Boost.Asio

(Asynchronous Input and Output)

Concurrent Programming

Scalable Computing Systems Laboratory
Hanyang University

Introduction

- What is Boost library?
- What is Boost.Asio?
- Installing Boost library
- Practice

What is Boost library?

- Set of libraries for the C++ programming language that provide support for tasks and structures such as
  - String and text processing
  - Containers
  - Iterators
  - Algorithms
  - Function objects and higher-order programming
  - Generic Programming
  - Template Metaprogramming
  - Preprocessor Metaprogramming
  - Concurrent Programming
  - Math and numerics
  - Correctness and testing
  - Data structures
  - Domain Specific
  - Input/Output
  - Inter-language support
  - Language Features Emulsation
  - Memory
  - Parsing
  - Patterns and Idioms
  - Programming Interfaces
  - State Machines
  - System
  - Miscellaneous
  - Broken compiler workarounds

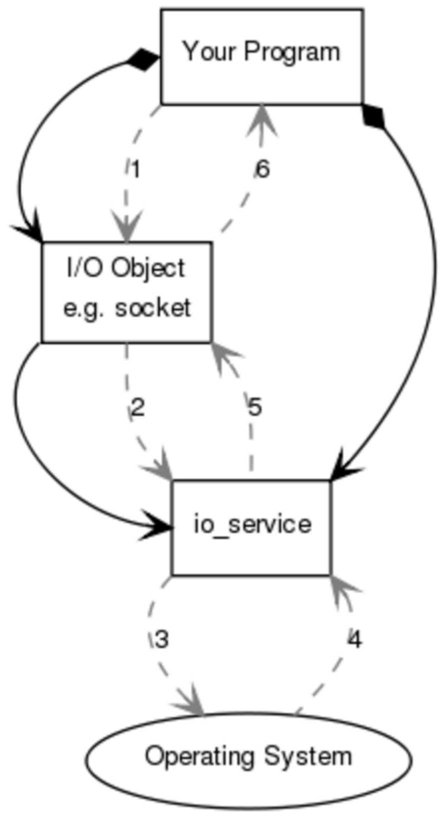
What is Boost library?

- Concurrent Programming
  - Asio
  - Atomic
  - Compute
  - Context
  - Coroutine
  - Coroutine2
  - Fiber
  - Interprocess
  - Lockfree
  - MPI
  - Thread

