Multithreading y recursos compartidos

Di Paola Martín

martinp.dipaola <at> gmail.com

Facultad de Ingeniería
Universidad de Buenos Aires

Multithreading

```cpp
int counter = 0;

void inc() {
    ++counter;
}

int main(int argc, char* argv[]) {
    std::thread t1 {inc};
    std::thread t2 {inc};

    t1.join(); t2.join();
    return counter;
}
```

Instrucciones no atómicas

```
void inc() {
    ++counter;
}
```

```
mov eax <-- counter
add eax <-- 1
mov counter <-- eax
```

Acceso concurrente: caso felíz

Thread 1

```
1 mov eax <-- counter
  add eax <-- 1
  mov counter <-- eax
  ---
  ---
  ---
```

Thread 2

```
---
---
---
mov eax <-- counter
add eax <-- 1
mov counter <-- eax
```

Registros del Thread 1

eax = 0

Registros del Thread 2

eax =

Data segment (Threads 1 y 2)

counter = 0

Acceso concurrente: caso felíz

Thread 1

```
1 mov eax <-- counter
2 add eax <-- 1
  mov counter <-- eax
  ---
  ---
  ---
```

Thread 2

```
---
---
---
mov eax <-- counter
add eax <-- 1
mov counter <-- eax
```

Registros del Thread 1

eax = 1

Registros del Thread 2

eax =

Data segment (Threads 1 y 2)

counter = 0

Acceso concurrente: caso felíz

Thread 1

```
mov eax <-- counter
add eax <-- 1
mov counter <-- eax
---
---
---
```

Thread 2

```
---
---
---
mov eax <-- counter
add eax <-- 1
mov counter <-- eax
```

Registros del Thread 1
eax = 1

Registros del Thread 2
eax =

Data segment (Threads 1 y 2)
counter = 1

Acceso concurrente: caso felíz

Thread 1

```
1 mov eax <-- counter
2 add eax <-- 1
3 mov counter <-- eax
  ---
  ---
  ---
```

Thread 2

```
  ---
  ---
  ---
4 mov eax <-- counter
  add eax <-- 1
  mov counter <-- eax
```

Registros del Thread 1
eax = 1

Registros del Thread 2
eax = 1

Data segment (Threads 1 y 2)
counter = 1

Acceso concurrente: caso felíz

Thread 1

```
1  mov eax <-- counter
2  add eax <-- 1
3  mov counter <-- eax
   ---
   ---
   ---
```

Thread 2

```
   ---
   ---
   ---
4  mov eax <-- counter
5  add eax <-- 1
   mov counter <-- eax
```

Registros del Thread 1
eax = 1

Registros del Thread 2
eax = 2

Data segment (Threads 1 y 2)
counter = 1

Acceso concurrente: caso felíz

Thread 1

```
1 mov eax <-- counter
2 add eax <-- 1
3 mov counter <-- eax
  ---
  ---
  ---
```

Thread 2

```
  ---
  ---
  ---
4 mov eax <-- counter
5 add eax <-- 1
6 mov counter <-- eax
```

Registros del Thread 1
eax = 1

Registros del Thread 2
eax = 2

Data segment (Threads 1 y 2)
counter = 2

Acceso concurrente: race condition

Thread 1

```
1 mov eax <-- counter
  add eax <-- 1
  ---
  ---
  ---
  mov counter <-- eax
```

Thread 2

```
---
---
mov eax <-- counter
add eax <-- 1
mov counter <-- eax
---
```

Registros del Thread 1

eax = 0

Registros del Thread 2

eax =

Data segment (Threads 1 y 2)

counter = 0

Acceso concurrente: race condition

Thread 1

```
1 mov eax <-- counter
2 add eax <-- 1
  ---
  ---
  ---
  mov counter <-- eax
```

Thread 2

```
---
---
mov eax <-- counter
add eax <-- 1
mov counter <-- eax
---
```

Registros del Thread 1
eax = 1

Registros del Thread 2
eax =

Data segment (Threads 1 y 2)
counter = 0

Acceso concurrente: race condition

Thread 1

```
1  mov eax <-- counter
2  add eax <-- 1
   ---
   ---
   ---
   mov counter <-- eax
```

Thread 2

```
   ---
   ---
3  mov eax <-- counter
   add eax <-- 1
   mov counter <-- eax
   ---
```

Registros del Thread 1
eax = 1

Registros del Thread 2
eax = 0

Data segment (Threads 1 y 2)
counter = 0

Acceso concurrente: race condition

Thread 1

```
1  mov eax <-- counter
2  add eax <-- 1
   ---
   ---
   ---
   mov counter <-- eax
```

Thread 2

```
   ---
   ---
3  mov eax <-- counter
4  add eax <-- 1
   mov counter <-- eax
   ---
```

