Programming in C/C++
Exercises set six: multi-threading 1

Christiaan Steenkist
Jaime Betancor Valado
Remco Bos

December 15, 2016

Exercise 38, basic multithreading with recursive program

We were tasked to design a program that recursively visits all elements of a starting directory and computes the total size of its regular files.

Code listings

Listing 1: main.ih

```
1  #include "main.h"
2
3  #include <thread>
4  #include <chrono>
5
6  #include <sys/types.h>
7  #include <sys/stat.h>
8  #include <dirent.h>
9
10 using namespace std;
```

Listing 2: main.cc

```
1  #include "main.ih"
2
3  int main(int argc, char **argv)
4  {
5    if (argc <= 1)
```

```
6     {
7       cerr << "Please supply a folder path.\n";
8       return 0;
9     }
10
11    mutex statusMutex;
12    bool completed = false;
13    size_t bytes = 0;
14    thread byteCounter(countBytes,
15      std::ref(statusMutex), std::ref(completed),
16      std::ref(bytes), argv[1]);
17    thread progressBar(timeProcess,
18      std::ref(statusMutex), std::ref(completed));
19
20    byteCounter.join();
21    progressBar.join();
22
23    cout << bytes << " bytes\n";
24  }
```

Listing 3: openfolder.cc

```
1   #include "main.ih"
2
3   size_t openFolder(string path)
4   {
5     DIR *dir = opendir(path.c_str());
6     if (dir == 0)
7       return 0;
8
9     size_t bytes = 0;
10    struct dirent *pent = 0;
11    while((pent = readdir(dir)))
12    {
13      struct stat sb;
14      lstat(pent->d_name, &sb);
15
16      bytes += sb.st_size;
17
18      if (S_ISDIR(sb.st_mode) == 0)
19      {
```

```
20        string newPath = path + '/' + pent->d_name;
21        bytes += openFolder(newPath);
22      }
23    }
24    closedir(dir);
25
26    return bytes;
27  }
```

Listing 4: timeprocess.cc

```
1  #include "main.ih"
2
3  void timeProcess(mutex &statusMutex, bool &status)
4  {
5    bool localStatus = false;
6    while (!localStatus)
7    {
8      cout << '.' << std::flush;
9
10     statusMutex.lock();
11     localStatus = status;
12     statusMutex.unlock();
13
14     this_thread::sleep_for(chrono::seconds(1));
15   }
16
17   cout << std::endl;
18 }
```

Exercise 39, using chrono/clock facilities

We were tasked to display the time at the beginning and end of a program.

Output

Listing 5: output

```
1  .
2  203509 bytes
3   Program starts at Thu Dec 15 14:44:36 2016
```

```
4
5  Program ends at Thu Dec 15 14:44:36 2016
6
7 Total time passed 0.0112944
```

Code listings

Listing 6: main.cc

```
1  #include "main.ih"
2  #include <iostream>
3  #include <chrono>
4  #include <ctime>
5
6  int main(int argc, char **argv)
7  {
8    // Time block
9    chrono::time_point<chrono::system_clock> start,
10     end;
11
12   start = chrono::system_clock::now();
13
14   if (argc <= 1)
15   {
16     cerr << "Please supply a folder path.\n";
17     return 0;
18   }
19
20   mutex statusMutex;
21   bool completed = false;
22   size_t bytes = 0;
23   thread byteCounter(countBytes,
24     std::ref(statusMutex), std::ref(completed),
25     std::ref(bytes), argv[1]);
26   thread progressBar(timeProcess,
27     std::ref(statusMutex), std::ref(completed));
28
29   byteCounter.join();
30
31   end = chrono::system_clock::now();
32   progressBar.join();
```

```
33    cout << bytes << " bytes\n";
34
35    // Print block
36
37    time_t startTime =
38      chrono::system_clock::to_time_t(start);
39    cout << " Program starts at "
40      << ctime(&startTime) << '\n';
41
42    time_t endTime =
43      chrono::system_clock::to_time_t(end);
44    cout << " Program ends at "
45      << ctime(&endTime) << '\n';
46
47    chrono::duration<double> totalTime =
48      end - start;
49    cout << "Total time passed "
50      << totalTime.count() << '\n';
51  }
```

Exercise 40, thread-safe queue

A proxy and a lot of lock guards attempt to make this queue thread-safe.

