Sistemas Operativos – Grado en Ingeniería Informática

Práctica obligatoria 2

Programación con threads

Descripción

El objetivo de la práctica es que el alumno se familiarice con la sincronización de procesos y *threads*. Para ello, se va a plantear un problema en el que se deberá implementar el sistema de acceso a un *parking* usando procesos ligeros o *threads*.

El programa a implementar debe gestionar las entradas y salidas de vehículos al *parking*. Los posibles vehículos que pueden entrar a dicho *parking* son automóviles y camiones. Un automóvil ocupará una plaza de aparcamiento dentro del *parking*, mientras que un camión ocupará dos plazas contiguas de aparcamiento. Las plazas del *parking* se sitúan de manera contigua de la siguiente forma:

0	1	2	3	…	N

El *parking* dispondrá de una única entrada y una única salida, por la que entrarán y saldrán los distintos tipos de vehículos permitidos. En la entrada de vehículos habrá un dispositivo de control que permita o impida el acceso de los mismos al *parking*, dependiendo del estado actual del mismo (plazas de aparcamiento disponibles). En caso de que el dispositivo de control permita el acceso a un vehículo, se le asignará el número de plaza de aparcamiento donde debe aparcar. Queda a decisión del control de acceso qué plaza asigna de entre todas las que haya libres.

Los tiempos de espera de los vehículos dentro del *parking* son aleatorios (para ello, se puede utilizar la función `rand()`). En el momento en el que un vehículo sale del *parking*, notifica al dispositivo de control el número de plaza que tenía asignada y se libera la plaza o plazas de aparcamiento que estuviera ocupando, quedando así éstas nuevamente disponibles.

Un vehículo que ha salido del *parking* esperará un tiempo aleatorio para volver a entrar nuevamente en el mismo. Por tanto los vehículos estarán entrando y saliendo indefinidamente del *parking*. Es importante que se diseñe el programa de tal forma que se asegure que, antes o después, un vehículo que permanece esperando a la entrada del *parking* entrará en el mismo.

La salida del programa debería ser algo del tipo:

```
ENTRADA: Coche 1 aparca en 0. Plazas libre: 5
ENTRADA: Coche 2 aparca en 1. Plazas libre: 4
ENTRADA: Coche 3 aparca en 2. Plazas libre: 3
ENTRADA: Coche 4 aparca en 3. Plazas libre: 2
ENTRADA: Coche 5 aparca en 4. Plazas libre: 1
SALIDA: Coche 1 saliendo. Plazas libre: 2
ENTRADA: Coche 6 aparca en 0. Plazas libre: 1
SALIDA: Coche 5 saliendo. Plazas libre: 2
SALIDA: Coche 2 saliendo. Plazas libre: 3
SALIDA: Coche 4 saliendo. Plazas libre: 4
ENTRADA: Coche 8 aparca en 1. Plazas libre: 3
SALIDA: Coche 3 saliendo. Plazas libre: 4
ENTRADA: Coche 7 aparca en 2. Plazas libre: 3
ENTRADA: Coche 10 aparca en 3. Plazas libre: 2
ENTRADA: Coche 9 aparca en 4. Plazas libre: 1
SALIDA: Coche 8 saliendo. Plazas libre: 2
SALIDA: Coche 6 saliendo. Plazas libre: 2
SALIDA: Coche 9 saliendo. Plazas libre: 3
SALIDA: Coche 10 saliendo. Plazas libre: 5
ENTRADA: Camión 101 aparcado en 2. Plazas libre: 3
ENTRADA: Coche 2 aparca en 1. Plazas libre: 1
ENTRADA: Camión 102 aparcado en 4. Plazas libre: 1
```

```
SALIDA: Coche 7 saliendo. Plazas libre: 4
SALIDA: Camión 101 saliendo. Plazas libre: 3
SALIDA: Coche 2 saliendo. Plazas libre: 2
ENTRADA: Camión 103 aparcado en 2. Plazas libre: 1
SALIDA: Camión 102 saliendo. Plazas libre: 4
ENTRADA: Coche 1 aparca en 0. Plazas libre: 3
ENTRADA: Coche 9 aparca en 1. Plazas libre: 2
ENTRADA: Coche 4 aparca en 4. Plazas libre: 1
SALIDA: Coche 1 saliendo. Plazas libre: 2
ENTRADA: Coche 5 aparca en 0. Plazas libre: 1
SALIDA: Camión 103 saliendo. Plazas libre: 4
ENTRADA: Coche 3 aparca en 1. Plazas libre: 3
ENTRADA: Coche 10 aparca en 2. Plazas libre: 2
SALIDA: Coche 9 saliendo. Plazas libre: 4
^C
```

Objetivos parciales

- Gestionar correctamente las entradas y salidas del *parking* únicamente para automóviles. Un automóvil podrá acceder al *parking* si hay, al menos, una plaza de aparcamiento libre. En el caso en el que el *parking* esté lleno, el automóvil deberá esperar en la barrera de entrada a que se libere una plaza de aparcamiento. Para dar por correcto el funcionamiento de este apartado, el programa tiene que estar libre de interbloqueos y no producir inanición (**2 puntos**).

