SISTEMAS OPERATIVOS - LABORATORIO
13 de Junio de 2025

Nombre _____ DNI _____

Apellidos _____ Grupo _____

INSTRUCCIONES

- Se debe entregar una carpeta individual con el código de cada ejercicio, para así garantizar la evaluación independiente de cada uno.
- Si el código no compila o su ejecución produce un error grave la puntuación de ese apartado será 0.

Cuestión 1. (5 puntos) Desarrollar un programa "mytime" que se comporte de forma similar al comando unix time. En el campus virtual se proporciona una breve plantilla para la solución. La forma de uso del programa es la siguiente:

```
$ ./mytime cmd
```

Como vemos el programa recibe como argumento una sola cadena de caracteres con el comando a ejecutar. El desarrollo se hará progresivamente en tres apartados evaluados de forma independiente y descritos a continuación.

a) **(2 puntos)** El programa creará un proceso que ejecutará un shell que interprete el comando (equivalente a ejecutar "/bin/sh -c cmd"), informará del pid del proceso creado y esperará a que dicho proceso termine. Se debe completar la función **`int`** `run_command(`**`char`** `*cmd)` que realizará esta tareas, así como el programa principal que invocará a run_command pasándole el comando que debe ejecutar.

```
$ ./mytime "ls -l"
child process pid 65633
total 48
-rw-r--r-- 1 christian christian   279 may 20 19:44 Makefile
-rwxr-xr-x 1 christian christian 22400 may 21 11:25 mytime
-rw-r--r-- 1 christian christian  2592 may 21 11:25 mytime.c
-rw-r--r-- 1 christian christian 14464 may 21 11:25 mytime.o
```

b) **(2 puntos)** Modificar la función run_command de forma que:
   - El proceso original ignore SIGINT y SIGTERM mientras el proceso hijo está en ejecución.
   - El proceso hijo usará el tratamiento por defecto para SIGINT (Ctrl+C) y SIGTERM.

   Cuando el proceso hijo haya terminado, el proceso original debe comprobar si el hijo terminó por la recepción de una señal, en cuyo caso mostrará por pantalla un mensaje que indique dicha señal.
   Nota: si el proceso padre recibe una señal, `wait` o `waitpid` se interrumpirán y `errno` tendrá el valor EINTR.

```
$ ./mytime "sleep 5"
child process pid 66196
^Cchild with pid 66196 terminated by signal 2   <-- ^C indica que se pulsó Ctrl+C
```

c) **(1 punto)** Modificar el programa principal para que nos de los tiempos real, de usuario y de sistema consumidos por el comando. Para ello se usarán dos llamadas al sistema (consultar su página de manual):
   - **gettimeofday:** nos da una marca de tiempo medida en segundos y microsegundos transcurridos desde el 1 de enero de 1970. Usaremos esta función para medir el tiempo "real" que tarda en ejecutarse `run_command`. La plantilla incluye una función de ayuda, `timeval_diff,` que calcula la diferencia entre dos marcas de tiempo obtenidas con `gettimeofday`, así como dos macros para obtener los segundos y milisegundos que representa un `struct timeval`.
   - **getrusage:** nos da estadísticas de uso de recursos. Usaremos como primer argumento `RUSAGE_CHILDREN` para obtener las estadísticas del proceso hijo. Los dos primeros campos de la estructura `struct rusage` nos dan el tiempo de usuario y el tiempo de sistema que necesitamos. Bastará con invocar esta función al terminar la ejecución de `run_commands`.

```
$ ./mytime ls
child process pid 66505
Makefile  mytime  mytime.c  mytime.o
real: 0.002 s  <-- puede fluctuar un poco
user: 0.001 s  <-- puede fluctuar un poco
sys:  0.001 s  <-- puede fluctuar un poco
```

Cuestión 2. (5 puntos) Desarrollar un programa pc-files con el siguiente modo de uso (se proporciona una plantilla con un Makefile y un fichero de entrada de prueba):

```
$ ./pc-files [-i fichero_entrada] [-o fichero_salida] [-h]
```

El reconocimiento de opciones se hará con **getopt** (1 punto). Nótese que ninguna de las opciones es obligatoria:

- En el caso de que no pasar la opción -i en la línea de comando, el fichero de entrada será la entrada estándar, que ya está abierta y su descriptor se corresponde al valor almacenado en la variable global stdin.
- En caso de no pasar la opción -o en la línea de comando, el fichero de salida será la salida estándar, que ya está abierta y su descriptor se corresponde al valor almacenado en la variable global stdout. Si por el contrario se especifica un fichero de salida con la opción -o y el fichero ya existe, su contenido se truncará; en caso contrario, se creará dicho fichero.
- La opción -h ofrece una ayuda de uso del programa

El programa pc-files creará dos hilos POSIX, que se comportan respectivamente como productor y consumidor. Ambos hilos se comunicarán usando un buffer global compartido de cadenas de caracteres, con capacidad para 4 cadenas:

```
#define MAX_SBUFFER_SIZE 4
char* shared_buffer[MAX_SBUFFER_SIZE];
```

Estos hilos se comportan del siguiente modo:

- (**2 puntos**) El productor leerá cadenas de caracteres terminadas en '\n' del fichero de entrada hasta detectar fin de fichero, insertando estas cadenas en el buffer para que puedan ser consumidas por el hilo consumidor. La lectura se hará sobre un buffer intermedio de tamaño `MAX_LINE_LENGTH` (256). Para realizar la inserción el hilo hará una copia de la cadena leída del fichero, e insertará la dirección de comienzo de esta copia en el buffer, respetando las condiciones habituales del productor (debe quedar sitio en el buffer, y no se deben producir carreras en la inserción). Para indicar al consumidor que no hay más cadenas que procesar, cuando se detecte el final de fichero el productor insertará un `NULL` en el buffer compartido.
- (**2 puntos**) El consumidor extraerá cadenas de este buffer compartido, y las escribirá en el fichero de salida en orden de recepción, liberando después la memoria del buffer que almacena dicha cadena (con `free`). El hilo consumidor terminará al extraer la cadena `NULL` del buffer.

El programa principal terminará cuando ambos hilos hayan acabado. Para la lectura por líneas del fichero se recomienda el uso de la función **`fgets()`** de la biblioteca estándar (consultar su página de manual).

Para implementar las restricciones de sincronización típicas del productor consumidor ha de usarse **un mutex, variables condicionales, y otras variables compartidas** necesarias para el correcto acceso al buffer compartido.

```
Ejemplos de uso:

$ ./pc-files -h
Uso: ./pc-files [-i fichero_entrada] [-o
fichero_salida] [-h]
$ cat in.txt
London
Barcelona
Madrid
Berlin
Lisbon
Paris
$ cat in.txt | ./pc-files
London
Barcelona
Madrid
Berlin
Lisbon
Paris
```

```
$ ./pc-files -i in.txt
London
Barcelona
Madrid
Berlin
Lisbon
Paris
$ ./pc-files -i in.txt -o out.txt
$ cat out.txt
London
Barcelona
Madrid
Berlin
Lisbon
Paris
```