Escuela Politécnica Superior de Elche

Grado en Ingeniería Informática
en Tecnologías de la Información

Sistemas Operativos

Examen Práctico
Convocatoria Diciembre Prácticas
29 Noviembre 2022

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 **procesar**.

Se dispone a su vez de un Shell script en **Bash** llamado **lanzar.sh** que lanza el ejecutable **procesar**, lee el fichero que éste genera y muestra un resultado por pantalla.

Para aprobar es necesario aprobar ambas partes por separado:

Parte C: 6 Puntos.
Parte Bash: 4 Puntos

Técnicas utilizadas:

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

Los ficheros **lanzar.sh** y **procesar.c** son plantillas o prototipos que forman una solución.

El alumno deberá entenderla y completarla.

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.

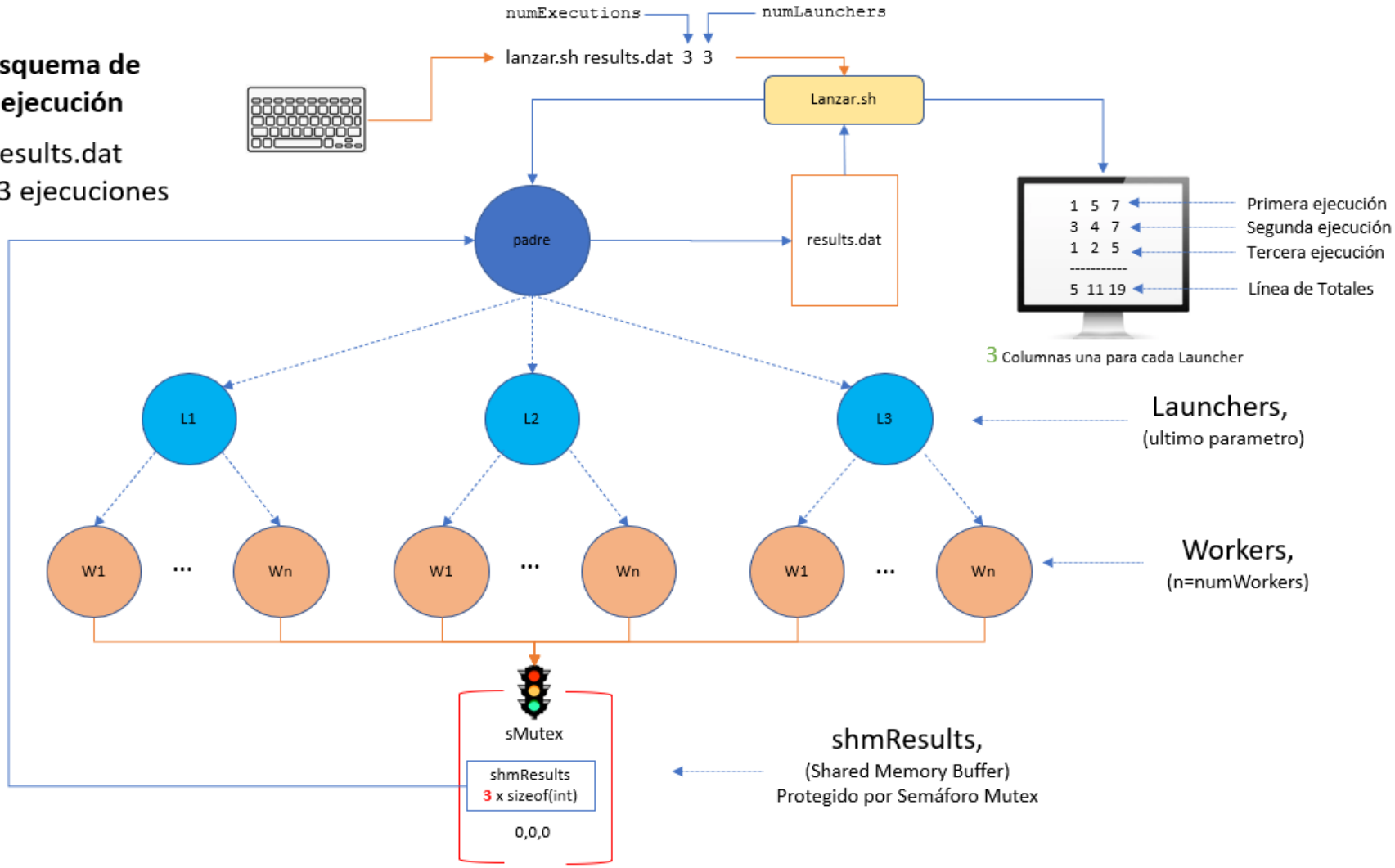
En el prototipo de los programas existen constantes y funciones definidas que no se deben cambiar, pero el alumno puede añadir las que necesite.

