Práctica 3: Procesos e hilos.

Índice

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.

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 ejercico. En las instrucciones de cada parte se asume que el ejercicio N se hace en un subdirectorio llamado ejercicio*N* 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 continuacion, 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 una 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 pthreads, 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 caracter, 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 Ejercicio 5: Manejos 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: 00000111112222233333444445555566666777778888899999

Un primer programador con poca experiencia en la programación de sistemas propone la siguiente implementación (fichero practica_2_5_inicial.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 produczcan carreras. El resultado, en la máquina del programador es:

```
$ for i in $(seq 10); do ./practica_2_5_inicial ; done
File contents are:
00000111112222233333555556666668888899999
File contents are:
00000111112222255555666666888899999
File contents are:
00000111112222233333555556666668888899999
File contents are:
00000111112222244444666667777799999
File contents are:
00000444447777755555666668888899999
File contents are:
00000222224444455555777778888899999
File contents are:
000001111122222444445555577777888899999
File contents are:
000001111122222555554444488888777799999
File contents are:
000001111122222444445555577777888899999
File contents are:
000001111122222555554444488888777799999
```

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: 00000111110000022222000003333300000444440000055555000006666660000077777000