Project 3 Supplementary

NP TA 誌佑

Outline

- Lambda Expressions
- Auto Specifier
- Shared Pointer
- Move
- Boost.Asio Example

Lambda Expressions (since C++11)

- An unnamed function object capable of capturing variables in scope
- A lambda expression consists of three parts

[ ] ( ) { }

- captures
- params
- body

```
/* without capture */
function<int(int)> square = [](int x) { return x * x; };
cout << square(5) << endl;  /* output: 25 */
```

Lambda Expressions (since C++11)

- An unnamed function object capable of capturing variables in scope

```
/* capture by reference */
int x = 0;
function<int(int)> add = [&x](int y) { x = 1; return x + y; };
cout << add(3) << endl;   /* output: 4 */
cout << x << endl;        /* output: 1 */

/* capture by value */
int x = 0;
function<int(int)> add = [x](int y) mutable { x = 1; return x + y;
};
cout << add(3) << endl;   /* output: 4 */
cout << x << endl;        /* output: 0 */
```

Lambda Expressions (since C++11)

- Without lambda expression

```
bool by_name(Person a, Person b) {
   return a.name < b.name;
}
bool by_age(Person a, Person b) {
   return a.age < b.age;
}
vector<Person> employees;

/* sort employees ordered by name */
sort(employees.begin(), employees.end(), by_name);

/* sort employees ordered by age */
sort(employees.begin(), employees.end(), by_age);
```

Lambda Expressions (since C++11)

- With lambda expression

```
vector<Person> employees;

/* sort employees ordered by name */
sort(employees.begin(), employees.end(), [](Person a, Person b) {
   return a.name < b.name;
});

/* sort employees ordered by age */
sort(employees.begin(), employees.end(), [](Person a, Person b) {
   return a.age < b.age;
});
```

Auto Specifier (since C++11)

- Let compiler automatically deduce types

```
// int
auto a = 1 + 2;

// int
auto b = a;

/* function<int(int)> */
auto square = [](int x) { return x * x; };

vector<int> arr;
/* vector<int>::iterator */
auto begin_it = arr.begin();
```

Shared Pointer (since C++11)

- **std::shared_ptr** is a smart pointer that retains shared ownership of an object through a pointer
- You do not have to **free** and **delete** manually

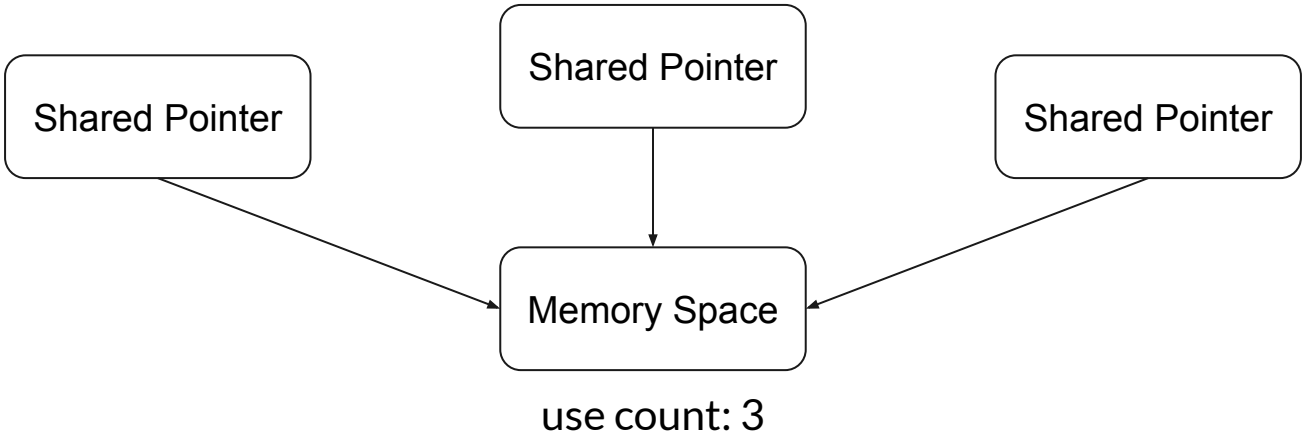
```
std::shared_ptr<int> sp(new int);
*sp = 5;
cout << *sp;     /* output: 5 */

auto sp2 = std::make_shared<int>(10);
cout << *sp2;    /* output: 10 */
```

