Sistemas Operativos

Examen Práctico
Convocatoria Ordinaria – 30 enero 2021

Disponemos de un conjunto de ficheros en **C** con sus respectivas cabeceras **.h** que una vez correctamente compilados mediante **make** generan un ejecutable llamado **procesar** que generara unos ficheros de resultados. Se dispone a su vez de un Shell script en Bash llamado **escribir.sh** que lanza al ejecutable **procesar**, recoge datos de los ficheros de resultados y presenta un informe por pantalla.

Para aprobar es necesario aprobar ambas partes por separado.
Parte C: 6 Puntos. Parte Bash: 4 Puntos

Técnicas utilizadas:

- **Pipes**
- **Semáforos**

Editar correctamente los bloques **//Begin TODO … //End TODO** colocados en los ficheros **procesar.c**, **func.c** y **escribir.sh** para que realicen la siguiente tarea:

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****.*

En el prototipo de los ficheros existen constantes definidas que no se deben cambiar. Entre otras cosas definen el número de hijos máximo, números de jobs máximo, etc… y deberían usarse entre otras cosas para el control de parámetros.

El alumno dispone del ejecutable ***procesar_profesor*** *que puede utilizar para llamarlo desde la parte bash aunque no haya desarrollado la parte C, o bien para que le genere los ficheros de resultados.*

Control de parámetros:

Dos parámetros obligatorios para el script bash y para el programa C. numHijos y numJobs con los rangos válidos como se muestra a continuación.

```
escribir.sh <numHijos> <numJobs>
                [1..26]     [1..100]
procesar <numHijos> <numJobs>
                [1..26]     [1..100]
```

Funcionamiento:

El usuario lanzará el script **escribir.sh**, pasándole el número de hijos y el número total de Jobs que tienen que realizar. Este script, tras el control de parámetros, lanzará una vez al ejecutable **procesar** pasándole los

parámetros. Cuando termina el proceso, se habrán generado dos ficheros en el directorio de trabajo, que el script leerá para mostrar un informe de resultados.

Parte C:

El programa **procesar** (parte C) simula la distribución de trabajos (Jobs) entre distintos procesos (procesos hijo) y que son asignados por el proceso padre de manera aleatoria a los hijos.

El proceso padre crea un el número solicitado de procesos hijos. A su vez recibe el número de trabajos que tienen que realizarse en total. Su tarea será distribuir los jobs entre los distintos hijos, esto lo hará de manera aleatoria, pudiendo haber hijos que no reciban Jobs y otros que reciban varios Jobs a procesar. Para cada Job el padre definirá una carga de trabajo de manera aleatoria entre 1 y 100, es decir, un job puede tener una carga 5 y otro job una carga 89.

Cada hijo, creado en orden por el padre, se encargará de imprimir una letra en un fichero común tantas veces como indique la carga de trabajo que le indica el padre. Así el hijo 0 imprimirá la letra 'A' tantas veces seguidas como indique la carga recibida por el padre. Por ejemplo, si el hijo 3 (correspondiente a la letra 'D' recibe un job con carga 30, imprimirá 30 veces el carácter 'D' en el fichero.

Este fichero es el mismo para todos los hijos, de forma que los hijos tendrán que conseguir acceso al fichero antes de poder escribir en él. Este control de acceso se realizará mediante un semáforo sMutex que habrá que declarar, crear y usar apropiadamente usando las funciones de la librería sem (sem.c y sem.h).

El padre comunica a los hijos la carga que tienen que ejecutar en cada job. Esto lo hará mediante un pipe. El padre declarará un array de estructuras HIJO que contienen los descriptores del pipe. Este array es heredado por los hijos. Cada hijo tendrá, en base al orden en que son creados por el padre, un índice en el array. A través de ese índice tiene acceso a su estructura y por tanto a su pipe por el que el padre le envía los Jobs. El hijo cerrará el canal de escritura y el padre el de lectura.

El hijo no sabe de antemano cuántos Jobs va a recibir. Cada hijo deberá entrar en bule para leer del pipe los Jobs que el padre le envíe. Saldrá del bucle cuando reciba un job cuya la carga sea 0 (constante JOB_END).

Cuando un hijo recibe un job, intentará acceder vía el semáforo sMutex a la función DoJob, a la que le pasará los parámetros apropiados y que internamente abrirá el fichero **jobsfile.txt** en modo 'append' y escribirá en él tantas letras como carga indique el job recibido. El hijo sabe que letra le corresponde por su orden de creación, al que le sumará 65 para obtener la letra apropiada. Analizar el código y usar las funciones que permiten obtener la letra asccii de un número y el número dada la letra.