Code listings

Listing 7: safequeue.ih

```
1  #include "main.h"
2
3  #include <thread>
4  #include <iostream>
5
6  using namespace std;
```

Listing 8: safequeue.h

```
1  #ifndef SAFEQUEUE_H
2  #define SAFEQUEUE_H
3
4  #include <mutex>
```

```
5  #include <queue>
6  #include <string>
7
8  class RefProxy
9  {
10   std::mutex *d_mutex;
11   std::string &d_string;
12
13   public:
14     RefProxy(std::mutex *mut, std::string &ref);
15     std::string &operator=(RefProxy const &rhs);
16     std::string &operator=(std::string const &rhs);
17     operator std::string const &() const;
18 };
19
20 class SafeQueue
21 {
22   std::mutex *d_mutex;
23   std::queue<std::string> d_queue;
24
25   public:
26     SafeQueue(std::mutex *mut);
27
28     bool empty();
29
30     RefProxy front();
31     RefProxy back();
32
33     void pop();
34     void push(std::string item);
35 };
36
37 #endif
```

Listing 9: back.cc

```
1  #include "safequeue.ih"
2
3  RefProxy SafeQueue::back()
4  {
5    lock_guard<mutex> lock(*d_mutex);
```

```
6     return RefProxy(d_mutex, d_queue.back());
7   }
```

Listing 10: empty.cc

```
1   #include "safequeue.ih"
2
3   bool SafeQueue::empty()
4   {
5     lock_guard<mutex> lock(*d_mutex);
6     return d_queue.empty();
7   }
```

Listing 11: front.cc

```
1   #include "safequeue.ih"
2
3   RefProxy SafeQueue::front()
4   {
5     lock_guard<mutex> lock(*d_mutex);
6     return RefProxy(d_mutex, d_queue.front());
7   }
```

Listing 12: opeartor1.cc

```
1   #include "safequeue.ih"
2
3   string &RefProxy::operator=(std::string const &rhs)
4   {
5     lock_guard<mutex> lock(*d_mutex);
6     return d_string = rhs;
7   }
```

Listing 13: operator2.cc

```
1   #include "safequeue.ih"
2
3   string &RefProxy::operator=(RefProxy const &rhs)
4   {
5     lock_guard<mutex> lock(*d_mutex);
6     return d_string = rhs.d_string;
7   }
```

Listing 14: pop.cc

```
1  #include "safequeue.ih"
2
3  void SafeQueue::pop()
4  {
5    lock_guard<mutex> lock(*d_mutex);
6    d_queue.pop();
7  }
```

Listing 15: promotor.cc

```
1  #include "safequeue.ih"
2
3  RefProxy::operator string const &() const
4  {
5    lock_guard<mutex> lock(*d_mutex);
6    return d_string;
7  }
```

Listing 16: proxyconstr.cc

```
1  #include "safequeue.ih"
2
3  RefProxy::RefProxy(mutex *mut, string &ref)
4  :
5    d_mutex(mut),
6    d_string(ref)
7  {
8  }
```

Listing 17: push.cc

```
1  #include "safequeue.ih"
2
3  void SafeQueue::push(string item)
4  {
5    lock_guard<mutex> lock(*d_mutex);
6    d_queue.push(item);
7  }
```

Listing 18: queueconstr.cc

```
1  #include "safequeue.ih"
```

```
2
3  SafeQueue::SafeQueue(mutex *mut)
4  :
5    d_mutex(mut)
6  {
7  }
```

Exercise 42, establish connection between a parent and child process

We were tasked to pass output from a child process to the parent process by using fork and exec.

Code listings

Listing 19: main.cc

```
1  #include "main.h"
2
3  int main()
4  {
5    int fileDescr[2];
6    if (pipe(fileDescr) == -1)
7      exit(1);
8
9    int pID = fork();
10   if (pID == 0) //Child
11   {
12     while ((dup2(fileDescr[1],
13       STDOUT_FILENO) == -1)) {}
14     close(fileDescr[1]);
15     close(fileDescr[0]);
16     execl("/bin/ls", "ls", (char *) 0);
17     _exit(1);
18   }
19
20   char buffer[4096];
21   while (1)
22   {
23     int count = read(fileDescr[0], buffer,
```

```
24        sizeof(buffer));
25      if (count == -1)
26      {
27        exit(1);
28      }
29      else if (count == 0)
30      {
31        break;
32      }
33      else
34      {
35        usechildoutput(count, buffer);
36        exit(1);
37      }
38    }
39    close(fileDescr[0]);
40    wait(0);
41  }
```

