Escuela Politécnica Superior de Elche

Grado en Ingeniería Informática
en Tecnologías de la Información

Sistemas Operativos

Examen Práctico
Convocatoria Adicional Prácticas
12 Enero 2023

Se dispone de un conjunto de ficheros en **C** con sus respectivas cabeceras **.h** que una vez correctamente compilados mediante **make** generan un ejecutable llamado **rps**.

Se dispone a su vez de un Shell script en **Bash** llamado **lanzar.sh** que lanza el ejecutable **rps** y lee y opera con los ficheros que éste genera, mostrando resultados por pantalla.

Para aprobar es necesario aprobar ambas partes por separado:

Parte C: 5 Puntos.
Parte Bash: 5 Puntos

Técnicas utilizadas:

- Semáforos usando la librería sem.h/sem.c
- Tuberías
- Memoria Compartida

Los ficheros **lanzar.sh** y **procesar.c** son plantillas o prototipos que forman una solución. Se adjunta el makefile que genera el ejecutable **rps**.

El alumno deberá entenderlas y completarlas.

Editará correctamente los bloques **//Begin TODO … //End TODO** colocados en los ficheros **procesar.c y lanzar.sh** para que el problema funcione correctamente.

Los prototipos de los ficheros están documentados para que el alumno tenga claro lo que realizan, a su vez los bloques TODO tienen comentarios de lo que realiza el bloque y los distintos pasos que se deben cumplir incluyendo el marcador "**…**" (tres puntos) que el alumno debe sustituir por el código apropiado. Puede haber líneas ya completadas entre bloques TODO.

En el prototipo de los programas existen constantes y funciones definidas que no se deben cambiar, pero el alumno puede añadir las que necesite.

> Ejecutando los programas **lanzar.sh** y **rps** sin argumentos deberá mostrarse una guía de uso o sintaxis del script bash el ejecutable.
>
> Se puede usar el ejecutable de la parte C, **rps_profesor,** para probar la parte Bash sin tener realizada la parte C.

Se dispone de una serie de funciones de utilidad en los ficheros **func.c** y **func.h**.
Se dispone de la librería básica de semáforos **sem.c** y **sem.h**

El problema:

Se trata de resolver un torneo del Juego Piedra, Papel o Tijera (**R**ock, **P**aper, **S**cissors en inglés, **rps**)

Se plantea realizar un torneo con hasta 128 jugadores en varias rondas. Tras cada ronda pasan a la siguiente sólo los ganadores. En una ronda se enfrentan aleatoriamente los jugadores de esa ronda por parejas para jugar la partida al mejor de 3, 5, 7 o 9 manos, en función de lo que se pida.

En cada ronda se reduce el número de jugadores en 2, por lo que tendremos que el número de rondas será $\log_2$(numPlayers)

El programa en bash se encargará de gestionar el torneo, y el programa en C se encargará de ejecutar una mano de Piedra-Papel-Tijera entre dos jugadores. Se le pasan los ids de los jugadores y el parámetro **bestOf** que indica cuantas manos jugar en la partida.

Parte Bash:

Tenemos un proceso BASH llamado **lanzar.sh**

El script **lanzar.sh** admite dos parámetros obligatorios, que serán

numPlayers: Número de jugadores
bestOf: Valor de "al mejor de" (numero de manos de una partida)
Sintaxis: **lanzar.sh <numPlayers> <bestOf>**

Con las siguientes restricciones que el programa debe controlar:

Ambos parámetros son obligatorios

numPlayers:

- Debe ser numérico mayor o igual que 16 y menor o igual que 128
- Debe ser múltiplo de 16
- Por tanto puede ser : 16, 32, 64 o 128

bestOf:

- Debe ser numérico impar mayor o igua que 3 y menor o igual que 9
- Por tanto puede ser: 3, 5, 7, o 9

Ejemplo de lanzamiento para un torneo con 16 jugadores iniciales siendo las partidas al mejor de 3.

```
$ lanzar.sh 16 3
```

El proceso se basa en el uso del array **players** que almacenará una lista aleatoria con los ids (número) de los numPlayer jugadores del torneo, en el ejemplo 16 jugadores, podría ser este el array:

```
Players : 1 15 5 12 14 0 10 2 4 9 13 3 8 6 7 11
```

La línea anterior la muestra una función de utilidad que el alumno debe completar

Para comenzar a jugar hay que eliminar primero los ficheros de las partidas que ha generado la parte C. Además tras cada ronda hay que volver a limpiar estos ficheros. Para ello usaremos una función de utilidad que el alumno debe completar.

