Exercises week 8 - Multi-threading II

Klaas Isaac Bijlsma s2394480

David Vroom s2309939

February 1, 2018

Exercise 57

Learn to design and implement a Semaphore class

We used the following code,

semaphore/semaphore.h

```
#ifndef INCLUDED_SEMAPHORE_H
#define INCLUDED_SEMAPHORE_H

#include <mutex>
#include <condition_variable>

class Semaphore
{
    mutable std::mutex d_mutex;
    std::condition_variable d_condition;
    size_t d_nAvailable;

    public:
        Semaphore(size_t nAvailable);
        void notify();
        void notify_all();
        size_t size() const;
        void wait();
};

```

```
inline Semaphore::Semaphore(size_t nAvailable)
:
    d_nAvailable(nAvailable)
{}

inline size_t Semaphore::size() const
{
    return d_nAvailable;
}

#endif
```

semaphore/semaphore.ih

```
#include "semaphore.h"

using namespace std;
```

semaphore/notify.cc

```
#include "semaphore.ih"

void Semaphore::notify()
{
    lock_guard<mutex> lg(d_mutex);

    if (d_nAvailable++ == 0)
        d_condition.notify_one();
}
```

semaphore/notifyall.cc

```
#include "semaphore.ih"

void Semaphore::notify_all()
{
    lock_guard<mutex> lg(d_mutex);

    if (d_nAvailable++ == 0)
        d_condition.notify_all();
}
```

semaphore/wait.cc

```
#include "semaphore.ih"

void Semaphore::wait()
{
    unique_lock<mutex> ul(d_mutex);
    while (d_nAvailable == 0)
        d_condition.wait(ul);

    --d_nAvailable;
}
```

Exercise 58

Become familiar with ***packaged_task***

We used the following code,

main.cc

```
#include <iostream>
#include <future>
#include <thread>
#include <iomanip>

using namespace std;

double lhs[4][5] =
{
    {1, 2, 3, 4, 1},
    {3, 4, 5, 7, 4},
    {2, 4, 5, 9, 3},
    {21, 8, 9, 42, 4}
};

double rhsT[6][5] =
{
    {1, 2, 3, 4, 2},
    {3, 4, 5, 7, 2},
    {2, 4, 5, 90, 3},
    {21, 8, 9, 42, 4},
    {1, 2, 3, 4, 8},
    {3, 4, 5, 7, 4}
};

enum
{
    ROWS = 4,
    COLS = 6,
    COMMON = 5,
};

future<double> fut[4][6];
```

```

double innerProduct(size_t row, size_t col)
{
    double sum = 0;
    for (size_t idx = 0; idx != COMMON; ++idx)
        sum += lhs[row][idx] * rhsT[col][idx];
    return sum;
}

void computeElement(size_t row, size_t col)
{
    packaged_task<double (size_t, size_t)> task(innerProduct);
    fut[row][col] = task.get_future();
    thread(move(task), row, col).detach();
}

int main()
{
    for (size_t row = 0; row != ROWS; ++row)
        for (size_t col = 0; col != COLS; ++col)
            computeElement(row, col);

    for (size_t row = 0; row != ROWS; ++row)
    {
        for (size_t col = 0; col != COLS; ++col)
        {
            try
            {
                cout << setw(5) << fut[row][col].get();
            }
            catch (exception &msg)
            {
                cout << "Exception: " << msg.what() << ’\n’;
            }
        }
        cout << ’\n’;
    }
}
```

Exercise 59

Become familiar with ***`packaged_task (2)`***

Exercise 60

Learn to implement a multi-threaded algorithm (2)

We used the following code,

main.cc

```
#include <iostream>
#include <algorithm>
#include <future>

using namespace std;

void quickSort(int *beg, int *end)
{
    if (end - beg <= 1)
        return;

    int lhs = *beg;
    int *mid = partition(beg + 1, end,
        [&](int arg)
        {
            return arg < lhs;
        }
    );

    swap(*beg, *(mid - 1));

    async(launch::async, quickSort, beg, mid);
    async(launch::async, quickSort, mid, end);
}

int main()
{
    int ia[] = {16, 2, 77, 40, 12071, 12, 3134, 42,
                5, 2453, 45, 3456, 35, 6, 56, 546, 2};

    size_t iaSize = 17;

    quickSort(ia, ia + iaSize);

    for (int el: ia)
        cout << el << ’\n’;
}
```

Exercise 62

Learn to inspect one or more ***`futures`*** *from inside a repeat-statement, even if the future has not yet been made ready*

We used the following code,

main.cc

```
#include <iostream>
#include <string>
#include <chrono>
#include <thread>
#include <future>

using namespace std;

string threadFun()
{
    cerr << "entry\n";

    this_thread::sleep_for(chrono::seconds(5));
    cerr << "first cerr \n";

    this_thread::sleep_for(chrono::seconds(5));
    cerr << "second cerr\n";

    return "done\n";
}

int main()
{
    future<string> fut = async(launch::async, threadFun);

    size_t count = 0;

    while (true)
    {
        this_thread::sleep_for(chrono::seconds(1));
        cerr << "inspecting: " << ++count << ’\n’;

        future_status status = fut.wait_for(0ms);
        if (status == future_status::ready)
            break;
    }

    try
    {
        cout << fut.get();
    }
    catch (exception const &msg)
    {
        cout << msg.what() << ’\n’;
    }
}
```