Enunciado de la Práctica de Aprendizaje : BASH + C

C Prácticas

Tabla de contenido

Objetivos

El objetivo de esta práctica es familiarizar al alumno con la programación concurrente básica en C sobre plataformas Unix-Like y su integración con la programación Shell Script de estos sistemas.

Descripción general de la práctica

La práctica consiste en la creación de un generador de sorteos de una lotería 6/15 con informes de resultados que tiene dos partes diferenciadas, aunque integradas en una única solución común. La parte de Shell Script y la parte de C.

La parte de Shell Script consistirá en un programa al que se le pasarán como parámetros el número de sorteos y el número de jugadores.
Con la sintaxis del siguiente ejemplo:

```
$ loteria 15 10
```

Donde 15 sería el número de sorteos y 10 el número de jugadores.

El programa **loteria** (Shell Script) lanzará 15 veces al programa **sorteo** (en C) al que le pasará en cada llamada el número de sorteo de que se trate y el número de jugadores (todos los sorteos se realizarán con los mismos jugadores).

El programa **sorteo** (en C) creará tantos hijos (jugadores) como se indique que comprobarán su apuesta con la combinación generada por el padre. Es decir el programa **sorteo** tendrá un código para el padre y un código para el hijo. El padre genera una combinación ganadora y los hijos generan una combinación (apuesta) cada uno. Cada hijo comparará su apuesta con la combinación del padre y contestará al padre diciéndole cuantos aciertos tiene. El padre entonces genera un fichero resultados del sorteo indicando el premio correspondiente a cada jugador en función de los aciertos reportados.

La parte Shell Script, (**loteria**) leerá todos los ficheros de resultado para ir sumando para cada jugador los premios recibidos en cada sorteo, con lo que confeccionará un informe de ganancias que mostrará por pantalla.

Hay que escribir por tanto dos programas, **loteria** y **sorteo**, en Bash y C respectivamente.
El programa **lotería** llamará n veces al programa **sorteo ,**tras las cuales leerá los ficheros generados por éste para confeccionar un resultado que mostrará por pantalla.

La comunicación básica entre los padres e hijos del programa sorteo será mediante tuberías (pipes) y mediante el valor de terminación del hijo (exit), es decir el padre genera la combinación ganadora y la volcará en un pipe. El hijo generará una combinación aleatoria y la comparará con la del padre que la lee del pipe. Con esto determinará cuantos aciertos tiene, informando al padre mediante la función de terminación *exit(aciertos)*.

La comunicación entre el shell script **loteria** y el programa C **sorteo** será mediante fichero.
El programa **sorteo** generará un fichero con el resultado en premios del sorteo, que leerá el programa **loteria** una vez terminados todos los sorteos. El nombre de los ficheros indica su contenido con la siguiente sintaxis (ver ejemplo). **SnR** Será el nombre del fichero de resultados del sorteo número n

Ejemplo: S1R (Resultados del sorteo 1 para 10 jugadores, uno por línea)

```
#Resultados Sorteo 1
0
10
0
50
500
0
10000
0
0
0
```

La primera línea es un comentario que se puede omitir. El programa lotería deberá funcionar correctamente tanto si el fichero tiene el comentario como si no.

Cada línea es el importe en premios de un jugador, línea 1 jugador 1, línea 2 jugador 2, etc... Las líneas a cero (0) indican que el jugador no tiene premio en el sorteo (no ha acertado al menos 3 números, ver más adelante la tabla de premios). El jugador 2 tiene 3 aciertos (10€), el jugador 4 tiene 4 aciertos, el 5 tiene 5 aciertos y el 7 tiene 6 aciertos.

Desarrollo de la práctica, descripción y especificaciones

Programa Lotería (Bash)

El programa **loteria** abrirá el fichero de resultados de cada sorteo y sumará acumulando los premios de cada jugador para finalmente informar por pantalla de los premios totales tras todos los sorteos. Puede leerlos todos al final o conforme se van realizando los sorteos, como se quiera. Cuando se vuelva a llamar al programa **loteria** primero eliminará los ficheros de resultados de la ejecución anterior si los hubiera.

- Realizará el control de los parámetros, de forma que mostrará un error si los rangos no se ajustan a lo especificado para el programa sorteo o no se pasa alguno de ellos puesto que son obligatorios. En caso de error mostrará la ayuda de utilización del programa.
- Analizará el directorio de trabajo para determinar si existen ficheros de resultado de una ejecución anterior, en cuyo caso procederá a borrarlos.
- Realizará un bucle para llamar tantas veces al programa sorteo como números de sorteos se indiquen como parámetro, pasando cada vez el número de sorteo correspondiente y el número de jugadores al programa sorteo según la sintaxis de este.
- Tras la ejecución de cada sorteo en el directorio de trabajo se encontrará el fichero de resultados.
- Abrirá uno a uno todos los ficheros de resultados e irá llevando la suma de las ganancias de cada jugador.
- Presentará un informe de resultados con el siguiente formato:

```
Informe de resultados
Numero de sorteos: 15 Numero de jugadores: 10
Jugador 1: Total premio nnnn Euros.
Jugador 2: Total premio nnnn Euros.
…….
Jugador 10: Total premio nnnn Euros.
```

