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 para la realización y evaluación de la práctica. El documento debe estar correctamente escrito, sin faltas de ortografía, y cumplir los mínimos exigibles de presentación. Caso contrario no se valorará.

3. Enunciado

En Villarriba de Abajo hay una gasolinera que dispone de seis surtidores de gasolina. Todos los surtidores sirven el mismo tipo de gasolina. El parque móvil de Villarriba de Abajo consta de 20 vehículos. Cada vehículo circula por el pueblo durante un cierto tiempo, lo que provoca que se le agote la gasolina del depósito, y después va a la gasolinera a repostar. En concreto, cada vehículo realizará diez repostajes. El proceso de repostaje comienza accediendo a un surtidor libre. Si los seis surtidores están ocupados en ese instante, el vehículo tendrá que esperar hasta que pueda repostar. Una vez acceda a un surtidor libre y el depósito esté lleno finaliza el repostaje y el vehículo vuelve a circular, repitiéndose el comportamiento anterior de manera continuada. El tiempo requerido para el respostaje es aleatorio, tardando entre 100 y 900 milisegundos.

Por otro lado, el dueño de la gasolinera es responsable del mantenimiento de los surtidores para garantizar su correcto funcionamiento, por lo que cada cierto tiempo decide revisar los seis surtidores. Para ello tiene que esperar a que todos los surtidores estén libres, momento en el que empieza la revisión. Esta dura cierto tiempo (entre 200 y 400 milisegundos) y, mientras se lleva a cabo, ningún vehículo puede entrar a repostar. El proceso que gestiona la revisión se realizará en diez ocasiones y tiene prioridad respecto a los vehículos, de manera que una vez se pone en marcha ningún nuevo vehículo entra a los surtidores. Cuando los que estén en ellos salgan se puede llevar a cabo el mantenimiento.

Se pide desarrollar un programa concurrente que simule el funcionamiento del sistema anteriormente descrito, la vida diaria del pueblo de Villarriba de Abajo. Para ello es necesario programar los procesos *vehículo* y *mantenimiento*, conforme al comportamiento descrito, y las estructuras de datos necesarias para representar la *gasolinera*, gestionando por medio de monitores la utilización correcta de los surtidores.

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 `ControlGasolinera` contiene los ficheros a modificar, los ficheros `ControlGasolinera.hpp` y `ControlGasolinera.cpp` y el fichero `main_p4_e1.cpp`. El primero es el programa principal y los dos siguientes corresponden con la especificación e implementación del monitor responsable de la gestión de la gasolinera. En el fichero `ControlGasolinera.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++. La programación del monitor *ControlGasolinera*, suministrado como parte del material de esta sesión, está basada en estas pautas.

Si leéis la especificación del monitor proporcionado veréis que es necesario implementar dos constructores. La principal diferencia entre ellos es que en el segundo caso es necesario pasar como parámetro de entrada un puntero a un objeto de la clase `Logger`. Al igual que en la práctica anterior, este objeto nos va a permitir almacenar en un fichero log la traza de ejecución del programa. No obstante, en esta sesión el interés está en almacenar los eventos que describen el uso que hacen los procesos del monitor. Esta información es útil para estudiar las ejecuciones de un programa, de cara a analizar su comportamiento y detectar posibles incorrecciones.

La clase `Logger` proporciona la operación `addMessage(...)` para almacenar en el fichero log un evento. En el caso de los monitores, los eventos a guardar se corresponden con el inicio y el final de la ejecución de una operación del monitor. Por este motivo, en cada operación del monitor habrá que generar dos eventos, uno al principio, después de coger el *mutex*, y otro al final, antes de acabar la operación, conforme el modelo siguiente. El primer evento contiene la cadena constante `BEGIN_FUNC_PROC`, el nombre de la operación invocada (que deberá ser cambiado para cada operación concreta) y el número de surtidores libres en ese instante, separados por comas. El evento final es similar, pero cambia la cadena constante `END_FUNC_PROC`. Nótese que se ha utilizado la macro `LOG` para invocar a la operación `addMessage(...)` desde el código del monitor.

```
void ControlGasolinera::dameSurtidor(...) {
  unique_lock<mutex> lck(mtx);
  LOG(log,"BEGIN_FUNC_PROC,dameSurtidor,"+to_string(nLibres));

  ...

  LOG(log,"END_FUNC_PROC,dameSurtidor,"+to_string(nLibres));
}
```

