SISTEMAS OPERATIVOS - LABORATORIO
23 de Enero de 2023

Nombre _____ DNI _____

Apellidos _____ Grupo _____

ADVERTENCIA: Si el código no compila o su ejecución produce un error grave la puntuación de ese apartado será 0.

Cuestión 1. (4 puntos) Crear un programa que reciba como argumento (opción -n) un número entero (*nbytes)* y opcionalmente (opción -o) un nombre de fichero de salida:

> ./cuestion1 - n <numBytes> [-o <outFileName>]

El programa recorrerá el directorio actual y leerá los <numBytes> primeros bytes de cada fichero regular (es decir, no tendrá en cuenta directorios, enlaces simbólicos…). Si un fichero tiene menos de <numBytes>, se leerán todos los bytes del fichero.

Si no se proporcionó la opción "-o <outFileName> ", se escribirán los bytes leídos directamente a la salida estándar, usando una línea nueva por cada fichero de entrada. En caso contrario, se escribirán en un fichero llamado <outFileName> que se creará si no existiese, y se truncará en caso de existir con anterioridad.

Para comprobar el programa anterior, ejecutar los siguientes comandos desde el directorio del ejecutable:

```
> mkdir prueba; cd prueba
> for ((i=0;i<20;i++)) ; do echo $i >> f1.txt ; done
> base64 /dev/urandom | head -c 20 > f2.txt
> ln -s f1.txt slink.sl
> ../cuestion1 -n 10
```

Deberían mostrarse 20 bytes en total (10 de f1.txt y 10 de f2.txt)

Cuestión 2. (6 puntos). Crear un programa que simule una ronda de entrenamiento de unos jugadores de baloncesto usando hilos POSIX como se describe a continuación.

Se creará un hilo por cada jugador (habrá NJUGADORES = 4). Cada jugador viene definido por la siguiente estructura:

```
struct jugador {
   int dorsal; // dorsal (único) del jugador
   int eficTiro; // entero entre 1 y 100 que determina su efectividad en el tiro
   int eficRebote; // entero entre 1 y 100 que determina su efectividad en el rebote
   int puntos; // puntos conseguidos durante el entrenamiento
};
```

Asimismo, se definirá un enumerado para modelar el balón:

```
// POSESION -> un jugador tiene el balon.  ENAIRE -> el balón se ha lanzado a canasta
enum estadoBalon_t {POSESION, ENAIRE};
```

Se crearán las siguiente variables globales, incluyendo un array de NJUGADORES jugadores inicializados como sigue:

```
#define MAXREBOTE 70
enum estadoBalon_t balon; // Modela el estado actual del balón
lint dificultadRebote =0; // var. Global para indicar la dificultad del rebote actual
struct jugador jugadores[NJUGADORES] = {
    {0,30,MAXREBOTE,0}, // jugador con dorsal 0, eficTiro 30, eficRebote MAXREBOTE
    {1,80,20,0} ,
    {2,40,MAXREBOTE,0} ,
    {3,50,50,0}};
```

Implementa el programa de modo que:

a) (1 punto) El programa principal cree NJUGADORES hilos (NJUGADORES será 4). Cada hilo comenzará en la función *jugadorInit(void*)* que recibirá como argumento un puntero al jugador correspondiente. En esta primera implementación el cuerpo de esta función solamente imprimirá por pantalla las características del jugador.

b) (4 puntos) Implementar las funciones ***tirar(struct jugador*)*** y ***rebotear(struct jugador*)*** y modificar la función ***jugadorInit()*** de acuerdo al pseudocódigo mostrado más abajo.

   En la función ***tirar(struct jugador*),*** el jugador realizará un tiro. Para ello, calculará un número aleatorio (*dificultadTiro*) entre 0 y 100. Si dicho número es menor que la eficacia de tiro del jugador, habrá canasta y el jugador se suma 2 puntos. Si no, no se suma nada. El balón pasará al estado ENAIRE y se calculará la dificultad del rebote como un aleatorio entre 1 y MAXREBOTE:

```
dificultadRebote = rand() % MAXREBOTE;
```

   Finalmente, avisará a todos los jugadores para que traten de conseguir el rebote.

   En la función ***rebotear(struct jugador*),*** cada jugador debe esperar a que el balón esté en el modo ENAIRE y a que la dificultad del rebote sea menor que su eficacia en el rebote. Si lo consigue, pondrá el balón en estado POSESION y procederá a *correr()*.

c) (1 punto) Finalización: el entrenamiento finalizará (un hilo escribirá *finished = 1*) cuando cualquier jugador llegue al tope de anotación definido (crear una constante TOPEANOTACION). Cuando finalice el entrenamiento y todos los hilos hayan finalizado, el programa principal (el hilo inicial) imprimirá por la salida estándar los puntos conseguidos por cada jugador. Modificar el código anterior para contemplar esta finalización.

Pseudo-código de las funciones principales para apartados b)

```
void rebotear(struct jugador *j )
{
   <esperar a conseguir rebote>
   balon = POSESION;
}

void correr() {
       usleep(100);
}
```

```
void tirar(struct jugador *j ) {
   int difTiro;
   balon = ENAIRE ;
   difTiro = rand() % 100;
   <si difTiro es menor que nuestra eficacia en tiro: canasta>
   difRebote = rand() % MAXREBOTE;
   <si hemos sobrepasado TOPEPUNTOS, termina el entrenamiento>
   <avisar a todos los  jugadores para que luchen por el rebote>
}
```

```
void* jugadorInit(void* arg) {
   struct jugador *j = ....;
   while (!finished) {
       rebotear(j);
       correr();
       tirar(j);
       correr();
   }
   return NULL;
}
```

Se permite utilizar cualquier mecanismo de sincronización para implementar las esperas/avisos.