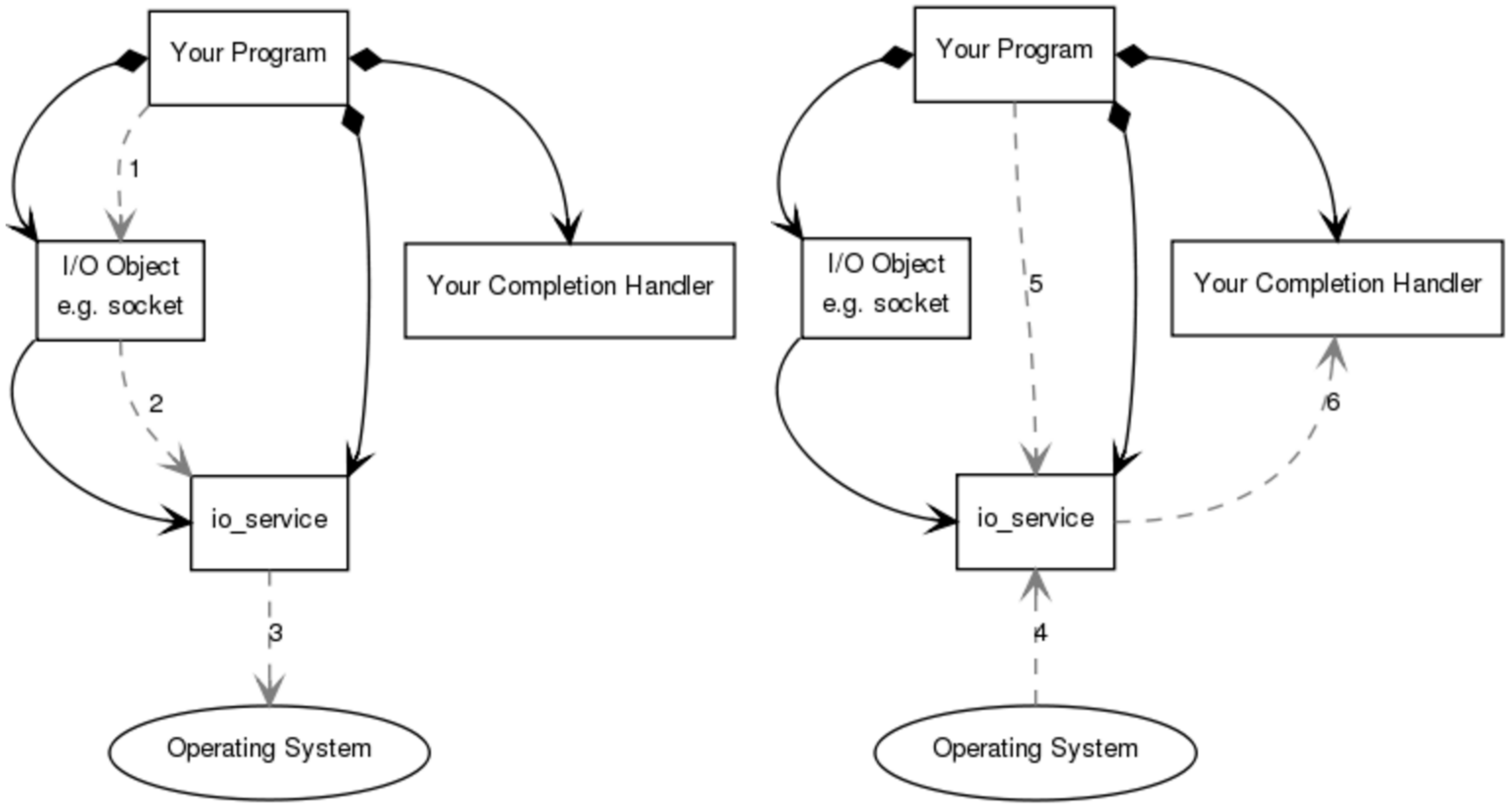
What is Boost.Asio?

- Cross-platform C++ library for network and low-level I/O programming that provides developers with a consistent asynchronous model using a modern C++ approach
  - Portability
  - Scalability
  - Efficiency
  - Model concepts from established APIs, such as BSD sockets
  - Ease of use
  - Basis for further abstraction

Boost.Asio Anatomy



Synchronous operation



Asynchronous operation

Installing Boost library

1. Download latest version (boost_1_65_1.tar.bz2)
   - https://sourceforge.net/projects/boost/files/boost/1.65.1/

2. Extract it

```
$ tar --bzip2 -xf ./boost_1_65_1.tar.bz2
```

3. Move to the extracted directory

```
$ cd boost_1_65_1
```

Installing Boost library

4. Run script files to install

```
$ ./bootstrap.sh
$ sudo ./b2 install
```

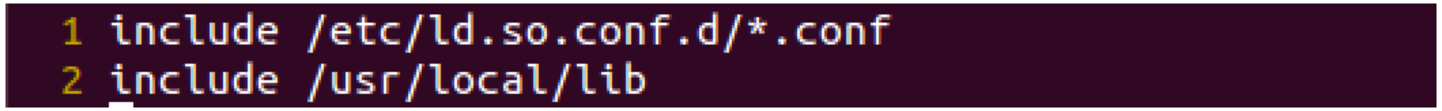
Default header files path : /usr/local/include/boost/

