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](#_14a3ftqman5y)

[Políticas de planificación](#_rk750rh0zbua)

[Grupos de procesos y sesiones](#_9jpm36obra1c)

[Ejecución de programas](#_oii4am5x6pl1)

[Señales](#_6kob7pg9jkqo)

# 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 de 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.

|  |
| --- |
| [cursoredes@localhost Prac3]$ nice  0  [cursoredes@localhost Prac3]$ renice  Usage:  renice [-n] <priority> [-p|--pid] <pid>...  renice [-n] <priority> -g|--pgrp <pgid>...  renice [-n] <priority> -u|--user <user>...  Options:  -g, --pgrp <id> interpret argument as process group ID  -n, --priority <num> specify the nice increment value  -p, --pid <id> interpret argument as process ID (default)  -u, --user <name|id> interpret argument as username or user ID  -h, --help display help text and exit  -V, --version display version information and exit  [cursoredes@localhost Prac3]$ sudo nice --10 bash  [root@localhost Prac3]# nice  -10  [root@localhost Prac3]# ps  PID TTY TIME CMD  12096 pts/0 00:00:00 sudo  12097 pts/0 00:00:00 bash  12348 pts/0 00:00:00 ps  [root@localhost Prac3]# chrt -f -p 12 12097  [root@localhost Prac3]# chrt -p 12097  pid 12097's current scheduling policy: SCHED\_FIFO  pid 12097's current scheduling priority: 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 <stdlib.h>  #include <string.h>  #include <stdio.h>  #include <sched.h>  int main(int argc, char \*\*argv){  struct sched\_param p;  int pid = atoi(argv[1]); //atoi() convierte a integer  int options = sched\_getscheduler(pid);  switch(options){  case SCHED\_FIFO:  printf("FIFO\n");  break;  case SCHED\_RR:  printf("RR\n");  break;  case SCHED\_OTHER:  printf("OTHER\n");  break;  }  sched\_getparam(pid,&p);  printf("Prioridad del proceso actual: %i\n", p.sched\_priority);  printf("Max priority %i\n", sched\_get\_priority\_max(pid));  printf("Min priority %i\n",sched\_get\_priority\_min(pid));  return 0;  }  [root@localhost Prac3]# gcc -Wall -g ej2.c -o ej2  [root@localhost Prac3]# ./ej2 bash  FIFO  Prioridad del proceso actual: 12  Max priority 0  Min priority 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?

|  |
| --- |
| [root@localhost Prac3]# gcc -Wall -g ej2.c -o ej2  [root@localhost Prac3]# ./ej2 bash  FIFO  Prioridad del proceso actual: 12  Max priority 0  Min priority 0  [root@localhost Prac3]# chrt --fifo 12 ./ej2 bash  FIFO  Prioridad del proceso actual: 12  Max priority 0  Min priority 0  La prioridad del programa es 12. Se heredan los atributos de planificación,  ya que esto se puede observar en los valores mínimo y máximo. |

# 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?

|  |
| --- |
| [root@localhost Prac3]# ps -l  F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD  4 S 0 12096 11769 0 80 0 - 55154 poll\_s pts/0 00:00:00 sudo  4 S 0 12097 12096 0 47 - - 29151 do\_wai pts/0 00:00:00 bash  0 R 0 13205 12097 0 47 - - 38300 - pts/0 00:00:00 ps  [root@localhost Prac3]# ps -o pid,gid,sid,state,comm  PID GID SID S COMMAND  12096 0 11769 S sudo  12097 0 11769 S bash  13196 0 11769 R ps  Se comparte el identificador de grupo y el identificador de sesión de la shell (bash) y ps. Se crea el GID 1000 al crear un nuevo proceso. |

***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 archivos que puede abrir el proceso y el directorio de trabajo actual.

|  |
| --- |
| #include <stdlib.h>  #include <string.h>  #include <stdio.h>  #include <unistd.h>  #include <sys/resource.h>  #include <sys/time.h>  int main(int argc, char \*\*argv){  struct rlimit limit;  printf("PID(identificador de proceso): %i\n", getpid());  printf("PPID(identificador de proceso padre): %i\n", getppid());  printf("PGID(identificador de proceso de grupo de procesos): %i\n", getpgid(getpid()));  printf("SID(identificador de sesión): %i\n", getsid(getpid()));  char \*buf = malloc(sizeof(char)\*2048);  char \*DirActual = getcwd(buf,sizeof(char)\*2048);    printf("Directoria de trabajo actual: %s\n", DirActual);  printf("Valor máximo: %li\n", limit.rlim\_max);  free(buf);  return 0;  }  [root@localhost Prac3]# ./ej5 bash  PID(identificador de proceso): 13614  PPID(identificador de proceso padre): 12097  PGID(identificador de proceso de grupo de procesos): 13614  SID(identificador de sesión): 11769  Directoria de trabajo actual: /home/cursoredes/Documents/Prac3  Límite actual: 4196368  Valor máximo: 4195888 |