Ejecutando los programas **lanzar.sh** y **procesar** sin argumentos deberá mostrarse una guía de uso o sintaxis del script bash.

Se puede usar el ejecutable **procesar_profesor** para probar la parte Bash sin tener realizada la parte C.

Esquema de ejecución

Results.dat
con 3 ejecuciones



Parte Bash:

La Figura muestra el esquema de la solución pedida, seguir la explicación con la figura.

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

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

- El nombre del fichero de resultados a generar (ejemplo: results.dat)
- El número de veces que se llama a **procesar** (numExecutions)
- El número de launchers que se utilizarán en **procesar** (numLaunchers).

Comprobar que los parámetros son correctos y que el parámetro `numExecutions` está dentro de su rango válido [1:10] y el parámetro numLaunchers está dentro del suyo [1:5]

```
$ lanzar.sh results.dat 3 3
```

El proceso llamará `numExecutions` veces al programa C **procesar** pasándole 4 parámetros cada vez, éstos son:

- numLaunchers // Número de procesos Launcher. Rango Válido []
- numWorkers // Número de procesos Worker a usar por cada Launcher. Rango Válido [1:20]
- workLoad // Número de segundos que emulan el trabajo de un Worker. Rango Válido [1:5]
- resultsFilename //Nombre del fichero de resultados (results.dat en el ejemplo anterior)

El script **lanzar.sh** generará aleatoriamente los parámetros `numWorkers` y `workLoad` dentro del rango permitido y llamará al proceso **procesar**, ejemplo:

```
Procesar $numLaunchers $numWorkers $workLoad $1  // $1: parámetro que recoge el nombre del fichero
```

El proceso **procesar** generará un fichero con el nombre requerido cada vez que se ejecute. El contenido del fichero son tres valores numéricos, uno en cada línea. Ejemplo de fichero:

```
346
141
1029
```

El Bash **lanzar.sh** deberá comprobar que el fichero no exista antes de llamar a procesar, **si existe debe borrarlo**.

Tras cada ejecución de **procesar**, **lanzar.sh** leerá el fichero de resultados (entre ejecuciones tendrá el mismo nombre, el requerido), mostrando en una línea los valores leídos correspondientes a cada launcher. A su vez acumulará el valor de cada launcher en un array para mostrar, en una línea final, el total obtenido por launcher. Un ejemplo con 3 ejecuciones de **procesar** sería: (se muestran los ficheros y el resultado)

Fichero results.dat (3 veces)

```
942
740
913
```

```
777
943
801
```

```
620
984
849
```

Salida por pantalla de **lanzar.sh.** Cada fila es una ejecución, cada columna corresponde a un Launcher. Una fila con el total de cada launcher.

```
   942      740      913
   777      943      801
   620      984      849
-------------------------
  2339     2667     2563
```

Parte C:

El ejecutable **procesar** deberá comprobar que le llegan los parámetros y están en su rango válido. Además, deberá comprobar que el fichero de resultados que le piden como parámetro no existe. **Si existe deberá terminar con error**.

Procesar consta de un proceso padre que genera 3 procesos hijo, que llamaremos Launchers.

Cada proceso Launcher generará tantos hijos, que llamaremos Workers, como indique el parámetro `numWorkers`.

En el esquema de la figura L1 a L3 son los 3 Launchers que genera el padre. Siempre genera 3 Launchers. W1 a Wn son los Workers que genera cada Launcher.

Cada proceso Worker simulará la ejecución de un trabajo con una duración de `workLoad` segundos, esto lo hace llamando a la función DoWork(). Esta función devolverá un valor entero simulando el resultado del trabajo.

Hay un array de 3 valores enteros de memoria compartida que el padre genera antes de crear los procesos. Cada índice de este array (shmResults) corresponde a un Launcher. Estos valores se inicializan a 0. Todos los Workers de un Launcher suman su valor obtenido de DoWork() en el índice del array correspondiente a su Launcher.

Por tanto, todos los Launchers deben acceder al array de memoria compartida en exclusión mutua. El padre declara un semáforo sMutex para tal fin. Utilizar las funciones wait y signal de la librería sem.h de manera apropiada para garantizar la actualización correcta en exclusión mutua.

Cuando todos los procesos Launcher y Workers han terminado, el padre lee el fichero de memoria compartida y genera el fichero de resultados, colocando los valores uno en cada línea.

En el bucle de creación de los hijos tener en cuenta que los procesos Launcher y Workers no iteren para crear nuevos procesos cuando no sea necesario.

En el código hay comentarios para ayudar a la resolución del problema.

Completar el código Bash y C para realizar estas tareas.