C++ smart pointers

1. std::unique_ptr
2. std::shared_ptr
3. std::weak_ptr

Shared Pointer (since C++11)

- When will the allocated resource be destroyed?
  - When the last remaining shared_ptr owning the object is destroyed (when use count is 0)



Shared Pointer (since C++11)

- When will the allocated resource be destroyed?
  - When the last remaining shared_ptr owning the object is destroyed (when use count is 0)

```
{
    std::shared_ptr< int> sp(new int);
    {
        std::shared_ptr< int> sp2(sp);
        cout << sp.use_count();   /* output: 2 */
    }
    cout << sp.use_count();       /* output: 1 */
} /* free the space */
```

enable_shared_from_this

- Allows an object that is currently managed by a shared_ptr safely generate additional shared_ptr instances

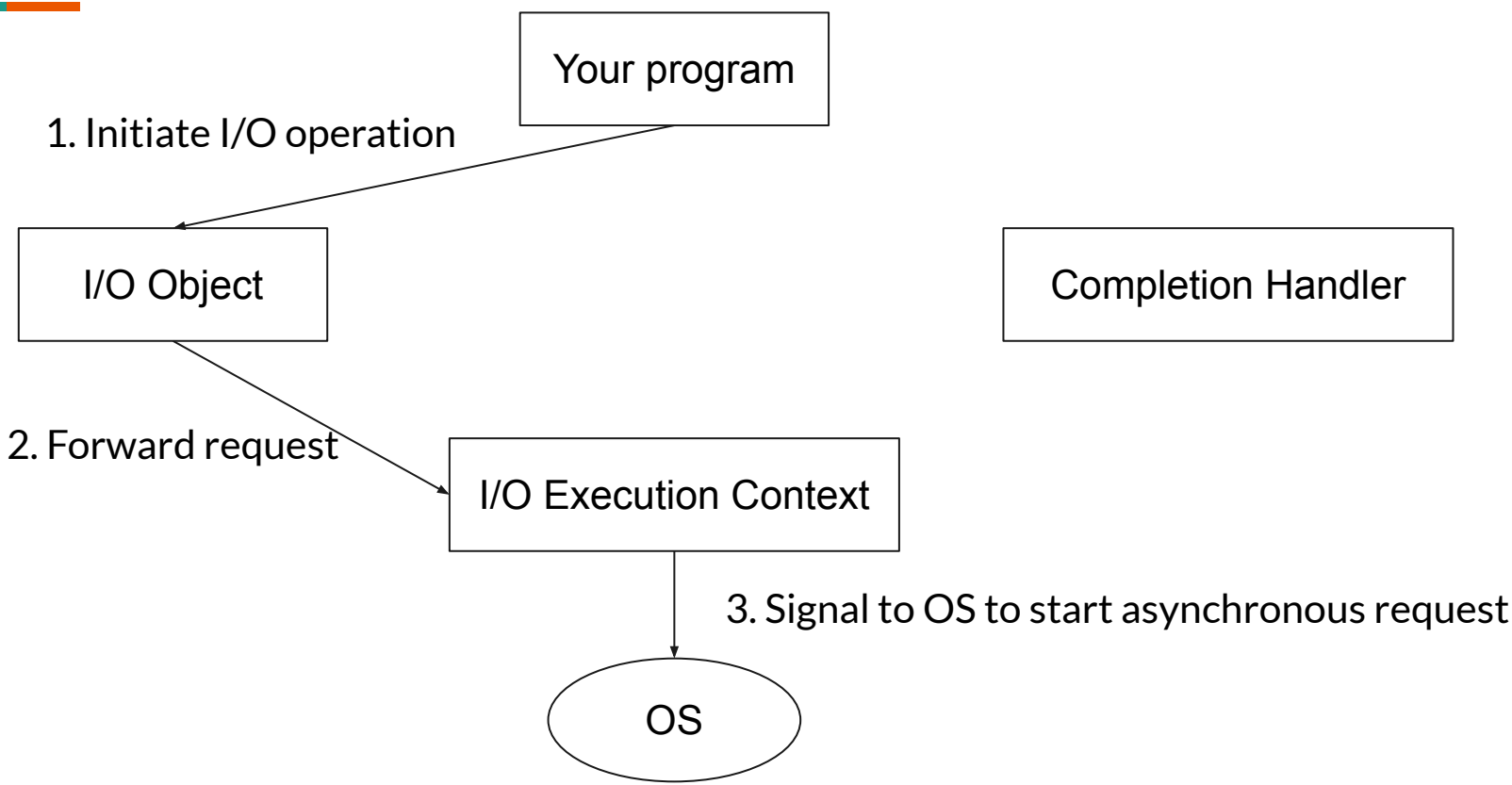
```
class MyClass : std::enable_shared_from_this<MyClass>
{
    std::shared_ptr<MyClass> get_ptr() {
        return shared_from_this(); // Good
        return this;               // Bad
    }
};
```