***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.

|  |
| --- |
| #include <stdlib.h>  #include <string.h>  #include <stdio.h>  #include <unistd.h>  #include <sys/resource.h>  #include <sys/time.h>  void getInfo() {  struct rlimit limit;  printf("PID(identificador de proceso): %i\n", getpid());  printf("PPID(identificador de proceso padre): %i\n", getppid());  printf("PGID(identificador de proceso de grupo de procesos): %i\n", getpgid(getpid()));  printf("SID(identificador de sesión): %i\n", getsid(getpid()));  char \*buf = malloc(sizeof(char)\*2048);  char \*DirActual = getcwd(buf,sizeof(char)\*2048);    printf("Directoria de trabajo actual: %s\n", DirActual);  printf("Valor máximo: %li\n", limit.rlim\_max);  free(buf);  }  int main(int argc, char \*\*argv){  pid\_t pid = fork(); //Creación de un proceso  if(pid == 0){  chdir("/tmp");  printf("Hijo\n");  getInfo();  }  else if(pid > 0){  sleep(3);  pid\_t mi\_sid = setsid(); //Creamos una nueva sesión  printf("Padre\n");  getInfo();  }  else {  printf("Ha habido un error!");  return -1;  }  return 0;  }  [root@localhost Prac3]# gcc -Wall -g ej6.c -o ej6  [root@localhost Prac3]# ./ej6 bash  Hijo  PID(identificador de proceso): 13912  PPID(identificador de proceso padre): 13911  PGID(identificador de proceso de grupo de procesos): 13911  SID(identificador de sesión): 11769  Directoria de trabajo actual: /tmp  Valor máximo: 0  Padre  PID(identificador de proceso): 13911  PPID(identificador de proceso padre): 12097  PGID(identificador de proceso de grupo de procesos): 13911  SID(identificador de sesión): 11769  Directoria de trabajo actual: /home/cursoredes/Documents/Prac3  Valor máximo: 0   * **¿Qué sucede si el proceso padre termina antes que el hijo (observar el PPID del proceso hijo)?**   Si el proceso padre termina antes, el hijo se queda huérfano y lo acoge init   * **¿Y si el proceso que termina antes es el hijo (observar el estado del proceso hijo con ps)?**   Como el hijo termina antes, no se queda huerfano.  estado del proceso puede ser:  Código significado  D Uninterruptible sleep (usualmente IO)  R Ejecutándose(running) o en cola de ejecución  S Interruptible sleep (p.e. esperando un evento)  T Detenido  Z Proceso zombie |

# 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 ./ejecuta ps -ely ./ejecuta “ps -el”?

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  int main( int argc, char \*\*argv){  //execvp(argv[1],argv + 1);  system(argv[1]);  printf("El comando terminó de ejecutarse \n");  return 0;  }  Solo con system se ejecuta al final “El comando terminó de ejecutarse”, ya que con execvp se crea una imagen de proceso y sustituye lo que haya debajo de él por el comando que ha sido insertado. |

***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*.

|  |
| --- |
| [root@localhost Prac3]# gcc -Wall -g ej8.c -o ej8  ej8.c: In function ‘main’:  ej8.c:35:9: warning: unused variable ‘mi\_sid’ [-Wunused-variable]  pid\_t mi\_sid = setsid(); //Creamos una nueva sesión  ^  [root@localhost Prac3]# ./ej8 bash  Hijo  PID(identificador de proceso): 14482  PPID(identificador de proceso padre): 14481  PGID(identificador de proceso de grupo de procesos): 14481  SID(identificador de sesión): 11769  Directoria de trabajo actual: /tmp  Valor máximo: 0  Padre  PID(identificador de proceso): 14481  PPID(identificador de proceso padre): 12097  PGID(identificador de proceso de grupo de procesos): 14481  SID(identificador de sesión): 11769  Directoria de trabajo actual: /home/cursoredes/Documents/Prac3  Valor máximo: 0  [root@localhost Prac3]# cd /tmp/  [root@localhost tmp]# ls  **daemon.err**  **daemon.out**  ssh-FeWkpV6Vkzba  systemd-private-e6f29b71632640868f06f1a46699b950-chronyd.service-CdRTyY  systemd-private-e6f29b71632640868f06f1a46699b950-cups.service-Y3unGB  systemd-private-e6f29b71632640868f06f1a46699b950-rtkit-daemon.service-atFcqC  [root@localhost tmp]# cd /dev/  [root@localhost dev]# ls  cpu\_dma\_latency network\_latency tty1 tty34 tty59 vcsa  crash network\_throughput tty10 tty35 tty6 vcsa1  disk **null**  tty11 tty36 tty60 vcsa2  kmsg sda2 tty23 tty48 urandom |