Default library files path : /usr/local/lib

Installing Boost library

5. Add boost library path to linux system library path

```
$ sudo vi /etc/ld.so.conf
```



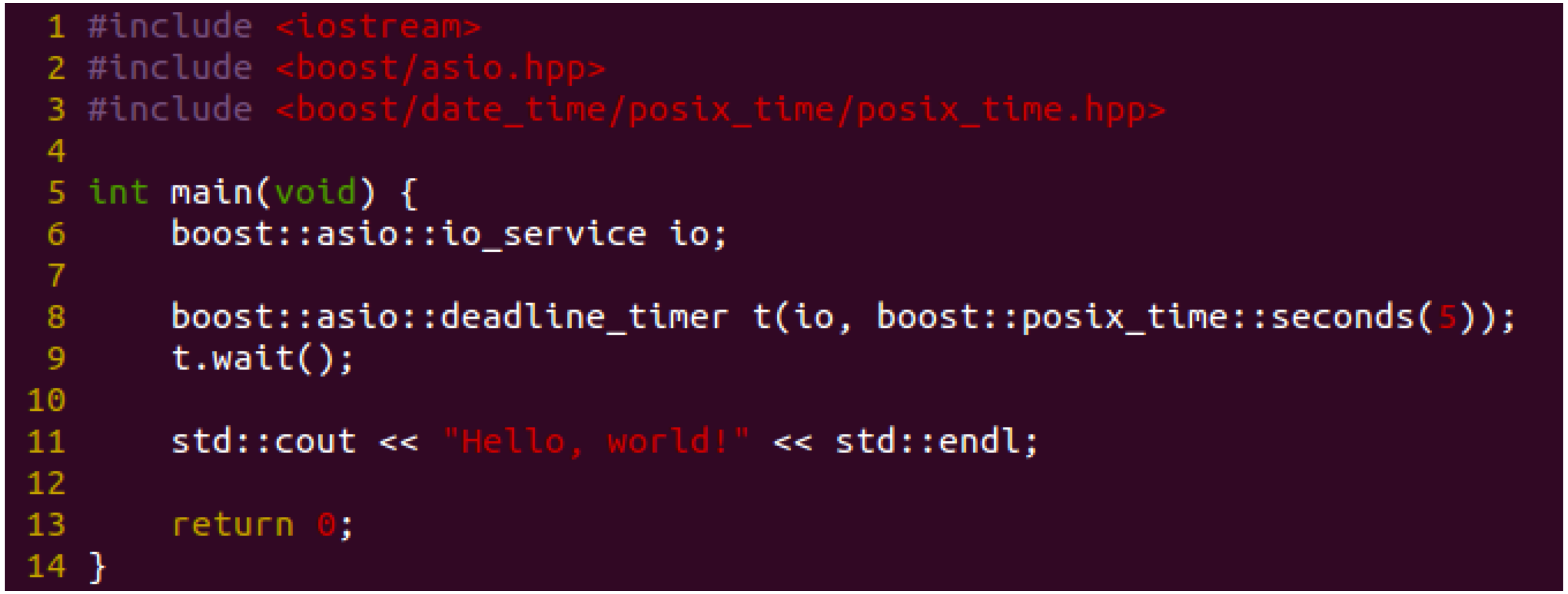
```
include /etc/ld.so.conf.d/*.conf
include /usr/local/lib
```

```
$ sudo ldconfig
```

Practice

(Using a timer synchronously)

[timer_sync.cpp]



