Programming in C/C++
Exercises set seven: multi-threading 2

Christiaan Steenkist
Jaime Betancor Valado
Remco Bos

January 13, 2017

Exercise 47, semaphore design

We designed a semaphore as shown in the annotations. This semaphore was upgraded for exercise 49.

Code listings

Listing 1: main.cc

```
1  #include "semaphore.h"
2  #include <thread>
3
4  using namespace std;
5
6  void waiter(Semaphore &semaphore)
7  {
8    semaphore.wait();
9  }
10
11 int main(int argc, char **argv)
12 {
13   Semaphore semaphore;
14
15   thread wait1(waiter, std::ref(semaphore));
16
17   semaphore.notify_all();
```

```
18
19    wait1.join();
20  }
```

Listing 2: semaphore.ih

```
1  #include "semaphore.h"
2
3  using namespace std;
```

Listing 3: semaphore.h

```
1  #ifndef SEMAPHORE_H
2  #define SEMAPHORE_H
3
4  #include <condition_variable>
5  #include <mutex>
6  #include <cstddef>
7
8  class Semaphore
9  {
10   std::mutex d_mutex;
11   std::condition_variable d_condition;
12   std::size_t d_nAvailable = 0;
13
14   public:
15     Semaphore() = default;
16     Semaphore(std::size_t nAvailable);
17
18     std::size_t size() const;
19
20     void notify();
21     void notify_all();
22     void wait();
23 };
24
25 #endif
```

Listing 4: constructor.cc

```
1  #include "semaphore.ih"
2
```

```
3  Semaphore::Semaphore(size_t nAvailable)
4  :
5    d_nAvailable(nAvailable)
6  {
7  }
```

Listing 5: notify.cc

```
1  #include "semaphore.ih"
2
3  void Semaphore::notify()
4  {
5    lock_guard<mutex> lock(d_mutex);
6    if (d_nAvailable++ == 0)
7      d_condition.notify_one();
8  }
```

Listing 6: notify_all.cc

```
1  #include "semaphore.ih"
2
3  void Semaphore::notify_all()
4  {
5    lock_guard<mutex> lock(d_mutex);
6    if (d_nAvailable++ == 0)
7      d_condition.notify_all();
8  }
```

Listing 7: size.cc

```
1  #include "semaphore.ih"
2
3  size_t Semaphore::size() const
4  {
5    return d_nAvailable;
6  }
```

Listing 8: wait.cc

```
1  #include "semaphore.ih"
2
3  void Semaphore::wait()
4  {
```

```
5     unique_lock<mutex> lock(d_mutex);
6     while (d_nAvailable == 0)
7       d_condition.wait(lock);
8
9     --d_nAvailable;
10  }
```

Exercise 48, async quicksort

Here is a quicksort algorithm using async threads called quickersort. Because async always returns something we decided to return the start of the array.

Code listings

Listing 9: main.ih

```
1  #include "main.h"
2
3  using namespace std;
```

Listing 10: main.h

```
1  #ifndef MAIN_H
2  #define MAIN_H
3
4  #include <future>
5  #include <algorithm>
6  #include <iostream>
7
8  int* quickersort(int *beg, int *end);
9
10 #endif
```

Listing 11: main.cc

```
1  #include "main.ih"
2
3  int main(int argc, char **argv)
4  {
5      int ia[] = {2,4,6,2,3,7,9,1,12};
6      size_t iaSize = 9;
```

```
7       auto fut = async(launch::async, quickersort, ia,
        ia + iaSize);
8       fut.get();
9
10      for (size_t i = 0; i != iaSize; ++i)
11          cout << ia[i] << ' ';
12      cout << '\n';
13  }
```

Listing 12: quickersort.cc

```
1   #include "main.ih"
2
3   int* quickersort(int *beg, int *end)
4   {
5       if (end - beg <= 1)
6           return beg;
7
8       int lhs = *beg;
9       int *mid = partition(beg + 1, end,
10          [&](int arg)
11          {
12              return arg < lhs;
13          }
14      );
15
16      swap(*beg, *(mid - 1));
17
18      auto leftPart = async(launch::async, quickersort,
19          beg, mid);
20      auto rightPart = async(launch::async, quickersort,
21          mid, end);
22
23      rightPart.get();
24
25      return leftPart.get();
26  }
```

Exercise 49, async quicksort

Here is a quicksort algorithm using async threads called quickersort. Because async always returns something we decided to return the start of the array.