Move (since C++11)

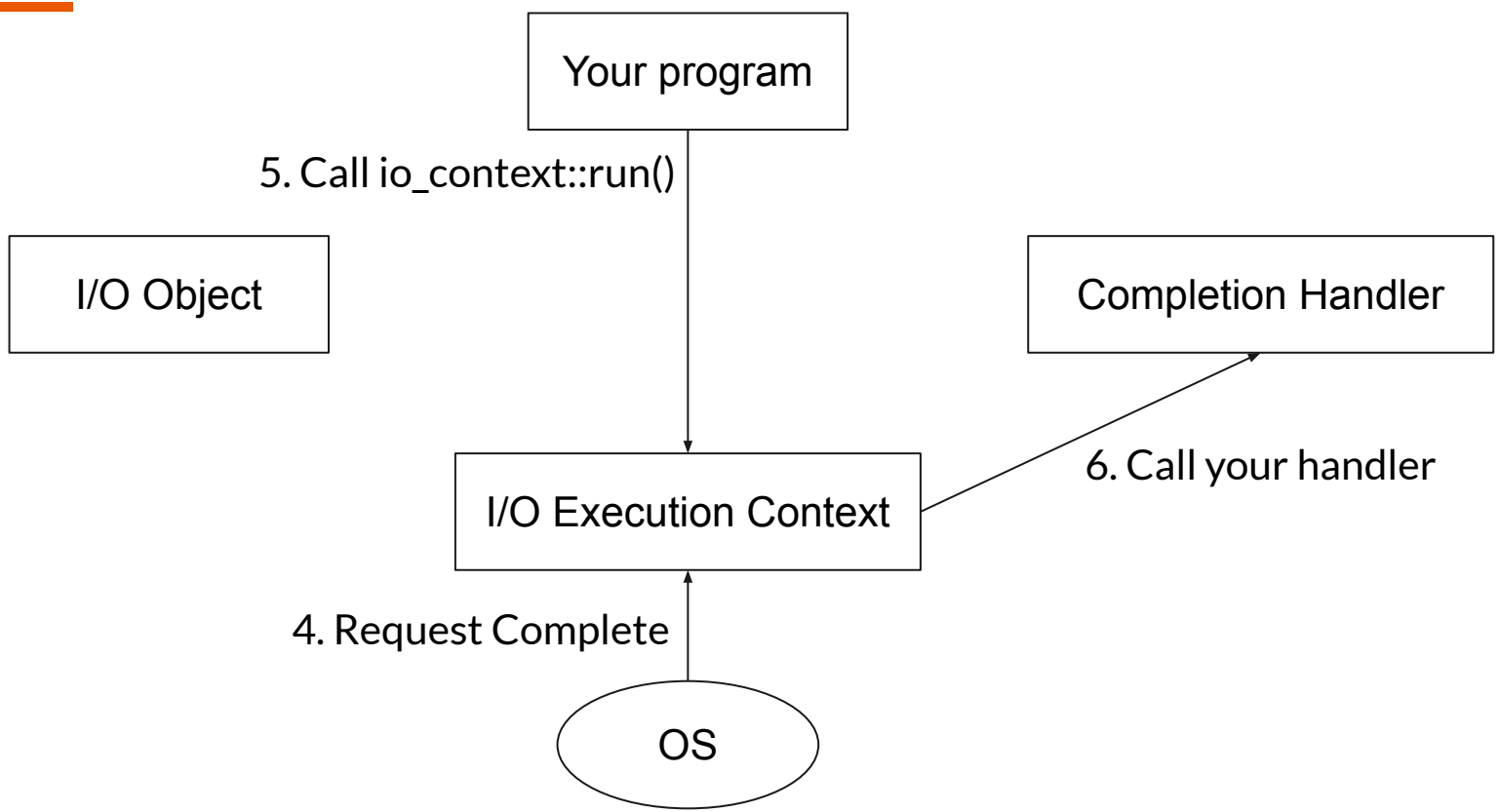
- **std::move** is used to indicate that an object may be "moved from", i.e. allowing the efficient transfer of resources from one object to another.

```
string a = "Hello";

/* extra cost of copying string a */
string b = a;

/* the content of string a will be moved into string c */
string c = move(a);

cout << '"' << a << '"' << endl; // output: ""
cout << '"' << b << '"' << endl; // output: "Hello"
cout << '"' << c << '"' << endl; // output: "Hello"
```