El programa calcula el número de rondas antes de la final e itera haciendo rondas hasta que el contador de rondas es igual a **numRounds** que se calcula como se ha indicado anteriormente.

En cada ronda, el programa llama al ejecutable C **rps** pasándole de dos en dos los ids de los jugadores en el array de **players**. Así por ejemplo en la primera ronda, con la lista de jugadores anterior se enfrentarían:

Jugador 1 vs Jugador 15
Jugador 5 vs Jugador 12
Jugador 14 vs Jugador 0
…

El programa llama en bucle al programa rps tantas veces como sea necesario para que se jueguen las partidas de esa ronda.

Tras el bucle de la ronda, en el directorio actual estarán los ficheros de resultados de cada partida, con el siguiente patrón de nombres: NNvsNN.dat

Así para las partidas anteriores los ficheros serían:

```
1vs15.dat
5vs12.dat
14vs0.dat
…
```

El contenido de cada uno de estos ficheros es el siguiente, por ejemplo, para el 1vs15.dat

```
1, 15, 0, 0,
R, P, 0, 1,
P, S, 1, 1,
S, P, 2, 1,
1
```

La primera línea tiene los dos jugadores y los contadores a cero, contadorA, contadorB
La segunda línea ya tiene el resultado de una mano válida (no son apuestas iguales), donde el primer jugador ha sacado Rock (piedra) y el segundo jugador Paper (papel) por lo que esa mano la gana el jugadorB, el jugador 15, por eso los contadores reflejan 0, 1
La tercera línea gana el JugadorA pues Rock gana a Scissors y los contadores quedan en 1,1
Finalmente en la tercera mano válida (bestOf=3) gana el jugadorA pues Scissors gana a Paper.
La última línea indica el jugador que ha ganado la partida, el jugadorA, en este caso el Jugador con id 1.

Tener en cuenta que **la última línea no tiene ni coma ni retorno de carro**.

El programa lee estos ficheros generados para esta ronda y determina el ganador de cada partida (el de la última fila). Entonces, reutilizando el mismo array **players**, coloca, empezando por el índice 0 los distintos ganadores. Rellena por tanto sólo la mitad del array **players** con los **numPlayers/2** que son ganadores.

Actualiza la variable **numPlayers**, borra los ficheros de la ronda anterior, muestra a los jugadores de la siguiente ronda y actualiza el contador de rondas.

Itera a por la siguiente ronda.

Cuando sale del bucle, estamos en la ronda final, sólo quedan dos jugadores, el índice 0 y el 1 del array **players**.

Llama una última vez a **rps** pasándole los dos finalistas y el valor de **bestOf**.

Muestra por pantalla el contenido del fichero de resultados de la final AvsB.dat y el ganador del torneo.

Parte C:

El ejecutable **rps** (compilado de procesar.c) deberá comprobar que le llegan los parámetros y que **bestOf** está en su rango válido. (el alumno puede añadir la comprobación de que los jugadores están en su rango válido.

La llamada sería: **rps <JugadorA> <JugadorB> <bestOf>**

La estrategia del programa se basa en:

- Crear un array de estructuras hijos (en este caso 2 jugadores)
- Crear un pipe en cada estructura hijo para leer los comandos del padre
- Usar un array de memoria compartida para guardar los resultados de cada mano válida.
    - Este array tendrá **numPlayers*besOf** enteros, 2 jugadores al mejor de 3 → 6 enteros
    - Se guardarán los enteros PIEDRA=0, PAPEL=1, TIJERAS=2 (definidos en el .h) en los índices de cada mano. Empieza en el índice 0, por lo que índices 0 y 1 para la primera mano, índices 2 y 3 para segunda mano e índices 4 y 5 para la última mano valida

