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

NAME
nice - change process priority

SYNOPSIS
#include <unistd.h>

int nice(int inc);

DESCRIPTION
nice() adds inc to the nice value for the calling thread. (A higher nice value means a low priority.)

The range of the nice value is +19 (low priority) to -20 (high priority). Attempts to set a nice value outside the range are clamped to the range.

Traditionally, only a privileged process could lower the nice value (i.e., set a higher priority).

However, since Linux 2.6.12, an unprivileged process can decrease the nice
value of a target process that has a suitable RLIMIT_NICE soft limit; see getrlimit(2) for details.

RETURN VALUE
On success, the new nice value is returned (but see NOTES below). On error, -1 is returned, and errno is set appropriately.

A successful call can legitimately return -1. To detect an error, set errno to 0 before the call, and check whether it is nonzero after nice() returns -1.

ERRORS
EPERM The calling process attempted to increase its priority by supplying a negative inc but has insufficient privileges. Under Linux, the CAP_SYS_NICE capability is required.
(But see the discussion of the RLIMIT_NICE resource limit in setrlimit(2).)

RENICE

NAME
renice - alter priority of running processes

SYNOPSIS
renice [-n] priority [-g|-p|-u] identifier...

DESCRIPTION
renice alters the scheduling priority of one or more running processes. The first argument is the priority value to be used. The other arguments are interpreted as process IDs
(by default), process group IDs, user IDs, or user names. renice'ing a process group causes all processes in the process group to have their scheduling priority altered.
renice'ing a user causes all processes owned by the user to have their scheduling priority altered.

OPTIONS
-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.

CHRT: sirve para la planificación y la prioridad de un proceso
`chrt -v -p`

```
$ sudo renice -n -10 -p $$
2778 (process ID) prioridad anterior 0, nueva prioridad -10
$ sudo chrt -f -p 12 $$
$ chrt -p $$
política actual de planificación del pid 2778: SCHED_FIFO
política actual de planificación del pid 2778: 12
```

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 <stdio.h>
#include <stdlib.h>
#include <sched.h>

int main(int argc, char **argv) {
   int sched_policy = sched_getscheduler(0);

   //Planificación
   switch (sched_policy) {
      case SCHED_OTHER:
         printf("SCHED_OTHER, ");
         break;
      case SCHED_FIFO:
         printf("SCHED_FIFO, ");
         break;
      case SCHED_RR:
         printf("SCHED_RR, ");
         break;
      default:
         printf("SCHED_ERROR, ");
         break;
   }

   //Max y min
   int max = sched_get_priority_max(sched_policy);
   int min = sched_get_priority_min(sched_policy);
   printf("MAX: %i - MIN: %i, ", max, min);

   //Prioridad
   struct sched_param p;
   sched_getparam(0, &p);
   printf("PRIORIDAD ACT: %i\n", p.sched_priority);
}
```

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 $$
$ ./ej2
SCHED_FIFO, MAX: 99, MIN: 1, PRIORIDAD ACT: 12
```

Grupos de procesos y sesiones

Los grupos de procesos y 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.
- Mostrar los procesos del sistema, incluyendo el identificador del proceso, el identificador del grupo de procesos, el identificador de sesión, el estado y la línea de comandos.
- 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?

