# Práctica 4: Procesos e hilos.

## Índice

**1 Objetivos** **1**

**2 Ejercicios** **1**
Ejercicio 1: Creación de procesos y ejecución de programas . . . . . 1
Ejercicio 2: Creación y paso de parámetros a hilos. . . . . 3
Ejercicio 3: Manejo de señales. . . . . 3
Ejercicio 4: Manejo de ficheros con varios procesos e hilos . . . . . 4

## 1 Objetivos

En esta práctica vamos a hacer varios ejercicos orientados a afianzar nuestro conocimiento del manejo del API POSIX de procesos e hilos, y cómo afecta el uso de hilos y procesos al manejo de ficheros.

Se aconseja al alumno que cree un directorio para la práctica y un subdirectorio por ejercicio. En las instrucciones se asume que el ejercicio N se hace en el subdirectorio ejercicio*N* dentro del directorio común para la práctica.

El archivo ficheros_p4.tar.gz contiene una serie de ficheros que pueden ser usados como punto de partida para el desarrollo los ejercicios de esta práctica, así como unos *makefiles* que pueden ser usados para la compilación de los distintos proyectos.

## 2 Ejercicios

### Ejercicio 1: Creación de procesos y ejecución de programas

Crear un programa, a partir de la plantilla suministrada en ficheros_p4.tar.gz, que liste el contenido de un directorio cuyo nombre se pasa como argumento al programa en la línea de órdenes. Para comprobar si el programa funciona correctamente podemos comparar su salida con la del programa `ls` con las opciones `-lf` y, opcionalmente, la opción `-R`. Es decir, el program aceptará dos párametros tipo Unix:

- *-h*: mostrará una ayuda del programa
- *-R*: activará la opción recursiva del programa

y una parámetro adicional que nos dará el nombre del directorio a listar. El diseño del programa se hará del siguiente modo:

1. El programa principal utilizará la función de biblioteca `getopt` para parsear las opciones cortas (-h y -R) e identificar si se ha pasado algún parámetro adicional, localizado por medio de la variable `optind`. En caso de que este último parámetro esté presente nos indicará el directorio que hay que listar, y si no está presente se asumirá que el directorio a listar es el directorio actual (“.”).

2. Una vez parseadas las opciones, el programa principal mostrará por pantalla el nombre del directorio a listar seguido de ":" si el usuario activó la opción recursiva del programa (pasó el parámetro -R); y a continuación invocará la función list_dir, pasándole el nombre del directorio a listar. Dicha función se encargará de leer todas las entradas del directorio que se le pasa como argumento, mostrando por la salida estándar el nombre de cada una de sus entradas.

   Ejemplo:

```
$ mkdir -p a/aa
$ echo "Hola mundo" | tee a/a.txt  a/aa/aa.txt > /dev/null
$ ./ej1
a
.
..
ej1.c
Makefile
ej1
ej1.o
```

   y con el comando `ls` obtendríamos el mismo resultado:

```
$ ls -1f
a
.
..
ej1.c
Makefile
ej1
ej1.o
```

3. A continuación, si se ha activado la opción recursiva del programa (flag `-R`), el programa principal invocará la función `list_dir_recurse`, pasándole como argumento el directorio que hay que listar. Esta función recorrerá el directorio que se le pasa como argumento y por cada subdirectorio creará un nuevo proceso, haciendo que dicho proceso vuelva a ejecutar el programa pero con el nombre del subdirectorio como argumento. La ruta del subdirectorio se forma concatenando el nombre del directorio original con el del subdirectorio separados por "/". El programa debe esperar la finalización del nuevo proceso antes de proseguir con la siguiente entrada del directorio original.

   Ejemplo:

```
$ ./ej1 -R
.:
a
.
..
ej1.c
Makefile
ej1
ej1.o

./a:
.
aa
..
a.txt

./a/aa:
.
..
aa.txt
```

y con el comando `ls -1fR` obtenemos el mismo resultado:

```
$ ls -1fR
.:
a
.
..
ej1.c
Makefile
ej1
ej1.o

./a:
.
aa
..
a.txt

./a/aa:
.
..
aa.txt
```

### Ejercicio 2: Creación y paso de parámetros a hilos.

En este ejercicio vamos a usar la biblioteca de pthreads, por lo que será necesario compilar y enlazar con la opción `-pthread`.

Escribir un programa *hilos.c* que va a crear hilos cuya funcionalidad vendrá determinada por los argumentos que se le pasen en la creación. Los hilos recibirán como argumentos el puntero a una estructura que contenga dos campos: un entero, que será el número de hilo, y un caracter, que indicará si el hilo es prioritario (P) o no (N).

El programa deberá crear una variable para el argumento de cada hilo usando memoria dinámica, inicializar dicha variable con el número de hilo y su prioridad (los pares serán prioritarios y los impares no lo serán), crear los hilos y esperar a que finalicen.

