J.B
13

Exercises week 8: Multi-threading II - Revision

Klaas Isaac Bijlsma
s2394480

David Vroom
s2309939

February 14, 2018

Exercise 58

1

Become familiar with `packaged_task`

In the previous attempt, we had made `lhs`, `rhsT` and `fut` global. Also, we didn't use the defined enums everywhere. Furthermore, we used our own function to compute an inner product and didn't return nonzero in the case of an exception. All these things are improved now.

main.cc

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

using namespace std;

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

void computeElement(double *rowPtr, double *colPtr, size_t row,
                    size_t col, future<double> (*fut)[COLS])
```

I like typedefs.

```
{
    packaged_task<double (double *, double *, double *, double)>
                    task(inner_product<double *, double *, double>);
    fut[row][col] = task.get_future();
    thread(move(task), rowPtr, rowPtr + COMMON, colPtr, 0).detach();
}

int main()
{
    double lhs[ROWS][COMMON] =
    {
        {1, 2, 3, 4, 1},
        {3, 4, 5, 7, 4},
        {2, 4, 5, 9, 3},
        {21, 8, 9, 42, 4}
    };

    double rhsT[COLS][COMMON] =
    {
        {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}
    };

    future<double> fut[ROWS][COLS];

    for (size_t row = 0; row != ROWS; ++row)
        for (size_t col = 0; col != COLS; ++col)
            computeElement(lhs[row], rhsT[col], row, col, fut);

    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';
                return 1;
            }
        }
        cout << '\n';
    }
}
```

Exercise 60

1

Learn to implement a multi-threaded algorithm (2)

In the previous attempt, it was not guaranteed that the threads finish before the function returns. This is fixed now.

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));

    future<void> fut1 = async(launch::async, quickSort, beg, mid);
    future<void> fut2 = async(launch::async, quickSort, mid, end);

    fut1.get(); // block current thread until nested threads are ready
    fut2.get();
}

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';
}
```