El padre lanzará aleatoriamente por los pipes de los hijos, todos los Jobs. El padre actualiza los contadores de los Jobs pendientes de cada hijo para llevar la cuenta (este control ya está implementado). Cuando haya lanzado todos los Jobs, escribirá en el pipe de cada hijo un job de terminación o JOB_END con carga 0.

Cuando un hijo termina un job comunica este hecho al padre vía otro pipe también definido en la estructura HIJO. Al tratarse de un esquema de comunicación bidireccional por pipes entre el padre y el hijo, necesitamos dos pipes, uno para pasar datos del padre al hijo (pipe **pPC**) y otro para pasar datos del hijo al padre (pipe **pCP**), ambos definidos en la estructura HIJO. Tanto el padre como el hijo cierran los descriptores no usados de los pipes. El leerá del pipe el indicador JOB_DONE, escrito por el hijo cuando termina de imprimir un job. En ese momento el padre incrementa los contadores de la estructura HIJO.

El padre esperará entonces la terminación de los procesos hijo. Cuando todos los hijos hayan terminado, generará un fichero **jobcounts.txt** donde en cada línea pondrá el número de Jobs que se han asignado a cada hijo. El número de Jobs de cada hijo lo obtiene de la estructura del HIJO, donde se fue incrementando el número de Jobs procesados conforme recibía las señales de los hijos.

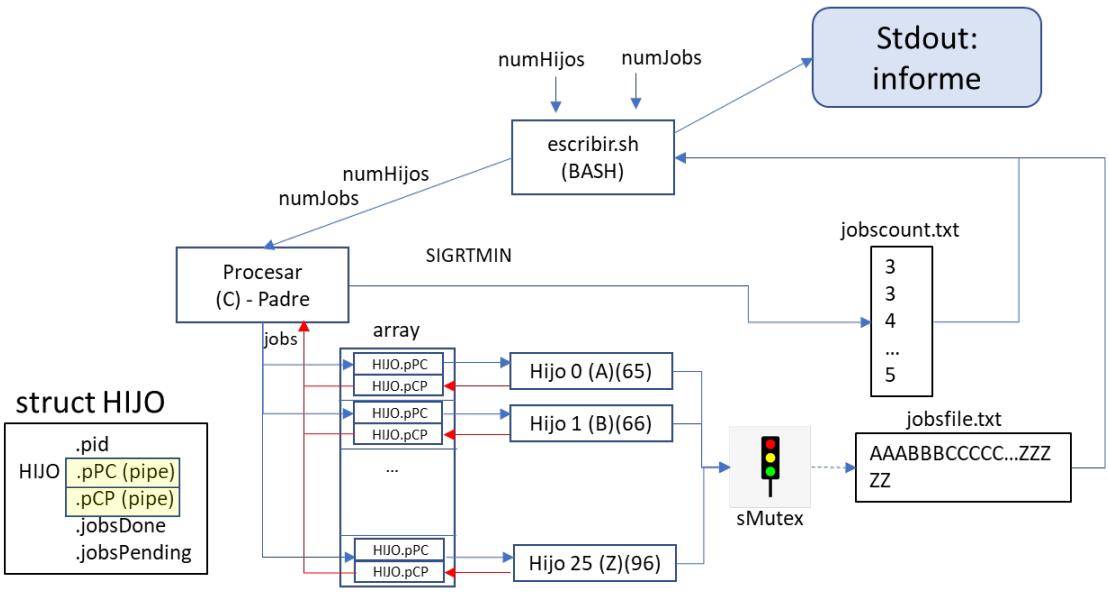
Parte Bash:

En el script escribir.sh (parte Bash), tras el control de parámetros, se declaran dos arryas, uno para contar las letras que ha escrito cada hijo y otro para almacenar el número de Jobs que ha procesado cada hijo. Esta información se obtiene respectivamente de los ficheros **jobsfile.txt** y **jobscount.txt**.

El script leerá letra a letra el fichero **jobsfile.txt** y en función de la letra leída, actualizará el array **letras** incrementando en uno el valor almacenado en el índice correspondiente del array. Recordar que cada letra corresponde a un hijo. El número de veces que aparece la letra A se actualizará en el índice 0 del array, la B en el 1 y así sucesivamente.