Cada hilo copiará sus argumentos en variables locales, liberará la memoria dinámica reservada para los mismos, averiguará cuál es su identificador e imprimirá un mensaje con su identificador, el número de hilo y su prioridad.

El alumno debe consultar las páginas de manual de: `pthread_create`, `pthread_join`, `pthread_self`.

Probar a crear solamente una variable para el argumento de todos los hilos, dándole el valor correspondiente a cada hilo antes de la llamada a `pthread_create`. Explicar qué sucede y cuál es la razón.

### Ejercicio 3: Manejo de señales.

En este ejercicio vamos a experimentar el envío de señales, haciendo que un proceso cree a un hijo, espere a una señal de un temporizador y, cuando la reciba, termine con la ejecución del hijo.

Al igual que en el ejercicio 1, el programa principal recibirá como argumento el ejecutable que se desea que ejecute el proceso hijo.

El proceso padre creará un hijo, que cambiará su ejecutable con una llamada a execvp. A continuación, el padre éstablecerá que el manejador de la señal SIGALRM sea una función que envíe una señal SIGKILL al proceso hijo y programará una alarma para que le envíe una señal a los 5 segundos. Antes de finalizar, el padre esperará a que finalice el hijo y comprobará la causa por la que ha finalizado el hijo (finalización normal o por recepción de una señal), imprimiendo un mensaje por pantalla.

El alumno debe consultar las páginas de manual de: sigaction, alarm, kill, wait.

Para comprobar el funcionamiento correcto de nuestro programa podemos usar como argumento un ejecutable que termine en menos de 5 segundos (como ls o echo) y uno que no finalice hasta que le llegue la señal (como xeyes).

Una vez funcione el programa, modificar el padre para que ignore la señal SIGINT y comprobar que, efectivamente, lo hace.

### Ejercicio 4: Manejo de ficheros con varios procesos e hilos

Se pretende crear un programa que utilice 10 procesos (el original y 9 procesos hijo) para escribir de manera concurrente un fichero "output.txt". La idea es que cada proceso escriba una cadena de caracteres con un número decimal repetido 5 veces. Así el proceso inicial escribira 5 ceros ("00000"), el primer proceso hijo 5 unos ("11111"), el segundo 5 doses ("22222") y así sucesivamente. De este modo el contenido del fichero al final será: 000001111122222333334444455555666667777788888999999

Un primer programador con poca experiencia en la programación de sistemas propone la siguiente implementación (fichero practica_2_5_inicial.c):

```c
int main(void)
{
    int fd1,fd2,i,pos;
    char c;
    char buffer[6];

    fd1 = open("output.txt", O_CREAT | O_TRUNC | O_RDWR, S_IRUSR | S_IWUSR);
    write(fd1, "00000", 5);
    for (i=1; i < 10; i++) {
        pos = lseek(fd1, 0, SEEK_CUR);
        if (fork() == 0) {
            /* Child */
            sprintf(buffer, "%d", i*11111);
            lseek(fd1, pos, SEEK_SET);
            write(fd1, buffer, 5);
            close(fd1);
            exit(0);
        } else {
            /* Parent */
            lseek(fd1, 5, SEEK_CUR);
        }
    }

    //wait for all childs to finish
    while (wait(NULL) != -1);

    lseek(fd1, 0, SEEK_SET);
    printf("File contents are:\n");
    while (read(fd1, &c, 1) > 0)
        printf("%c", (char) c);
    printf("\n");
    close(fd1);
    exit(0);
}
```

Tras esta implementación el programador comprueba el funcionamiento, ejecutando 10 veces seguidas el programa con la esperanza de que no se produzcan carreras. El resultado, en la máquina del programador es:

```
$ for i in $(seq 10); do ./practica_2_5_inicial ; done
File contents are:
000001111122222333335555566666888899999
File contents are:
000001111122222555556666668888899999
```

```
File contents are:
000001111122222333335555566666888899999
File contents are:
00000111112222244444666667777799999
File contents are:
000004444477777555556666668888899999
File contents are:
000002222244444555557777778888899999
File contents are:
0000011111222224444455555777778888899999
File contents are:
000001111122222555554444488887777799999
File contents are:
0000011111222224444455555777778888899999
File contents are:
000001111122222555554444488887777799999
```

Al parecer el programa tiene algunos errores, puesto que se producen carreras y el resultado es incorrecto en todos los casos.

- Cuestión A - Soluciona la implementación inicial, manteniendo la escritura concurrente en el fichero. Es decir, el proceso padre escribirá los cinco ceros iniciales, el hijo uno los cinco unos, etc, sin necesidad de sincronizar los procesos. Es decir, se desea que no sea necesario imponer un orden en la ejecución de los procesos.
- Cuestión B - Proponer una solución en la que el padre escriba su número entre la escritura de los hijos, de modo que el contenido del fichero al final será el siguiente:

  000001111100000222220000033333000004444400000555550000066666000007777700000888880000099999