Práctica 4: Programación con monitores en C++

Programación de Sistemas Concurrentes y Distribuidos

Dpto. de Informática e Ingeniería de Sistemas,
Grado de Ingeniería Informática
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

1. Objetivos

En esta práctica se estudiará la resolución de problemas de sincronización mediante monitores.

En concreto, los objetivos de esta práctica son:

- comprender y profundizar en la sincronización de procesos,
- resolver problemas de sincronización de procesos utilizando monitores,
- y profundizar en el modelo de concurrencia de C++.

2. Trabajo previo a la sesión en el laboratorio

Antes de la correspondiente sesión en el laboratorio, cada estudiante deberá leer el enunciado, analizar los problemas que en él se proponen y realizar un diseño previo de las soluciones sobre las que va a trabajar. Los resultados de su trabajo de análisis y diseño los tendrá que expresar en un documento que entregará en papel a los profesores antes del inicio de la sesión (en el mismo laboratorio). El documento debe contener como mínimo el nombre completo y el NIP del estudiante y, para cada ejercicio,

- la descripción de los datos compartidos, enumeración de los procesos que los comparten y especificación de los monitores necesarios para sincronizar la actividad de estos procesos.
- un esbozo de alto nivel del código de los procesos indicando las zonas que están afectadas por la sincronización.

La entrega de este documento es un pre-requisito únicamente si la práctica se realiza de forma presencial durante la sesión de laboratorio. El documento debe estar correctamente escrito, sin faltas de ortografía, y cumplir los mínimos exigibles de presentación. Caso contrario no se valorará.

3. Enunciado

Una empresa de alquiler de bicicletas dispone para una ciudad dada de un punto de alquiler con cinco bicicletas individuales y tres bicicletas tándem (para dos personas).

Una particularidad del funcionamiento del proceso de alquiler es el criterio que utilizan para asignar las bicicletas a los clientes. Cuando un cliente solicita una bicicleta, se le proporciona una bicicleta individual, si la hay disponible. En caso contrario se le proporcionará un tándem cuando un segundo cliente solicite una bicicleta. De esta forma, los dos clientes podrán conocerse y disfrutar de un agradable paseo. No obstante, si antes de llegar un segundo cliente una bicicleta individual es devuelta, ésta será asignada al cliente que estaba esperando.

El sistema de alquiler consta de 20 clientes registrados. Su comportamiento es el siguiente: en primer lugar, solicitan una bicicleta; una vez han logrado su alquiler, dan un agradable paseo y, posteriormente, devuelven la bicicleta. Este comportamiento se repite en 5 ocasiones.

Se pide desarrollar un programa concurrente que simule el funcionamiento del sistema anteriormente presentado. Para ello es necesario programar el proceso `Cliente`, conforme al comportamiento descrito, y el monitor `PuntoAlquiler` responsable de gestionar las bicicletas disponibles y su alquiler y devolución.

La interfaz del monitor consiste de dos operaciones: `solicitarBicicleta` y `devolverBicicleta`. Estas operaciones son invocadas por los clientes para alquilar y restituir una bicicleta, respectivamente. La primera operación tiene dos parámetros: el parámetro de entrada `id` que especifica el identificador del proceso que solicita el alquiler, y el parámetro de salida `idPareja` que corresponderá con el identificador del acompañante en el caso de alquilar un tándem (si se alquila una bicicleta individual, este último parámetro valdrá cero). Por otro lado, la operación de devolución también tiene dos parámetros de entrada: el identificador del cliente que devuelve la bicicleta y el de su acompañante, si lo hubiera (este último parámetro tendrá valor cero si se devuelve una bicicleta de uso individual). Además, cuando se trata de la devolución de un tándem, ambos clientes deben acudir al punto de alquiler para restituir la bicicleta, es decir, el monitor debe programar una barrera de sincronización que simule la devolución conjunta del tándem compartido. Sólo en ese instante, el tándem estará disponible para alquilarlo de nuevo.

4. Material proporcionado para resolver el ejercicio

En la página web de la asignatura hay disponible un fichero `practica4_NIP.zip` que contiene los ficheros que hay que completar y entregar, adecuadamente organizados. También hay un *script*, llamado `pract4_correcta.bash`, que prueba la correcta organización del zip a entregar.

En concreto, la carpeta `AlquilerBicicletas` contiene los ficheros a modificar, los ficheros `PuntoAlquiler.hpp` y `PuntoAlquiler.cpp` y el fichero `main_p4.cpp`. Los dos primeros corresponden con la especificación e implementación del monitor responsable de la gestión de las bicicletas y el último es el programa principal. En el fichero `PuntoAlquiler.hpp` se especifican las operaciones que debe ofrecer el

monitor. Estas operaciones deberán generar eventos de log, conforme se explica posteriormente.

Por otro lado, recuérdese que también es necesario entregar un documento `informe_p4.pdf` conforme a las pautas especificadas en la sección de entrega. Este informe debe añadirse al fichero `practica4_NIP.zip`. En caso contrario, el *script* de prueba generará el correspondiente error.

