Real-Time Software (RTSOF)

EVMINX9 Week 2

DE HAAGSE HOGESCHOOL

C++ Threads

- C++ heeft (nog) geen standaard library voor concurrent programmeren.
  - Boost Thread library http://www.boost.org/
  - Intel Threading Building Blocks (TBB) http://www.threadingbuildingblocks.org/
  - Parallel Pattern Library (PPL) http://msdn.microsoft.com/en-us/magazine/dd434652.aspx
- Er zijn ook uitbreidingen van C++ die concurrency aan de taal toevoegen (met taalconstructies).
  - µC++ voegt onder andere taalconstructies task en monitor toe. http://plg.uwaterloo.ca/~usystem/uC++.html

Real-Time?

Boost Thread

```
#include <boost/thread.hpp>
//…
void print1() {
    for (int i(0); i<10; ++i) {
        boost::this_thread::sleep(boost::posix_time::milliseconds(10));
        std::cout<<"print1"<<std::endl;
    }
}
void print2() {
    for (int i(0); i<10; ++i) {
        boost::this_thread::sleep(boost::posix_time::milliseconds(20));
        std::cout<<"print2"<<std::endl;
    }
}
```

```
int main() {
    boost::thread t1(&print1), t2(&print2);
    t1.join(); t2.join();
    std::cin.get(); return 0;
}
```

Boost Thread


```
#include <boost/thread.hpp>
//…
void print(int d, const std::string& m) {
    for (int i(0); i<10; ++i) {
        boost::this_thread::sleep(boost::posix_time::milliseconds(d));
        std::cout<<m<<std::endl;
    }
}

int main() {
    boost::thread t1(&print, 10, "print1");
    boost::thread t2(&print, 20, "print2");
    t1.join();
    t2.join();
    std::cin.get();
    return 0;
}
```

Wel type-safe (in tegenstelling tot pthreads).

```
print1
print2
print1
print1
print2
print1
print1
print2
print1
print1
print2
print1
print1
print2
print1
print2
print2
print2
print2
print2
```

Boost Thread

- thread::hardware_concurrency()
  - Geeft het aantal hardware threads wat op het huidige systeem beschikbaar is (= aantal CPU's of cores of hyperthreading units), of 0 als deze informatie niet beschikbaar is.

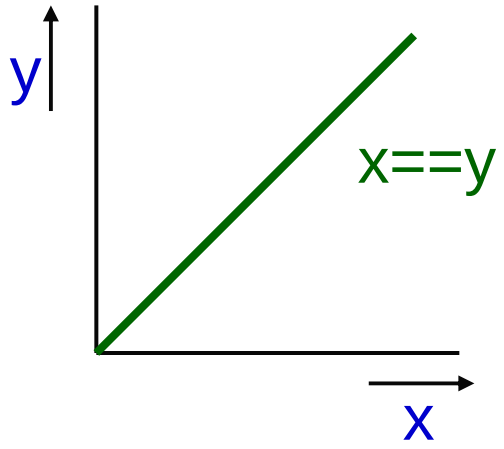
- t.native_handle();
  - Geeft een instantie van native_handle_type wat gebruikt kan worden met de platform-specific API om de onderliggende implementatie te programmeren.
  - Kan in een real-time pthread omgeving gebruikt worden om de prioriteit in te stellen.

Boost Synchronization

- Mutex
  - mutex
    - void lock();
    - bool try_lock();
    - void unlock();
  - recursive_mutex
    - Idem maar telt aantal locks (en aantal unlocks).
  - shared_mutex (Multiple-reader / single-writer)
    - void lock();
    - bool try_lock();
    - void unlock();
    - void lock_shared();
    - bool try_lock_shared();
    - bool unlock_shared();
- Lock
- Conditional variable
- Barrier

Mutex voorbeeld

```
class Point {
public:
    Point(): x(0), y(0) {
    }
    void stepNorthEast() {
        x++;
        y++;
    }
    bool isNorthEast() {
        return x==y;
    }
private:
    int x, y;
};
```



Wat gebeurt er als deze class in een multi-threaded omgeving gebruikt wordt?

Mutex voorbeeld

```
class TestPoint {
private:
    Point p;
    void thread1() {
        for (int i(0); i < 20000000; i++)
            p.stepNorthEast();
    }
    void thread2() {
        for (int i(0); i < 20000000; i++)
            if (!p.isNorthEast())
                std::cout<<"Probleem!"<<std::endl;
    }
public:
};
```

```
void test() {
    boost::thread t1(&TestPoint::thread1, this);
    boost::thread t2(&TestPoint::thread2, this);
    t1.join(); t2.join();
    std::cout<<"Einde."<<std::endl;
}
```

Probleem!
Probleem!
Einde.