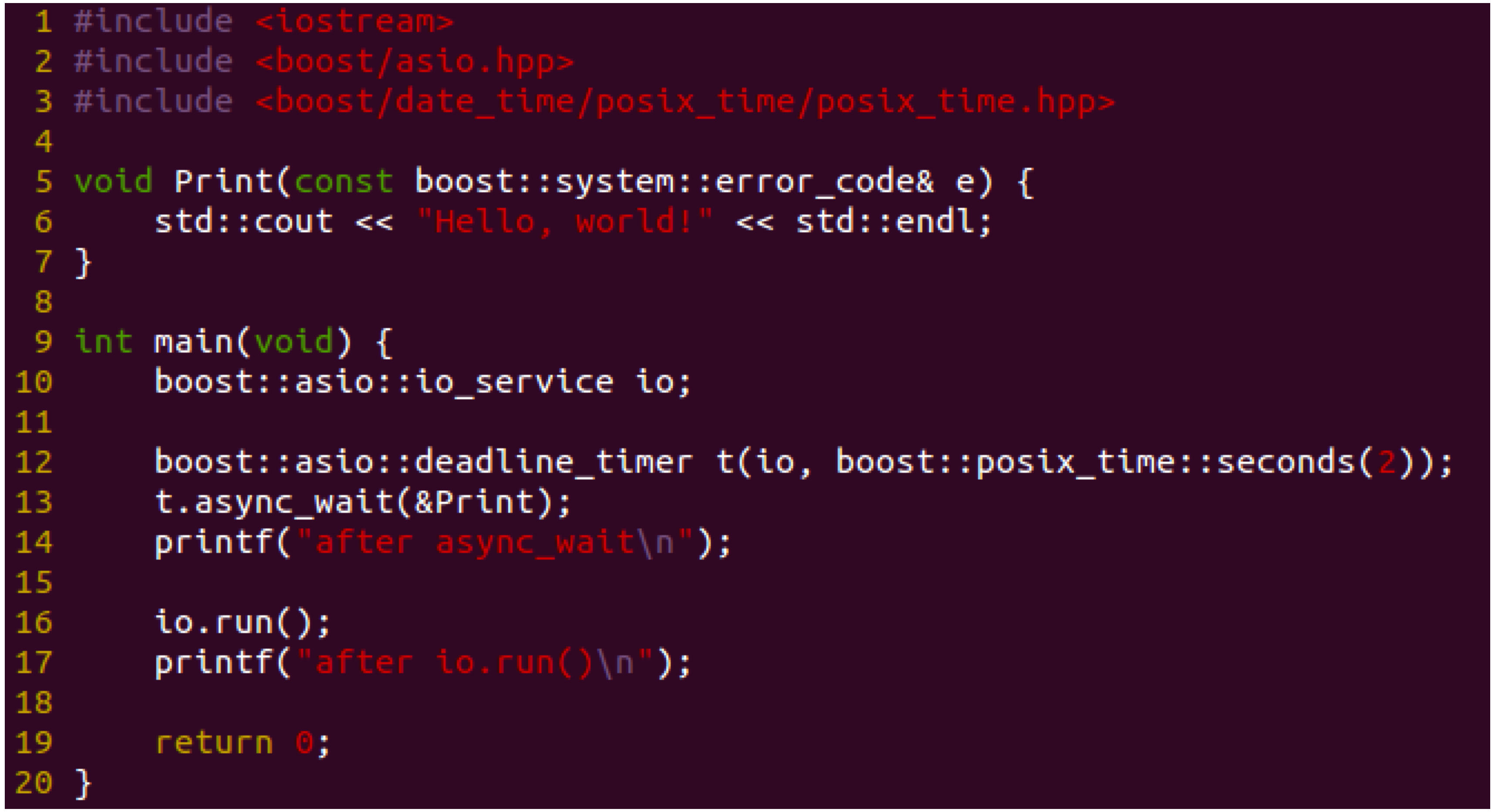
```cpp
#include <iostream>
#include <boost/asio.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>

int main(void) {
    boost::asio::io_service io;

    boost::asio::deadline_timer t(io, boost::posix_time::seconds(5));
    t.wait();

    std::cout << "Hello, world!" << std::endl;

    return 0;
}
```

```
$ g++ timer_sync.cpp -lboost_system
```

Practice

(Using a timer asynchronously)