El array **jobs** almacenará el número de Jobs que cada hijo ha procesado. Esto lo hace leyendo línea a línea el fichero **jobscounts.txt** donde en cada línea sólo hay un valor, que corresponde al número de Jobs del hijo de esa línea, cada línea es de un hijo.

Esquema:



Señales:

- El hijo utilizará la señal SIGRTMIN para comunicar al padre que ha terminado un job.
- Se utilizará un manejador ya programado para gestionar la recepción de las señales.

Semáforos:

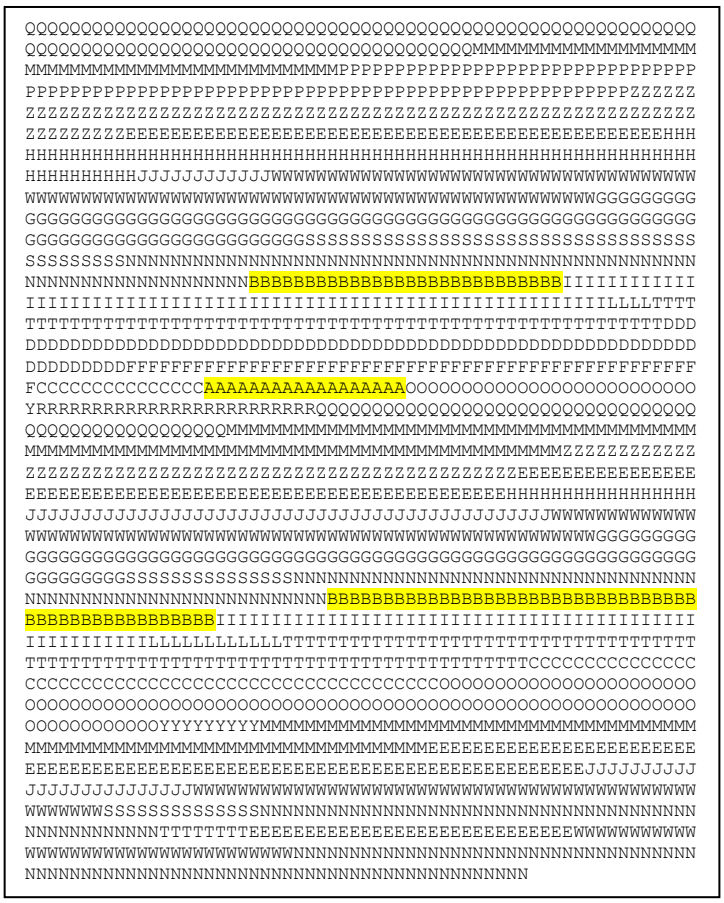
- Los hijos utilizarán el semáforo sMutex para acceder al fichero compartido de escritura.
- El semáforo se crea apropiadamente por el padre antes de crear a los hijos, con lo que lo heredan y se destruye cuando termine el padre.

- Se dispone de la librería sem.c y sem.h con las funciones para gestionar los semáforos.

Pipes:

- El padre creará un array de estructuras HIJO, donde se crearán los pipes en el campo apropiado.
- El padre escribe Jobs en los pipes de los hijos.
- Los hijos leen, cada uno de su pipe.

Ficheros de resultados:



La parte C genera dos ficheros, el fichero que utilizan todos los hijos para escribir es el fichero **jobsfile.txt.**

A la izquierda un ejemplo del fichero de resultados, **jobsfile.txt** lanzado para 26 hijos con 100 jobs:

Como vemos faltan letras (K,X,etc.), es decir, hay hijos que no reciben Jobs y otros reciben varios Jobs. El primer hijo (hijo 0) recibe 1 job, corresponde a la letra A, por eso en el fichero **jobsfile.txt** solo hay una cadena de letras 'A'. Sin embargo, hay 2 jobs para la letra B como se ve en el fichero hay dos cadenas de Bs.

Otras letras (hijos) no han recibido jobs, en el fichero **jobscount.txt** aparece un cero en su posición (Ejemplo a la derecha para 26 hijos y 100 job

```
1
2
2
1
4
1
2
2
2
3
0
2
3
5
2
1
2
1
3
3
0
0
4
0
2
2
```

A completar:

Las siguientes funciones **tienen bloques TODO** para completar por el alumno (indicado en el código):

escribir.sh:

- `CheckArguments()`
- `InitDAta()`
- `LoadResults()`
- `MAIN del programa.`

procesar.c:

- `int main( int argc, char *argv[] )`

El resto de funciones de dichos ficheros ya están completas y no deben modificarse.
El alumno utilizará las variables y constantes definidos en el código y podrá definir los que necesite.