$ ps -u $USER -f : Mostrar todos los procesos del usuario actual en formato extendido

```
UID        PID    PPID  C STIME TTY          TIME CMD
usuario+    874       1  0 nov21 ?       00:00:00 /lib/systemd/systemd --user
usuario+    875     874  0 nov21 ?       00:00:00 (sd-pam)
usuario+    892     874  0 nov21 ?       00:00:00 /usr/bin/pulseaudio --daemonize=no
--log-target=journal
usuario+    895       1  0 nov21 ?       00:00:03 /usr/bin/gnome-keyring-daemon --daemonize --login
usuario+    898     858  0 nov21 ?       00:00:03 lxqt-session
usuario+    914     874  0 nov21 ?       00:00:03 /usr/bin/dbus-daemon --session --address=systemd:
--nofork --nopidfile --systemd-activation --syslog-only
usuario+    949       1  0 nov21 ?       00:00:00 /usr/bin/VBoxClient --clipboard
usuario+    950     949  0 nov21 ?       00:00:00 /usr/bin/VBoxClient --clipboard
usuario+    961       1  0 nov21 ?       00:00:00 /usr/bin/VBoxClient --seamless
usuario+    962     961  0 nov21 ?       00:00:00 /usr/bin/VBoxClient --seamless
usuario+    968       1  0 nov21 ?       00:00:00 /usr/bin/VBoxClient --draganddrop
usuario+    969     968  0 nov21 ?       00:01:54 /usr/bin/VBoxClient --draganddrop
usuario+    976       1  0 nov21 ?       00:00:00 /usr/bin/VBoxClient --vmsvga
usuario+    977     976  0 nov21 ?       00:00:00 /usr/bin/VBoxClient --vmsvga
usuario+    980     898  0 nov21 ?       00:00:00 /usr/bin/ssh-agent /usr/bin/im-launch env
LXQT_DEFAULT_OPENBOX_CONFIG=/etc/xdg/xdg-Lubuntu/openbox/lxqt-rc.xml
/usr/bin/startlxqt
usuario+    998     898  0 nov21 ?       00:00:04 /usr/bin/openbox --config-file
/home/usuarioso/.config/openbox/lxqt-rc.xml
usuario+   1001       1  0 nov21 ?       00:00:00 /usr/libexec/at-spi-bus-launcher --launch-immediately
usuario+   1003       1  0 nov21 ?       00:00:00 /usr/libexec/geoclue-2.0/demos/agent
usuario+   1008    1001  0 nov21 ?       00:00:01 /usr/bin/dbus-daemon
--config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 3
usuario+   1017     898  0 nov21 ?       00:00:02 /usr/bin/lxqt-globalkeysd
usuario+   1019     874  0 nov21 ?       00:00:00 /usr/libexec/gvfsd
usuario+   1020     898  0 nov21 ?       00:00:02 /usr/bin/lxqt-notificationd
usuario+   1025     874  0 nov21 ?       00:00:00 /usr/libexec/gvfsd-fuse /run/user/1000/gvfs -f -o
big_writes
usuario+   1032     898  0 nov21 ?       00:00:40 /usr/bin/lxqt-panel
usuario+   1033     898  0 nov21 ?       00:00:02 /usr/bin/lxqt-policykit-agent
usuario+   1036     898  0 nov21 ?       00:00:03 /usr/bin/lxqt-runner
usuario+   1038       1  0 nov21 ?       00:00:05 /usr/bin/xscreensaver -no-splash
usuario+   1041       1  0 nov21 ?       00:00:01 /usr/bin/python3
/usr/share/system-config-printer/applet.py
```

```
usuario+   1057      1  0 nov21 ?       00:00:00 /bin/sh
/usr/lib/lubuntu-update-notifier/lubuntu-upg-notifier.sh
usuario+   1093   1019  0 nov21 ?       00:00:00 /usr/libexec/gvfsd-trash --spawner :1.8
/org/gtk/gvfs/exec_spaw/0
usuario+   1107    874  0 nov21 ?       00:00:00 /usr/libexec/gvfs-udisks2-volume-monitor
