PROGRAMACIÓN CON THREADS

Práctica 2 – Sistemas Operativos

2018-2019

Mikayel Mardanyan Petrosyan
3º GII

Índice de contenidos

1. Introducción

El objetivo de esta práctica es la adquisición de conocimientos en el uso de threads. Haremos uso de estos para implementar un problema que simule el comportamiento de un parking de vehículos. Tendremos que incluir características como plazas ocupadas y control de entrada y salida.

A nuestro parking entrarán dos tipos de vehículos: coches y camiones. Los coches ocuparán una plaza y los camiones, dos plazas contiguas. El parking dispondrá de entrada y salida, y para acceder y salir existirá un dispositivo de control.

El comportamiento de los vehículos será el siguiente. Un vehículo llega al control de acceso, si hay alguna plaza en la que pueda aparcar, la ocupará y permanecerá aparcado un tiempo aleatorio. Después, intentará salir pidiendo permiso al control de acceso. Una vez haya salido, volverá a esperar un tiempo aleatorio y volverá a intentar entrar. Esto se repetirá hasta que detengamos la ejecución del programa.

2. Autoría

El autor de la práctica y de esta memoria es únicamente Mikayel Mardanyan Petrosyan.

3. Descripción del código

Para el planteamiento del programa se ha intentado que este sea lo más parecido posible a un parking real.

Una de las decisiones que se ha tomado es el número de entradas. ¿Hay una única vía de acceso que sirve de entrada y salida, o son dos vías diferentes? Pues bien, la decisión tomada es que se implementará una sola vía de acceso, para entrar y salir.

Cuando un vehículo llega y no tiene sitio para aparcar, se queda esperando en el control y deja que salgan vehículos hasta que pueda aparcar él, pero no deja “entrar” a más vehículos. Este es un aspecto trivial que ha llevado al uso de dos mutex, uno para entrar y otro para salir, ya que con uno solo no se puede implementar el comportamiento buscado (se explica más adelante).

Cuando se ejecuta el programa, este se encarga de crear el número de plazas que habrá disponibles y los vehículos. Las plazas se implementan mediante un array de enteros y cada vehículo será un thread con un número de identificación. Ni el número de plazas ni el número de vehículos se conoce en tiempo de compilación, de modo que se reserva memoria dinámica para ellos mediante la función ‘malloc’. Se parsean los argumentos usando la función ‘atoi’ y se obtienen estos parámetros.

Al inicio, el programa realiza un control de errores para los datos introducidos. Nos obliga a introducir al menos dos argumentos, el número de plazas y el número de plantas, y comprueba que ambos valores sean mayores que 0. En nuestro caso, solo se acepta ‘1’ como el valor para plantas; en caso contrario se muestra un mensaje de error por pantalla. Si no se introduce ningún argumento más, el número de coches se calcula automáticamente (el doble de las plazas por el número de plantas) y el numero de camiones se establece en 0. Si se quiere especificar el número de coches y/o camiones, se debe introducir valores mayores o iguales que 0 y que cumplan la condición ‘número de coches’ + ‘numero de camiones’ > 0.

A continuación, se inicializan los mutex de entrada y salida a NULL y se crean los threads de coches, que llamarán a la función ‘coche’, y los threads de camiones, que llamarán a la función

‘camión’. Una vez realizadas estas acciones, se entra en un bucle infinito y se deja que se ejecuten los threads hasta que el usuario detenga la ejecución.

Los vehículos reciben por referencia un parámetro, un identificador que en caso de que sea un coche empieza en 1, y en caso de que sea un camión empieza en 101. Cada vehículo tiene dos variables locales, una para guardar su identificador y otra para guardar el número de plaza donde aparca. Los camiones, ya que aparcan en dos plazas, tienen una variable local más. El vehículo va a estar siempre entrando y saliendo del parking, por lo que su código va a estar metido dentro de un bucle infinito. Primero se indica que no tiene ninguna plaza asignada y se hace que espere un tiempo aleatorio. Cuando este tiempo haya pasado, intentará entrar al parking. Aquí es cuando se tomó la crucial decisión de usar dos mutex. El vehículo que quiera entrar, tanto si es coche como camión, necesita coger los dos mutex, el de entrada y el de salida, en ese orden. El vehículo que coja el de entrada, intentará coger el de salida también. Si hay algún coche saliendo, esperará a que salga y volverá a intentar cogerlo. Esto se repite hasta que lo consiga. Cuando lo haga, comprueba si hay plazas libres. Si no hay, suelta el mutex de salida hasta que haya al menos una plaza libre o dos, dependiendo de si es un coche o un camión, respectivamente. Es importante que suelte el de salida y no el de entrada, ya que si suelta los dos (o si solamente tenemos un mutex para entrar/salir) está permitiendo que otro vehículo le ‘adelante’ e intente entrar antes que él. Esto no solo supone un problema porque los coches ‘se cuelan’ y porque queremos hacerlo lo más parecido a la realidad (los vehículos hacen fila en orden de llegada hasta entrar), sino que, en el caso de los camiones, estos pueden sufrir inanición. Si un camión llega, coge los dos mutex para entrar y se encuentra con una sola plaza vacía, debe soltar solo el de salida porque si permite entrar a algún coche, este le puede quitar la plaza, volviendo a coger el mutex ahora con 0 plazas libres. Esto es un problema, y el camión seguiría así hasta que se dé la casualidad de que salen dos coches seguidos. Con valores similares de plazas y coches es difícil que esto ocurra, pero con valores mayores sería bastante probable. Este comportamiento no es aceptable, así que se hace uso de dos, lo que también garantiza que los vehículos hagan una cola ordenada para entrar (un coche que sale del parking no puede volver a entrar antes que uno que ya estaba en la cola para entrar cuando este salió).