- Gestionar correctamente las entradas y salidas del *parking* para todo tipo de vehículos permitidos (tanto automóviles como camiones). Un camión sólo podrá acceder al *parking* si hay, al menos, dos plazas contiguas de aparcamiento libre. En el caso de que no haya plazas disponibles, el vehículo deberá esperar en la barrera de entrada hasta que el control de acceso le permita entrar. Para dar por correcto el funcionamiento de este apartado, el programa tiene que estar libre de interbloqueos y no producir inanición (**5 puntos**).

- Mostrar por pantalla el estado del *parking* después de que entre un vehículo (**1 punto**):

```
ENTRADA: Coche 1 aparca en 0. Plazas libres: 5
Parking: [1] [0] [0] [0] [0] [0]
ENTRADA: Coche 2 aparca en 1. Plazas libres: 4
Parking: [1] [2] [0] [0] [0] [0]
ENTRADA: Coche 3 aparca en 2. Plazas libres: 3
Parking: [1] [2] [3] [0] [0] [0]
ENTRADA: Coche 4 aparca en 3. Plazas libres: 2
Parking: [1] [2] [3] [4] [0] [0]
ENTRADA: Coche 5 aparca en 4. Plazas libres: 1
Parking: [1] [2] [3] [4] [5] [0]
SALIDA: Coche 2 saliendo. Plazas libres: 2
SALIDA: Coche 3 saliendo. Plazas libres: 3
ENTRADA: Coche 7 aparca en 2. Plazas libres: 2
Parking: [1] [0] [7] [4] [5] [0]
ENTRADA: Coche 6 aparca en 1. Plazas libres: 1
Parking: [1] [6] [7] [4] [5] [0]
SALIDA: Coche 1 saliendo. Plazas libres: 2
SALIDA: Coche 4 saliendo. Plazas libres: 3
ENTRADA: Coche 9 aparca en 0. Plazas libres: 2
Parking: [9] [6] [7] [0] [5] [0]
ENTRADA: Coche 10 aparca en 3. Plazas libres: 1
Parking: [9] [6] [7] [10] [5] [0]
SALIDA: Coche 5 saliendo. Plazas libres: 2
ENTRADA: Coche 8 aparca en 4. Plazas libres: 1
Parking: [9] [6] [7] [10] [8] [0]
SALIDA: Coche 7 saliendo. Plazas libres: 2
SALIDA: Coche 6 saliendo. Plazas libres: 3
ENTRADA: Coche 1 aparca en 1. Plazas libres: 2
Parking: [9] [1] [0] [10] [8] [0]
SALIDA: Coche 9 saliendo. Plazas libres: 3
SALIDA: Coche 8 saliendo. Plazas libres: 4
SALIDA: Coche 10 saliendo. Plazas libres: 5
ENTRADA: Camión 103 aparcado en 3 y 4. Plazas libres: 3
Parking: [0] [1] [0] [103] [103] [0]
ENTRADA: Coche 2 aparca en 2. Plazas libres: 2
Parking: [0] [1] [2] [103] [103] [0]
ENTRADA: Coche 7 aparca en 0. Plazas libres: 1
Parking: [7] [1] [2] [103] [103] [0]
SALIDA: Coche 1 saliendo. Plazas libres: 2
ENTRADA: Coche 8 aparca en 1. Plazas libres: 1
Parking: [7] [8] [2] [103] [103] [0]
```

```
SALIDA: Coche 2 saliendo. Plazas libres: 2
ENTRADA: Coche 9 aparca en 2. Plazas libres: 1
Parking: [7] [8] [9] [103] [103] [0]
SALIDA: Camión 103 saliendo. Plazas libres: 3
ENTRADA: Camión 102 aparcado en 3 y 4. Plazas libres: 1
Parking: [7] [8] [9] [102] [102] [0]
^C
```

- 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 (**1 punto**). Este apartado implica reescribir el método de escritura del *parking* para mostrar su estado para el número de plantas especificado.