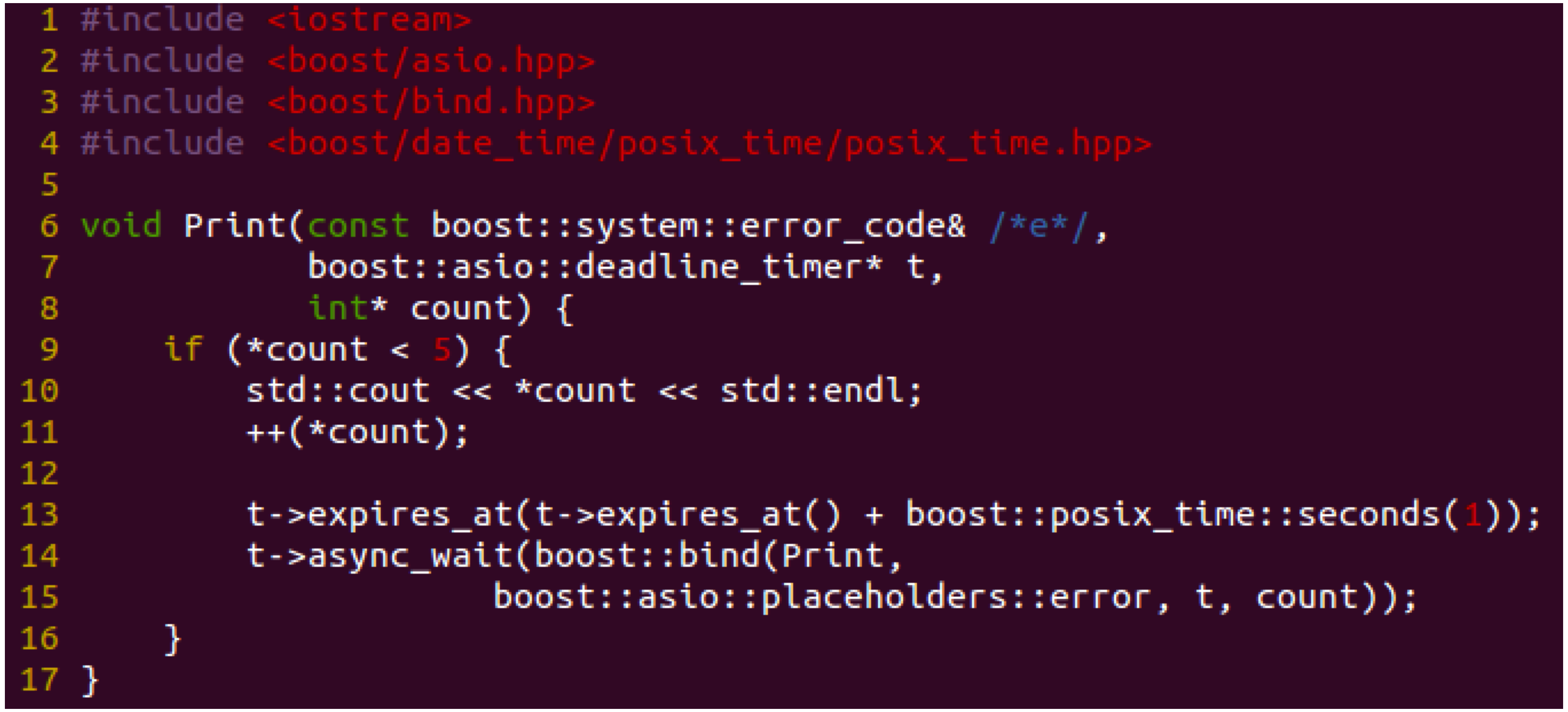
[timer_async.cpp]



```
#include <iostream>
#include <boost/asio.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>

void Print(const boost::system::error_code& e) {
    std::cout << "Hello, world!" << std::endl;
}

int main(void) {
    boost::asio::io_service io;

    boost::asio::deadline_timer t(io, boost::posix_time::seconds(2));
    t.async_wait(&Print);
    printf("after async_wait\n");

    io.run();
    printf("after io.run()\n");

    return 0;
}
```

Practice
(Binding arguments to a handler)

