Threads

PRÁCTICA 2

Sistemas Operativos – Ingeniería Informática

Iván Conde Carretero – 03940213Z

Lara Camila Sánchez Correa – Y0479214T

Tabla de contenido

En el siguiente documento explicaremos el desarrollo de la segunda práctica de la asignatura: Threads. Explicando sus distintos aspectos, como elecciones de diseño, funcionalidad etc.

En primer lugar, hablaremos de la autoría.

Autoría

La siguiente práctica ha sido desarrollada por dos alumnos del Grado en Ingeniería Informática de Móstoles del tercer curso.

Iván Conde Carretero – 03940213Z – Expediente: 1497

Lara Camila Sánchez Correa – Y0479214T – Expediente: 1520

La totalidad de la práctica ha sido especificada por ambos estudiantes.

A continuación, se explicará el código elegido.

Descripción del código

Podremos separar el código presentado según la funcionalidad de cada aspecto. A grandes rasgos nos encontramos con los siguientes apartados:

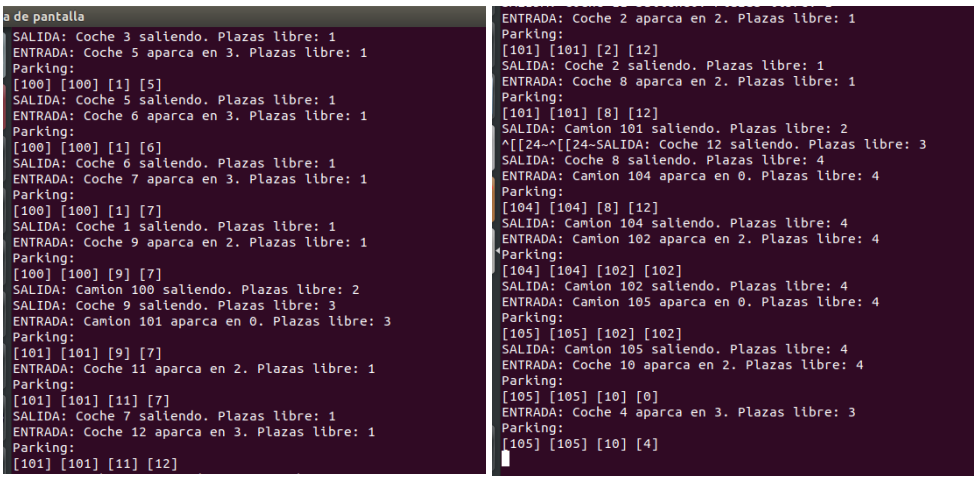
Main

En el main nos encontraremos con los siguientes aspectos:

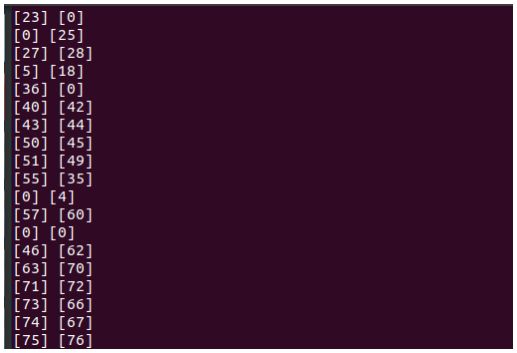
Argumentos

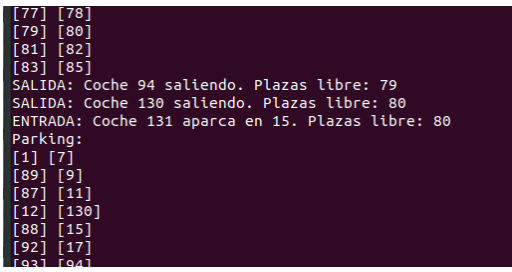
En esta sección de código se tendrá en cuenta cuántos argumentos de entrada han sido pasados para plazas, plantas, coches y camiones. Existirán cuatro posibilidades:

- Si no se pasa **ningún argumento**: se pondrán los valores por defecto para todos los valores.