Programa sorteo (C):

El proceso padre:

- Recibirá como parámetro dos valores:
  - el número de sorteo
  - el número de hijos (jugadores) que participan en el sorteo.
- Realizará la validación oportuna de los parámetros de tal forma que:
  - El número de sorteo puede ser cualquier numero del rango 1..15
  - El número de jugadores puede ser cualquier número del rango 1..10
  - Los parámetros son obligatorios.
  - Cualquier error de parámetros provocara la terminación del programa devolviendo -1 (e informando del error por pantalla), lo que será tenido en cuenta por el programa **loteria**.
- El padre creará tantos procesos hijos como se indiquen, y registrará el pid de cada hijo en el array en su estructura correspondiente.
- Llevará el control del pid y el número de orden de cada hijo en un array de estructuras HIJO (Ver más adelante).
- El padre no esperará la terminación del hijo para crear otro hijo.
- Cuando el padre haya terminado de crear los hijos esperará la confirmación de apuesta de todos los hijos.
- Generará una combinación ganadora que le hará llega a los hijos, mediante Pipe o memoria compartida
- El padre esperará la terminación de todos sus hijos que en su valor de retorno, *exit(&status)*, devolverán el número de aciertos que han tenido.
- El padre generará un fichero **SnR** con los resultados obtenidos por todos los hijos en ese sorteo n

- El padre generará el fichero de resultados con la sintaxis especificada y con la siguiente tabla de premios.
  - Menos de tres aciertos premio 0€
  - Tres aciertos premio 10€
  - Cuatro aciertos premio 50€
  - Cinco aciertos premio 500€
  - Seis aciertos premio 10.000€

El proceso hijo:

- Generará una combinación o apuesta
- Cada hijo tendrá una apuesta propia
- En el caso de implementar una comunicación bidireccional con señales:
  - Comunicará (enviará una señal) al padre que le indica que ya ha realizado su apuesta
  - Esperará una señal del padre para poder leer la combinación ganadora
- En el caso de implementar una comunicación unidireccional con señales:
  - Cuando reciba la señal el hijo podrá leer la combinación ganadora
- Deberá leer la combinación generada por el padre (combinación ganadora)
- Esta lectura la podrá hacer mediante un Pipe o mediante memoria compartida
- Comparará su apuesta con la combinación del padre para determinar el número de aciertos
- Terminará su ejecución pasando el número de aciertos como parámetro de *exit()*.

Sincronización:

En la práctica se producirán dos sincronizaciones entre padre e hijos que serán resueltas de distintas formas (en distintas versiones de la práctica). El siguiente esquema recoge las dos sincronizaciones que se producen en el programa sorteo (en negrita) y algunas formas para poder realizar dichas sincronizaciones. En función de qué combinaciones seleccionemos tendremos una u otra versión

- **Sincronización 1 : Recoger información de los hijos**
  - Mediante Señales
  - Mediante Semáforos
- **Sincronización 2 : Lectura de la combinación ganadora del padre**
  - Mediante Señales + Pipe
  - Mediante lectura del Pipe solamente
  - Mediante Señales + Memoria Compartida
  - Mediante Semáforos + Memoria Compartida

Versiones

Con las distintas sincronizaciones y formas de implementar cada una de ellas se plantea al alumno que realice varias versiones del programa sorteo, que aunque la funcionalidad sea la misma, su implementación difiere en la forma de realizar las sicronizaciones y comunicaciones entre el proceso padre y los procesos hijo.

- Versión 1 : Comunicación y sincronización con señales bidireccionales entre padre e hijos y el uso de Pipes
- Versión 2 : Comunicación y sincronización con señales unidireccionales entre padre e hijos y el uso de Pipes
- Versión 3 : Comunicación y sincronización con semáforos y Pipes.
- Versión 4 : Comunicación y sincronización con semáforos y memoria compartida.

Versión 1

Sincronización entre padre e hijos utilizando señales en ambas direcciones, es decir, señales que envía el padre a todos los hijos y señales que cada hijo envía al padre.

El prototipo del código a utilizar en esta versión sería Esqueleto Versión 1

En esta versión, la comunicación con señales de los hijos al padre no puede hacerse con señales normales. Cuando la señal es recibida, el proceso padre pasa a la función manejadora de la misma. Como todos los hijos enviarán la señal al mismo tiempo, el padre recibirá un aluvión de señales de los hijos. Mientras está en el manejador de la señal, pueden llegar nuevas señales. Las señales normales no se encolan y si el padre recibe señales mientras está en la función manejadora, estas se pierden y por tanto perderá la cuenta de cuantos hijos se la han enviado. Esta problemática se resuelve si los hijos envían señales de tiempo real, que el sistema si encola y no deja que se pierdan. Revisar esto en Señales de Tiempo Real.

