Arquitectura de Sistemas

Práctica 9: Exclusión mutua

Gustavo Romero López

Updated: 14 de febrero de 2019

Arquitectura y Tecnología de Computadores

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 sabemos que **no**.
- Implementar **otras** soluciones de la literatura.
- **Comparar** el rendimiento de cada uno de ellos midiendo...
  - El número de **mensajes correctos** impresos.
  - El número de **mensajes totales** impresos.
  - La justicia, el número de **hebras diferentes** ejecutadas.
  - El **tiempo** empleado (real/usuario/sistema).

Soluciones propuestas

- Algoritmo de la panadería de Lamport.
- Cerrojo: versión incorrecta, con condición de carrera.
- Cerrojo: versión correcta con `test_and_set`.
- Cerrojo: versión correcta con `test_and_set` (`builtin`).
- Cerrojo: versión correcta con `test_test_and_set`.
- Semáforos binarios (`std::mutex`) de C++11.
- Semáforos binarios (`std::lock_guard`) de C++11.
- Ticket locks.
- Exponential backoff.

makefile

```
EXE = $(basename $(SRC))
ATT = $(EXE:=.att)
DAT = $(EXE:=.dat)
LOG = $(EXE:=.log)
ST  = $(EXE:=.st)

CFLAGS = -march=native -O3 -pthread -Wall -Wl,--no-as-needed # -
    fno-inline-functions -g
CXXFLAGS = $(CFLAGS) --std=c++14

all: stat

att: $(ATT)

clean:
  -rm -fv $(ATT) $(DAT) $(EXE) $(LOG) $(ST) core.* *~
  -find -mindepth 2 -iname 'makefile' -execdir make $@ \;

sort: stat
```

- Con "`make stat`" podrá comparar las soluciones que vaya implementando.

Ejemplo: `secuencial.cc` I

- Programa de partida.
- Al intentar paralelizarlo van a aparecer problemas.
- `mensaje.cc` es una versión paralela y errónea.

```
//----------------------------------------------------
// secuencial.cc
//----------------------------------------------------

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

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

using namespace std;
```

Ejemplo: secuencial.cc II

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

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

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

void hebra()
{
  while(true)
  {
    seccion_critica();
  }
}
```

Ejemplo: secuencial.cc III

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

int main()
{
  alarm(1);
  hebra();
}

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

Ejemplo: `mensaje.cc`

- **Verifique** la existencia de una condición de carrera en el programa `mensaje.cc`.
- **Analice** cuidadosamente la salida del programa para comprender por qué falla.

```
    thread t[N];

    alarm(1);
    for(auto& i: t) i = thread(hebra);
    for(auto& i: t) i.join();

}
```

- **Compare** los resultados de `secuencial.cc` y `mensaje.cc`.

Algoritmo de la panadería de Lamport: lamport.cc

- **Descargue** el fichero `lamport.cc`.
- **Modifique** `lamport.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 `lamport.cc` observando todos los parámetros señalados como objetivos.

Cerrojo incorrecto (con condición de carrera): cerrojo.cc

- **Descargue** `cerrojo.cc` y complete la implementación de la clase tal y como hemos visto en teoría.
- Utilice la primera versión vista en clase que es **incorrecta** por contener una condición de carrera.