usuario+   1113    874  0 nov21 ?       00:00:00 /usr/libexec/gvfs-mtp-volume-monitor
usuario+   1117    874  0 nov21 ?       00:00:00 /usr/libexec/gvfs-gphoto2-volume-monitor
usuario+   1121    874  0 nov21 ?       00:00:02 /usr/libexec/gvfs-afc-volume-monitor
usuario+   1128    874  0 nov21 ?       00:00:00 /usr/libexec/gvfs-goa-volume-monitor
usuario+   1153    898  0 nov21 ?       00:00:04 /usr/bin/lxqt-powermanagement
usuario+   1155      1  0 nov21 ?       00:00:04 /usr/bin/qlipper
usuario+   1157      1  0 nov21 ?       00:00:02 /usr/bin/nm-tray
usuario+   1395      1  0 nov21 ?       00:00:04 /usr/libexec/at-spi2-registryd --use-gnome-session
usuario+   1479    874  0 nov21 ?       00:00:00 /usr/bin/gnome-keyring-daemon --start
--foreground --components=secrets
usuario+   1499   1019  0 nov21 ?       00:00:00 /usr/libexec/gvfsd-network --spawner :1.8
/org/gtk/gvfs/exec_spaw/1
usuario+   1509    874  0 nov21 ?       00:00:00 /usr/libexec/dconf-service
usuario+   1518   1019  0 nov21 ?       00:00:00 /usr/libexec/gvfsd-dnssd --spawner :1.8
/org/gtk/gvfs/exec_spaw/3
usuario+   2775      1  0 nov21 ?       00:01:11 /usr/bin/qterminal
usuario+   2778   2775  0 nov21 pts/0    00:00:00 /bin/bash
usuario+   6415    874  0 04:10 ?       00:00:00 /usr/libexec/gvfsd-metadata
usuario+  10335   1057  0 10:55 ?       00:00:00 sleep 3600
usuario+  10337      1  0 10:55 ?       00:00:13 /usr/share/code/code --no-sandbox --unity-launch
usuario+  10339  10337  0 10:55 ?       00:00:00 /usr/share/code/code --type=zygote --no-sandbox
usuario+  10361  10337  0 10:55 ?       00:00:17 /usr/share/code/code --type=gpu-process
--field-trial-handle=6694135585693413256,15008579315086192752,131072
--disable-features=LayoutNG,
usuario+  10375  10337  0 10:55 ?       00:00:00 /usr/share/code/code --type=utility
--field-trial-handle=6694135585693413256,15008579315086192752,131072
--disable-features=LayoutNG,Pict
usuario+  10383  10337  3 10:55 ?       00:01:19 /usr/share/code/code --type=renderer
--disable-color-correct-rendering --no-sandbox
--field-trial-handle=6694135585693413256,150085793150
usuario+  10412  10383  0 10:56 pts/1    00:00:00 /bin/bash
usuario+  10414  10337  0 10:56 ?       00:00:02 /usr/share/code/code --type=renderer
--disable-color-correct-rendering --no-sandbox
--field-trial-handle=6694135585693413256,150085793150
usuario+  10429  10383  0 10:56 ?       00:00:09 /usr/share/code/code --inspect-port=0
/usr/share/code/resources/app/out/bootstrap-fork --type=extensionHost
usuario+  10455  10383  0 10:56 ?       00:00:00 /usr/share/code/code
/usr/share/code/resources/app/out/bootstrap-fork --type=watcherService
usuario+  10481  10429  0 10:56 ?       00:00:05
/home/usuarioso/.vscode/extensions/ms-vscode.cpptools-1.0.1/bin/cpptools
usuario+  10689  10481  0 11:32 ?       00:00:00
/home/usuarioso/.vscode/extensions/ms-vscode.cpptools-1.0.1/bin/cpptools-srv 10481 1
usuario+  10729   2778  0 11:39 pts/0    00:00:00 ps -u usuarioso -f
```

