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 pareja de estudiantes 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 y presentará a los profesores antes del inicio de la sesión.* El documento debe contener como mínimo el nombre completo y el NIP de los dos estudiantes y, para cada ejercicio,

- un diseño de alto nivel de la solución que incluya la enumeración de los procesos involucrados y una descripción completa del monitor necesario 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.

El documento deberá llamarse `informe_P4_NIP1_NIP2.pdf` (donde `NIP1` **es el NIP menor** y `NIP2` **es el NIP mayor** de la pareja), y deberá entregarse antes del comienzo de la sesión de prácticas utilizando el comando `someter` en la máquina `hendrix.cps.unizar.es`

```
someter progc_20 informe_P4_NIP1_NIP2.pdf
```

Su entrega es un pre-requisito para la realización y evaluación de la práctica.

3. Generación de ficheros de log

En muchas situaciones es interesante almacenar información sobre la historia de ejecución, con el fin de poder analizar propiedades comportamentales, detectar problemas e inconsistencias, etc. En el caso de un programa secuencial es sencillo: se escriben eventos en un fichero conforme van sucediendo. En el caso de un programa concurrente, por cuestiones de entrelazados no es tan sencillo, ya que puede que los eventos no se guarden en el orden en que ocurrieron. Para ello suministramos la librería `Logger_V3`. La generación del log se puede usar solo durante la etapa de desarrollo, para poner a punto el programa, pero también se puede dejar para la etapa de explotación, pues podría ser útil para detectar posibles causas de problemas. En cualquier caso, que el programa genere un fichero de log o no se puede activar/desactivar en la compilación del mismo.

Una forma de compilar programas con y sin la opción de generar el log es la que se muestra en el ejemplo `pruebaSemaforos.cpp` que acompaña a la librería de semáforos, 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. Así, si en la invocación al compilador en el makefile aparece la opción `-D LOGGING_MODE`, entonces se define la variable interna `LOGGING_MODE`. Si no aparece, no está definida. Por otro lado, las líneas 20 a 27 del fichero `pruebaSemaforos.cpp` analizan si dicha variable está definida. Si no lo está, se ejecutará la línea 26, con lo que `ADD_EVENT(e)` será la instrucción vacía. En este caso, la línea 95 es lo mismo que una línea vacía. En caso contrario se ejecutará las líneas 21 (incluye las definiciones del logger), la 22 (generando el fichero de log `_log_.log`) con un buffer de capacidad 1024 (lo eventos se van guardando en una estructura de datos intermedia y son pasados al fichero cuando se hayan generado 1024 eventos o cuando se acabe el programa), y haciendo que la instrucción `ADD_EVENT` se defina como la línea 24 establece. Ahora la línea 95 ya no será vacía, sino que será, exactamente, {`_logger.addMessage(e);`}. Esta instrucción almacena un evento en el fichero de log. El evento se forma a partir del parámetro `e` (un string) junto con información del sistema (identificador del thread que lo ha generado, el timestamp del momento en que ha sucedido, ...). El contenido de `e` dependerá del programa concreto, que es lo que le dará el significado.

4. Ejercicio a desarrollar

Hay un tren turístico, para visitar parajes protegidos, compuesto por un único vagón con capacidad para 6 viajeros. Cuando un usuario llega al arcén, si las puertas de entrada

están abiertas y hay hueco en el vagón entra y se sienta. Si al llegar el vagón este está lleno o las puertas de entrada están cerradas, tendrá que esperar al siguiente viaje. Cada vez que un viajero sube a un vagón pasa su billete por un lector, que actualiza el contador del sistema del número de viajeros en el vagón.

Por su parte, el conductor espera a que el tren esté lleno (consultando el valor del contador de ocupación del vagón), momento en el que cierra las puertas de entrada y parte de viaje. Asumimos que el viaje dura un tiempo aleatorio de entre 10 y 15 milisegundos (que simulan una duración entre 10 y 15 minutos).

Una vez termina el viaje, el conductor abre las puertas de salida y espera a que el sistema detecte que cada uno de los viajeros ha descendido del vagón. En este momento avisa al servicio de limpieza, que lleva a cabo una desinfección del vagón. Una vez ha finalizado, el servicio avisa de su terminación al conductor. En este momento se puede iniciar un nuevo viaje.

Se trata de implementar un programa concurrente que simule el comportamiento del sistema descrito, asumiendo que hay 48 viajeros. Cada viajero, lo mismo que el conductor y el servicio de limpieza, se implementará mediante un thread.

