

Parking MPI

Ampliación de Sistemas Operativos

Tabla de contenido

PARKING MPI.....	1
Opciones de diseño utilizadas.....	3
Subsistemas de comunicación.....	3
Conclusiones y comentarios personales.....	4
Ejemplos prácticos de ejecución.....	5

Opciones de diseño utilizadas

Al principio, pensamos en utilizar solo un ejecutable y separar el flujo de la ejecución con varios *If* para distinguir entre maestro (el parking) y esclavos (coche y camión) al resultarnos más intuitivo. Nos decidimos por la opción de separarlo en varios ejecutables ya que dejaba más claro el código y es más eficiente.

Nos planteamos usar varios hilos en el código del parking para que uno de ellos se encargara de gestionar la recepción y el envío de mensajes MPI continuamente (así conseguiríamos un funcionamiento óptimo), mientras que el otro gestionaba las colas de entrada y salida de vehículos. Sin embargo, dado a la acumulación de tareas y falta de tiempo, no implementamos esta mejora.

Subsistemas de comunicación

Las instrucciones MPI que usamos en nuestro programa son las instrucciones básicas de envío y recepción de datos de manera bloqueante (MPI_Send y MPI_Recv).

En el contenido del envío (el buffer) le pasamos un array con el número de la planta y la plaza que ocupa, la matrícula (que sería el rango) y el tipo de petición que le hacen al parking (entrada o salida).

Usamos también varios tags para diferenciar el tipo de mensaje (tanto peticiones como la confirmación de que se ha llevado a cabo una entrada o una salida del parking).

Conclusiones y comentarios personales

La práctica nos ha enseñado un mecanismo sencillo de paralelización de algoritmos. Además, el haber usado un cluster real nos ha permitido familiarizarnos con un sistema UNIX desde la terminal (aunque al principio, tuvimos bastantes problemas con la ejecución en el cluster debido a nuestros pocos conocimientos sobre este tipo de sistemas).

Tuvimos también dudas sobre si usar instrucciones de envío y recepción bloqueantes o no bloqueantes. Tras varias pruebas nos decantamos por el uso de las bloqueantes ya que así llevamos un mejor control sobre la ejecución del programa.

Ejemplos prácticos de ejecución

1- Creación del parking de forma dinámica

```
danny@danny:~/Carrera/ASO/Practica2$ gcc parking.c -o test_init
danny@danny:~/Carrera/ASO/Practica2$ ./test_init 10 5
Reservado espacio para el parking
Parking inicializado
Parking: planta 0-> [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Parking: planta 1-> [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Parking: planta 2-> [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Parking: planta 3-> [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Parking: planta 4-> [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Plazas totales: 50
Plazas libres disponibles: 50
danny@danny:~/Carrera/ASO/Practica2$ ./test_init 20 6
Reservado espacio para el parking
Parking inicializado
Parking: planta 0-> [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Parking: planta 1-> [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Parking: planta 2-> [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Parking: planta 3-> [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Parking: planta 4-> [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Parking: planta 5-> [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Plazas totales: 120
Plazas libres disponibles: 120
danny@danny:~/Carrera/ASO/Practica2$ ./test_init 20
Reservado espacio para el parking
Parking inicializado
Parking: planta 0-> [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Plazas totales: 20
Plazas libres disponibles: 20
danny@danny:~/Carrera/ASO/Practica2$
```

En el apartado c del enunciado de la práctica se nos pide crear las dimensiones del parking en función de dos parametros que le pasamos como entrada (plazas y plantas respectivamente). En esta primera captura de pantalla, ofrecemos una demo del primer ejecutable que compilamos con el comando gcc y ejecutamos sin usar mpirun, para comprobar que las dimensiones de la matriz creada dinámicamente se ajustaban a los parámetros que le pasamos como entrada junto con el ejecutable de la demo, al que llamamos “test_init”.

