Práctica 7: Exclusión mutua

Gustavo Romero

Arquitectura y Tecnología de Computadores

1 de diciembre de 2008

Objetivos

- **Verificar** la existencia de una **condición de carrera** en un programa que accede a un recurso compartido, el terminal, para imprimir un cierto mensaje.
- **Implementar** las diferentes **soluciones** vista en teoría para resolver el problema, tanto las que **funcionan** como las que **no**:
  - Algoritmo de la panadería.
  - Cerrojo: versión con condición de carrera.
  - Cerrojo: versión correcta con test_and_set.
- Programar otra solución que emplee los semáforos binarios (**mutex**) de la bibliteca Pthreads.
- Comparar el rendimiento de cada uno de ellos comparando:
  - El número de **mensajes correctos** impresos.
  - El número de **mensajes totales** impresos.
  - La justicia entre implementaciones: el **número de hebras diferentes** que son capaces de imprimir un mensaje.
  - El **tiempo** empleado (real/usuario/sistema).

Makefile

http://atc.ugr.es/~gustavo/aco/practicas/practica7/Makefile

Makefile

```
SRC = $(wildcard *.cc)
EXE = $(basename $(SRC))

CXXFLAGS += -g3 -O3 -Wall
LDFLAGS += -lpthread

all:    $(EXE)

clean:
        $(RM) $(EXE) *~ core.*
```

El Makefile completo tiene una opción "`make stat`" que permite comprobar la correción, el rendimiento y la justicia de las soluciones que vaya implementando.

Ejemplo: mensaje.cc I

http://atc.ugr.es/~gustavo/aco/practicas/practica7/mensaje.cc

Copie el programa mensaje.cc y **verifique** la existencia de una condición de carrera en su interior.

```
//-------------------------------------------------------------------
// mensaje.cc
//-------------------------------------------------------------------

#include <pthread.h>
#include <unistd.h>
#include <iostream>

//-------------------------------------------------------------------

using namespace std;

//-------------------------------------------------------------------

void seccion_critica()
{
  cout << "[" << pthread_self() << "]: ";
  for(unsigned i = 0; i < 10; ++i)
    cout << i;
  cout << endl;
}

//-------------------------------------------------------------------

void* hebra(void*)
```

Ejemplo: mensaje.cc II

http://atc.ugr.es/~gustavo/aco/practicas/practica7/mensaje.cc

```
{
  while(true)
    {
      seccion_critica();
    }
  return NULL;
}

//-------------------------------------------------------------------------

int main()
{
  const unsigned N = 100;
  pthread_t id[N];

  alarm(2); // para tras 2 segundos

  for(unsigned i = 0; i < N; ++i)
    pthread_create(&id[i], NULL, hebra, NULL);

  for(unsigned i = 0; i < N; ++i)
    pthread_join(id[i], NULL);
}

//-------------------------------------------------------------------------
```


Algoritmo de la panadería

http://atc.ugr.es/~gustavo/aco/practicas/practica7/panaderia.cc

- **Copie mensaje.cc** en otro fichero que debe llamar panaderia.cc.
- **Modifique panaderia.cc** de forma que se utilice el algoritmo de la panadería para conseguir la exclusión mutua en el acceso a la sección crítica por parte de las hebras.
- **Compare mensaje.cc con panaderia.cc** siguiendo todos los parámetros indicados en la página 2 (objetivos).

Cerrojo (con condición de carrera)

http://atc.ugr.es/~gustavo/aco/practicas/practica7/cerrojo.cc

- **Copie mensaje.cc** en otro fichero que debe llamar cerrojo.cc.
- **Modifique cerrojo.cc** de forma que se utilice un cerrojo para conseguir la exclusión mutua en el acceso a la sección crítica por parte de las hebras.
- Utilice la primera versión vista en clase que es **incorrecta** por contener una condición de carrera.
- **Compare mensaje.cc y panadería.cc con cerrojo.cc** siguiendo todos los parámetros indicados en la página 2 (objetivos).

```
class cerrojo
{
public:
    void adquirir() {}
    void liberar() {}
};
```


Cerrojo (con test_and_set)

http://atc.ugr.es/~gustavo/aco/practicas/practica7/tas.cc

- **Copie cerrojo.cc** en otro fichero que debe llamar tas.cc.
- **Modifique tas.cc** de forma que ahora el cerrojo funcione correctamente mediante el empleo de alguna instrucción atómica del procesador.
- **Compare mensaje.cc, panadería.cc y cerrojo.cc y tas.cc** siguiendo todos los parámetros indicados en la página 2 (objetivos).

```
bool testandset (volatile bool *spinlock)
{
  bool ret;
  __asm__ __volatile__("lock xchgb %0, %1"
                       : "=r"(ret), "=m"(*spinlock)
                       : "0"(true), "m"(*spinlock)
                       : "memory");
  return ret;
}
```

Pthreads: API de **mutex**

`pthread_mutex_t` Tipo mutex, inicializable a PTHREAD_MUTEX_INITIALIZER.

`pthread_mutex_init(mutex, attr)` Crea e inicializa `mutex` con los atributos `attr`.

`pthread_mutex_destroy(mutex)` Destruye `mutex`.

`pthread_mutex_lock(mutex)` Adquiere `mutex` en caso de estar libre. En otro caso bloquea la hebra.

`pthread_mutex_unlock(mutex)` Desbloquea `mutex`.

`pthread_mutex_trylock(mutex)` Intenta bloquear `mutex` y en caso de no poder continua sin bloquear la hebra.

Semáforos binarios (mutex) de Pthreads

`http://atc.ugr.es/~gustavo/aco/practicas/practica7/mutex.cc`

- **Copie mensaje.cc** en otro fichero que debe llamar mutex.cc.
- **Modifique mutex.cc** de forma que ahora se consiga la exclusión mutua mediante el uso de mutex de Pthreads.
- **Compare mensaje.cc, panaderia.cc, cerrojo.cc y tas.cc con mutex.cc** siguiendo todos los parámetros indicados en la página 2 (objetivos).

evaluación

La evaluación se realizará en función de dos parámetros:

- La corrección en la resolución de los problemas: 0-5.
- El rendimiento y la justicia de cada solución: 0-5.