# 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.

|  |
| --- |
| NAME  kill - terminate a process  kill [-s signal|-p] [-q sigval] [-a] [--] pid...  [root@localhost dev]# kill -l (obtienes la lista de 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  [cursoredes@localhost Prac3]$ ps  PID TTY TIME CMD  15009 pts/0 00:00:00 bash  15112 pts/0 00:00:00 ps  [cursoredes@localhost Prac3]$ sleep 600  Terminal 2  [cursoredes@localhost Prac3]$ ps -aux  cursore+ 15285 0.0 0.0 107948 352 pts/0 S+ 19:45 0:00 sleep 600  [cursoredes@localhost Prac3]$ sudo kill -SIGABRT 15009 |

***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.

Después de despertar de sleep(3), 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 <stdio.h>  #include <stdlib.h>  #include <signal.h>  #include <unistd.h>  int main(int argc, char \*\*argv){  sigset\_t blk\_set;  sigemptyset(&blk\_set);  sigaddset(&blk\_set, SIGINT);  sigaddset(&blk\_set, SIGTSTP);  //ACTIVACIÓN DE CERROJO  sigprocmask(SIG\_BLOCK, &blk\_set, NULL);    char \*sleep\_sec = getenv("SLEEP\_SECS");  int s = atoi(sleep\_sec);  sleep(s);    //DESACTIVACIÓN DE CERROJO  sigprocmask(SIG\_UNBLOCK, &blk\_set, NULL);    printf("La señal SIGINT ha sido capturada: %i\n", SIGINT);  printf("La señal SIGTSTP ha sido capturada: %i\n", SIGTSTP);  return 0;  }  Terminal 1  [cursoredes@localhost Prac3]$ export SLEEP\_SECS="3"  [cursoredes@localhost Prac3]$ gcc -Wall -g ej11.c -o ej11  [cursoredes@localhost Prac3]$ ./ej11  La señal SIGINT ha sido capturada: 2  La señal SIGTSTP ha sido capturada: 20  Terminal 2  [cursoredes@localhost Prac3]$ sudo kill -SIGINT 14859 |

***Ejercicio 12.*** Escribir un programa que instale un manejador sencillo 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 <string.h>  #include <stdlib.h>  #include <stdio.h>  #include <signal.h>  volatile int contarSIGINT = 0;  volatile int contarSIGSTP = 0;  void contar(int state){  if(state == SIGINT)  contarSIGINT++;  if (state == SIGTSTP)  contarSIGSTP++;  }  int main(){  struct sigaction sa;  sa.sa\_handler = contar;  sigaction(SIGINT, &sa, NULL);    sa.sa\_flags = SA\_RESTART;  sigaction(SIGTSTP, &sa, NULL);  sigemptyset(&sa.sa\_mask);  while((contarSIGINT + contarSIGSTP) < 10)  sigsuspend(&sa.sa\_mask);    printf("La señal SIGINT ha sido capturada %i\n", contarSIGINT);  printf("La señal SIGTSTP ha sido capturada %i\n",contarSIGSTP);  return 0;  }  Terminal 1  [cursoredes@localhost Prac3]$ gcc -Wall -g ej12.c -o ej12  [cursoredes@localhost Prac3]$ ./ej12  Terminal 2  [cursoredes@localhost Prac3]$ ps -aux  cursore+ 17728 0.0 0.0 4208 348 pts/0 S+ 20:51 0:00 ./ej12  [cursoredes@localhost Prac3]$ sudo kill -SIGINT 17728 (10 veces)  Terminal 1  La señal SIGINT ha sido capturada 10  La señal SIGTSTP ha sido capturada 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.

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

|  |
| --- |
| #include<string.h>  #include<stdio.h>  #include<stdlib.h>  #include <signal.h>  #include <unistd.h>  #define TRUE 1  #define FALSE 0  volatile int borrar = TRUE;  void hand(int state){  if(state == SIGUSR1)  borrar = FALSE;  }  int main( int argc, char \*\*argv){  struct sigaction sa;  sa.sa\_handler = hand;  sigaction(SIGUSR1,&sa,NULL);  sigemptyset(&sa.sa\_mask);  int s = atoi(argv[1]);  sleep(s);  sigemptyset(&sa.sa\_mask);  if(borrar){  remove(argv[0]);  printf("El fichero ha sido borrado \n");    }  else  printf("No se ha borrado el fichero\n");  return 0;  }  Terminal 1  [cursoredes@localhost Prac3]$ gcc -Wall -g ej13.c -o ej13  [cursoredes@localhost Prac3]$ ./ej13 60  Terminal 2  [cursoredes@localhost Prac3]$ ps -aux  cursore+ 18580 0.0 0.0 4208 348 pts/0 S+ 21:14 0:00 ./ej13 60  [cursoredes@localhost Prac3]$ sudo kill -SIGUSR1 18580  Terminal 1  No se ha borrado el fichero |