$ ps -eo pid,gid,sid,s,command : Mostrar los procesos del sistema, incluyendo PID, del grupo, la sesión, el estado y la línea de comandos

```
  PID  GID    SID S COMMAND
    1    0      1 S /sbin/init splash
    2    0      0 S [kthreadd]
    3    0      0 I [rcu_gp]
    4    0      0 I [rcu_par_gp]
    6    0      0 I [kworker/0:0H-kblockd]
```

```
    9    0      0 I [mm_percpu_wq]
   10    0      0 S [ksoftirqd/0]
   11    0      0 R [rcu_sched]
   12    0      0 S [migration/0]
   13    0      0 S [idle_inject/0]
   14    0      0 S [cpuhp/0]
   15    0      0 S [kdevtmpfs]
   16    0      0 I [netns]
   17    0      0 S [rcu_tasks_kthre]
   18    0      0 S [kauditd]
   19    0      0 S [khungtaskd]
   20    0      0 S [oom_reaper]
   21    0      0 I [writeback]
   22    0      0 S [kcompactd0]
   23    0      0 S [ksmd]
   24    0      0 S [khugepaged]
   70    0      0 I [kintegrityd]
   71    0      0 I [kblockd]
   72    0      0 I [blkcg_punt_bio]
   73    0      0 I [tpm_dev_wq]
   74    0      0 I [ata_sff]
   75    0      0 I [md]
   76    0      0 I [edac-poller]
   77    0      0 I [devfreq_wq]
   78    0      0 S [watchdogd]
   81    0      0 S [kswapd0]
   82    0      0 S [ecryptfs-kthrea]
   84    0      0 I [kthrotld]
   85    0      0 I [acpi_thermal_pm]
   86    0      0 S [scsi_eh_0]
   87    0      0 I [scsi_tmf_0]
   88    0      0 S [scsi_eh_1]
   89    0      0 I [scsi_tmf_1]
   91    0      0 I [vfio-irqfd-clea]
   93    0      0 I [ipv6_addrconf]
  102    0      0 I [kstrp]
  105    0      0 I [kworker/u3:0]
  118    0      0 I [charger_manager]
  119    0      0 I [kworker/0:1H-kblockd]
  159    0      0 S [scsi_eh_2]
  161    0      0 I [scsi_tmf_2]
  163    0      0 I [cryptd]
  169    0      0 S [irq/18-vmwgfx]
  171    0      0 I [ttm_swap]
  236    0      0 S [jbd2/sda1-8]
  237    0      0 I [ext4-rsv-conver]
  281    0    281 S /lib/systemd/systemd-journald
  314    0    314 S /lib/systemd/systemd-udevd
  359  103    359 S /lib/systemd/systemd-resolved
  363    0    363 S /usr/sbin/haveged --Foreground --verbose=1 -w 1024
  370    0    370 S /usr/lib/accountsservice/accounts-daemon
  371    0    371 S /usr/sbin/acpid
  375  120    375 S avahi-daemon: running [usuarioso-virtualbox.local]
  377    0    377 S /usr/sbin/cron -f
  379    0    379 S /usr/sbin/cupsd -l
  380  106    380 S /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile
--systemd-activation --syslog-only
```

```
  381    0    381 S /usr/sbin/NetworkManager --no-daemon
  383    0    383 S /usr/sbin/dundee -n
  392    0    392 S /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
  395    0    395 S /usr/sbin/ofonod -n
  398    0    398 S /usr/lib/policykit-1/polkitd --no-debug
  409  110    409 S /usr/sbin/rsyslogd -n -iNONE
  412    0    412 S /lib/systemd/systemd-logind
  416    0    416 S /usr/lib/udisks2/udisksd
  423    0    423 S /sbin/wpa_supplicant -u -s -O /run/wpa_supplicant
  435  120    375 S avahi-daemon: chroot helper
  456    7    379 S /usr/lib/cups/notifier/dbus dbus://
  461    0    461 S /usr/sbin/cups-browsed
  475    0      0 I [iprt-VBoxWQueue]
  476    0    476 S /usr/sbin/ModemManager --filter-policy=strict
  507    0    506 S /usr/sbin/VBoxService
  522    0    522 S /usr/bin/python3
/usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal
  556  123    556 S /usr/bin/whoopsie -f
  561    4    561 S /usr/sbin/kerneloops --test
  563    4    563 S /usr/sbin/kerneloops
  790    0    790 S /usr/bin/sddm
  808    0    808 R /usr/lib/xorg/Xorg -nolisten tcp -auth
/var/run/sddm/{772ad5eb-6b15-4743-ab21-282b6a144b1f} -background none -noreset -displayfd
17 -seat seat0 vt1
  848  116    848 S /usr/libexec/rtkit-daemon
  854    0    854 S /usr/lib/bluetooth/bluetoothd
  858    0    790 S /usr/lib/x86_64-linux-gnu/sddm/sddm-helper --socket
/tmp/sddm-authafb6cbd3-fcbb-4300-9968-abd8384819b7 --id 1 --start env
LXQT_DEFAULT_OPENBOX_CONFIG="/etc/xdg/xdg-L
  874 1001    874 S /lib/systemd/systemd --user
  875 1001    874 S (sd-pam)
  892 1001    892 S /usr/bin/pulseaudio --daemonize=no --log-target=journal
  895 1001    894 S /usr/bin/gnome-keyring-daemon --daemonize --login
  898 1001    790 S lxqt-session
  914 1001    914 S /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfile
--systemd-activation --syslog-only