Se proponen dos opciones para el desarrollo de la práctica. La primera opción consiste en implementar el sistema concurrente sin considerar el servicio de limpieza. La nota máxima que se podrá obtener será de 7,5 puntos sobre 10. La segunda opción requiere programar el sistema completo, incluida la limpieza del tren (ésta puntuará sobre 10 puntos). Sólo se podrá entregar una de las opciones, dado que la segunda ya incluye a la primera. A continuación se detalla el trabajo a realizar en cada caso particular.

5. Opción 1

Para el desarrollo de la primera opción se deben tener en cuenta las siguientes cuestiones:

- En el Anexo-I se muestra el código del `main` del programa (casi completo), en el que se ve el comportamiento de cada uno de los procesos involucrados. Toda la sincronización entre los procesos se lleva a cabo mediante el monitor `MonitorTren`, cuyo fichero de especificación se muestra en el Anexo-II. La práctica requiere que se completen los fuentes `practica_4_opc_1.cpp` y `MonitorTren.hpp`, así como que se desarrolle el código del monitor en `MonitorTren.cpp`.

- Los fuentes de los distintos anexos contienen el código necesario para programar el sistema completo. Por lo tanto, deberéis comentar todo aquello que sea relativo al servicio de limpieza: la creación del correspondiente proceso en el programa principal (líneas 49 y 55 del Anexo-I), el procedimiento que ejecutaría ese proceso (líneas 30 a 37 del Anexo-I), y las operaciones de sincronización en el monitor (líneas 37 a 42 del Anexo-II).

- Es también necesario que el programa desarrollado incluya la generación de un fichero de log, denominado `_log_.log`, de la misma manera que se hizo para la

práctica anterior. Además de los eventos que incluye el `main`, es necesario que cada operación pública del monitor genere un evento justo después de haber adquirido el cerrojo del monitor, y otro justo antes de terminar la ejecución de la operación. Por ejemplo

```
1 void MonitorTren::monta(const int i) {
2     //adquirir el cerrojo del monitor
3         ADD_EVENT("BEGIN_monta,"+to_string(i)+","+to_string(w));
4
5     ...
6         ADD_EVENT("END_monta,"+to_string(i)+","+to_string(w));
7 }
```

Las operaciones que toman como parámetro un identificador de usuario tendrán que generar los eventos de manera análoga a este. En el caso de las operaciones que tienen como parámetro el identificar del viaje, deberán generarlo análogamente al caso de `iniciaViaje`:

```
1 void MonitorTren::iniciaViaje(const int nV) {
2     //adquirir el cerrojo del monitor
3         ADD_EVENT("BEGIN_iniciaViaje,"+to_string(nV)+","+to_string(w));
4
5     ...
6         ADD_EVENT("END_iniciaViaje,"+to_string(nV)+","+to_string(w));
7 }
```

6. Opción 2

Para esta segunda opción se deberán tener en cuenta todas las cuestiones mencionadas en la opción anterior, y además:

- En este caso, la práctica requiere implementar el sistema completo. Este consistirá de los fuentes `practica_4_opc_2.cpp`, `MonitorTren.hpp` y `MonitorTren.cpp`, conforme se presentan en los anexos de este enunciado. Observad que contienen todo el código necesario para programar el servicio de limpieza.

7. Entrega de la práctica

Cuando la práctica se finalice, los dos componentes de la pareja deben entregar, cada uno desde su cuenta, el mismo fichero comprimido `practica_4_NIP1_NIP2.zip` (donde `NIP1` **es el NIP menor** y `NIP2` **es el NIP mayor** de la pareja) con el siguiente contenido:

1. El fichero `practica_4_opc_1.cpp` o el fichero `practica_4_opc_2.cpp` con el main de programa. Sólo se debe entregar uno de ellos.

2. Los ficheros `MonitorTren.hpp` y `MonitorTren.cpp` con el monitor programado.
3. El directorio `librerias` que se suministra, con la librería para generar ficheros de log.
4. El fichero `Makefile_p4` que compila el fuente, generando el ejecutable `practica_4`.
5. Todos los demás ficheros requeridos para que la ejecución de `make -f Makefile_p4` genere el ejecutable `practica_4`.

Generación del fichero `.zip` a entregar

Con el objetivo de homogeneizar los contenidos del fichero `.zip` vamos a proceder como sigue:

1. Creamos un directorio `practica_4_NIP1_NIP2` que contenga los ficheros que hay que entregar. Es importante tener presente que **se ha de hacer exactamente de esta manera**.
2. Con el botón derecho del ratón sobre la carpeta seleccionamos la opción "Compress..." y le damos en nombre requerido, `practica_4_NIP1_NIP2.zip`.
3. Alternativamente lo podemos hacer desde la terminal como sigue. Una vez creado el directorio `practica_4_NIP1_NIP2` con los ficheros pedidos ejecutamos lo siguiente desde la terminal:

```
zip -r practica_4_NIP1_NIP2.zip practica_4_NIP1_NIP2
```

Con el fin de comprobar que el `zip` contiene todos los ficheros que debe, y organizados adecuadamente, podéis ejecutar el script `pract_4_entrega_correcta.bash`. Leed la cabecera del fichero, que explica cómo utilizarlo.