Mutex Voorbeeld

```
class Point {
public:
    Point(): x(0), y(0) { }
    void stepNorthEast() {
        m.lock();
        x++; y++;
        m.unlock();
    }
    bool isNorthEast() {
        m.lock();
        bool res=x==y;
        m.unlock();
        return res;
    }
private:
    boost::mutex m;
    int x, y;
};
```

Einde.

	Run-time
Zonder mutex	1.5 s
Met mutex	101.2 s

Mutex voorbeeld

```
class Point {
public:
    Point(): x(0), y(0) { }
    void stepNorthEast() {
        m.lock(); x++; y++; m.unlock();
    }
    bool isNorthEast() {
        m.lock(); bool r(x==y); m.unlock(); return r;
    }
    bool isWithinLimits() {
        m.lock();
        bool r(x>=0&&x<=1000000&&y>=0&&y<=1000000);
        m.unlock(); return r;
    }
private:
    boost::mutex m;
    int x, y;
};
```

Je wilt niet onnodig wachten!

Mutex voorbeeld

isNorthEast() en isWithinLimits kunnen nu parallel draaien!

```
class Point {
public:
    Point(): x(0), y(0) { }
    void stepNorthEast() {
        m.lock(); x++; y++; m.unlock();
    }
    bool isNorthEast() {
        m.lock_shared(); bool r(x==y); m.unlock_shared(); return r;
    }
    bool isWithinLimits() {
        m.lock_shared();
        bool r(x>=0&&x<=1000000&&y>=0&&y<=1000000);
        m.unlock_shared(); return r;
    }
private:
    boost::shared_mutex m;
    int x, y;
};
```

Boost Synchronization

Een lock maakt het gebruik van een mutex **eenvoudiger** en **exception safe**.

Zie C++ exceptions (OGOPRG dictaat par.6.6).

- Mutex
- Lock
  - lock_guard
    - Gebruikt RAII (Resource Acquisition Is Initialization):
      - Lock mutex in constructor, unlock mutex in destructor
  - unique_lock
    - Idem als lock_guard + lock(), try_lock() en unlock() memberfuncties
  - shared_lock
    - Idem als unique_lock maar gebruikt _**shared** versies van lock, try_lock en unlock.
- Conditional variable
- Barrier

Mutex Voorbeeld

```
class Point {
public:
    Point(): x(0), y(0) {
    }
    void stepNorthEast() {
        boost::lock_guard<boost::mutex> lock(m);
        x++;
        y++;
    }
    bool isNorthEast() {
        boost::lock_guard<boost::mutex> lock(m);
        return x==y;
    }
private:
    boost::mutex m;
    int x, y;
};
```

Mutex voorbeeld

```
…
std::vector<int> v;
boost::mutex m;
…
    m.lock();
    v.at(10)=27;
    m.unlock();
…
```

```
…
    m.lock();
    try() {
        v.at(10)=27;
        m.unlock();
    catch(…) {
        m.unlock();
        throw;
    }
…   …
        {
            boost::lock_guard<boost::mutex> l(m);
            v.at(10)=27;
        }
        …
```

Niet handig!

Wat gaat er **mis** als at een exception gooit?

Oplossing !

Boost Synchronization

- Mutex
- Lock
- Conditional variable
  - conditional_variable
    - void notify_one();
    - void notify_all();
    - void wait(boost::unique_lock<boost::mutex>& lock);
    - void wait(boost::unique_lock<boost::mutex>& lock, pred_type pred);
  - conditional_variable_any
    - Idem maar werkt met elk type lock
- Barrier

Monitor voorbeeld

```
#include <boost/thread.hpp>
using namespace boost;

int ei_teller(0);

mutex m;
condition_variable c;

//  consumer thread
...
   {
      unique_lock<mutex> u(m);
      while (ei_teller<12)
         c.wait(u);
      ei_teller-=12;
   }
...
```

```
//  producer thread
...
   {
      lock_guard<mutex> g(m);
      ei_teller+=n;
   }
   c.notify_one();
...
```

Monitor voorbeeld

```
    while (ei_teller<12)
        c.wait(u);

// alternatief met predicate functie:
bool pred() {
    return ei_teller>=12;
}
…
    c.wait(u, &pred);

// alternatief met lambda functie (C++0x):
…
    c.wait(u, []() {
        return ei_teller>=12
    });
```

lambda functies zijn behandeld bij de cursus ALDAT

Boost Synchronization

- Mutex
- Lock
- Conditional variable
- Barrier
  - barrier
    - barrier(unsigned count);
    - bool wait();

Een barrier is een ontmoetingspunt (rendezvous). Pas nadat count threads een wait hebben gedaan mogen ze samen verder.