Registros del Thread 1

eax = 1

Registros del Thread 2

eax = 1

Data segment (Threads 1 y 2)

counter = 0

Acceso concurrente: race condition

Thread 1

```
1 | mov eax <-- counter
2 | add eax <-- 1
  | ---
  | ---
  | ---
  | mov counter <-- eax
```

Thread 2

```
  | ---
  | ---
3 | mov eax <-- counter
4 | add eax <-- 1
5 | mov counter <-- eax
  | ---
```

Registros del Thread 1
eax = 1

Registros del Thread 2
eax = 1

Data segment (Threads 1 y 2)
counter = 1

Acceso concurrente: race condition

Thread 1

```
1 mov eax <-- counter
2 add eax <-- 1
  ---
  ---
  ---
6 mov counter <-- eax
```

Thread 2

```
  ---
  ---
3 mov eax <-- counter
4 add eax <-- 1
5 mov counter <-- eax
  ---
```

Registros del Thread 1
eax = 1

Registros del Thread 2
eax = 1

Data segment (Threads 1 y 2)
counter = 1

Sincronización: mutual exclusion

```cpp
int counter = 0;
std::mutex m;

void inc() {
    m.lock();
    ++counter;
    m.unlock();
}
```

Acceso atómico

Thread 1

```
1 siz m
  mov eax <-- counter
  add eax <-- 1
  ---
  mov counter <-- eax
  zer m
  ---
```

Thread 2

```
---
---
---
siz m
---
---
siz m
```

Registros del Thread 1

eax =

Registros del Thread 2

eax =

Data segment (Threads 1 y 2)

counter = 0 mutex = 1

Acceso atómico

Thread 1

```
1 siz m
2 mov eax <-- counter
  add eax <-- 1
  ---
  mov counter <-- eax
  zer m
  ---
```

Thread 2

```
---
---
---
siz m
---
---
siz m
```

Registros del Thread 1

eax = 0

Registros del Thread 2

eax =

Data segment (Threads 1 y 2)

counter = 0 mutex = 1

Acceso atómico

Thread 1

```
1 siz m
2 mov eax <-- counter
3 add eax <-- 1
  ---
  mov counter <-- eax
  zer m
  ---
```

Thread 2

```
---
---
---
siz m
---
---
siz m
```

Registros del Thread 1

eax = 1

Registros del Thread 2

eax =

Data segment (Threads 1 y 2)

counter = 0   mutex = 1

Acceso atómico

Thread 1

```
1  siz m
2  mov eax <-- counter
3  add eax <-- 1
   ---
   mov counter <-- eax
   zer m
   ---
```

Thread 2

```
   ---
   ---
   ---
4  siz m
   ---
   ---
   siz m
```

Registros del Thread 1

eax = 1

Registros del Thread 2

eax =

Data segment (Threads 1 y 2)

counter = 0 mutex = 1

Acceso atómico

Thread 1

```
1 | siz m
2 | mov eax <-- counter
3 | add eax <-- 1
  | ---
5 | mov counter <-- eax
6 | zer m
  | ---
```

Thread 2

```
  | ---
  | ---
  | ---
4 | siz m
  | ---
  | ---
  | siz m
```

Registros del Thread 1

eax = 1

Registros del Thread 2

eax =

Data segment (Threads 1 y 2)

counter = 1   mutex = 0

Protección de los recursos: monitor

```cpp
class ProtectedCounter {
  int counter;
  std::mutex m;

  public:
  void inc() {
      m.lock();
      ++counter;
      m.unlock;
  }
};
```

```cpp
ProtectedCounter counter;

void inc() {
    counter.inc();
}
```

Proteger es más que usar mutexs

```
class ProtectedList {
  std::list<int> list;
  std::mutex m;

  public:
  bool has(int x) {
      m.lock();
      bool b = list.has(x);
      m.unlock;
      return b;
  }

  void add(int x) {
      m.lock();
      list.add(x);
      m.unlock;
  }
};
```

```
ProtectedList list;

void add_uniq(int x) {
    if (not list.has(x)) {
        list.add(x);
    }
}
```

Métodos de un monitor: critical sections

```cpp
class ProtectedList {
  std::list<int> list;
  std::mutex m;

  public:
  void add_if_hasnt(int x) {
      m.lock();
      if (not list.has(x))
          list.add(x);
      m.unlock;
  }
};
```

```cpp
ProtectedList list;

void add_uniq(int x) {
    list.add_if_hasnt(x);
}
```

Appendix

Referencias

Referencias I

- Bjarne Stroustrup.
  ***The C++ Programming Language*.**
  Addison Wesley, Fourth Edition.
- Andrew S. Tanenbaum.
  ***Modern Operating System*.**
  Prentice Hall, Second Edition.
- Mordechai Ben-Ari.
  Principles of Concurrent and Distributed Programming
  Addison Wesley, Second Edition.