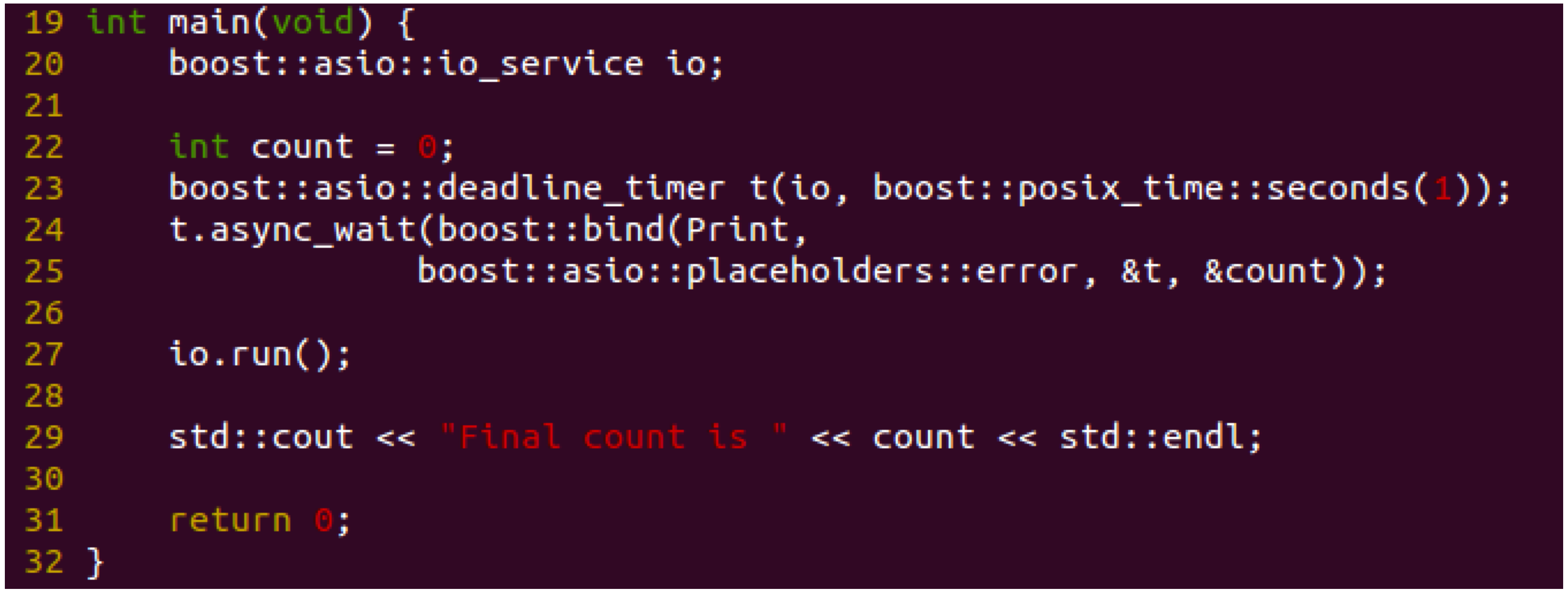
[timer_async_arg.cpp]



```cpp
#include <iostream>
#include <boost/asio.hpp>
#include <boost/bind.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>

void Print(const boost::system::error_code& /*e*/,
           boost::asio::deadline_timer* t,
           int* count) {
    if (*count < 5) {
        std::cout << *count << std::endl;
        ++(*count);

        t->expires_at(t->expires_at() + boost::posix_time::seconds(1));
        t->async_wait(boost::bind(Print,
                    boost::asio::placeholders::error, t, count));
    }
}
```

Practice
(Binding arguments to a handler)

[timer_async_arg.cpp] continue…

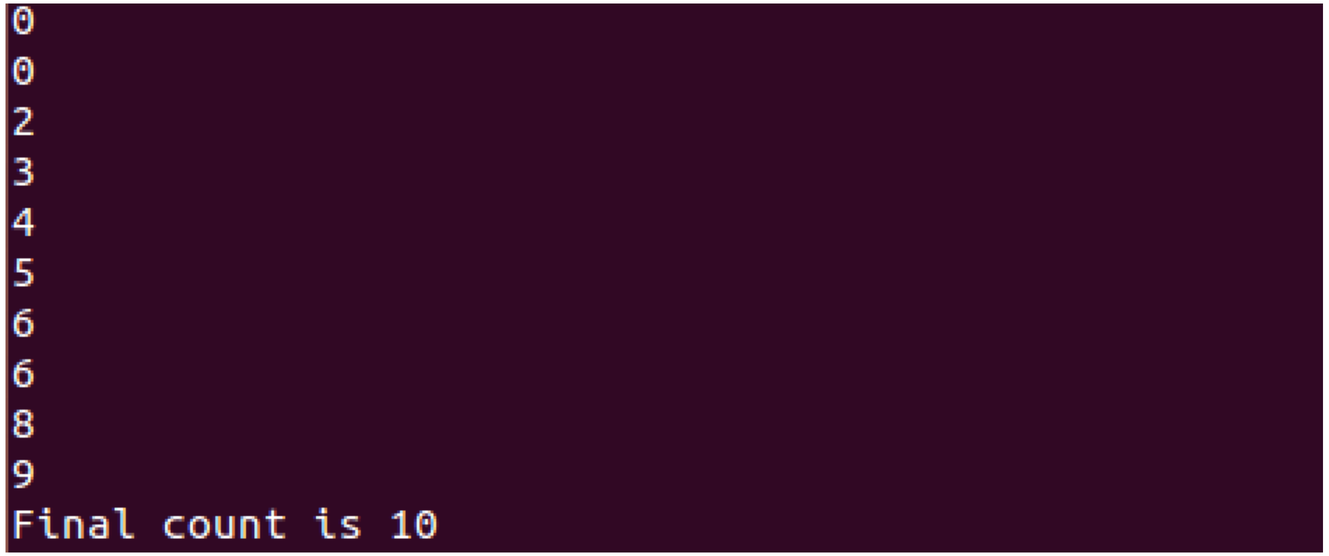


```cpp
int main(void) {
    boost::asio::io_service io;

    int count = 0;
    boost::asio::deadline_timer t(io, boost::posix_time::seconds(1));
    t.async_wait(boost::bind(Print,
                boost::asio::placeholders::error, &t, &count));

    io.run();

    std::cout << "Final count is " << count << std::endl;

    return 0;
}
```