  949 1001    948 S /usr/bin/VBoxClient --clipboard
  950 1001    948 S /usr/bin/VBoxClient --clipboard
  961 1001    960 S /usr/bin/VBoxClient --seamless
  962 1001    960 S /usr/bin/VBoxClient --seamless
  968 1001    967 S /usr/bin/VBoxClient --draganddrop
  969 1001    967 S /usr/bin/VBoxClient --draganddrop
  976 1001    975 S /usr/bin/VBoxClient --vmsvga
  977 1001    975 S /usr/bin/VBoxClient --vmsvga
  980 1001    980 S /usr/bin/ssh-agent /usr/bin/im-launch env
LXQT_DEFAULT_OPENBOX_CONFIG=/etc/xdg/xdg-Lubuntu/openbox/lxqt-rc.xml
/usr/bin/startlxqt
  998 1001    790 S /usr/bin/openbox --config-file /home/usuarioso/.config/openbox/lxqt-rc.xml
 1001 1001   1000 S /usr/libexec/at-spi-bus-launcher --launch-immediately
 1003 1001   1002 S /usr/libexec/geoclue-2.0/demos/agent
 1008 1001   1000 S /usr/bin/dbus-daemon
--config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 3
 1017 1001    790 S /usr/bin/lxqt-globalkeysd
 1019 1001   1019 S /usr/libexec/gvfsd
 1020 1001    790 S /usr/bin/lxqt-notificationd
 1025 1001   1019 S /usr/libexec/gvfsd-fuse /run/user/1000/gvfs -f -o big_writes
 1032 1001    790 S /usr/bin/lxqt-panel
```

```
  1033  1001    790 S /usr/bin/lxqt-policykit-agent
  1036  1001    790 S /usr/bin/lxqt-runner
  1038  1001   1037 S /usr/bin/xscreensaver -no-splash
  1041  1001   1040 S /usr/bin/python3 /usr/share/system-config-printer/applet.py
  1057  1001   1056 S /bin/sh /usr/lib/lubuntu-update-notifier/lubuntu-upg-notifier.sh
  1093  1001   1019 S /usr/libexec/gvfsd-trash --spawner :1.8 /org/gtk/gvfs/exec_spaw/0
  1107  1001   1107 S /usr/libexec/gvfs-udisks2-volume-monitor
  1113  1001   1113 S /usr/libexec/gvfs-mtp-volume-monitor
  1117  1001   1117 S /usr/libexec/gvfs-gphoto2-volume-monitor
  1121  1001   1121 S /usr/libexec/gvfs-afc-volume-monitor
  1128  1001   1128 S /usr/libexec/gvfs-goa-volume-monitor
  1142     0   1142 S /usr/lib/upower/upowerd
  1153  1001    790 S /usr/bin/lxqt-powermanagement
  1155  1001   1154 S /usr/bin/qlipper
  1157  1001   1156 S /usr/bin/nm-tray
  1395  1001   1000 S /usr/libexec/at-spi2-registryd --use-gnome-session
  1479  1001    914 S /usr/bin/gnome-keyring-daemon --start --foreground --components=secrets
  1499  1001   1019 S /usr/libexec/gvfsd-network --spawner :1.8 /org/gtk/gvfs/exec_spaw/1
  1509  1001    914 S /usr/libexec/dconf-service
  1518  1001   1019 S /usr/libexec/gvfsd-dnssd --spawner :1.8 /org/gtk/gvfs/exec_spaw/3
  2775  1001   2774 R /usr/bin/qterminal
  2778  1001   2778 S /bin/bash
  6415  1001   6415 S /usr/libexec/gvfsd-metadata
  9459     0      0 I [kworker/0:1-events]
  9473     0   9473 S /usr/libexec/fwupd/fwupd
  9882     0      0 R [kworker/u2:0-events_unbound]
 10225     0      0 I [kworker/0:2-events]
 10335  1001   1056 S sleep 3600
 10337  1001  10336 S /usr/share/code/code --no-sandbox --unity-launch
 10339  1001  10336 S /usr/share/code/code --type=zygote --no-sandbox
 10361  1001  10336 S /usr/share/code/code --type=gpu-process
--field-trial-handle=6694135585693413256,15008579315086192752,131072
--disable-features=LayoutNG,PictureInPicture,SpareRender
 10375  1001  10336 S /usr/share/code/code --type=utility
--field-trial-handle=6694135585693413256,15008579315086192752,131072
--disable-features=LayoutNG,PictureInPicture,SpareRendererFo
 10383  1001  10336 S /usr/share/code/code --type=renderer --disable-color-correct-rendering
--no-sandbox --field-trial-handle=6694135585693413256,15008579315086192752,131072
--disable-fe
 10412  1001  10412 S /bin/bash
 10414  1001  10336 S /usr/share/code/code --type=renderer --disable-color-correct-rendering
--no-sandbox --field-trial-handle=6694135585693413256,15008579315086192752,131072
--disable-fe
 10429  1001  10336 S /usr/share/code/code --inspect-port=0
/usr/share/code/resources/app/out/bootstrap-fork --type=extensionHost
 10455  1001  10336 S /usr/share/code/code /usr/share/code/resources/app/out/bootstrap-fork
--type=watcherService
 10481  1001  10336 S
/home/usuarioso/.vscode/extensions/ms-vscode.cpptools-1.0.1/bin/cpptools
 10649     0      0 R [kworker/u2:1+events_unbound]
 10689  1001  10336 S
/home/usuarioso/.vscode/extensions/ms-vscode.cpptools-1.0.1/bin/cpptools-srv 10481 1
 10701     0      0 I [kworker/u2:3-flush-8:0]
 10730  1001   2778 R ps -eo pid,gid,sid,s,command
```

