SISTEMAS OPERATIVOS - LABORATORIO
19 de Diciembre de 2024, Turno 2

Nombre _____ DNI _____

Apellidos _____ Grupo _____

INSTRUCCIONES

- Se debe entregar una carpeta individual con el código de cada apartado (1A, 1B, etc.), 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) Extender la implementación de la práctica 5 (disco.c) para incorporar un servicio de limpieza a la discoteca, que entrará a limpiarla cada vez que 3 nuevos usuarios (clientes de cualquier tipo) hayan estado en la discoteca. El servicio de limpieza de la discoteca se implementará como un nuevo hilo POSIX, que tendrá el siguiente comportamiento:

```
void* cleaning_thread(void* arg){
   while(1){
         enter_cleaning();
         //Limpiar la discoteca
         printf("Cleaning...\n");
         sleep(2);
         printf("Done!\n");
         exit_cleaning();
   }
   return NULL;
 }
```

Por simplicidad, se aconseja mantener al menos una variable global extra llamada `clean_pending` que estará a 1 cuando el hilo de limpieza esté dentro de `enter_cleaning()` o bien esté llevando a cabo la limpieza en ese momento (no haya acabado aún).

Al implementar las funciones `enter_cleaning()` y `exit_cleaning()` se han de cumplir las siguientes restricciones:

- `enter_cleaning()` bloqueará al servicio de limpieza hasta que al menos 3 clientes hayan estado en la discoteca desde la última limpieza efectuada. Análogamente, la función no retornará tampoco hasta que todos los clientes que ya estuvieran dentro de la discoteca la hayan abandonado.

- El sistema no debe permitir el acceso a la discoteca de ningún cliente esperando mientras (i) la variable `clean_pending` esté a 1 y (ii) al menos 3 clientes hayan estado en la discoteca desde la última limpieza efectuada. Garantizar esta restricción implica modificar funciones del programa concurrente invocadas por los clientes.

- `exit_cleaning()` actualizará las variables pertinentes y despertará a los clientes esperando a entrar, respetando turnos y preferencia de los clientes vip frente a los normales.

En los siguientes apartados del ejercicio (a entregar por separado para su evaluación individualizada) se implementará gradualmente la funcionalidad solicitada:

a) (0,75 p.) Modificar el código de la práctica 5 para que se cree el nuevo thread de limpieza en el programa principal. Este hilo ejecutará `cleaning_thread()`. No obstante, por simplicidad en este apartado las funciones `enter_cleaning()` y `exit_cleaning()` se dejarán vacías.

b) (3p.) Modificar el código desarrollado en el apartado anterior para satisfacer todas las restricciones descritas más arriba. El hilo de limpieza se terminará de forma forzosa con `pthread_cancel()` cuando todos los hilos cliente hayan acabado.

c) (1,25 p) Alterar la implementación de la parte b) del ejercicio de tal forma que el hilo de la discoteca termine normalmente retornando de su función principal (sin usar pthread_cancel() ni pthread_exit()) cuando todos los hilos cliente hayan acabado.