Entrega del fichero en `hendrix`

Para la entrega del fichero `.zip` se utilizará el comando `someter` en la máquina `hendrix.cps.unizar.es`

```
someter progc_20 practica_4_NIP1_NIP2.zip
```

Fechas de entrega de la práctica

Los alumnos pertenecientes a grupos de prácticas cuya primera sesión de prácticas se celebra el día 18 de noviembre del 2020 deberán someter la práctica no más tarde del día 27 de noviembre a las 23:59. Los alumnos pertenecientes a grupos de prácticas cuya primera sesión de prácticas se celebra el día 25 de noviembre deberán someter la práctica no más tarde del día 4 de diciembre a las 23:59.

Hay que asegurarse de que la práctica funciona correctamente en los ordenadores del laboratorio (hay que 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.

Anexo-I

Esbozo del código de los procesos

```
1  //Fichero: renombrarAdecuadamente.cpp
2  ...
3  #include "MonitorTren.hpp"
4
5  using namespace std;
6
7  const int N = 48; //número de usuarios
8  const int N_W = 6; //capacidad del vagón
9  ...
10 //-----------------------------------------------------
11 void usuario(MonitorTren& monTren, const int i) {
12     monTren.monta(i);
13     //simula cierto retraso a la hora de bajarse
14     //sleep entre 5 y 10 ms
15     monTren.desmonta(i);
16 }
17 //-----------------------------------------------------
18 void conductor(MonitorTren& monTren, const int numViajes) {
19     for (int i=0; i<numViajes; i++) {
20         monTren.iniciaViaje(i);
21         //simula tiempo de duración del viaje
22         //sleep entre 10 y 15 ms
23         monTren.avisaFinViaje(i);
24         monTren.esperaHayanBajado(i);
25         monTren.avisaLimpieza(i);
26         monTren.esperaFinLimpieza(i);
27     }
28 }
29 //-----------------------------------------------------
30 void limpieza(MonitorTren& monTren, const int numViajes) {
31     for (int i=0; i<numViajes; i++) {
32         monTren.esperaAvisoInicio(i);
33         //simula tiempo de duración de la limpieza
34         //sleep entre 5 y 10 ms
35         monTren.avisaFinLimpieza(i);
36     }
37 }
38 //-----------------------------------------------------
39 int main() {
40     MonitorTren mT(N_W);
41
42     thread t_usuario[N];
43     thread t_conductor;
44     thread t_limpieza;
45     //-------------------------------------------------
46         ADD_EVENT("BEGIN_MAIN,0,0");
```

```
47
48      t_conductor = thread(conductor, ...);
49      t_limpieza = thread(limpieza, ...);
50      for(int i=0; i<N; i++) {
51          t_usuario[i] = thread(usuario, ...);
52      }
53
54      t_conductor.join();
55      t_limpieza.join();
56      for(int i=0; i<N; i++) {
57          t_usuario[i].join();
58      }
59          ADD_EVENT("END_MAIN,0,0");
60      //------------------------------------------------
61      return 0;
62  }
```

Anexo-II

Fichero (parcial) de especificación del monitor de control del sistema del tren:

```
1  //Fichero: MonitorTren.hpp
2  #ifndef MONITOR_TREN_HPP
3  #define MONITOR_TREN_HPP
4
5  #include <mutex>
6  #include <condition_variable>
7  //-------------------------------------------------------
8  using namespace std; //mutex, condition_variable, etc.
9
10 class MonitorTren {
11 private:
12     //Variables permanentes. Inicializar en el constructor.
13     int N_W; //tamaño del tren
14     int w; //personas dentro del vagón
15     bool puertasSalidaAbiertas;
16     bool puertasEntradaAbiertas;
17     int hanBajado;
18     ... //añadir las que se consideren necesarias
19
20     mutex mtx; //para la ejecución de procs en exclusión mutua
21     condition_variable trenVacio; //para esperar a que el tren esté vacío
22     ... //añadir las que se consideren necesarias
23
24 public:
25     //------------------------- constructores
26     //el parámetro "N_W" se copiará sobre el atributo "N_W" del monitor
27     MonitorTren(const int N_W);
28     //------------------------- destructor
29     ~MonitorTren();
30     //------------------------- usuario
31     //"i" es el identificador de usuario
32     void monta(const int i);
33     void desmonta(const int i);
34     //------------------------- conductor
35     //"nV" es el identificador del viaje
36     void iniciaViaje(const int nV);
37     void avisaFinViaje(const int nV);
38     void esperaHayanBajado(const int nV);
39     void avisaLimpieza(const int nV);
40     void esperaFinLimpieza(const int nV);
41     //------------------------- servicio de limpieza
42     //"nV" es el identificador del viaje
43     void esperaAvisoInicio(const int nV);
44     void avisaFinLimpieza(const int nV);
45 };
46 #endif
```