Comparten el GID (1001). El SID del nuevo proceso es el PID(y el SID) de la shell.

Ejercicio 5. Escribir un programa que muestre los identificadores del proceso: identificador de proceso, de proceso padre, de grupo de procesos y de sesión. Mostrar además el número máximo de ficheros que puede abrir el proceso y el directorio de trabajo actual.

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

int main() {
   printf("PID: %i\n", getpid());
   printf("PPID: %i\n", getppid());
   printf("GID: %i\n", getpgid(getpid()));
   printf("SID: %i\n", getsid(getpid()));

   struct rlimit param;
   if (getrlimit(RLIMIT_NOFILE, &param) == -1) {
      perror("ERROR RLIMIT");
      return -1;
   }
   printf("MAX FICHEROS: %ld\n", param.rlim_max);

   char *path = (char*)malloc(sizeof(char)*(4096 + 1));
   getcwd(path, 4096 + 1);
   printf("DIRECTORIO: %s\n", path);
   free (path);

   return 0;
}
```

$ ./ej5
PID: 11050
PPID: 2778
GID: 11050
SID: 2778
MAX FICHEROS: 1048576
DIRECTORIO: /home/usuarioso/Desktop/ASOR/PRACTICA 3

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 ejecutar la lógica del servicio y crear la nueva sesión. 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`.

¿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`)?

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

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

void mostrarAtributos(char *proceso) {
   printf("[%s] PID: %i\n", proceso, getpid());
   printf("[%s] PPID: %i\n", proceso, getppid());
   printf("[%s] GID: %i\n", proceso, getpgid(getpid()));
   printf("[%s] SID: %i\n", proceso, getsid(getpid()));

   struct rlimit param;
   getrlimit(RLIMIT_NOFILE, &param);
   printf("[%s] MAX FICHEROS: %ld\n", proceso, param.rlim_max);

   char *path = (char*)malloc(sizeof(char)*(4096 + 1));
   getcwd(path, 4096 + 1);
   printf("[%s] DIRECTORIO: %s\n", proceso, path);
   free (path);
}

int main(){
   pid_t pid = fork();
   switch(pid){
      case -1: //Error
         perror("fork");
         exit(-1);
      case 0: //Hijo
         chdir("/tmp");
         setsid();
         //sleep(10); //1
         printf("Proceso Hijo %i -> Padre: %i\n",getpid(),getppid());
         mostrarAtributos("Hijo");
         exit(0);
      default:
         sleep(10); //2
         printf("Proceso Padre %i\n",getpid());
         mostrarAtributos("Padre");
         exit(0);
         break;
   }

   return 0;
}

//Acaba antes el padre descomentando 1
Proceso Padre 11137
[Padre] PID: 11137
[Padre] PPID: 2778
[Padre] GID: 11137
[Padre] SID: 2778
[Padre] MAX FICHEROS: 1048576
[Padre] DIRECTORIO: /home/usuarioso/Desktop/ASOR/PRACTICA 3
Proceso Hijo 11138 -> Padre: 1
```

```
[Hijo] PID: 11138
[Hijo] PPID: 1
[Hijo] GID: 11138
[Hijo] SID: 11138
[Hijo] MAX FICHEROS: 1048576
[Hijo] DIRECTORIO: /tmp

//Acaba antes el hijo descomentando 2
Proceso Hijo 11163 -> Padre: 11162
[Hijo] PID: 11163
[Hijo] PPID: 11162
[Hijo] GID: 11163
[Hijo] SID: 11163
[Hijo] MAX FICHEROS: 1048576
[Hijo] DIRECTORIO: /tmp
Proceso Padre 11162 -> Padre: 2778
[Padre] PID: 11162
[Padre] PPID: 2778
[Padre] GID: 11162
[Padre] SID: 2778
[Padre] MAX FICHEROS: 1048576
[Padre] DIRECTORIO: /home/usuarioso/Desktop/ASOR/PRACTICA 3

------

Si el padre termina antes el hijo se queda huérfano y el ppid lo recoge la shell o init
Si el hijo termina antes
```

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?

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"`?

CON EXEC:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv){
   if (argc < 2) {
      printf("ERROR: Tienes que pasar el comando.\n");
      return -1;
   }

   //Ejecutamos el comando correspondiente
   if (execvp(argv[1], argv + 1) == -1) {
```

```
    printf("ERROR: Ejecución incorrecto\n");
  }
  printf("El comando terminó de ejecutarse.\n");

  return 0;
}
```

CON SYSTEM

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

int main(int argc, char **argv){
  if (argc < 2) {
    printf("ERROR: Tienes que pasar el comando.\n");
    return -1;
  }

  int cmdSize = 1;
  for (int i = 1; i < argc; i++)
    cmdSize += strlen(argv[i]) + 1;
  char *cmd = (char *)malloc(sizeof(char)*cmdSize);
  strcpy(cmd, "");

  //Concatenamos los argumentos
  for (int i = 1; i < argc; i++) {
    strcat(cmd, argv[i]);
    strcat(cmd, " ");
  }

  //Ejecutamos el comando correspondiente
  if (system(cmd) == -1) {
    printf("ERROR: Ejecución incorrecta.\n");
  }
  printf("El comando terminó de ejecutarse.\n");

  return 0;
}
```

La cadena solo se imprime cuando se usa system porque al ejecutar exec sustituye la imagen del programa a la imagen del programa que hemos pasado por argumento

Si pasamos ps -el estamos pasando dos argumentos y si se quiere ejecutar como system habría que unirlos. Si pasamos "ps -el" estamos pasando un único string y nos permitie ejecutar directamente el comando system sin unirlo

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

int main(int argc, char **argv){
   if (argc < 2) {
      printf("ERROR: Tienes que pasar el comando.\n");
      return -1;
   }

   pid_t pid = fork();

   switch(pid){
      case -1: //Error
         perror("fork");
         exit(-1);

      case 0: //Hijo
         setsid(); //Creamos una nueva sesión
         chdir("/tmp");

         int fdout = open("/tmp/daemon.out",O_CREAT | O_RDWR, 0777);
         int fderr = open("/tmp/daemon.err", O_CREAT | O_RDWR, 0777);
         int fdnull = open("/dev/null", O_CREAT | O_RDWR, 0777);
         int fd2 = dup2(fdout, 1);
         int fd3 = dup2(fderr, 2);
         int fd4 = dup2(fdnull, 0);

