Sistemas Operativos

Examen Práctico
Convocatoria Extraordinaria – 5 Septiembre 2020

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

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

Técnicas utilizadas:

- **Señales**
- **Semáforos**
- **Memoria Compartida**

Editar correctamente los bloques **//Begin TODO … //End TODO** colocados en los ficheros **pintar.c**, **func.c** y **pintar.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****. A su vez hay un conjunto de sentencias de salida a pantalla cuyo objetivo es facilitar la traza pero que están comentadas, el alumno las puede descomentar y añadir las que considere para poder hacer un seguimiento de la ejecución del programa.*

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

El alumno dispone del ejecutable ***pintarprofesor*** *que puede utilizar para llamarlo desde la parte bash para que le genere los ficheros de resultados.*

Control de parámetros:

```
pintar.sh <numHijos> <maxPiezas>
          [2..12]     [1..10]
pintar <numHijos> <maxPiezas>
        [2..12]     [1..10]
```

Funcionamiento:

El programa pintar (parte C) simula la distribución de tareas de pintura de piezas entre distintos procesos (procesos hijo) que son gobernados por un proceso dispatcher (proceso padre).

El proceso padre crea un numero determinado de procesos hijos y en el momento de su creación les indica el número de piezas que cada proceso debe pintar.

Tanto el proceso padre como los hijos comparten un array de memoria donde cada hijo utiliza una posición.

Simulamos que cada hijo selecciona una pieza a pintar e indica en su posición del array compartido cuanto tiempo le va a llevar pintar dicha pieza y espera a que el padre le diga que la pinte. Esto lo realizan todos los hijos. El proceso padre, seleccionará un hijo basándose en los tiempos que figuran en el array compartido. En esta versión, el algoritmo utilizado por el padre es seleccionar el mayor tiempo de entre los disponibles.

Una vez que el padre ha seleccionado el tiempo del array, le comunica al hijo que usa esa posición que comience la pintura de la pieza. Cuando el hijo termina de pintar, si tiene mas piezas pendientes, vuelve a colocar en su posición del array el tiempo que le llevará pintar la siguiente. Así hasta que no le queden más piezas por pintar y el proceso hijo termine.

El proceso padre no espera a que todos los hijos comuniquen su tiempo. Mientras haya piezas pendientes de pintar, el padre seleccionará periódicamente un tiempo de entre los que haya disponibles en el array compartido. Esta selección la hace en un bucle con una espera entre iteraciones de 1 segundo, trascurrido el cual vuelve a seleccionar el mayor de los tiempos disponibles.

El tiempo total de pintura de las piezas depende del delay del padre, del algoritmo utilizado para seleccionar al hijo que debe pintar y del orden en que los hijos seleccionan las piezas.

Para que en la fábrica puedan analizar posteriormente qué algoritmo es el más apropiado, nuestro sistema debe proporcionar un indicador del retraso que acumula cada hijo para todas sus piezas desde que indica el tiempo que le va a llevar pintar hasta que el padre le da la orden y la pinta finalmente. Los cálculos del delay para cada pieza y su control en un array de tiempos de pieza (tPiezas) ya están codificados y comentados en el código.

El proceso pintar (parte C) genera un fichero de resultados mediante una función ya codificada que utiliza ese array tPiezas para generarlo.

En el script pintar.sh (parte Bash) nos limitamos a recoger la información de los ficheros de resultados de todos los hijos y mostrar un informe por pantalla de los delays de cada hijo y el total de todos.

Consideraciones:

- Para implementar al array compartido, hay que utilizar un segmento de memoria compartida debidamente inicializado y dimensionado.
- Para acceder al array de memoria compartida hay que utilizar un semáforo mutex tanto por el padre como por los hijos
- Para que los hijos esperen la señal del padre utilizarán pause en bucle a la espera de la llegada de una señal del padre
- Para que el padre sepa cuantas piezas quedan por pintar los hijos enviarán una señal al padre, indicando que han terminado el trabajo de la pieza.
- El padre realizará la selección de un tiempo del array cada 1 segundo, para lo que utilizará un bucle con la función sleep(1) para esperar ese tiempo.
- El padre saldrá del bucle de selección de piezas cuando no haya más piezas, cosa que sabrá porque la función manejadora de la señal del hijo decrementará el contador de piezas pendientes.

Señales:

- El padre utilizará la señal SIGUSR1 para comunicar al hijo que debe pintar. No es una señal de tiempo real.
- El hijo utilizará la señal SIGRTMIN para comunicar al padre que ha terminado una pieza. Es una señal de tiempo real.
- Se utilizará el mismo manejador para gestionar la recepción de los dos tipos de señales.

Semáforos:

- Tanto padre como hijos utilizarán el semáforo shmMuthex para acceder al array compartido de tiempos.
- 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.

Memoria compartida:

- Antes de la creación de los hijos se dimensiona y crea el array de memoria compartida para que los hijos lo hereden.
- Una vez creado, tanto el padre como los hijos deben vincularse al mismo para usarlo, y desvincularse cuando lo dejen de usar
- El padre destruirá el segmento de memoria apropiadamente.

Fichero de resultados

Nombre: ChildNNData.txt donde NN es el número de hijo, 01, 02 …09,10, etc..
Formato y ejemplo:

```
<NumPiezas>                                   03
<tPrevisto>; <tFinal>; <tDelay>               10.000000; 10.000321; 0.000321
Repetir línea anterior si más piezas          8.000000; 8.000252; 0.000252
                                              5.000000; 5.000243; 0.000243
```

Funciones a completar:

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

pintar.sh:

- `CheckArguments()`
- `LoadPiezas()`
- `LoadDelays()`
- `LoadPiezas()`
- `MAIN del programa.`

pintar.c:

- `void SignalWorkDone(int numHijo)`
- `void SignalWorkStart(int pid)`
- `void Handler(int signo, siginfo_t *info, void *context)`
- `void CodigoHijo(int ixChild, int piezasHijo)`
- `int main( int args, char *argv[] )`

func.c:

- `void InitializeSharedMemory(int *shmTiempos, int numHijos)`

Funciones disponibles ya completas:

Comentadas en el código

sem.c:

- `int sCreate(int seed, int value);`
- `void sWait(int semID);`
- `void sSignal(int semID);`
- `void sDestroy(int semID);`

func.c:

- `void LeerArrayPipe(int fdPipe, int *arrayDatos, int longitud);`
- `void EscribirArrayPipe(int fdPipe, int *arrayDatos, int longitud);`
- `bool InArray(int *v, int len, int num);`
- `void PrintArray(char *name, int *array, int length);`
- `int RandInt(int M, int N);`
- `double Seconds(struct timeval start, struct timeval stop);`

pintar.c:

- `int SelectWork(int *array, int length)`
- `void LaunchWork()`
- `void Paint(int tiempo)`
- `void SaveChildData(int ixChild, int piezasHijo, t_datos_pieza *tPiezas)`
-

pintar.sh:

- `Number()`
- `Sintaxis()`
- `StrLen()`
- `ToNumber()`
- `GetFileName()`
- `MostrarResultados()`