Practice
(io_service event processing with multi-thread)

- Download timer_2thread.cpp from Piazza resources page

```
$ g++ timer_2thread.cpp -lboost_system -lboost_thread
```

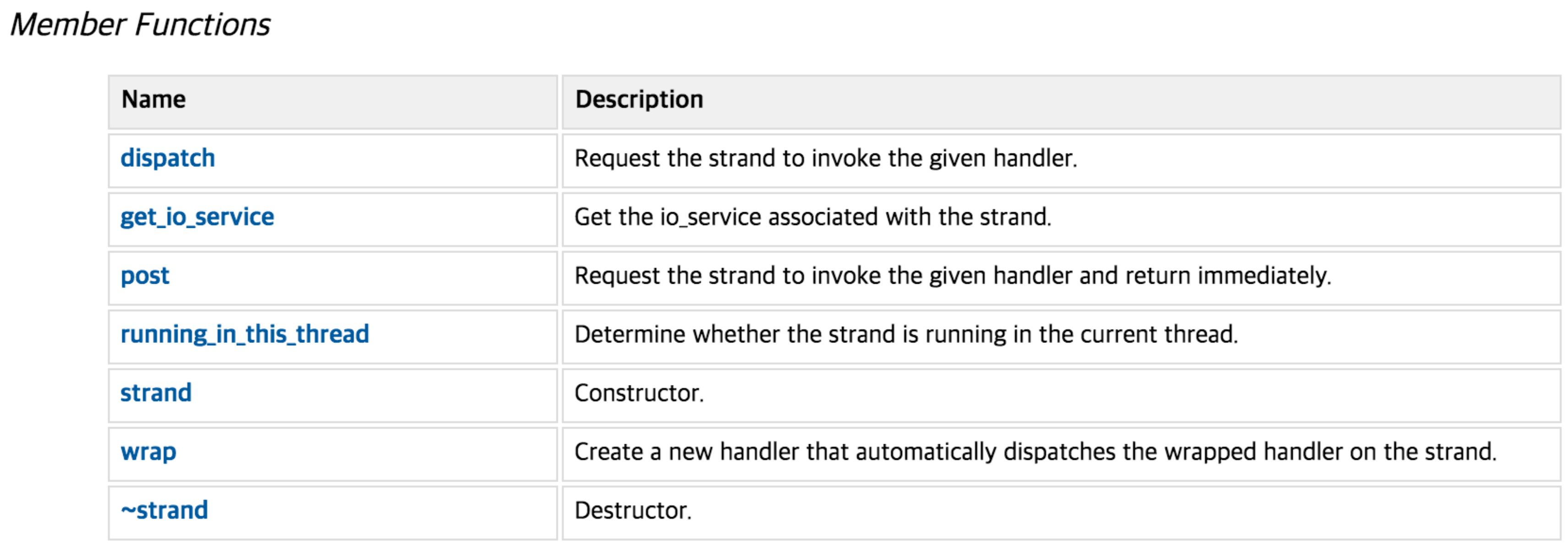


```
0
0
2
3
4
5
6
6
8
9
Final count is 10
```

