Multithreading y recursos compartidos

Di Paola Martín

martinp.dipaola <at> gmail.com

Facultad de Ingeniería Universidad de Buenos Aires

Multithreading

```
int counter = 0;
3
    void inc() {
4
        ++counter;
5
6
    int main(int argc, char* argv[]) {
8
        std::thread t1 {inc};
9
        std::thread t2 {inc};
10
11
        t1.join(); t2.join();
12
        return counter;
13
```

Instrucciones no atómicas

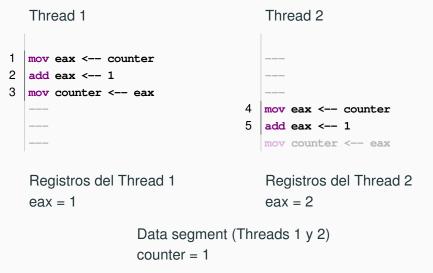
Thread 1 Thread 2 mov eax <-- counter add eax <-- 1 mov counter <-- eax mov eax <-- counter add eax <-- 1 mov counter <-- eax Registros del Thread 1 Registros del Thread 2 eax = 0eax =

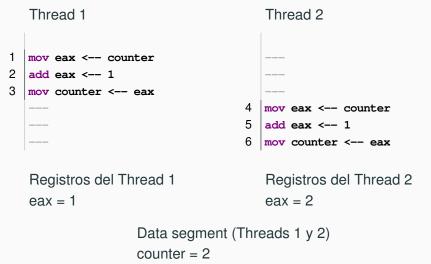
Data segment (Threads 1 y 2) counter = 0

```
Thread 1
                                      Thread 2
mov eax <-- counter
add eax \leftarrow 1
mov counter <-- eax
                                     mov eax <-- counter
                                      add eax <-- 1
                                     mov counter <-- eax
Registros del Thread 1
                                      Registros del Thread 2
eax = 1
                                      eax =
                 Data segment (Threads 1 y 2)
                 counter = 0
```

```
Thread 1
                                      Thread 2
mov eax <-- counter
add eax \leftarrow 1
mov counter <-- eax
                                     mov eax <-- counter
                                      add eax <-- 1
                                     mov counter <-- eax
Registros del Thread 1
                                      Registros del Thread 2
eax = 1
                                      eax =
                 Data segment (Threads 1 y 2)
                 counter = 1
```

```
Thread 1
                                      Thread 2
mov eax <-- counter
add eax \leftarrow 1
mov counter <-- eax
                                      mov eax <-- counter
                                      add eax <-- 1
                                      mov counter <-- eax
Registros del Thread 1
                                      Registros del Thread 2
eax = 1
                                      eax = 1
                 Data segment (Threads 1 y 2)
                 counter = 1
```





Thread 1

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

Registros del Thread 1 eax = 0

Data segment (Threads 1 y 2) counter = 0

Thread 2

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

Registros del Thread 2 eax =

Thread 1 Thread 2 mov eax <-- counter add eax \leftarrow 1 mov eax <-- counter add eax <-- 1 mov counter <-- eax mov counter <-- eax Registros del Thread 1 Registros del Thread 2 eax = 1eax = Data segment (Threads 1 y 2) counter = 0

Thread 1 mov eax <-- counter add eax <-- 1 --mov counter <-- eax

Registros del Thread 1 eax = 1

Data segment (Threads 1 y 2) counter = 0

Thread 2



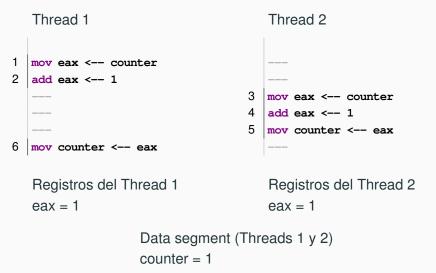


Data segment (Threads 1 y 2) counter = 0

Registros del Thread 2 eax = 1



counter = 1



Sincronización: mutual exclusion

```
1  int counter = 0;
2  std::mutex m;
3  
4  void inc() {
    m.lock();
    ++counter;
    m.unlock();
8  }
```



Data segment (Threads 1 y 2)

counter = 0 mutex = 1

8

eax = 0



Data segment (Threads 1 y 2)

counter = 0 mutex = 1

eax =



Data segment (Threads 1 y 2) counter = 0 mutex = 1



Data segment (Threads 1 y 2) counter = 0 mutex = 1

eax = 1



Data segment (Threads 1 y 2) counter = 1 mutex = 0

eax =

Protección de los recursos: monitor

```
class ProtectedCounter {
      int counter;
3
      std::mutex m;
4
5
      public:
6
      void inc() {
          m.lock();
8
          ++counter;
          m.unlock;
10
```

```
1 ProtectedCounter counter;
2 
3 void inc() {
4    counter.inc();
5 }
```

Proteger es más que usar mutexs

```
class ProtectedList {
2
      std::list<int> list;
3
      std::mutex m:
4
5
      public:
6
      bool has(int x) {
                                          ProtectedList list;
          m.lock();
                                       2
8
          bool b = list.has(x);
                                       3
                                          void add uniq(int x) {
9
          m.unlock;
                                       4
                                               if (not list.has(x)) {
10
          return b;
                                                   list.add(x);
11
                                       6
12
13
      void add(int x) {
14
          m.lock();
15
          list.add(x);
16
          m.unlock;
17
18
```

Métodos de un monitor: critical sections

```
class ProtectedList {
      std::list<int> list;
3
      std::mutex m;
4
                                          ProtectedList list;
5
      public:
                                       2
6
      void add_if_hasnt(int x) {
                                       3
                                          void add uniq(int x) {
          m.lock();
                                       4
                                              list.add_if_hasnt(x);
8
          if (not list.has(x))
                                       5
9
              list.add(x);
10
          m.unlock;
11
12
```

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 ProgrammingAddison Wesley, Second Edition.