|  |  |
| --- | --- |
|  | **2014** |
|  | The Company  Huijsmans, Martien |

|  |
| --- |
| **[Threading]** |
| Threading in C++, Java and Python land |

# C++

## Atomic

This is a great references (for some parts)  
<http://en.cppreference.com/w/cpp/atomic/memory_order>

Question:  
Why do all the examples (here and elsewhere) have the same  
while (!(p2 = ptr.load(std::memory\_order\_acquire)));  
for the consumer.  
Is that how it is supposed to work / to be used?  
  
Question:  
There is no discussion about time for Release-acquire and release-consume.  
So if thread1 performs a: ptr.store(p, std::memory\_order\_release);  
and terminates  
and thread2 performs  
thread1.join();  
std::string\* p2 = ptr.load(std::memory\_order\_acquire);  
Will thread2 see correct value of ptr?  
Or is it seen eventually and thus you need a loop (like in previous bullet)?

This link: <http://stackoverflow.com/questions/20130571/c11-ensuring-defined-semantics-of-write-once-read-many?rq=1> holds following question and answer:

**To question how fast other threads see writes on your atomic variable, the standard says the following: (§ 29.3.13)**

**Implementations should make atomic stores visible to atomic loads within a reasonable amount of time.**

Following code run ok (but doesn’t mean anything in MT-land).   
I think that it is wrong and that it must be a while (!(p2 = ptr.load(std::memory\_order\_acquire))); But is that a free running (cpu burner)?  
  
#include <thread>  
#include <atomic>  
#include <cassert>  
#include <string>  
#include <iostream>  
  
std::atomic<std::string\*> ptr;  
int data;  
  
void producer()  
{  
 std::string\* p = new std::string("Hello");  
 data = 42;  
 ptr.store(p, std::memory\_order\_release);  
}

void consumer()  
{  
 std::string\* p2 = ptr.load(std::memory\_order\_acquire);  
 assert(\*p2 == "Hello"); // never fires  
 assert(data == 42); // never fires  
}

int main()  
{  
 std::cout << "starting" << std::endl;   
 for(int i=0; i<1000; i++)   
 {  
 std::thread t1(producer);  
 t1.join();  
 consumer();  
 }  
 std::cout << "ready" << std::endl;   
}