Code listings

Listing 13: main.ih

```
1  #include "main.h"
2
3  using namespace std;
```

Listing 14: main.h

```
1  #ifndef MAIN_H
2  #define MAIN_H
3
4  #include <future>
5  #include <algorithm>
6  #include <iostream>
7  #include "semaphore.h"
8
9  void quickerSort(Semaphore &nextRange);
10 void quickerSorter(Semaphore &nextRange);
11
12 #endif
```

Listing 15: main.cc

```
1  #include "main.ih"
2  #include <thread>
3
4  int main(int argc, char **argv)
5  {
6    Semaphore nextRange;
7
8    int ia[] = {2,4,6,2,3,7,9,1,12};
9      size_t iaSize = 9;
10
11   nextRange.push(Range(ia, ia + iaSize));
12
```

```
13    thread sorter1(quickerSorter,
14      std::ref(nextRange.subscribeThread()));
15    thread sorter2(quickerSorter,
16      std::ref(nextRange.subscribeThread()));
17    thread sorter3(quickerSorter,
18      std::ref(nextRange.subscribeThread()));
19
20    sorter1.join();
21    sorter2.join();
22    sorter3.join();
23
24    for (size_t i = 0; i != iaSize; ++i)
25          cout << ia[i] << ' ';
26      cout << '\n';
27  }
```

The sorter and the sorting algorithm

Listing 16: quickersorter.cc

```
1  #include "main.ih"
2
3  void quickerSorter(Semaphore &nextRange)
4  {
5    while (true)
6    {
7      nextRange.wait();
8
9      if (nextRange.size() == 0 &&
10       nextRange.waiters() == nextRange.subscribers())
11     {
12       nextRange.desubscribeThread();
13       nextRange.notify_all();
14       return;
15     }
16
17     quickerSort(nextRange);
18   }
19 }
```

Listing 17: quickersort.cc

```
#include "main.ih"

void quickerSort(Semaphore &nextRange)
{
  Range range = nextRange.next();
  int* beg = range.beg;
  int* end = range.end;

    if (end - beg <= 1)
        return;

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

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

  Range newRange1(beg, mid);
  Range newRange2(mid, end);

    nextRange.push(newRange1);
    nextRange.push(newRange2);
}
```

Semaphore 2.0, please like and subscribe to our queue

Listing 18: semaphore.ih

```
#include "semaphore.h"
#include <iostream>

using namespace std;
```

Listing 19: semaphore.h

```
#ifndef SEMAPHORE_H
#define SEMAPHORE_H
```

```
3
4  #include <condition_variable>
5  #include <mutex>
6  #include <queue>
7  #include <cstddef>
8
9  struct Range
10 {
11   int *beg = 0;
12   int *end = 0;
13   Range() = default;
14         Range(int *b, int *e)
15         :
16             beg(b),
17             end(e)
18         {}
19 };
20
21 class Semaphore
22 {
23   std::mutex d_mutex;
24   std::condition_variable d_condition;
25   std::size_t d_nAvailable = 0;
26
27   std::size_t d_subscribers = 0;
28   std::size_t d_waiters = 0;
29
30   std::queue<Range> d_queue;
31
32   public:
33     Semaphore() = default;
34     Semaphore(std::size_t nAvailable);
35
36     Semaphore &subscribeThread();
37     void desubscribeThread();
38
39     std::size_t size() const;
40     std::size_t subscribers() const;
41     std::size_t waiters() const;
42
```

```
43      void push(Range range);
44      Range next();
45
46      void notify();
47      void notify_all();
48      void wait();
49  };
50
51  #endif
```

Listing 20: desubscribethread.cc

```
1  #include "semaphore.ih"
2
3  void Semaphore::desubscribeThread()
4  {
5    --d_subscribers;
6    --d_waiters;
7  }
```

Listing 21: next.cc

```
1  #include "semaphore.ih"
2
3  Range Semaphore::next()
4  {
5    Range range = d_queue.front();
6    d_queue.pop();
7    return range;
8  }
```

Listing 22: notify_all.cc

```
1  #include "semaphore.ih"
2
3  void Semaphore::notify_all()
4  {
5    lock_guard<mutex> lock(d_mutex);
6    d_condition.notify_all();
7  }
```

Listing 23: push.cc

```
1  #include "semaphore.ih"
2
3  void Semaphore::push(Range range)
4  {
5    {
6      lock_guard<mutex> lock(d_mutex);
7      d_queue.push(range);
8      ++d_nAvailable;
9    }
10   notify_all();
11 }
```