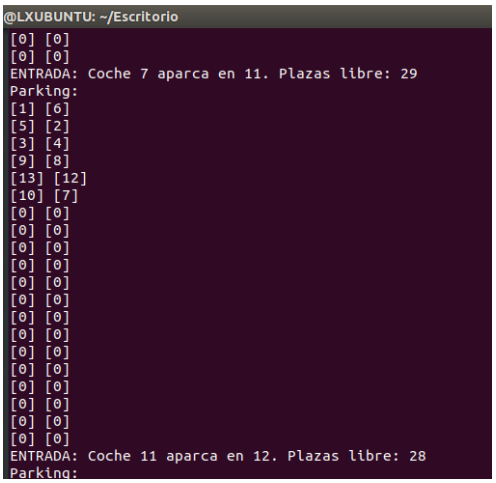


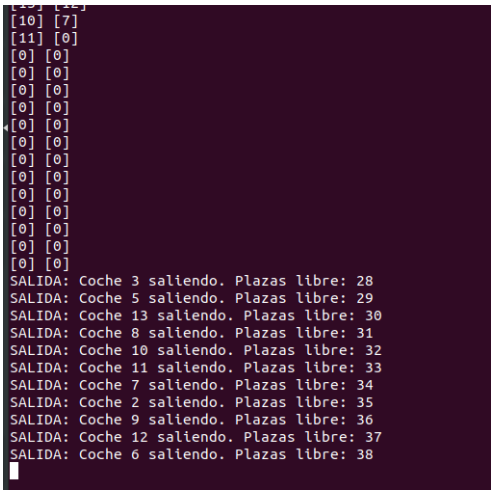
- Si se pasan **dos argumentos**: el número de coches y camiones será por defecto. En este ejemplo la entrada es: 100 plazas y 2 plantas.



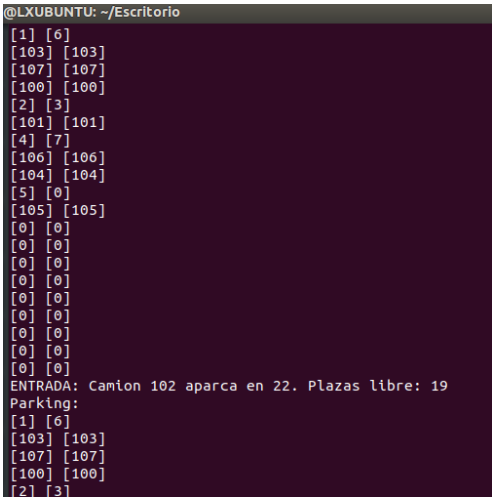


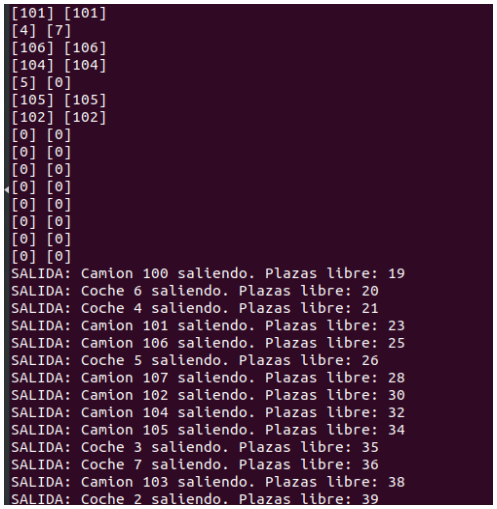
- Si se pasan **tres argumentos**: el número de camiones será por defecto, es decir, 0. En este ejemplo: 40 plazas, 2 plantas y 10 coches.





- Última opción es que el usuario introduzca las cuatro variables por entrada estándar, siguiendo el orden indicado en el enunciado: Plazas - Plantas - Coches - Camiones.





Reserva de memoria e inicialización del entorno

Se debe reservar en memoria, ya que los tamaños que vamos a utilizar serán conocidos en tiempo de compilación. Por tanto, nos manejaremos con memoria dinámica.

Con la función **malloc** se reservará en memoria el espacio para el *parking* completo, conteniendo las plantas y plazas.

Así como los identificadores de los automóviles y camiones, con la creación de sus respectivos threads según la cantidad declarada anteriormente.

Se inicializarán todas las plazas a cero, ya que cero significan que están vacías.

Crearemos las condiciones para los coches y camiones, así como el mutex que nos gestionará la sección crítica.

Uno de los mayores desafíos que se han enfrentado en esta práctica ha sido que los camiones y los coches accedan de forma aleatoria, evitar el interbloqueo en el parking para que no acceda un solo tipo. Es decir, que accedan de manera intercalada, ya que, si constantemente accedían los camiones, los coches no podrían entrar jamás y viceversa.

El problema anterior se ha solucionado de forma eficaz mediante dos condiciones: si hay más coches que camiones en el parking se introducirán intercalándose, pero primero se introducirá un coche y después un camión, y si hay más camiones en el parking se hará, al contrario.