A continuación, se muestra un fragmento de un fichero log similar al que se genera en esta práctica. La primera línea contiene la cabecera del log, donde se especifica el formato y los campos de información almacenados en el fichero. Estos campos son los siguientes: el identificador del evento (en esta práctica este ID no se utiliza y se le ha dado el valor constante "idUnico"), el tipo de evento de interés (`BEGIN_FUNC_PROC` o `END_FUNC_PROC`), la operación invocada del monitor, el número de surtidores libres, el identificador del proceso invocador, el *timestamp* y un *ticket* interno que se utiliza para garantizar la escritura ordenada de los mensajes en el fichero log compartido (por legibilidad en el ejemplo de fragmento de log los dos últimos campos de cada evento han sido sustituidos por puntos suspensivos).

```
ID,event,sectionID,val,procID,ts,ticket
...
idUnico,BEGIN_FUNC_PROC,beginMantenimiento,0,66350556928,...
idUnico,BEGIN_FUNC_PROC,dejoSurtidor,0,66633561856,...
idUnico,END_FUNC_PROC,dejoSurtidor,1,66633561856,...
```

```
idUnico,BEGIN_FUNC_PROC,dameSurtidor,1,66633561856,...
idUnico,BEGIN_FUNC_PROC,dejoSurtidor,1,66625169152,...
idUnico,END_FUNC_PROC,dejoSurtidor,2,66625169152,...
idUnico,BEGIN_FUNC_PROC,dameSurtidor,2,66625169152,...
idUnico,BEGIN_FUNC_PROC,dejoSurtidor,2,66641954560,...
idUnico,END_FUNC_PROC,dejoSurtidor,3,66641954560,...
idUnico,BEGIN_FUNC_PROC,dameSurtidor,3,66641954560,...
idUnico,BEGIN_FUNC_PROC,dejoSurtidor,3,66616776448,...
idUnico,END_FUNC_PROC,dejoSurtidor,4,66616776448,...
idUnico,BEGIN_FUNC_PROC,dameSurtidor,4,66616776448,...
idUnico,BEGIN_FUNC_PROC,dejoSurtidor,4,66599991040,...
idUnico,END_FUNC_PROC,dejoSurtidor,5,66599991040,...
idUnico,BEGIN_FUNC_PROC,dameSurtidor,5,66599991040,...
idUnico,BEGIN_FUNC_PROC,dejoSurtidor,5,66367342336,...
idUnico,END_FUNC_PROC,dejoSurtidor,6,66367342336,...
idUnico,BEGIN_FUNC_PROC,dameSurtidor,6,66367342336,...
idUnico,END_FUNC_PROC,beginMantenimiento,6,66350556928,...
idUnico,BEGIN_FUNC_PROC,endMantenimiento,6,66350556928,...
...
```

Finalmente, recordad que una forma de generar programas con y sin la opción de generar el log es mediante la compilación condicional definida por `#if DEBUG_MODE ... #else ... #endif` en el main, junto con `-D DEBUG_MODE=true` en la invocación al compilador en el Makefile. 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_e1` 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 14 de noviembre de 2018 deberán someter la práctica no más tarde del día 27 de noviembre de 2018 a las 23:59. Los alumnos pertenecientes a grupos de prácticas cuya cuarta sesión de prácticas se celebra el día 21 de noviembre de 2018 deberán someter la práctica no más tarde del día 4 de diciembre de 2018 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.