Anexo-III

Fichero (parcial) del log generado por la ejecución del programa considerado:

```
...
id_140527229663040,BEGIN_MAIN,0,0,1605027265640823525,140527229663040,1
id_140527229663040,BEGIN_iniciaViaje,0,0,1605027265641288292,140527229658880,2
id_140527229663040,BEGIN_esperaAvisoInicio
    ↪ ,0,0,1605027265641416803,140527221266176,3
id_140527229663040,BEGIN_monta,0,0,1605027265641485131,140527212873472,4
id_140527229663040,END_monta,0,1,1605027265641608747,140527212873472,5
id_140527229663040,BEGIN_monta,1,1,1605027265641666321,140527204480768,6
...
id_140527229663040,BEGIN_monta,3,5,1605027265642281763,140527187695360,14
id_140527229663040,END_monta,3,6,1605027265642369767,140527187695360,15
id_140527229663040,END_iniciaViaje,0,6,1605027265642396722,140527229658880,16
id_140527229663040,BEGIN_monta,9,6,1605027265642431736,140526934275840,17
...
id_140527229663040,BEGIN_desmonta,5,6,1605027265647425085,140526967846656,63
id_140527229663040,BEGIN_desmonta,3,6,1605027265648599899,140527187695360,64
id_140527229663040,BEGIN_avisaFinViaje
    ↪ ,0,6,1605027265652565580,140527229658880,65
id_140527229663040,END_avisaFinViaje
    ↪ ,0,0,6,1605027265652715987,140527229658880,66
id_140527229663040,BEGIN_esperaHayanBajado
    ↪ ,0,6,1605027265652750001,140527229658880,67
...
id_140527229663040,END_desmonta,2,0,1605027265653261124,140527196088064,73
id_140527229663040,END_esperaHayanBajado
    ↪ ,0,0,1605027265653324600,140527229658880,74
id_140527229663040,BEGIN_avisaLimpieza
    ↪ ,0,0,1605027265653352957,140527229658880,75
id_140527229663040,END_avisaLimpieza,0,0,1605027265653379650,140527229658880,76
id_140527229663040,BEGIN_esperaFinLimpieza
    ↪ ,0,0,1605027265653388567,140527229658880,77
id_140527229663040,END_esperaAvisoInicio
    ↪ ,0,0,1605027265653438720,140527221266176,78
id_140527229663040,BEGIN_avisaFinLimpieza
    ↪ ,0,0,1605027265658611063,140527221266176,79
id_140527229663040,END_avisaFinLimpieza
    ↪ ,0,0,1605027265658669667,140527221266176,80
...
id_140527229663040,END_desmonta,43,1,1605027265774305749,140525130729216,296
id_140527229663040,END_desmonta,45,0,1605027265774332672,140525113943808,297
id_140527229663040,END_esperaHayanBajado
    ↪ ,7,0,1605027265774352309,140527229658880,298
id_140527229663040,BEGIN_avisaLimpieza
    ↪ ,7,0,1605027265774360457,140527229658880,299
id_140527229663040,END_avisaLimpieza
    ↪ ,7,0,1605027265774366884,140527229658880,300
```

```
34 id_140527229663040,BEGIN_esperaFinLimpieza
       ↪ ,7,0,1605027265774370033,140527229658880,301
35 id_140527229663040,END_esperaAvisoInicio
       ↪ ,7,0,1605027265774384196,140527221266176,302
36 id_140527229663040,BEGIN_avisaFinLimpieza
       ↪ ,7,0,1605027265779460786,140527221266176,303
37 id_140527229663040,END_avisaFinLimpieza
       ↪ ,7,0,1605027265779478719,140527221266176,304
38 id_140527229663040,END_esperaFinLimpieza
       ↪ ,7,0,1605027265779515087,140527229658880,305
39 id_140527229663040,END_MAIN,0,0,1605027265779713568,140527229663040,306
```