SISTEMAS OPERATIVOS

Memoria – Práctica 3

Grupo 80

Iván Aguado Perulero – 100405871

Javier Cruz del Valle – 100383156

Jorge Serrano Pérez - 100405987

Índice

Introducción 2
Descripción del código 2
queue.h 2
queue.c 3
costCalculator.c 4
Batería de Pruebas 5
Conclusión 7

Introducción

El objetivo de esta práctica es la familiarización del alumno con los servicios para la gestión de procesos que proporciona POSIX. Para ello se utilizarán las llamadas al sistema *pthread_create*, *pthread_join*, y *pthread_exit*, y para su sincronización se hará uso de mutex y variables condicionales. Se pretende codificar un sistema multihilo concurrente que calcule el coste de utilización de las máquinas de un centro de procesamiento. Es decir, se debe calcular cuánto hay que cobrar al cliente según el tipo de máquinas y el tiempo que las quiere utilizar, que se especificará en un fichero que sigue un formato específico.

Para ello se nos ha proporcionado un archivo .h donde codificar la cola, además de un código fuente de apoyo que contiene el Makefile que nos permite compilar el código, un probador que realiza una autocorrección y el fichero fuente de C que es donde deberemos codificar nuestras funciones.

Descripción del código

A continuación, expondremos la implementación utilizada para resolver el problema planteado. Para ello dividiremos la explicación en tres partes: la primera hará referencia al archivo "queue.h" donde se encontrará la interfaz que más tarde será utilizada en la cola, en segundo lugar, se resumirá el archivo "queue.c" que contiene los métodos de la cola y, por último, explicaremos el archivo principal "costCalculator.c" que se encargará de recoger y aunar todos los elementos anteriormente descritos para su correcta ejecución.

queue.h

Inicialmente, definiremos tres estructuras: la estructura "elemento" se encargará de dar forma a los objetos que serán introducidos en la cola por lo que para ello será necesario dos atributos enteros que guarden el tipo de máquina y el tiempo que es utilizada la misma. La siguiente estructura que encontraremos es la del "nodo" la cual se compondrá por un atributo elemento y dos atributos nodos que definirán el elemento anterior y posterior recursivamente. Siguientemente, tendremos la estructura de "cola" compuesta por dos atributos nodo que

definirán el inicio y final de la cola, dos valores enteros encargados de definir la capacidad total de la cola además del tamaño actual de la misma y tres propiedades definidas como mutex/variables condición cuya función será administrar la concurrencia de los hilos dentro de la cola.

En último lugar, encontraremos planteados los seis métodos de la cola que se encargarán de su creación y destrucción, inserción y obtención de elementos y la comprobación tanto de si se encuentra la cola vacía como llena.

queue.c

Primeramente, encontramos tres métodos encargados de la creación, inserción y eliminación de los nodos correspondientes a las estructuras previamente definidas. A continuación, explicaremos las seis funciones de la cola:

Creación de la cola, este método obtendrá un tamaño inicial expresado como valor entero y su función será la de instanciar y asignar memoria para cada elemento que componga la cola. Finalmente comprobaremos la concurrencia y en último lugar devolveremos la cola.

Destrucción de la cola, este método cogerá la cola dada y eliminará cada nodo que la componga además de la propia cola, liberando el espacio de memoria asignado previamente a ambos. Comprobará la concurrencia y devolverá un entero para el control de errores.

Inserción en la cola, este método pasará como parámetros la cola donde se desea insertar y el elemento a insertar. Para ello comprobamos la concurrencia al inicio y final de la función e insertaremos un nodo en la cola con el elemento deseado aumentando en uno el tamaño actual.

Extracción de la cola, este método tomará la cola dada y devolverá el primer elemento de la cola eliminando el nodo en el que se encuentre y decrementando en uno el valor del tamaño actual de la cola. Para ello será necesario comprobar al inicio y final la concurrencia que pueda darse.

Confirmar cola vacía, este método pasará la cola como parámetro y comprobará si su tamaño actual es cero, devolverá un entero con valor uno en

caso positivo y cero en caso negativo. A su vez, será necesario comprobar la concurrencia al inicio y final de la comprobación del tamaño actual.

Confirmar cola llena, este método pasará la cola como parámetro y comprobará si su tamaño actual es igual a la capacidad máxima de la cola, devolverá un entero con valor uno en caso positivo y cero en caso negativo. A su vez, será necesario comprobar la concurrencia al inicio y final de la comprobación del tamaño actual.

costCalculator.c

El archivo principal se estructurará del siguiente modo:

Encontraremos un método encargado de abrir el archivo para almacenar todos sus valores, dentro de este método se harán las correctas comprobaciones de errores y se cargarán los datos para su posterior uso. En última instancia cerraremos el archivo y devolveremos el número de operaciones.

A continuación, encontraremos dos estructuras con un método asociado, serán:

Productores, compuesta por un atributo de elementos, dos enteros que marquen el inicio y final de actuación, y un atributo de tipo cola. Su función asociada será la de insertar las operaciones que se encuentren dentro de su intervalo definido por los valores enteros y finalizar el hilo correspondiente.