Cuando no haya camiones no se presentará el problema anterior y accederán todos los coches de golpe.

Liberación de memoria

Último apartado a tratar en el main es la liberación de la memoria que se ha reservado en el desarrollo de la práctica, se debe tener especial cuidado en los threads y el mutex para que no queden constantemente ejecutando.

Hilos

En esta sección explicaremos la creación de los hilos correspondientes para cada vehículo que entra en el entorno.

Entrada y salida del camión

En primera instancia se debe especificar el identificador único del camión, para poder identificarlo cuando se muestre por salida estándar a la hora de realizar movimientos.

Seguidamente, el hilo intentará coger el mutex, si está cerrado se esperará hasta que pueda acceder, si está abierto continuará ejecutando la siguiente sección:

Mirará si tiene plaza libre, se ha tenido en cuenta que los camiones requieren de dos plazas contiguas para poder aparcar, es por ello por lo que se ha desarrollado un método que nos diga si estas dos plazas seguidas existen. Si no existen plazas libres, el hilo dormirá hasta que lo despierten.

Si existen plazas libres, se colocará en dicha plaza poniendo el identificador sobre ella. Se imprimirá por pantalla el movimiento que ha realizado, y restará dos plazas libres al parking.

Una vez que ha terminado de ejecutar, se abrirá el mutex de la sección crítica, para que los demás hilos puedan continuar su flujo de ejecución.

El camión dentro de la plaza realizará un sleep, que será el tiempo que permanece en dicha plaza.

En el método mencionado anteriormente, se realiza un random que nos indicará cuánto tiempo permanecerá dormido el hilo.

Una vez que despierte el camión, este deberá actualizar la variable de plazas libres que se encuentran en el parking, pero dado que es un recurso compartido por todos los hilos para acceder a ella se utilizará un semáforo y así evitar que se actualice por otro lado.

Se imprimirá por pantalla el movimiento que ha realizado el camión para salir. Y así como la apertura del semáforo.

Una vez finalizado el método, el camión ya no está en el parking así que deberá despertar al siguiente hilo, dando preferencia a los camiones y así evitar la inanición, que se encuentre durmiendo para que este acceda, para dar mayor preferencia a los camiones, hemos lanzado un broadcast a todos los camiones, para que comprueben si pueden acceder, y a los coches un signal, para solo avisar a uno.

Entrada y salida del coche.

Se tratará de una ejecución muy similar a la del camión, pero con algunos aspectos diferentes, ya que el coche no necesita dos plazas contiguas para poder aparcar, sino que únicamente una.

Realizará las mismas operaciones que las explicadas anteriormente:

Se le asignará un identificador único, intentará poder entrar en el mutex, si no lo consigue esperará hasta que esté abierto, si consigue entrar seguirá ejecutando.

Una vez dentro del mutex tendrá dos opciones, si no existe plaza libre se quedará esperando hasta que lo despierten, si puede entrar en el parking, se colocará el identificador único en dicha plaza.

Las acciones que se realizan al entrar son las mismas que con el camión: Se restarán las plazas libres correspondientes, se imprimirá por pantalla qué coche y dónde aparca, en última instancia abrimos el mutex para que los demás hilos puedan entrar en el recurso compartido.

En este apartado, el coche ya se encuentra dentro del parking y deberá realizar un sleep aleatorio que nos diga cuánto tiempo permanecerá en el mismo. El siguiente paso será salir del parking, el método utilizado realizará lo siguiente:

Se colocará el semáforo en rojo, se aumentarán las plazas libres del parking, se imprimirá por pantalla el movimiento realizado por el coche y finalmente se volverá a colocar el semáforo en verde para los demás hilos.

Después del coche salir, se comprobará que haya dos plazas contiguas, para poder así despertar a los camiones, para que estos puedan acceder al parking, si no hay plazas contiguas, se despertara un solo coche, como en el caso de hilo camión.

Conclusión

En esta práctica se ha podido ver de primera mano cómo funcionan los threads en C. Siendo estos muy interesantes dado que muestran un comportamiento en paralelo, y como al tratarse de una ejecución paralela se han tenido que tratar de forma excepcional las secciones críticas del código, es decir, los recursos compartidos a los que accedían los hilos de forma constante.

Una de las partes más tediosas del código ha sido evitar la inanición, dado podía darse el caso de que un hilo jamás accediese a un recurso que estaba siendo utilizado constantemente.

También se ha podido volver a explorar el entorno de Linux, así como la programación en C, teniendo en esta práctica más soltura gracias a la anterior entrega.