Boost.Asio - Strand

- To synchronize callback handler in multi-threaded program

- It guarantees that, for those handlers that are dispatched through it, an executing handler will be allowed to complete before the next one is started



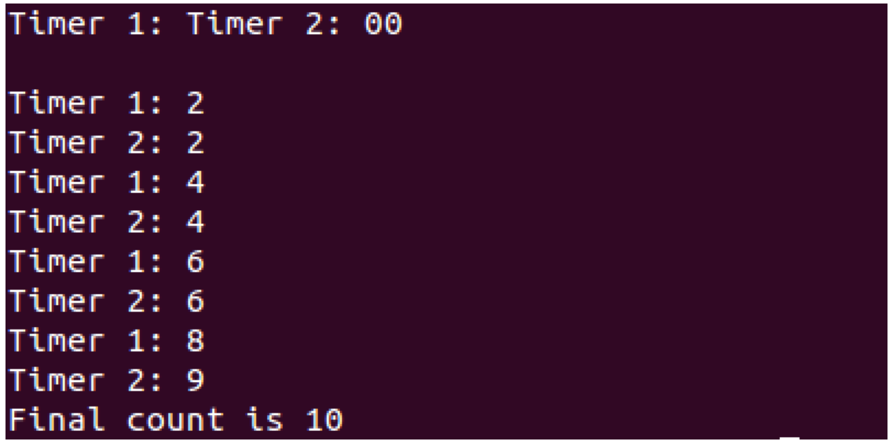
Member Functions

Name	Description
dispatch	Request the strand to invoke the given handler.
get_io_service	Get the io_service associated with the strand.
post	Request the strand to invoke the given handler and return immediately.
running_in_this_thread	Determine whether the strand is running in the current thread.
strand	Constructor.
wrap	Create a new handler that automatically dispatches the wrapped handler on the strand.
~strand	Destructor.

Practice
(Task: fix race condition)

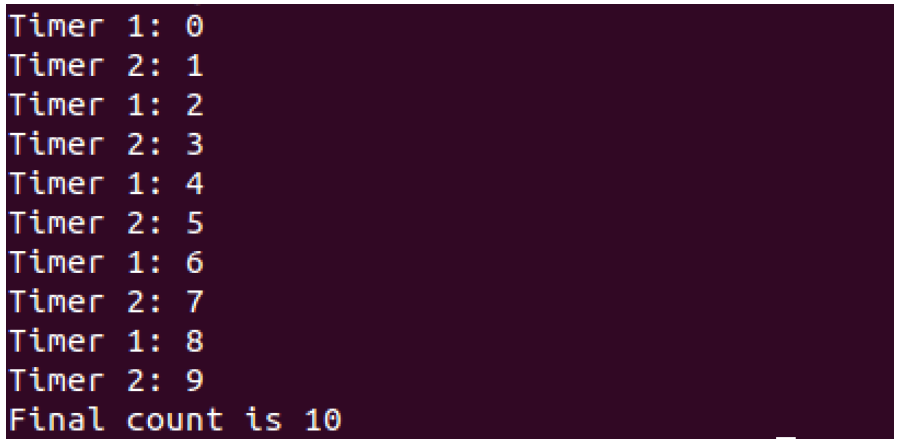
- Download timer_fix_race.cpp from Piazza resources page

```
$ g++ timer_fix_race.cpp -lboost_system -lboost_thread
```



```
Timer 1: Timer 2: 00

Timer 1: 2
Timer 2: 2
Timer 1: 4
Timer 2: 4
Timer 1: 6
Timer 2: 6
Timer 1: 8
Timer 2: 9
Final count is 10
```

Before



```
Timer 1: 0
Timer 2: 1
Timer 1: 2
Timer 2: 3
Timer 1: 4
Timer 2: 5
Timer 1: 6
Timer 2: 7
Timer 1: 8
Timer 2: 9
Final count is 10
```

After

Thank You