         //Ejecutamos el comando correspondiente
         if (execvp(argv[1], argv + 1) == -1) {
            printf("ERROR: Ejecución incorrecta\n");
            exit(-1);
         }
         printf("El comando terminó de ejecutarse.\n");
         break;
   }

   return 0;
}
```

```
$ ./ej8 ps -el
$ cat /tmp/daemon.out
F S   UID    PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY        TIME CMD
4 S     0      1       0  0  80   0 - 25472 -     ?      00:00:03 systemd
1 S     0      2       0  0  80   0 -     0 -     ?      00:00:00 kthreadd
1 I     0      3       2  0  60 -20 -     0 -     ?      00:00:00 rcu_gp
……..
```

Señales

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

KILL

NAME
kill - send a signal to a process

SYNOPSIS
kill [options] <pid> [...]

DESCRIPTION
The default signal for kill is TERM. Use -l or -L to list available signals. Particularly useful signals include HUP, INT, KILL, STOP, CONT, and 0. Alternate signals may be
specified in three ways: -9, -SIGKILL or -KILL. Negative PID values may be used to choose whole process groups; see the PGID column in ps command output. A PID of -1 is spe-
cial; it indicates all processes except the kill process itself and init.

OPTIONS
<pid> [...]
Send signal to every <pid> listed.
-<signal>
-s <signal>
--signal <signal>
Specify the signal to be sent. The signal can be specified by using name or number. The behavior of signals is explained in signal(7) manual page.
-l, --list [signal]
List signal names. This option has optional argument, which will convert signal number to signal name, or other way round.
-L, --table
List signal names in a nice table.

PKILL

NAME
pgrep, pkill - look up or signal processes based on name and other attributes

SYNOPSIS
pgrep [options] pattern
pkill [options] pattern

DESCRIPTION

pgrep looks through the currently running processes and lists the process IDs which match the selection criteria to stdout. All the criteria have to match. For example,

$ pgrep -u root sshd

will only list the processes called sshd AND owned by root. On the other hand,

$ pgrep -u root,daemon

will list the processes owned by root OR daemon.

pkill will send the specified signal (by default SIGTERM) to each process instead of listing them on stdout.

OPTIONS

-signal
--signal signal
Defines the signal to send to each matched process. Either the numeric or the symbolic signal name can be used. (pkill only.)

-c, --count
Suppress normal output; instead print a count of matching processes. When count does not match anything, e.g. returns zero, the command will return non-zero value.

-d, --delimiter delimiter
Sets the string used to delimit each process ID in the output (by default a newline). (pgrep only.)

-f, --full
The pattern is normally only matched against the process name. When -f is set, the full command line is used.

-g, --pgroup pgrp,...
Only match processes in the process group IDs listed. Process group 0 is translated into pgrep's or pkill's own process group.

-G, --group gid,...
Only match processes whose real group ID is listed. Either the numerical or symbolical value may be used.

-i, --ignore-case
Match processes case-insensitively.

-l, --list-name
List the process name as well as the process ID. (pgrep only.)

-a, --list-full
List the full command line as well as the process ID. (pgrep only.)

-n, --newest
Select only the newest (most recently started) of the matching processes.

-o, --oldest
Select only the oldest (least recently started) of the matching processes.

-P, --parent ppid,...
Only match processes whose parent process ID is listed.

-s, --session sid,...
Only match processes whose process session ID is listed. Session ID 0 is translated into pgrep's or pkill's own session ID.

-t, --terminal term,...
Only match processes whose controlling terminal is listed. The terminal name should be specified without the "/dev/" prefix.

-u, --euid euid,...
Only match processes whose effective user ID is listed. Either the numerical or symbolical value may be used.

-U, --uid uid,...
Only match processes whose real user ID is listed. Either the numerical or symbolical

value may be used.
-v, --inverse
Negates the matching. This option is usually used in pgrep's context. In pkill's context the short option is disabled to avoid accidental usage of the option.
-w, --lightweight
Shows all thread ids instead of pids in pgrep's context. In pkill's context this option is disabled.
-x, --exact
Only match processes whose names (or command line if -f is specified) exactly match the pattern.
-F, --pidfile file
Read PID's from file. This option is perhaps more useful for pkill than pgrep.
-L, --logpidfile
Fail if pidfile (see -F) not locked.
-r, --runstates D,R,S,Z,...
Match only processes which match the process state.
--ns pid
Match processes that belong to the same namespaces. Required to run as root to match processes from other users. See --nslist for how to limit which namespaces to match.
--nslist name,...
Match only the provided namespaces. Available namespaces: ipc, mnt, net, pid, user,uts.