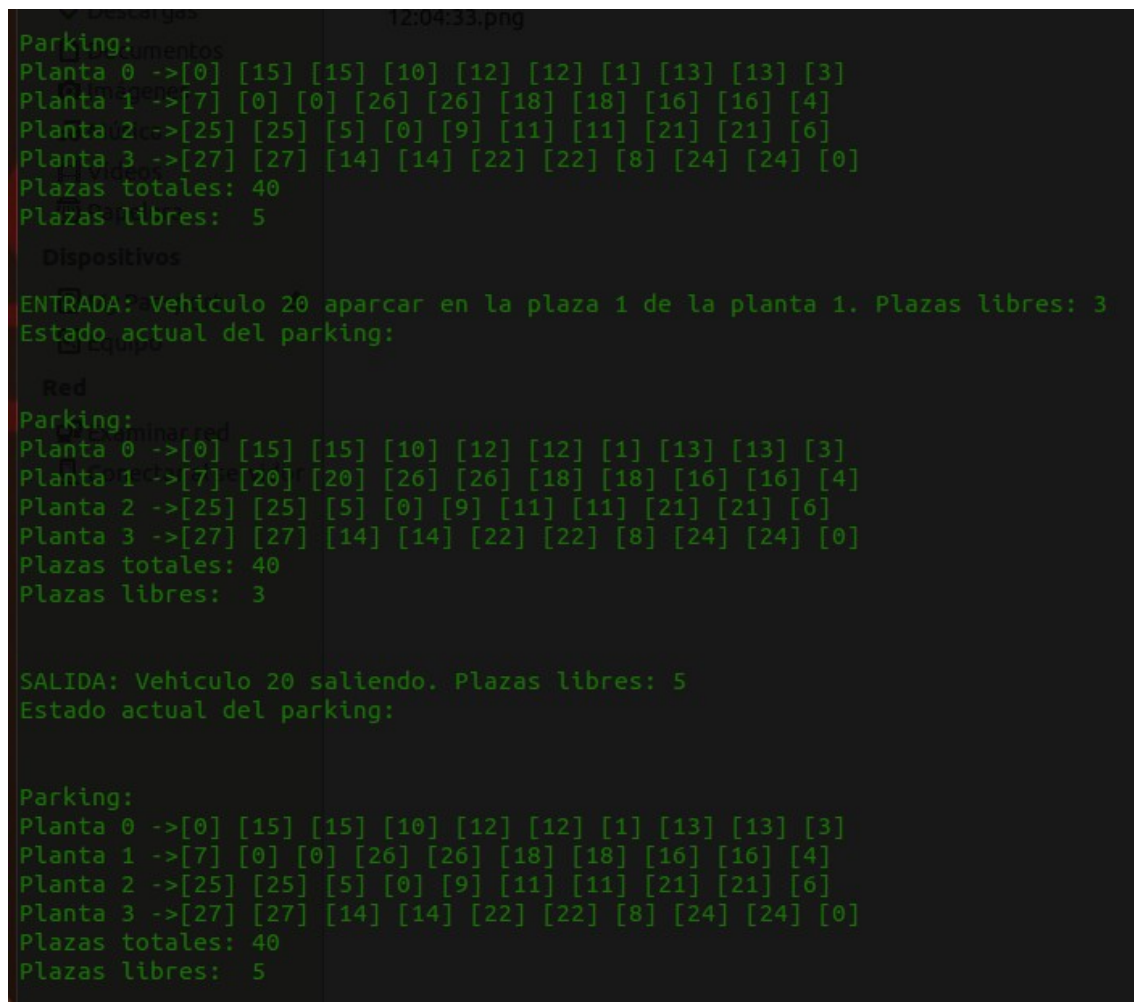
2- Estacionamiento y salida de coche

```
Parking:eta personal
Planta 0 ->[2] [6] [8] [1] [3] [7] [13] [13] [14] [14]
Planta 1 ->[10] [9] [12] [12] [15] [15] [17] [17] [18] [18]
Planta 2 ->[20] [20] [24] [24] [19] [19] [22] [22] [27] [27]
Planta 3 ->[26] [26] [21] [21] [23] [23] [16] [16] [0] [0]
Plazas totales: 40
Plazas libres: 2
Música
Videos
SALIDA: Vehículo 3 saliendo. Plazas libres: 3
Estado actual del parking:
Dispositivos
Parking:assport
Planta 0 ->[2] [6] [8] [1] [0] [7] [13] [13] [14] [14]
Planta 1 ->[10] [9] [12] [12] [15] [15] [17] [17] [18] [18]
Planta 2 ->[20] [20] [24] [24] [19] [19] [22] [22] [27] [27]
Planta 3 ->[26] [26] [21] [21] [23] [23] [16] [16] [0] [0]
Plazas totales: 40
Plazas libres: 3
ENTRADA: Vehículo 5 aparcar en la plaza 4 de la planta 0. Plazas libres: 2
Estado actual del parking:
Parking:
Planta 0 ->[2] [6] [8] [1] [5] [7] [13] [13] [14] [14]
Planta 1 ->[10] [9] [12] [12] [15] [15] [17] [17] [18] [18]
Planta 2 ->[20] [20] [24] [24] [19] [19] [22] [22] [27] [27]
Planta 3 ->[26] [26] [21] [21] [23] [23] [16] [16] [0] [0]
Plazas totales: 40
Plazas libres: 2
SALIDA: Vehículo 5 saliendo. Plazas libres: 3
Estado actual del parking:
```

