

# Práctica 2: API de ficheros, directorios, procesos e hilos

## Índice

<b>1</b>	<b>Objetivos</b>	<b>1</b>
<b>2</b>	<b>API de procesos e hilos</b>	<b>1</b>
	Ejercicio 1: Creación de procesos . . . . .	1
	Ejercicio 2: Creación de procesos que cumplan un grafo de dependencias. . . . .	2
	Ejercicio 3: Creación y paso de parámetros a hilos. . . . .	2
	Ejercicio 4: Manejo de señales. . . . .	2
<b>3</b>	<b>Manejos de ficheros con varios procesos e hilos</b>	<b>3</b>

## 1 Objetivos

En esta práctica vamos a hacer varios ejercicios 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.

La práctica está organizada en 2 partes:

- API de procesos e hilos
- Manejos de ficheros con varios procesos e hilos

Cada parte consiste en uno o más ejercicios independientes. Se aconseja al alumno que cree un directorio por parte con un subdirectorio por ejercicio. En las instrucciones de cada parte se asume que el ejercicio N se hace en un subdirectorio llamado ejercicioN dentro del directorio común para dicha parte.

El archivo ficheros\_p3.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 API de procesos e hilos

En esta parte de la práctica trabajaremos las llamadas al sistema: fork, exec, wait, waitpid, getpid, getppid, sigaction, alarm, kill. Además, usaremos las funciones de la biblioteca de pthreads: pthread\_create, pthread\_join, pthread\_self.

### Ejercicio 1: Creación de procesos

Diseña un programa fork1.c que cree dos procesos hijos. El primero de ellos no cambiará de ejecutable, pero el segundo sí lo hará, mediante una llamada a execvp. El programa recibirá como parámetros el nombre del ejecutable que deberá ejecutar y los argumentos que necesite pasarle.

El programa realizará una primera llamada a fork. Después de ella, tanto el programa padre como el hijo imprimirán un mensaje indicando si son padre o hijo, su identificador y el de su padre. A continuación, ambos procesos realizarán

una segunda llamada a `fork`, después de la cual cada proceso imprimirá un mensaje indicando si es padre o hijo, su identificador y el de su padre. Cada hijo generado en la segunda llamada cambiará su ejecutable por el que se haya pasado como argumento al `main` usando `execvp`. Cada padre esperará a que sus hijos finalicen.

El alumno debe consultar las páginas de manual de las siguientes llamadas al sistema: `fork`, `getpid`, `getppid`, `execvp`, `waitpid`.

Para comprobar el funcionamiento correcto de nuestro programa podemos usar como argumento cualquier ejecutable que imprima algo por pantalla, por ejemplo `echo` o `ls`.

## **Ejercicio 2: Creación de procesos que cumplan un grafo de dependencias.**

En este ejercicio tendremos un proceso padre, que creará  $N$  hijos siguiendo un grafo de dependencias, de forma que no se pueda crear un determinado hijo hasta que terminen todos los procesos de los que dicho hijo depende. En nuestro caso  $N$  será 8 y el grafo de dependencias será el del problema 5 de la hoja de procesos. Los hijos se crearán usando llamadas a `fork` seguidas de `execl` (pares) y `execlp` (impares) y cada uno ejecutará el comando `echo`, imprimiendo por pantalla su nombre (P0, P1, etc.).

El alumno debe consultar las páginas de manual de las siguientes llamadas al sistema: `fork`, `execl`, `execlp`, `waitpid`.

## **Ejercicio 3: Creación y paso de parámetros a hilos.**

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

Escribir un programa `hilos.c` que cree un hilo para cada usuario, pasándole a cada uno como argumento el puntero a una estructura que contenga dos campos: un entero, que será el número de usuario, y un carácter, que indicará si el usuario 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 usuario 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 usuario 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 4: 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 establecerá 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.

### **3 Manejos de ficheros con varios procesos e hilos**

*Under construction*