```
$kill -s  #Manda una señal: sustituir s por un número según el tipo de señal
$kill -l  #Lista todas las señales
```

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:
- *SIGKILL (9): Terminación brusca*

$ kill -9 11988
- *SIGTRAP (5): Ejecución paso a paso, enviada después de cada instrucción*

$ kill -5 11990
- *SIGHUP (1): Desconexión de terminal*

$ kill -1 12080
- *SIGSTOP (19): Parar proceso. No se puede capturar, bloquear o ignorar*

$ kill -19 12080
- *SIGINT(2): Interrupción del proceso*

$ kill -2 12084

TERMINAL 2:
$ sleep 600
Terminado (killed)
$ sleep 600
«trap» para punto de parada/seguimiento (`core' generado)
$ sleep 600
Colgar (hangup)
$ sleep 600
[1]+ Detenido sleep 600

$ 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).

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


int main() {
   sigset_t set;
   //Inicializamos un conjunto de señales y añadimos las señales int y tstp
   sigemptyset(&set);
   sigaddset(&set, SIGINT);
   sigaddset(&set, SIGTSTP);
   //Bloqueamos las señales
   sigprocmask(SIG_BLOCK, &set, NULL);

   /*NO FUNCIONA: da violación de segmento
   //Variable de entorno
   char* sleep_secs = getenv("SLEEP_SECS");
   int secs = atoi(sleep_secs);
   */

   int secs = 10;
   printf("Durmiendo %d segundo(s)\n", secs);
   //Dormimos el proceso
   sleep(secs);

   //Comprobamos las señales pendientes
   sigset_t pend;
   sigpending(&pend);

   if (sigismember(&pend, SIGINT) == 1) {
      printf("Recibida la señal SIGINT\n");
   }
   if (sigismember(&pend, SIGTSTP) == 1) {
      printf("Recibida la señal SIGTSTP\n");
   }
   sigprocmask(SIG_UNBLOCK, &set, NULL);

   printf("Cadena de prueba\n");
```

```
    return 0;
}

$ ./ej11
Durmiendo 10 segundo(s)
Cadena de prueba
```

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

volatile int count_INT = 0;
volatile int count_TSTP = 0;

//Modificamos el modificador para que haga lo que queremos cuando recibe una señal: en este caso queremos ir sumando al contador
void handler(int signal){
    if (signal == SIGINT){
        count_INT++;
    }
    if (signal == SIGTSTP){
        count_TSTP++;
    }
}

int main(){
    struct sigaction act;
    sigaction(SIGINT, NULL, &act);
    act.sa_handler = handler;
    sigaction(SIGINT, &act, NULL);
    sigaction(SIGTSTP, NULL, &act);
    act.sa_handler = handler;
    sigaction(SIGTSTP, &act, NULL);

    sigset_t set;
    sigemptyset(&set);

    while (count_INT + count_TSTP < 10){
        sigsuspend(&set); //Espera a recibir una señal fuera conjunto set (que esta vacio)
        printf("Señal capturada\n");
    }
```

```
   printf("Señales SIGINT capturadas: %i\n", count_INT);
   printf("Señales SIGTSTP capturadas: %i\n", count_TSTP);

   return 0;
}
```

TERMINAL 1:

```
$ ./ej12
Señales SIGINT capturadas: 10
Señales SIGTSTP capturadas: 1
```

TERMINAL 2: Señal 2 es SIGINT, Señal 19 es SIGTSTOP

```
$ kill -2 12482
$ kill -2 12482
$ kill -2 12482
$ kill -2 12482
$ kill -2 12482
$ kill -2 12482
$ kill -2 12482
$ kill -19 12482
$ kill -2 12482
$ kill -2 12482
```

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`.

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

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

volatile int SIGUSR1_received = 0;

void handler(int signal){
  if (signal == SIGUSR1){
      SIGUSR1_received = 1;
  }
}

int main(int argc, char **argv) {
            if (argc != 2) {
      printf("ERROR: Tienes que pasar los segundos!\n");
      return -1;
   }
```

```
   sigset_t set;
   sigemptyset(&set);
   sigaddset(&set, SIGUSR1);
   sigprocmask(SIG_UNBLOCK, &set, NULL);

   struct sigaction act;
   sigaction(SIGUSR1, NULL, &act);
   act.sa_handler = handler;
   sigaction(SIGUSR1, &act, NULL);

   int secs = atoi(argv[1]);
   sleep(secs);

   if (SIGUSR1_received == 0) {
      unlink(argv[0]);
      printf("Fichero borrado!!\n");
   } else {
      printf("No se ha borrado el fichero");
   }

   return 0;
}

 -------

$ ./ej13 10
Fichero borrado!!
```