```
class cerrojo {
public:
  cerrojo(): cerrado(false) {}
  void adquirir() {
    while (cerrado);
    cerrado = true;
  }
  void liberar () { cerrado = false; }
private:
  bool cerrado;
};
```

- **Compare** los resultados de `mensaje.cc`, `lamport.cc` y `cerrojo.cc`.

Cerrojo con `test_and_set`: tas.cc

- **Descargue** `tas.cc`.
- **Modifique** `tas.cc` de forma que ahora el cerrojo funcione correctamente mediante el empleo de instrucciones atómicas.

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

- **Compare** `mensaje.cc`, `lamport.cc`, `cerrojo.cc` y `tas.cc`.

Cerrojo con el `builtin` de `test_and_set`: tasb.cc

- **Descargue** `tasb.cc`.
- Por comodidad, **modifique** `tasb.cc` de forma que ahora el cerrojo funcione correctamente mediante el empleo del **builtin** `__sync_lock_test_and_set()`[1].
- **Compare** `mensaje.cc`, `lamport.cc`, `cerrojo.cc`, `tas.cc` y `tasb.cc`.

[1] `https://gcc.gnu.org/onlinedocs/gcc-4.1.2/gcc/Atomic-Builtins.html`

- Investigue por su cuenta qué es `test_and_test_and_set`.
- La idea es poder verificar si el cerrojo está cerrado accediendo a una variable en la caché local de cada procesador sin tener que acceder al cerrojo en memoria y así evitar tráfico en el bus del sistema.
- **Descargue** `ttas.cc`.
- **Modifique** `ttas.cc` de forma que ahora el cerrojo emplee la versión de `test_and_set` con doble comprobación.
- **Compare** `mensaje.cc`, `lamport.cc`, `cerrojo.cc`, `tas.cc`, `tasb.cc` y `ttas.cc`.

`std::atomic` de C++11: operaciones atómicas

- Las operaciones atómicas no deberían ser tan difíciles de usar y por eso han aparecido clases para poder utilizarlas fácilmente.
- Lo que hemos hecho hasta ahora puede conseguirse de manera más práctica empleando `std::atomic<bool>` o su especialización `std::atomic_flag`.
- **Descargue** `atomic.cc`.
- **Modifique** `atomic.cc` de forma que ahora el cerrojo emplee en su interior el tipo `std::atomic<bool>` para conseguir funcionar adecuadamente.
- **Compare** `mensaje.cc`, `lamport.cc`, `cerrojo.cc`, `tas.cc`, `tasb.cc`, `ttas.cc` y `atomic.cc`.

- Encontrarse el trabajo ya hecho es lo mejor... :)
- **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 los semáforos binarios de C++11, `std::mutex`[2].
- **Compare** `mensaje.cc`, `lamport.cc`, `cerrojo.cc`, `tas.cc`, `tasb.cc`, `ttas.cc`, `atomic.cc` y `mutex.cc`.

[2] `http://en.cppreference.com/w/cpp/thread/mutex`

- ¿Le suena **RAII** ("Resource Acquisition Is Initialization")[3]?
- `std::lock_guard`[4] es una versión RAII de `std::mutex`.
- **Copie** `mensaje.cc` en otro fichero que debe llamar `lock_guard.cc`.
- **Modifique** `lock_guard.cc` de forma que ahora se consiga la exclusión mutua mediante el uso de `std::lock_guard`.
- **Compare** `mensaje.cc`, `lamport.cc`, `cerrojo.cc`, `tas.cc`, `tasb.cc`, `ttas.cc`, `atomic.cc`, `mutex.cc` y `lock_guard.cc`.

[3] https://es.wikipedia.org/wiki/RAII

[4] http://en.cppreference.com/w/cpp/thread/lock_guard

Cerrojos basados en turnos... añadiendo algo de justicia

- Investigue qué son los cerrojos basados en turnos o *ticket locks*[5]
- **Copie** `mensaje.cc` en otro fichero que debe llamar `ticketlock.cc`.
- **Modifique** `ticketlock.cc` de forma que ahora se consiga la exclusión mutua mediante el uso de "ticket locks".
- **Compare** `mensaje.cc`, `lamport.cc`, `cerrojo.cc`, `tas.cc`, `tasb.cc`, `ttas.cc`, `atomic.cc`, `mutex.cc`, `lock_guard.cc` y `ticketlock.cc`.

[5] `http://academic.research.microsoft.com/Publication/303472/algorithms-for-scalable-synchronization-on-shared-memory-multiprocessors`

Marcha atrás exponencial: más eficiente todavía

- La mayoría de los métodos vistos hasta ahora pueden mejorarse aun más si aplicamos la técnica conocida como marcha atrás exponencial (*"exponential backoff"*):
  - **Dejar libre el procesador** en caso de encontrar el cerrojo ocupado.
  - La cantidad de **tiempo va aumentando** a medida que crece el número de reintentos.
- **Modifique** las soluciones vistas hasta ahora para incorporar esta mejora.
- **Compare** ambos tipos de soluciones.

¿Qué es más importante, el algoritmo o la implementación?

¿Se le ocurre algo mejor?

- ¿Sabes de alguna técnica/truco/implementación capaz de mejorar lo visto hasta ahora?
- ¡Cuéntanosla!