Una mano válida es aquella en que las apuestas de ambos jugadores son distintas. Sólo los resultados de las manos válidas se almacenan en el buffer (array de memoria compartida)

El padre

El padre espera a que los hijos (jugadores) jueguen la primera mano y guarden sus apuestas en los índices correspondientes del buffer. La espera del padre se realiza sincronizando con los hijos en un semáforo **sSincro** que hace las veces de barrera. Lee los resultados de la mano y decide que hay que hacer en esta primera mano. Lo que decida será el comando **cmd** que enviará por el pipe a los dos hijos (esto también lo decide para las siguientes manos)

- Si ambas apuestas son iguales hay que repetir, el comando será cmd=**CMD_REPEAT** (en el .h)
- Si es una mano válida el cmd=**CMD_BET** (comando nueva apuesta)
- Si ya se ha llegado al número de manos válidas **bestOf** el comando será cmd=**CMD_END**

Una vez jugada la primera mano, entra en bucle hasta que el número de manos válidas sea igual a **bestOf** (se utilizará un contador que se actualizará sólo cuando la mano sea válida), momento en que el comando será **CMD_END** y se saldrá del bucle.

El hecho de una primera apuesta y un bucle es necesario puesto que no se sabe lo que va a durar la partida, dependerá de las veces que toque repetir si hacen la misma apuesta.

El padre abrirá un fichero de resultados para guardar los resultados de las manos, habiendo escrito la primera línea primero, donde se indica los jugadores con los contadores a cero. Durante la partida el padre lleva en los contadores (**contadorA** y **contadorB**) como está el *marcador* de la partida. Conforme las manos sean válidas el padre añadirá una línea al fichero.

Cuando finalice la partida el padre sabrá que jugador ha ganado comparando los contadores, al ser impar el número de jugadas siempre uno será mayor que otro. Añade por tanto la última línea al fichero (recordar, sin coma y sin fin de línea).

Recordar también que el padre debe esperar por la terminación de sus hijos.

Los hijos (jugadores)

Como en otros programas, el padre utiliza un bucle para crear a los hijos que terminan con exit y por tanto no iteran en el bucle.

Dentro del bucle se comprueba que fork ha funcionado y que estamos en el código del hijo.

Cada hijo primero inicializa la semilla del generador de números aleatorios con su pid.

Realiza su apuesta utilizando una función de utilidad **RandInt** que ya está programada (func.c func.h) y la guarda en el índice apropiado en el buffer de memoria compartida.

Cada hijo sabe en que índice guardar porque el iterador del bucle de creación de los hijos, variable **i**, así lo indica, 0 el primer hijo y 1 el segundo hijo. Cuando actualice para guardar su siguiente apuesta incrementará en 2 su índice.

Una vez guardada su apuesta en al Buffer señaliza al padre mediante el semáforo **sSincro**.

Entonces el hijo (hecha su primera apuesta), entra en bucle para leer lo que tiene que hacer ahora, puesto que es el padre el que sabe qué han apostado ambos jugadores. El padre enviará por el pipe de cada hijo el comando de lo que tienen que hacer.

Si ambos apostaron igual, recibirán el comando CMD_REPEAT (los comandos son enteros definidos en el .h). En este caso el hijo deberá volver a apostar pero no debe incrementar el índice donde dejar el resultado puesto que está repitiendo apuesta. Por tanto realiza su nueva apuesta y la vuelve a dejar en el mismo índice del Buffer, señaliza al padre y vuelve a esperar qué hacer (leyendo del pipe).

Si la apuesta es válida, ambos no han apostado lo mismo, el padre puede enviarles el comando CMD_BET para que vuelvan a apostar, pero esta vez incrementando el índice donde dejar el resultado en el buffer, harán su nueva apuesta y la almacenarán en la nueva ubicación señalizando también al padre de que ya ha apostado.

Pero también puede recibir CMD_END que indica que la apuesta que habían hecho antes ya era la última y por tanto deben salir del bucle de apuestas.

Al salir del bucle sólo le queda hacer **exit** para terminar su ejecución.