- Cada hijo enviará una señal al padre indicando que ha realizado su apuesta.
- El padre espera la llegada de todas las señales, para lo que entra en bucle hasta que un contador de señales llegadas sea igual que el número de hijos. El contador se incrementa un contador en la función manejadora.
- Los hijos esperan la llegada de la señal del padre que les permite leer del Pipe.
- Cuando llegue esa señal leen del pipe, comprueban sus aciertos y terminan.

Versión 2

Esta versión es simplemente una mejora respecto a la anterior. Como una lectura en un pipe, donde no hay datos escritos, provoca un bloqueo en el proceso lector, podemos usar este hecho para evitar que el padre envíe las señales a los hijos que les permiten leer del Pipe.

El prototipo del código a utilizar en esta versión sería Esqueleto Versión 2

- Cada hijo tras generar su apuesta envía su señal al padre indicando que ha realizado su apuesta.
- El padre espera la llegada de todas las señales, para lo que entra en bucle hasta que un contador de señales llegadas sea igual que el número de hijos. El contador se incrementa un contador en la función manejadora.
- Los hijos leen del pipe, pero si lo hacen antes de que el padre vuelque en el Pipe la combinación se bloquearán.
- Cuando el padre escriba en el Pipe los hijos se desbloquean, comprueban sus aciertos y terminan.

Versión 3

En esta versión introducimos el uso de semáforos para realizar la sincronización entre padres e hijos y quitamos la necesidad de sincronizarlos con las señales que llegan de los hijos. Es decir, quitamos las señales en tiempo real y las sustituimos por semáforos.

El prototipo del código a utilizar en esta versión sería Esqueleto Versión 3

- El padre crea un conjunto de semáforos, uno para cada hijo y otro para el mismo.
- El mecanismo de sincronización será una barrera múltiple donde el padre actúa como controlador.
- El padre quedará parado sin generar la combinación hasta que todos los hijos hayan apostado.
- Cuando un hijo apuesta señaliza la barrera y continúa (un signal no bloquea) quedando bloqueado en la lectura del Pipe
- Cuando todos los hijos han señalizado la barrera el padre podrá continuar, con lo que generará la combinación ganadora sacando a los hijos del bloqueo del Pipe.

Versión 4

En esta versión vamos a prescindir del Pipe. Por tanto necesitamos otro mecanismo para hacerle llegar la combinación ganadora a los hijos. Utilizaremos memoria compartida entre el padre y los hijos. El padre escribirá en la memoria compartida la combinación ganadora que leerán los hijos directamente de la memoria. Ahora, como ya no utilizamos el pipe donde los

hijos se quedaban esperando, es necesario sincronizar con semáforos a los hijos para que esperen por la combinación ganadora. Utilizaremos un semáforo para cada hijo, donde se quedarán esperando a que el padre les de paso para acceder a la memoria.

El prototipo del código a utilizar en esta versión sería Esqueleto Versión 4

- El padre crea un segmento de memoria compartida, ver Sincronización Padre-Hijos con semáforos y memoria compartida.
- El padre crea un conjunto de semáforos, uno para cada hijo y otro para el mismo.
- El mecanismo de sincronización será una barrera múltiple donde el padre actúa como controlador.
- El padre quedará parado sin generar la combinación hasta que todos los hijos hayan apostado.
- Cuando un hijo apuesta señaliza la barrera y continúa (un signal no bloquea).
- El hijo se quedará bloqueado en su semáforo esperando que el padre le de paso para acceder a la memoria compartida.
- Cuando todos los hijos han señalizado la barrera el padre podrá continuar, con lo que generará la combinación ganadora sacando a los hijos del bloqueo del semáforo.

Requisitos

- Hacer un código modular y documentado :
  - main.c : Incluirá únicamente la función main que llamará al resto de funciones
  - func.c : Tiene las funciones, con cabecera
  - func.h : Prototipos y constantes
- Utilizar make para compilar todo el proyecto utilizando el makefile correspondiente.
- Eliminar correctamente los ficheros temporales
- Cerrar correctamente los pipes (si procede en función de la versión)
- Procesos hijos deben terminar correctamente y no después que el padre.
- Generar funciones robustas a los posibles fallos en su funcionamiento.
- El código fuente debe estar comentado adecuadamente. Ni excesos ni defecto.

Estructura básica HIJO.

Puede ser ampliada y modificada convenientemente.

```
typedef struct
{
int pid ; //pid del hijo
int num ; //numero de orden del hijo
long premio; //premio del hijo en un sorteo
} HIJO
```

← Entrada anterior

Entrada siguiente →

Escuela Politécnica Superior de Elche

Universidad Miguel Hernández

Miguel Onofre Martínez Rach