Listing 24: subscribers.cc

```
1  #include "semaphore.ih"
2
3  size_t Semaphore::subscribers() const
4  {
5    return d_subscribers;
6  }
```

Listing 25: subscribethread.cc

```
1  #include "semaphore.ih"
2
3  Semaphore &Semaphore::subscribeThread()
4  {
5    ++d_subscribers;
6    return *this;
7  }
```

Listing 26: wait.cc

```
1  #include "semaphore.ih"
2
3  void Semaphore::wait()
4  {
5    ++d_waiters;
6    unique_lock<mutex> lock(d_mutex);
7    while (d_nAvailable == 0)
8    {
```

```
9       if (d_waiters == d_subscribers)
10        return;
11      d_condition.wait(lock);
12    }
13
14    --d_waiters;
15    --d_nAvailable;
16  }
```

Listing 27: waiters.cc

```
1  #include "semaphore.ih"
2
3  size_t Semaphore::waiters() const
4  {
5    return d_waiters;
6  }
```

Exercise 50, package task design

We are asked to pack a function that calculates inner products of a matrix multiplication and send it to 24 detached threads.

Code listings

Listing 28: main.cc

```
1  #include <thread>
2  #include <iostream>
3  #include <future>
4  #include <utility>
5  #include <iomanip>
6
7  double lhs[4][5] = {{1, 2, 3, 4, 5},
8                      {1, 2, 3, 4, 5},
9                      {1, 2, 3, 4, 5},
10                     {1, 2, 3, 4, 5}};
11
12 double rhsT[6][5] = {{1, 2, 3, 4, 5},
13                     {1, 2, 3, 4, 5},
14                     {1, 2, 3, 4, 5},
```

```
15              {1, 2, 3, 4, 5},
16              {1, 2, 3, 4, 5},
17              {1, 2, 3, 4, 5}};
18
19  double innerProduct(int row, int col)
20  {
21      double sum = 0;
22
23      for (int idx = 0; idx != 5; ++idx)
24        sum +=  lhs[row][idx] *
25          rhsT[col][idx];
26
27      return sum;
28  }
29
30  int main()
31  {
32
33      std::future<double> fut[4][6];
34
35      for (int row = 0; row != 4; ++row)
36      {
37        for (int col = 0; col != 6; ++col)
38        {
39            std::packaged_task<double(int,int)>
40              Task (innerProduct);
41            fut[row][col] = Task.get_future();
42            std::thread(std::move(Task), row,
43              col).detach();
44        }
45      }
46
47      for (int row = 0; row != 4; ++row)
48      {
49        for (int col = 0; col != 6; ++col)
50            std::cout << fut[row][col].get()
51              << std::setw(6);
52        std::cout << '\n';
53      }
54  }
```

Exercise 52, std::promise design

This exercise is the same as 50 but using std::promise instead of package task.

Code listings

Listing 29: main.cc

```
1  #include <thread>
2  #include <iostream>
3  #include <future>
4  #include <utility>
5  #include <iomanip>
6
7  double lhs[4][5] = {{1, 2, 3, 4, 5},
8              {1, 2, 3, 4, 5},
9              {1, 2, 3, 4, 5},
10             {1, 2, 3, 4, 5}};
11
12 double rhsT[6][5] = {{1, 2, 3, 4, 5},
13             {1, 2, 3, 4, 5},
14             {1, 2, 3, 4, 5},
15             {1, 2, 3, 4, 5},
16             {1, 2, 3, 4, 5},
17             {1, 2, 3, 4, 5}};
18
19 void innerProduct(std::promise<double> &ref, int row,
      int col)
20 {
21     double sum = 0;
22     for (int idx = 0; idx != 5; ++idx)
23       sum += lhs[row][idx] * rhsT[col][idx];
24
25     ref.set_value(sum);
26 }
27
28 int main()
29 {
30     std::promise<double> result[4][6];
31
```

```
32      for (int row = 0; row != 4; ++row)
33      {
34        for (int col = 0; col != 6; ++col)
35            std::thread(innerProduct,
36              ref(result[row][col]),
37              row, col).detach();
38      }
39
40      for (int row = 0; row != 4; ++row)
41      {
42        for (int col = 0; col != 6; ++col)
43            std::cout
44              << result[row][col].get_future().get()
45              << std::setw(6);
46        std::cout << '\n';
47      }
48  }
```