptyset(&set1);

    sigaddset(&set1, SIGINT);

    sigaddset(&set1, SIGTSTP);
```

```
//tomamos la variable de entorno SLEEP_SECS
char* sleeping = getenv("SLEEP_SECS");
if (sleeping == NULL) {
    printf("Error en getenv \n");
    exit(1);
}
//bloqueamos las señales de set1
sigprocmask(SIG_BLOCK, &set1, NULL);
//suspendemos la ejecución
sleep(atoi(sleeping));
//--------------------------------//
//Miramos las señales que tenemos pendientes
sigset_t set2;
int pending = sigpending(&set2);
if (pending == -1) {
    printf("Error en sigpending \n");
    exit(1);
}
//Está SIGINT en pendientes??
if (sigismember(&set2, SIGINT)) {
    printf("SIGINT recibida \n");
}
//Está SIGTSTP en pendientes??
if (sigismember(&set2, SIGTSTP)) {
    printf("SIGTSTP recibida \n");
    sigset_t set3;
    sigemptyset(&set3);
    sigaddset(&set3, SIGTSTP);
    //las señales en set3 se desbloquean
```

```
        sigprocmask(SIG_UNBLOCK, &set3, NULL);

    }

    printf("Fin del ejercicio 11 \n");

    return 0;

}
```

[Terminal 1]

Comando:

export SLEEP_SECS="30"

./a.out

Salida:

SIGINT recibida

SIGTSTP recibida [1]+ Detenido ./a.out

[Terminal 2]

Comandos:

kill -s 2 1690

kill -s 20 1690

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 <stdlib.h>

#include <signal.h>

int nSIGINT = 0;

int nSIGTSTP = 0;

void manejadorSIGINT() {

    //printf("Señal SIGINT %i \n", nSIGINT);

    nSIGINT++;

}

void manejadorSIGTSTP() {

    //printf("Señal SIGTSTP %i \n", nSIGTSTP);

    nSIGTSTP++;

}

int main() {
```

```
    //creamos structs para los nuevos manejadores
    struct sigaction sSIGINT;
    sSIGINT.sa_handler = manejadorSIGINT;
    struct sigaction sSIGTSTP;
    sSIGTSTP.sa_handler = manejadorSIGTSTP;
    //modificar acciones por defecto de ambas señales
    sigaction(SIGINT, &sSIGINT, NULL);
    sigaction(SIGTSTP, &sSIGTSTP, NULL);
    //en bucle hasta recibir 10 señales
    while (nSIGINT + nSIGTSTP < 10) {}
    //mostrar número de señales de cada tipo
    printf("Señales SIGINT: %i \n", nSIGINT);
    printf("Señales SIGTSTP: %i \n", nSIGTSTP);
    return 0;
}
```

[Terminal 1]

Comando:

./a.out

Salida:

Señales SIGINT: 9

Señales SIGTSTP: 1

[Terminal 2]

Comandos:

kill -s 2 1749 (9 veces)

kill -s 20 1749

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

```c
#include <unistd.h>

volatile int borrado = 1;

void manejador() {

    borrado = 0;

}

void manejadorAlarma() {

    printf("Alarma enviada\n");

}

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

    if (argc < 2) {

        printf("Número de argumentos insuficientes \n");

        exit(1);

    }

    //creamos el nuevo manejador para SIGUSR

    struct sigaction sSIGUSR1;

    sSIGUSR1.sa_handler = manejador;

    sigaction(SIGUSR1, &sSIGUSR1, NULL);



    //creamos el nuevo manejador para SIGALRM

    struct sigaction sSIGALRM;

    sSIGALRM.sa_handler = manejadorAlarma;

    sigaction(SIGALRM, &sSIGALRM, NULL);

    //espera antes de borrar el fichero

    alarm(atoi(argv[1])); //cuando pasen los segundos manda una señal SIGALARM

    sigset_t set;

    sigemptyset(&set);

    int sus = sigsuspend(&set);

    if (borrado == 1) {

        printf("Borramos \n");
```

```
        unlink(argv[0]);
    }
    else {
        printf("Ejecutable no borrado \n");
    }
    return 0;
}
```

[Terminal 1]

Comando:

./a.out 30

Salida:

Ejecutable no borrado

Comando:

./a.out 30

Salida:

Alarma enviada

Borramos

[Terminal 2]

Comando:

kill -s SIGUSR1 24999

[No hay comando]

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