Práctica 2

Parking

Rubén Moreno Martín

Jaime Llanos Melchor

Índice

Descripción del código

En esta segunda práctica, teníamos que realizar un parking con entradas y salidas gestionadas automáticamente mediante procesos ligeros (threads).
La realización la esquematizamos en 3 partes: main, función coche y función camiones.
Para empezar, hemos incluido las librerías necesarias, incluida la pthread que era nueva para esta parte y 3 constantes que son: las plazas del parking, los coches y los camiones disponibles para entrar a aparcar.
Tenemos una variable que es un array de enteros, para que en cada posición se ponga el número del coche o camión que este aparcado en su plaza, y un mutex para que bloquee el recurso compartido del parking cuando este aparcando algún vehículo, ya que no permitimos la entrada de varios vehículos a la vez para no provocar conflictos. Cuando algún vehículo este saliendo también lo bloqueamos pero lo explicaremos más adelante. Tenemos también una variable ocupación, que nos dirá cuantas plazas ocupadas hay en el parking y una variable de condición para el mutex que nos será muy útil luego como ya veremos.
Por último, la sección crítica elegida es el propio parking, el cual se bloqueará cada vez que un solo vehículo (de cualquier tipo), acceda a él, y que tendrá una condición de espera, la cual cuando el parking esté lleno y un vehículo quiera entrar, hará que el vehículo espere a que algún vehículo salga del parking y haya espacio para aparcar.
Para poder compilar bien el programa, es necesario hacerlo con "-pthreads".

-Función Main

Esta parte es muy sencilla, con una variable i para la inicialización de las diferentes variables que necesitemos, y con dos arrays de enteros, uno para los coches y otro para los camiones, para darle número a cada uno de ellos.
Inicializamos el mutex con la variable que queremos y la condición para el mutex, e inicializamos también los diferentes arrays declarados:
En el de plazas a inicializamos sus elementos a 0 (con esto queremos decir que están libres y no están ocupadas por nadie).
Los arrays para los vehículos los nombramos de 1 a número de coches para el array de coches y de 101 a 100+nº de camiones para los camiones, es decir, con la nomenclatura que así se indica en el enunciado de la práctica.
Con pthread_t creamos las variables thread para cada vehículo y con los 2 ultimos for las inicializamos pasándole en cada caso los valores correspondientes (identificador de cada thread, la función, y el número del vehículo), no es necesario realizar un join por cada thread creado, ya que nunca se saldrá de la función salvo con Ctrl + c o Ctrl + z.

-Función Coche y función Camión

Se van a explicar a la par ya que son similares aunque hayamos considerado oportuno realizarlas en 2 funciones distintas.

En primer lugar tenemos una variable tipo entero, la cual recoge el número del vehículo el cual obtenemos mediante un casting al void que nos pasa la función "pthread_create()", también tenemos una variable entera i y un variable entera para las plazas libres para mayor comodidad a la hora de programar, aunque no es estrictamente necesaria.
A continuación entramos en el bucle while (1), bucle que se repite indefinidamente. Dentro del bucle comenzamos inicializando la variable i a 0 (es necesario hacerlo aquí y no antes para que con cada nueva iteración la tengamos a 0 desde el principio) y ponemos el vehículo a dormir un tiempo aleatorio.

Después bloqueamos el mutex y con el bucle while miramos si el parking está lleno para los coches, y para los camiones miramos si está lleno o si solo tiene un plaza libre (ya que el camión necesita 2 plazas libres para aparcar) y ponemos que espere por la condición para que cuando este libre vuelva a activarse.

Después comprobamos la primera plaza libre disponible, y para los camiones además, ponemos una condición para que no se pueda aparcar en la última plaza exclusivamente, y para que solo aparque cuando haya dos sitios libres consecutivos.
Esta condición es indispensable en la función de los camiones, dado que estos tendrán que aparcar en dos plazas consecutivas, y si no se cumple desbloquea el mutex y vuelve a empezar, si se cumple entramos en el if, asignamos a esa plaza el número del vehículo (y también a la plaza siguiente en el caso de los camiones). La ocupación la incrementamos en 1 para los coches y en 2 para los camiones, y por último calculamos las plazas libres y mostramos por pantalla la ocupación de parking. Después debloqueamos el mutex y ponemos a dormir al vehículo.
Ahora entramos en la segunda parte de la función que es cuando se quiere ir el vehículo. Bloqueamos de nuevo el mutex, volvemos a poner la variable i a 0 y buscamos en la posición donde está el vehículo. Para el camión ponemos a 0 el valor del array parking en la posición i y en la i+1 (ya que ocupa 2 posiciones) y para el coche la posición i. Restamos la ocupación (1 para los coches y 2 para los camiones) y calculamos las plazas libres. Por último, mostramos la ocupación del parking, mandamos el signal de la condición para decir que hay plazas libres, y después desbloqueamos el mutex. Así termina la iteración del bucle, el cual volverá a repetirse indefinidamente.

Comentarios personales

La realización de esta práctica, a diferencia de la anterior, nos ha resultado más sencilla y hemos sido capaces de realizarla en su mayoría prácticamente en una tarde.
Cabe destacar que nos peleamos bastante con el problema de la cena de los filósofos y eso ha ayudado a que nos resultara más sencillo entenderlo.
Como problemas a la hora de realizar la práctica nos hemos encontrado la falta de tiempo para realizar los apartados que nos quedan ya que la minishell nos ha quitado mucho tiempo y que trabajábamos los 2 durante las navidades.
Otro problema mínimo es los meticuloso que hay que ser sobre todo con los camiones para no cometer ningún violación del segmento con la condición if creada sobre todo, pero la pusimos para todo el bucle y lo solucionamos.