Ejemplo de ejecución (apartados b y c):

```
$ cat example_10clients.txt
10
0
0
1
0
1
0
0
1
0
1

$  ./disco example_10clients.txt
Cleaning thread waiting to enter disco
Starting client  2
Client  2 (  vip  ) entering the disco
Client  2 (  vip  ) dancing in disco
Starting client  0
Client  0 (not vip) entering the disco
Client  0 (not vip) dancing in disco
Starting client  3
Client  3 (not vip) entering the disco
Client  3 (not vip) dancing in disco
Starting client  4
Client  4 (  vip  ) waiting on the queue
Starting client  1
Client  1 (not vip) waiting on the queue
Starting client  5
Client  5 (not vip) waiting on the queue
Starting client  6
Client  6 (not vip) waiting on the queue
Starting client  7
Client  7 (  vip  ) waiting on the queue
Starting client  8
Client  8 (not vip) waiting on the queue
Starting client  9
Client  9 (  vip  ) waiting on the queue
Client  0 (not vip) exits the disco
Client  4 (  vip  ) waiting on the queue
Client  7 (  vip  ) waiting on the queue
Client  9 (  vip  ) waiting on the queue
Client  2 (  vip  ) exits the disco
Client  4 (  vip  ) waiting on the queue
Client  7 (  vip  ) waiting on the queue
Client  9 (  vip  ) waiting on the queue
Client  3 (not vip) exits the disco
Cleaning...
Client  4 (  vip  ) waiting on the queue
Client  9 (  vip  ) waiting on the queue
Client  7 (  vip  ) waiting on the queue
Done!
Cleaning thread waiting to enter disco
Client  4 (  vip  ) entering the disco
Client  4 (  vip  ) dancing in disco
Client  9 (  vip  ) waiting on the queue
Client  7 (  vip  ) entering the disco
Client  7 (  vip  ) dancing in disco
Client  9 (  vip  ) entering the disco
Client  9 (  vip  ) dancing in disco
Client  6 (not vip) waiting on the queue
Client  1 (not vip) waiting on the queue
Client  5 (not vip) waiting on the queue
Client  8 (not vip) waiting on the queue
Client  7 (  vip  ) exits the disco
Client  6 (not vip) waiting on the queue
Client  5 (not vip) waiting on the queue
Client  1 (not vip) waiting on the queue
Client  8 (not vip) waiting on the queue
Client  4 (  vip  ) exits the disco
Client  9 (  vip  ) exits the disco
Client  6 (not vip) waiting on the queue
Client  5 (not vip) waiting on the queue
Client  1 (not vip) waiting on the queue
Client  8 (not vip) waiting on the queue
Cleaning...
Done!
…
```

Cuestión 2. (5 puntos) Ampliar la funcionalidad del programa `run_commands` (desarrollado en el ejercicio 1 de la práctica 4) de la siguiente forma:

a) (2,5p.) Se incorporará una nueva opción -i que permitirá ejecutar comandos en primer plano de forma interactiva. El programa imprimirá la cadena "> " a modo de *prompt* para invitar al usuario que teclee un nuevo comando. Al teclear el comando, este se ejecutará usando la función `launch_command()` --ya implementada en la práctica original--, y se esperará al proceso hijo correspondiente. Cuando el hijo haya terminado se volverá a mostrar el *prompt*, y así sucesivamente hasta que el usuario introduzca el fin de fichero en la entrada estándar, tecleando CTRL+D. Esto terminará la ejecución del programa. Se aconseja usar la función `fgets()` para la implementación; consultar `man fgets()`. Asimismo, se recomienda ejecutar `fflush(stdout)` tras la sentencia de impresión del *prompt,* para que este salga inmediatamente por pantalla.

b) (2,5p.) Una limitación de la función `launch_command()` es el hecho de que no permite ejecutar comandos de shell que contengan extensiones típicas de bash, como expansión de variables, redirecciones o *pipes*. Para subsanar esta limitación, se pide implementar una nueva opción -B que permita ejecutar comandos del shell BASH. Esta nueva opción se usará en combinación con el modo interactivo (opción -i) desarrollada en el apartado anterior. Para implementar la nueva opción -B se creará la siguiente función que ejecutará comandos del shell BASH:

```
pid_t launch_shell_command(char* cmd);
```

Esta función se comportará de forma similar a `launch_command()` (creación de proceso hijo y retorno de su PID), pero haciendo que el hijo creado ejecute el comando `/bin/bash -c <cmd>`, donde <cmd> es el argumento de la función (comando especificado por el usuario). Nótese que ahora el programa `bash` se ocupará del parsing del comando (argumento de "-c"), gestionando adecuadamente las extensiones de shell arriba mencionadas.

Ejemplo de ejecución:

```
$ ./run_commands -i
> ls
Makefile     run_commands run_commands.c  run_commands.o  test1         test2
> sleep 2
> echo hola > a.txt
hello > a.txt
> echo $HOME
$HOME
> ^D

$ ./run_commands -i -B
> ls
Makefile     run_commands run_commands.c  run_commands.o  test1         test2
> echo $HOME
/home/usuarioso
> sleep 1; echo Uno; sleep 1; echo Dos
Uno
Dos
> echo hello | grep o
hello
> echo hello > a.txt
> cat a.txt
hello
> echo $$
23532
> ^D
$
```