Práctica 8: Barreras

Gustavo Romero

Arquitectura y Tecnología de Computadores

14 de febrero de 2014

Objetivos

- Aprender a utilizar las **variables condición** y las **barreras** de la biblioteca Pthreads.
- Diseñar e implementar una barrera:
  - Barrera que no funciona (sólo probar)
  - Barrera que funciona sólo una vez (sólo probar).
  - Barrera con espera ocupada (sólo probar).
  - Barrera sin espera ocupada.
  - Barrera con variables condición.
  - Barrera de Pthread.
- Comprobar la funcionalidad y el rendimiento de las diferentes versiones (“`make all`”).
- La nota se calculará en función de...
  - La corrección las soluciónes: 0-5.
  - El rendimiento de las soluciones: 0-5 (variables condición).

Makefile

http://pccito.ugr.es/~gustavo/aco/practicas/practica8/Makefile

Makefile

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

CXXFLAGS = -D _GLIBCXX_USE_NANOSLEEP -O3 -std=c++0x -Wa
LDFLAGS  = -lpthread -lrt

default: $(EXE)

all:     stat
    @for d in $(DIR); do $(MAKE) -C $$d $@; done
```

Mediante "make all" podrá comparar fácilmente las soluciones que vaya programando.

Semáforos (POSIX)

`#include <semaphore.h>` Cabecera C/C++.

`sem_t` Tipo semáforo.

`sem_init(sem, attr, valor)` Inicializa el semáforo `sem` al calor `valor` con los atributos `attr`.

`sem_destroy(sem)` Destruye el semáforo `sem`.

`sem_wait(sem)` Si el valor del semáforo `sem` es positivo lo decrementa y retorna inmediatamente. En otro se bloquea hasta poder hacerlo.

`sem_trywait(sem)` Versión no bloqueante de `sem_wait(sem)`. En cualquier caso retorna inmediatamente. Es necesario comprobar la salida antes de continuar.

`sem_post(sem)` Incrementa el valor del semáforo `sem`. En caso de cambiar a un valor positivo desbloquea a alguno de los llamadores bloqueados en `sem_wait(sem)`.

Pthreads: API de variables condición

`pthread_cond_t` Tipo variable condición. Inicializable a PTHREAD_COND_INITIALIZER.

`pthread_cond_init(cond,attr)` Inicializa la variable condición `cond` con los atributos `attr`.

`pthread_cond_destroy(cond)` Destruye la variable condición `cond`.

`pthread_cond_wait(cond, mutex)` Bloque a la hebra llamadora hasta que se señale `cond`. Esta función debe llamarse mientras `mutex` está ocupado y ella se encargará de liberarlo automáticamente mientas espera. Despué de la señal la hebra es despertada y el cerrojo es ocupado de nuevo. El programador es responsable de desocupar `mutex` al finalizar la sección crítica para la que se emplea.

`pthread_cond_signal(cond)` Función para despertar a otra hebra que espera que se señale sobre la variable condición `cond`. Debe llamarse después de que `mutex` esté ocupado y se encarga de liberarlo en `pthread_cond_wait(cond, mutex)`.

`pthread_cond_broadcast(cond)` Igual que la función anterior para el caso de que queramos desbloquear a varias hebras que esperan.

Pthreads: API de barreras

`pthread_barrier_t` Tipo barrera.

`pthread_barrier_init(barrier, attr, n)` Inicializa la barrera `barrier` con los atributos `attr` para que funcione con `n` hebras.

`pthread_barrier_destroy(barrier)` Destruye la barrera `barrier`.

`pthread_barrier_wait(barrier)` Bloque a la hebra llamadora hasta que se `n` hebras ejecuten `pthread_barrier_wait(barrier)`.

Ejemplo: barrera.cc I

http://pccito.ugr.es/~gustavo/aco/practicas/practica8/barrera.cc

Copie el programa barrera.cc y **verifique** que la secuencia de ejecución no es correcta.

```
//------------------------------------------------------------------------
// barrera.cc
//------------------------------------------------------------------------

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

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

using namespace std;

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

const int HEBRAS = 10;                                  // numero de hebras

pthread_mutex_t display = PTHREAD_MUTEX_INITIALIZER;    // exclusion mutua display

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

class barrera_t
{
public:
    barrera_t(): contador(0) {}
    void esperar() {}
```

Ejemplo: barrera.cc II

http://pccito.ugr.es/~gustavo/aco/practicas/practica8/barrera.cc

```
private:
    int contador;
} barrera;

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

void* hebra(void*)
{
    while(true)
    {
        pthread_mutex_lock(&display);
        cout << pthread_self() << ": antes" << endl;
        pthread_mutex_unlock(&display);

        barrera.esperar();

        pthread_mutex_lock(&display);
        cout << pthread_self() << ": despues" << endl;
        pthread_mutex_unlock(&display);
    }
    return NULL;
}

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

int main()
{
```

Ejemplo: barrera.cc III

http://pccito.ugr.es/~gustavo/aco/practicas/practica8/barrera.cc

```
    pthread_t id[HEBRAS];

    alarm(1);

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

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

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