Boost.Asio Overview



Boost.Asio Overview



Boost.Asio Example

- An asynchronous echo server example from Boost.Asio documentation
- The following codes are simplified because of space limitation

Boost.Asio Example

- Call io_context::run()

```
int main(int argc, char* argv[])
{
    boost::asio::io_context io_context;
    server s(io_context, std::atoi(argv[ 1]));

    /* VERY IMPORTANT! */
    io_context.run();

    return 0;
}
```

Boost.Asio Example

```
class server {
private:
 tcp::acceptor acceptor_;

public:
 server(boost::asio::io_context & io_context, short port)
   : acceptor_(io_context, tcp::endpoint(tcp::v4(), port)) {
   do_accept();
 }


 .
 .
 .

};
```

Boost.Asio Example

forward request to io_context
provide handler for I/O completion

io_context

```
void do_accept() {
   acceptor_.async_accept(
      [this](error_code ec, tcp::socket socket) {
        if (!ec) {
          std::make_shared<session>(std::move(socket))->start();
        }
        do_accept();
      });
 }
```

socket cannot be copied, so move is used here

Boost.Asio Example

```
class session : public std::enable_shared_from_this<session> {
private:
 tcp::socket socket_;
 enum { max_length = 1024 };
 char data_[max_length];

public:
 session(tcp::socket socket) : socket_(std::move(socket)) {}
 void start() { do_read(); }

 .
 .
 .
```

socket cannot be copied, so move is used here

Boost.Asio Example

```
void do_read() {
  auto self(shared_from_this());
  socket_.async_read_some(
      boost::asio::buffer(data_, max_length),
      [this, self](error_code ec, std::size_t length) {
        if (!ec) do_write(length);
      });
}

void do_write(std::size_t length) {
  auto self(shared_from_this());
  boost::asio::async_write(
      socket_, boost::asio::buffer(data_, length),
      [this, self](error_code ec, std::size_t length) {
        if (!ec) do_read();
      });
}
```

forward request to io_context
provide handler for I/O completion

io_context

forward request to io_context
provide handler for I/O completion

Reference

- https://en.cppreference.com/w/cpp/language/lambda
- https://en.cppreference.com/w/cpp/language/auto
- https://en.cppreference.com/w/cpp/memory/shared_ptr
- https://en.cppreference.com/w/cpp/memory/enable_shared_from_this
- https://en.cppreference.com/w/cpp/utility/move
- https://www.boost.org/doc/libs/1_70_0/doc/html/boost_asio.html