Advanced Programming Techniques – Lecture 21

- Dr.-Ing. Harald Köstler

- 15.1.2013

C++ Threads

PARALLEL PROGRAMMING

Parallel Programming

- Multithreaded programming allows you to perform multiple calculations in parallel
- Typical Problems:
  - Race conditions: can occur when multiple threads want to read/write to a shared memory location
  - Deadlocks: threads that are blocking indefinitely because they are waiting to acquire access to resources currently locked by other blocked threads

Atomics (C++11)

- Allow atomic accesses, which means that concurrent reading and writing without additional synchronization is possible
- In this way race conditions can be solved
- Example
  - atomic<int> counter(0); // global variable
  - ++counter; // executed in multiple threads

C++ Threads (C++11)

```cpp
#include <iostream>
#include <thread>
using namespace std;

void counter(int id, int numIterations) {
    for (int i = 0; i < numIterations; ++i) {
      cout << "Counter " << id << " has value "; cout << i << endl;
} }

int main() {
  cout.sync_with_stdio(true); // Make sure cout is thread-safe

  thread t1(counter, 1, 6);
  thread t2(counter, 2, 4);

  t1.join();
  t2.join();

  return 0;
}
```

Mutual Exclusion (mutex) (C++11)

- Step 1: A thread wants to read/write to memory shared with another thread and tries to lock a mutex object. If another thread is currently holding this lock, the thread blocks until the lock is released
- Step 2: Once the thread has obtained the lock, it is free to read/write to shared memory
- Step 3: After the thread is finished with reading/writing it releases the lock. If two or more threads are waiting on the lock, there are no guarantees as to which thread will be granted the lock

Locks (C++11)

```
#include <mutex>
using namespace std;

mutex mut1;
mutex mut2;

void process() {
  unique_lock<mutex> lock1(mut1, defer_lock_t());
  unique_lock<mutex> lock2(mut2, defer_lock_t());
  lock(lock1, lock2); // Locks acquired
}

int main() {
  process();
  return 0;
}
```

Inter-thread communication (C++11)

```
#include <iostream>
#include <future>
using namespace std;

int calculate() {
  return 123;
}

int main() {
  auto fut = async(calculate);
  //auto fut = async(launch::async, calculate);
  //auto fut = async(launch::deferred, calculate);

  // Do some more work...

  // Get result
  int res = fut.get();
  cout << res << endl;
  return 0;
}
```