5. Monitores en C++

En clase se ha explicado cómo programar un monitor en C++ (Lección 7, transparencias 20-23) y se ha presentado un ejemplo concreto (transparencias 27-30 de la misma lección). La programación del monitor *PuntoAlquiler*, suministrado como parte del material de esta sesión, tendrá que estar basada en estas pautas.

Si leéis la especificación del monitor proporcionado veréis que es necesario implementar un constructor, un destructor y las dos operaciones descritas en el enunciado del problema. Para estudiar el uso que hacen los procesos del monitor durante la ejecución del programa, es necesario generar y almacenar eventos de *logging* de interés. Concretamente, en el caso de los monitores, estamos interesado en generar un evento al inicio y al final de la ejecución de cada una de sus operaciones. A continuación, se muestra el ejemplo concreto de la operación `solicitarBicicleta`:

```
void PuntoAlquiler::solicitarBicicleta(...) {
  unique_lock<mutex> lck(mtx);
     ADD_EVENT("solicitarBicicleta,BEGIN_FUNC_PROC," +
               to_string(nBicicletas) + "," +
               to_string(nTandems));

  ...

     ADD_EVENT("solicitarBicicleta,END_FUNC_PROC,␣"+
               to_string(nBicicletas) + "," +
               to_string(nTandems));
}
```

Como se puede observar, cada evento consiste de el nombre de la operación invocada, el tipo de evento de interés ("BEGIN_FUNC_PROC" al inicio de la operación y "END_FUNC_PROC" al final) y el estado actual del monitor (número de bicicletas individuales y tándems disponibles en ese instante). Estos eventos serán almacenados en un fichero log, como se explicó en la práctica anterior, que será utilizado para analizar el correcto comportamiento del sistema.

Finalmente, recordad que una forma de generar programas con y sin la opción de generar el log es la que se mostró en el ejemplo `pruebaSemaforos.cpp` de la práctica anterior, mediante la compilación condicional definida por `#ifdef LOGGING_MODE ... #else ... #endif` en el main, junto con `-D LOGGING_MODE` en la invocación al compilador en el Makefile correspondiente, que se encarga de definir `LOGGING_MODE`. Esta combinación hace que al compilar el programa y generar el ejecutable, este se genere como si el fuente tuviera o bien las instrucciones en el caso del `#if` o del `#else`.

6. Entrega de la práctica

La práctica se realizará de forma individual. Cuando se finalice se debe entregar un fichero comprimido `practica4_miNIP.zip` (donde `miNIP` es el NIP del autor de los ejercicios) con el siguiente contenido:

1. Los ficheros proporcionados, tanto los que hay que modificar como los que no, como aquellos ficheros adicionales que el alumno considere necesario. Si esto último ocurriera, el alumno deberá adaptar el fichero `Makefile_p4` para que la compilación se lleve a cabo correctamente.

2. Un documento en formato PDF, denominado `informe_p4.pdf` que debe contener la siguiente información:

   - el nombre completo del alumno y su NIP
   - una breve explicación de las decisiones de diseño adoptadas en su solución
   - una descripción de las principales dificultades encontradas para la realización de la práctica
   - una explicación del comportamiento observado en las ejecuciones del ejercicio
   - para cada uno de los ejercicios, el listado de los nombres de los ficheros fuente que conforman la solución solicitada.

Para la entrega del fichero `.zip` se utilizará el comando `someter` en la máquina `hendrix.cps.unizar.es`. Los alumnos pertenecientes a grupos de prácticas cuya cuarta sesión de prácticas se celebra el día 13 de noviembre de 2019 deberán someter la práctica no más tarde del día 27 de noviembre de 2019 a las 23:59. Los alumnos pertenecientes a grupos de prácticas cuya cuarta sesión de prácticas se celebra el día 20 de noviembre de 2019 deberán someter la práctica no más tarde del día 4 de diciembre de 2019 a las 23:59.

Hay que asegurarse de que la práctica funciona correctamente en los ordenadores del laboratorio (vigilar aspectos como los permisos de ejecución, juego de caracteres utilizado en los ficheros, etc.). También es importante someter código limpio (donde se ha evitado introducir mensajes de depuración que no proporcionan información al usuario). El tratamiento de errores debe ser adecuado, de forma que si se producen debería informarse al usuario del tipo de error producido. Además se considerarán otros aspectos importantes como calidad del diseño del programa, adecuada documentación de los fuentes, correcto formateado de los fuentes, etc.

Para el adecuado formateado de los fuentes, es conveniente seguir unas pautas. Hay varias, y es posible que podáis configurar el entorno de desarrollo para cualquiera de ellas. Una posible, sencilla de seguir, es la "Google C++ Style Guide", que se puede encontrar en

`https://google.github.io/styleguide/cppguide.html`

Alternativamente, cualquiera que uséis en otras asignaturas de programación.