En este apartado vamos a explicar brevemente el funcionamiento que hace un coche para entrar y salir del parking. Se imprime un mensaje de salida avisando de que

el coche 3 estacionado en la plaza 5 de la planta baja del parking, sale y una vez que ha dejado la plaza libre (la plaza es igual a 0), la va a ocupar el coche 5, entonces imprimimos un mensaje por pantalla avisando de la entrada del coche 5, más el número de plazas libres que queden una vez que éste ocupe la plaza. A continuación imprimimos el estado global del parking.

3- Estacionamiento y salida camión



```

Parking:
Planta 0 ->[0] [15] [15] [10] [12] [12] [1] [13] [13] [3]
Planta 1 ->[7] [0] [0] [26] [26] [18] [18] [16] [16] [4]
Planta 2 ->[25] [25] [5] [0] [9] [11] [11] [21] [21] [6]
Planta 3 ->[27] [27] [14] [14] [22] [22] [8] [24] [24] [0]
Plazas totales: 40
Plazas libres: 5

Dispositivos

ENTRADA: Vehiculo 20 aparcar en la plaza 1 de la planta 1. Plazas libres: 3
Estado actual del parking:

Red

Parking:
Planta 0 ->[0] [15] [15] [10] [12] [12] [1] [13] [13] [3]
Planta 1 ->[7] [20] [20] [26] [26] [18] [18] [16] [16] [4]
Planta 2 ->[25] [25] [5] [0] [9] [11] [11] [21] [21] [6]
Planta 3 ->[27] [27] [14] [14] [22] [22] [8] [24] [24] [0]
Plazas totales: 40
Plazas libres: 3

SALIDA: Vehiculo 20 saliendo. Plazas libres: 5
Estado actual del parking:

Parking:
Planta 0 ->[0] [15] [15] [10] [12] [12] [1] [13] [13] [3]
Planta 1 ->[7] [0] [0] [26] [26] [18] [18] [16] [16] [4]
Planta 2 ->[25] [25] [5] [0] [9] [11] [11] [21] [21] [6]
Planta 3 ->[27] [27] [14] [14] [22] [22] [8] [24] [24] [0]
Plazas totales: 40
Plazas libres: 5
  
```

En este caso en la planta 1 del parking, encontramos la plaza 1 y 2 libres, es decir, según lo que tenemos codificado en nuestro algoritmo, se da la condición de que existen dos plazas libres consecutivas, y que por tanto el primer camión

que esté esperando en la cola de acceso será el que ocupe estas plazas. Imprimimos un mensaje por pantalla avisando de que el vehículo con matrícula 20 (es un camión) va a estacionar. Al volver a mostrar el estado global del parking, observamos que las plazas 1 y 2 de la primera planta del parking se les asignado el valor 20, que se corresponde con el número de matrícula de del camión.

Una vez que el camión quiere salir se muestra un mensaje de salida por pantalla avisando de la salida del vehículo 20, y nuevamente al mostrar el estado del parking vemos que las plazas 1 y 2 de la primera planta vuelven a estar a 0.