**Sistemas Operativos**

*Prácticas curso 2021-2022*

Programación con llamadas al sistema: Práctica 5

Señales

1. En **/home/asignaturas/so/Seniales** tienes unos ficheros ejemplo de cómo programar señales en UNIX (transparencias 50 a 60). Copia la carpeta a tu cuenta, luego compila, observa el código y ejecuta (en orden) dichos ejemplos. **Notas**:
   * Para enviar la señal SIGUSR1 al ejemplo 2 abre otra terminal en *gollum*, averigua el PID (con el comando “**ps lxw”**) del ejemplo y usa el comando “**kill -s USR1 <pid>”**
   * El ejemplo 3 es como el 2 pero se crea un hijo. El padre se dedica a dormir en el sofá y el hijo, de vez en cuando, le llama desde el móvil para despertarlo (le envía automáticamente la señal SIGUSR1).
2. ¿Qué tres acciones diferentes se pueden indicar cuando llega una señal al proceso?

Se puede:

1. Realizar la acción por defecto de dicha señal.
2. Invocar a una función concreta del programa.
3. Ignorar dicha señal.
4. Implementa un comando externo **delay** que dados unos segundos y una orden (programa y argumentos), comience la ejecución del programa una vez hayan transcurridos dichos segundos. Sintaxis:

delay <segundos> <orden>

#include <iostream>

#include <sys/types.h>

#include <unistd.h>

#include <signal.h>

using namespace std;

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

// Comprobar que se introducen dos argumentos.

if (argc != 3) {

cout << "Uso: " << argv[0] << " <segundos> <orden> " << endl;

return 1;

}

sleep(atoi(argv[1]));

execvp(argv[2], argv + 2);

}

1. Implementa un comando externo **alarma** que dados unos segundos y un mensaje, imprima en la salida estándar dicho mensaje cuando transcurran los segundos indicados. **El comando no debe impedir que se sigan ejecutando otros comandos y no se puede implementar usando la llamada al sistema sleep()**. Sintaxis:

alarma <segundos> <mensaje>

#include <iostream>

#include <sys/types.h>

#include <unistd.h>

#include <signal.h>

using namespace std;

string mensaje;

void Manejador(int sig) {

cout << mensaje << endl;

exit(0);

}

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

// Comprobar que se introducen dos argumentos.

if (argc != 3) {

cout << "Uso: " << argv[0] << " <segundos> <mensaje> " << endl;

return 1;

}

mensaje = argv[2];

struct sigaction act;

act.sa\_handler = Manejador;

sigaction(SIGALRM, &act, NULL);

if(fork()) {

exit(0); // Si es el proceso padre, salir.

} else {

alarm(atoi(argv[1]));

pause();

}

}

1. Amplía el código del intérprete (Versión 4) de forma que el intérprete sea inmune al CTRL-C **(PERO NO LOS COMANDOS QUE SE EJECUTAN EN ÉL).** En definitiva, que se comporte como un Shell del sistema (obsérvese como el **bash** es inmune al CTRL-C pero no así los comandos que ejecuta).

En la línea 173:

1. struct sigaction act;
2. act.sa\_handler = SIG\_IGN;
3. sigaction(SIGINT, &act, NULL);
4. [OPCIONAL] Implementa un comando externo **limit** que dados unos segundos y una orden (programa y argumentos), ejecute la orden. El comando **limit** terminará por uno de estos dos motivos: la orden finaliza o bien si transcurridos los segundos indicados la orden no ha finalizado se procederá a su terminación (enviando la señal adecuada) y terminará el comando. Además, si durante la ejecución de la orden, el comando **limit** recibe la señal SIGUSR1 dejará que la orden se ejecute sin restricción de tiempo. Sintaxis:

limit <segundos> <orden>

#include <iostream>

#include <sys/types.h>

#include <unistd.h>

#include <signal.h>

using namespace std;

bool terminar = true;

void Manejador(int sig) {

terminar = false;

}

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

// Comprobar que se introducen dos argumentos.

if (argc != 3) {

cout << "Uso: " << argv[0] << " <segundos> <orden> " << endl;

return 1;

}

struct sigaction act;

act.sa\_handler = Manejador;

sigaction(SIGUSR1, &act, NULL);

pid\_t pid = fork();

if (pid == 0) {

// Código del hijo

execvp(argv[2], argv + 2);

} else {

sleep(atoi(argv[1]));

if(terminar) kill(pid, SIGKILL);

}

}