- Ser capaz de pasar como argumentos el número de plazas por cada planta (PLAZAS), el número de plantas del *parking* (PLANTAS), el número de automóviles (COCHES) y de camiones (CAMIONES) que van a acceder al mismo. Es decir, el programa tendría que entender (**1 punto**):

  ```
  parking PLAZAS PLANTAS
  parking PLAZAS PLANTAS COCHES
  parking PLAZAS PLANTAS COCHES CAMIONES
  ```

  Si no se ha realizado el apartado anterior deberá escribirse 1 como argumento de número de PLANTAS. Si no se especifica el número de coches (en el primer caso), éste será el doble del número de plazas por cada planta del *parking* por el número de plantas, es decir: COCHES = 2*PLAZAS*PLANTAS. Si no se especifica el número de camiones, éste será 0.

Nota: Las puntuaciones para cada objetivo parcial son las puntuaciones máximas que se pueden obtener si se cumplen esos objetivos.

Nota: No se debe hacer un programa separado para cada objetivo, sino un único programa genérico que cumpla con todos los objetivos simultáneamente.

Nota: Para evitar que se produzcan interbloqueos o inanición, no se considerarán válidas soluciones a medida implementadas por el alumno, que solucionen casos concretos.

Entrega de prácticas

La entrega se realizará a través del Campus Virtual en las fechas anunciadas en el mismo. Se debe entregar un único archivo *parking.c* con el código de toda la práctica, debidamente comentado y una memoria de la misma en formato PDF. La memoria debe incluir:

- Índice de contenidos
- Autores
- Descripción del código: incluyendo descripción de las principales funciones implementadas, decisiones de diseño tomadas y sección crítica elegida.
- Comentarios personales: incluyendo problemas encontrados, críticas constructiva, propuesta de mejoras y evaluación del tiempo dedicado.
- No incluir código fuente
- NO DESCUIDE LA CALIDAD DE LA MEMORIA DE SU PRÁCTICA. Aprobar la memoria es tan imprescindible para aprobar la práctica, como el correcto funcionamiento de la misma. Si al evaluar la memoria se considera que no alcanza el mínimo admisible, la práctica se considerará SUSPENSA.

Evaluación de la práctica

La práctica se evaluará comprobando el correcto funcionamiento de los distintos objetivos, y valorando la simplicidad del código, los comentarios, la óptima gestión de recursos, la gestión de errores y la calidad de la memoria. El profesor podrá solicitar una defensa oral de la práctica si lo considerase necesario.

A la hora de codificar la solución pedida, se deberán respetar una serie de normas de estilo:

- Las variables locales de las funciones se declararán inmediatamente después de la llave de comienzo del cuerpo de la misma. Se penalizará la declaración de variables entre sentencias ejecutables de las funciones.
- No se admitirán asignaciones iniciales a variables cuyo valor dependa del valor de otras variables. El valor asignado en estos casos siempre deberá ser conocido en tiempo de compilación.
- Cuando se declare una variable de tipo *array*, su tamaño deberá ser conocido en tiempo de compilación. Si se quiere utilizar un *array* de tamaño variable, deberá crearse en memoria dinámica mediante las funciones correspondientes (malloc, calloc o realloc).
- Las operaciones sobre *strings* (copia, comparación, duplicación, concatenación, etc) se realizarán en lo posible mediante las funciones indicadas en *string.h* (ver referencias a la biblioteca de C en el Campus Virtual).
- La práctica deberá estar programada en ANSI C.
- En general, se penalizará el uso de construcciones propias de C++.
- Al compilar el código fuente, deberá producirse el menor número posible de *warnings* (mejor que no se produzca ninguno).

El incumplimiento de estas normas de estilo, así como de otras normas que puedan ser anunciadas por el profesor a través del Campus Virtual, conllevará una penalización en la nota obtenida.

Autoría de la práctica

La práctica se debe realizar en grupos de 2 personas como máximo.

El hecho de detectar copia en las prácticas expondrá a los alumnos a la posibilidad de una apertura de expediente disciplinario y expulsión. En caso de detectar copia, los alumnos afectados serán suspendidos en la TOTALIDAD de la asignatura. Una práctica será considerada copia en caso de que contenga la totalidad o una parte de la práctica de otro alumno. Se considerará copia en caso de:

- Archivos que contengan la totalidad o fragmentos de código de otro alumno
- Memorias con la totalidad o fragmentos de frases e imágenes de otro alumno

El profesor podrá hacer uso de detectores automáticos de plagio en las prácticas, tanto en la parte referente al código como a la memoria.