Una vez se tienen los mutex, se entra en sección crítica, segmento de código que modifica la memoria compartida. Cuando el coche consigue que haya plazas, recorre el array que representa al parking buscando la libre y la almacena en la variable local mencionada antes. Se marca esta plaza como ocupada, se resta 1 al número de plazas libres y se muestra por pantalla que el coche ha aparcado y cuál es el estado del parking. Se liberan los dos mutex y se sale de sección crítica. Ahora el coche permanece aparcado un tiempo aleatorio y se dispone a salir. Para poder salir necesita únicamente el mutex de salida. Esto puede hacer pensar que, si solo coge ese, paralelamente puede entrar otro vehículo, lo cual no debe pasar. Sin embargo, para poder entrar, un vehículo necesita poseer ambos, de modo que la implementación es correcta. En este momento se vuelve a entrar en sección crítica. Se marca la plaza ocupada como libre, se suma 1 al numero de plazas libres y se muestra por pantalla que el coche ha salido y cuál es el estado del parking. Se lanza una señal de que hemos salido, por si hubiese un vehículo esperando a esta condición, y se libera el mutex. Se sale de sección crítica.

El comportamiento de los camiones es algo distinto. Estos ocupan y desocupan dos plazas contiguas, así que recorrerán y buscarán plazas en el array de una forma un poco distinta. Por ejemplo, los coches aparcan en la primera posición libre que encuentran, mientras que los camiones lo hacen en la última. No es por nada especial, simplemente se debe a la implementación del bucle for.

4. Objetivos cumplidos

a. Gestionar correctamente las entradas y salidas del parking únicamente para automóviles.
b. Gestionar correctamente las entradas y salidas del parking para todo tipo de vehículos permitidos (tanto automóviles como camiones).
c. Mostrar por pantalla el estado del parking después de que entre un vehículo.
d. Ser capaz de pasar como argumentos [el número de plazas por cada planta (PLAZAS),] el número de automóviles (COCHES) y de camiones (CAMIONES) que van a acceder al mismo. No se ha realizado el apartado de las plantas del parking, por lo que habrá que escribir '1' como el argumento del número de plantas.

5. Objetivos no cumplidos

a. Implementar un algoritmo para gestionar un parking con varias plantas, todas ellas con el mismo número de plazas sabiendo que un camión no puede aparcar entre dos plantas.

6. Comentarios personales

Los problemas encontrados durante la realización de la práctica fueron los siguientes:

- A la hora de crear los threads para los vehículos, se pasaba como argumento la dirección de una variable entera, que se iba sobrescribiendo en cada iteración del bucle for. El problema que daba era que cuando se creaba el hilo, tomaba el valor de la variable indicada, pero al sobrescribirse esta, el valor de la variable podía no ser el correcto. Se solucionó este problema sustituyendo la variable entera por un array de enteros, de manera que cada thread recibía una posición de memoria distinta del array y los valores no se sobrescribían.
- Los camiones no entraban correctamente por lo que se mencionó. Una primera versión del programa contaba únicamente con un mutex, de manera que cuando un camión no tenía hueco, soltaba el mutex para que salieran vehículos, pero podía ocurrir que no solo salieran, sino que también entraran, o que otro vehículo se adueñara del mutex y se quedara esperando. Una primera solución consistió en que los camiones reservaran una plaza en cuanto estuviera libre y esperaran a que se vaciara la contigua, pero a veces se daba el caso de que la contigua tardaba mucho en vaciarse mientras se liberaban plazas consecutivas en otras posiciones. Esta solución resultó ser muy poco eficiente y se desechó.
- Problemas como que los camiones aparcaban con una posición fuera del parking o que los coches tenían identificadores propios de camiones fueron problemas menores que encontramos. El primero se debía a que la condición de parada del bucle for para buscar plazas no estaba bien implementado. Se pudieron resolver ambos.

El tiempo dedicado ha sido mayor al esperado y al que se debía haber dedicado, debido a los problemas encontrados. El segundo problema fue de los que más tiempo nos robó, ya que no se tenía claro cuál era el problema y se probaron cosas como poner la señal 'signal' después de soltar el mutex. Foros aseguraban que entre estas dos formas había pequeñas diferencias y ninguna prevalecía sobre la otra.

No hay ninguna propuesta ni mejora, la práctica ha sido muy interesante y didáctica.