Listing 20: main.h

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4  #include <errno.h>
5  #include <sys/wait.h>
6  #include <iostream>
7
8  using namespace std;
9
10 void usechildoutput(int count, char buffer[]);
```

Listing 21: usechildoutput.cc

```
1  #include "main.h"
2
3  void usechildoutput(int count, char buffer[])
4  {
5    int lines = 0;
6    for (int bufferElement = 0;
7      bufferElement <= count; ++bufferElement)
```

```
8    {
9      cout << buffer[bufferElement];
10     if (buffer[bufferElement] == '\n')
11       lines += 1;
12   }
13   cout << "Number of characters: " << count << "\n";
14   cout << "Number of lines: " << lines << "\n";
15 }
```

Exercise 43: design a simple multi-thread program

We were tasked to make a program with threads counting vowels, hexadecimals, digits and punctuation character in a file passed to the program.

Code listings

Listing 22: task.h

```
1  #ifndef TASK_H
2  #define TASK_H
3
4  #include <iostream>
5  #include <vector>
6
7  class Task: public std::vector<char>
8  {
9      public:
10         Task(std::istream &file);
11         void countVowel();
12         void countDigit();
13         void countHexDec();
14         void countPunctChar();
15 };
16
17 #endif
```

Listing 23: task.ih

```
1  #include "task.h"
2  #include <algorithm>
3  #include <iterator>
```

```
4  #include <cctype>
5
6  using namespace std;
```

Listing 24: main.cc

```
1  #include "task.ih"
2  #include <thread>
3  #include <chrono>
4
5  int main(int argc, char **argv)
6  {
7    using namespace std::chrono;
8    time_point<system_clock> start,end;
9    start = system_clock::now();
10
11   Task task(cin);
12
13   if (argc >=2) //perform threads in sequence
14   {
15     thread vowelThread(&Task::countVowel,
16       &task);
17     vowelThread.join();
18     thread digitThread(&Task::countDigit,
19       &task);
20     digitThread.join();
21     thread hexdecThread(&Task::countHexDec,
22       &task);
23     hexdecThread.join();
24     thread punctThread(&Task::countPunctChar,
25       &task);
26     punctThread.join();
27   }
28   else //perform threads in parallel
29   {
30     thread vowelThread(&Task::countVowel,
31       &task);
32     thread digitThread(&Task::countDigit,
33       &task);
34     thread hexdecThread(&Task::countHexDec,
35       &task);
```

```
36      thread punctThread(&Task::countPunctChar,
37        &task);
38      vowelThread.join();
39      digitThread.join();
40      hexdecThread.join();
41      punctThread.join();
42    }
43
44    end = system_clock::now();
45    duration<double> program_runtime = end-start;
46    cout << "Program runtime: "
47      << program_runtime.count() << "s \n";
48  }
```

Listing 25: task.cc

```
1  #include "task.ih"
2
3  Task::Task(std::istream &file)
4  {
5    copy(istream_iterator<char>(file),
6      istream_iterator<char>(),
7      back_inserter(*this));
8  }
```

Listing 26: countdigit.cc

```
1  #include "task.ih"
2
3  void Task::countDigit()
4  {
5    size_t ret = 0;
6    for_each(begin(), end(),
7      [&] (char c)
8      {
9        if (string("1234567890").find(c)
10         != string::npos)
11
12         ++ret;
13     }
14   );
```

```
15    cout << "Digits: " << ret << "\n";
16  }
```

Listing 27: counthexdec.cc

```
1   #include "task.ih"
2
3   void Task::countHexDec()
4   {
5     size_t ret = 0;
6     for_each(begin(), end(),
7       [&] (char c)
8       {
9         if (isxdigit(c))
10          ++ret;
11      }
12    );
13    cout << "Hexadecimals: " << ret << "\n";
14  }
```

Listing 28: countpunctchar.cc

```
1   #include "task.ih"
2
3   void Task::countPunctChar()
4   {
5     size_t ret = 0;
6     for_each(begin(), end(),
7       [&] (char c)
8       {
9         if (ispunct(c))
10          ++ret;
11      }
12    );
13    cout << "Punctuation characters: "
14      << ret << "\n";
15  }
```

Listing 29: countvowelcc

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

```
3  void Task::countVowel()
4  {
5    size_t ret = 0;
6    for_each(begin(), end(),
7      [&] (char c)
8      {
9        if (string("aeiouAEIOU").find(c)
10         != string::npos)
11
12         ++ret;
13     }
14   );
15   cout << "Vowels: " << ret << "\n";
16 }
```