Consumidor , compuesta por un atributo de tipo cola y un valor entero. Su función asociada será la de extraer las operaciones de la cola para poder calcular el coste asociado a cada una de ellas y guardar el total que posteriormente se imprimirá por pantalla a su vez se finalizará el hilo correspondiente.

En último lugar encontramos el "main" donde leeremos los valores introducidos por pantalla e implementaremos las funciones y estructuras que se han definido anteriormente, además, controlaremos los posibles errores imprimiendo por pantalla el tipo de error y forzando el final de la ejecución de ser necesario. Se repartirán las operaciones y crearemos los hilos productores para que encolen concurrentemente las operaciones, crearemos el hilo consumidor encargado

de calcular el coste total e imprimir por pantalla y una vez se espere para la correcta finalización de cada uno los hilos finalizaremos con la ejecución.

Batería de Pruebas

Prueba	Resultado esperado	Resultado obtenido
Mas de 4 argumentos	[ERROR]: El número de argumentos debe ser 4.	Correcto
Fichero inexistente	[ERROR]: No se pudo abrir el fichero.	Correcto
Fichero con texto	El fichero se lee con normalidad.	Correcto
Fichero imagen	[ERROR]: No se pudo leer el número de operaciones.	Correcto
Fichero sin texto	[ERROR]: No se pudo leer el número de operaciones.	Correcto
Directorio	[ERROR]: No se pudo abrir el fichero.	Correcto
Número de productores no valido (0 o negativo)	[ERROR]: Número de productores no válido.	Correcto
Número de productores no valido (valor no numérico)	[ERROR]: Número de productores no válido.	Correcto
Tamaño del buffer no válido (0 o negativo)	[ERROR]: Tamaño del buffer no válido.	Correcto
Tamaño del buffer no válido (valor no numérico)	[ERROR]: Tamaño del buffer no válido.	Correcto
Número de operaciones no válido (0 o negativo)	[ERROR]: Debe insertar al menos una operación a realizar.	Correcto

Número de operaciones no válido (valor no numérico)	[ERROR]: No se pudo leer el número de operaciones.	Correcto
Id no válido (valor no numérico)	[ERROR]: No se pudo leer el valor de id.	Correcto
Tipo no válido (valor distinto de 1, 2 o 3)	ERROR]: El valor de tipo es incorrecto.	Correcto
Tipo no válido (valor no numérico)	[ERROR]: No se pudo leer el valor de tipo.	Correcto
Tiempo no válido (valor no numérico)	[ERROR]: No se pudo leer el valor de tiempo.	Correcto
Id no válido (valor double)	[ERROR]: No se pudo leer el valor de tipo.	Correcto: la función fscanf lee el valor entero. Por ello, cuando llega a la coma da error del siguiente valor.
Tipo no válido (valor double)	[ERROR]: No se pudo leer el valor de tiempo.	Correcto: la función fscanf lee el valor entero. Por ello, cuando llega a la coma da el error del siguiente valor.
Tiempo no válido (valor double)	[ERROR]: No se pudo leer el valor de id.	Correcto: la función fscanf lee el valor entero. Por ello, cuando llega a la coma da el error del siguiente valor.

Conclusión

Para concluir, añadimos aquí los problemas principales y nuestras valoraciones personales.

En primer lugar, hay que destacar que el enunciado proporcionado y la ayuda con el pseudocódigo han sido de mucha utilidad para estructurar de manera correcta la práctica y saber los pasos a seguir.

Sin embargo, una parte del código proporcionado, en concreto el *queue.h* y el *queue.c*, creemos que podría haber sido explicado con más claridad (qué debe contener, por qué se implementa de esa manera, como se utiliza, etc.), aunque también es cierto que debido a la situación actual del COVID-19, se han quitado clases de prácticas en las que se podría haber resuelto esta cuestión.

Por otro lado, el uso de distintas instrucciones no empleadas anteriormente, como puede ser la creación y manejo de hilos, los mutex y las variables condicionales, ralentiza el proceso al principio, pero simplemente por la experiencia que supone enfrentarse a algo nuevo. Se trata de un problema leve, pero para tener en cuenta, a pesar de que los profesores no podáis hacer mucho al respecto para solucionarlo.

Por último, creemos que deberían revisarse posibles errores antes de entregar a los alumnos el código inicial. Me refiero a los errores que ha habido por ejemplo con el Makefile o el corrector, tanto en esta práctica como en las anteriores. Queremos destacar que nuestra intención es informar de problemas que existen y que se pueden solucionar con facilidad. En ningún momento pretendemos quejarnos ni menospreciar el trabajo que hacen los profesores.

En cuanto a las valoraciones personales, al comienzo parece una práctica sencilla. Sin embargo, conforme vas avanzando te das cuenta de pequeños matices que parecen poca cosa pero que son importantes a la hora de la ejecución. Además, creemos que a pesar de la situación actual que estamos viviendo, esta tercera práctica se ha manejado lo mejor posible.