Проектирование сетевых приложений

Подготовительное отделение C/C++ (открытый курс)

образование

Асинхронность

```
int get_value();

int value = get_value();
```

```
void on_value_got(int val)
{
    ...
}

void get_value();

get_value();
```

```
void on_value_got1(int value)
{ ... }

void on_value_got2(int value)
{ ... }

void get_value(void(*cb)(int));

void f() {
    get_value(on_value_got1);
}

void g() {
    get_value(on_value_got2);
}
```

```
struct Context {
    ...
};

void on_value_got1(int value, void* udata)
{
    Context* ctx = (Context*)udata;
    ...
    delete ctx;
}

void
get_value(void(*cb)(int, void*), void* udata);

void f() {
    get_value(on_value_got1, new Context {
        ...
    });
}
```

Функциональный объект (функтор)

```cpp
class LessThan {
    int val;

public:
    LessThan(int val): val(val) {}
    bool operator()(int n) const { return n < val; }
};


LessThan less_than_five(5);


std::cout << less_than_five(3) << std::endl; // 1
std::cout << less_than_five(7) << std::endl; // 0
```

Лямбда-функции

```
class Vector {
    ...
public:
    ...
    void for_each(std::function<void(int elem)>);
};

void print_elem(int elem) {
    std::cout << elem << std::endl;
}

class Summator {
    int& sum;
public:
    Summator(int& sum): sum(sum) {}
    void operator()(int elem) const { sum += elem; }
};


Vector v;

v.for_each(print_elem);
```

```
int sum = 0;
Summator summator(sum);
v.for_each(summator);
```

```
int sum = 0;

v.for_each([&sum] (int elem) { sum += elem; });
```

Асинхронность в C++

```
struct Context {
    ...
};


void on_value_got1(int value, void* udata)
{
    Context* ctx = (Context*)udata;
    ...
    delete ctx;
}
```

```
void get_value(void(*cb)(int, void*), void* udata);


void f() {
    ...
    get_value(on_value_got1, new Context {
        ...
    });
}
```

```
void get_value(std::function<void(int value)>);


void f() {
    ...
    get_value([...] (int value) {
        ...
    });
}
```

Протокол HTTP 1.1

```
GET /resource1 HTTP/1.1\r\n
Host: example.com\r\n
Connection: keep-alive\r\n
Keep-Alive: 300\r\n
\r\n

GET /resource2 HTTP/1.1\r\n
Host: example.com\r\n
Connection: keep-alive\r\n
Keep-Alive: 300\r\n
\r\n

GET /resource3 HTTP/1.1\r\n
Host: example.com\r\n
Connection: close\r\n
\r\n
```

```
HTTP/1.1 200 OK\r\n
Content-Type: text/plain\r\n
Content-Length: 15\r\n
Connection: keep-alive\r\n
Keep-Alive: 300\r\n
\r\n
Hello, World!!!

HTTP/1.1 404 Not Found\r\n
Connection: keep-alive\r\n
Keep-Alive: 300\r\n
\r\n

HTTP/1.1 200 OK\r\n
Content-Type: video/mp4\r\n
Content-Length: 13268\r\n
Connection: close\r\n
\r\n
<<13268 bytes of binary data>>
```

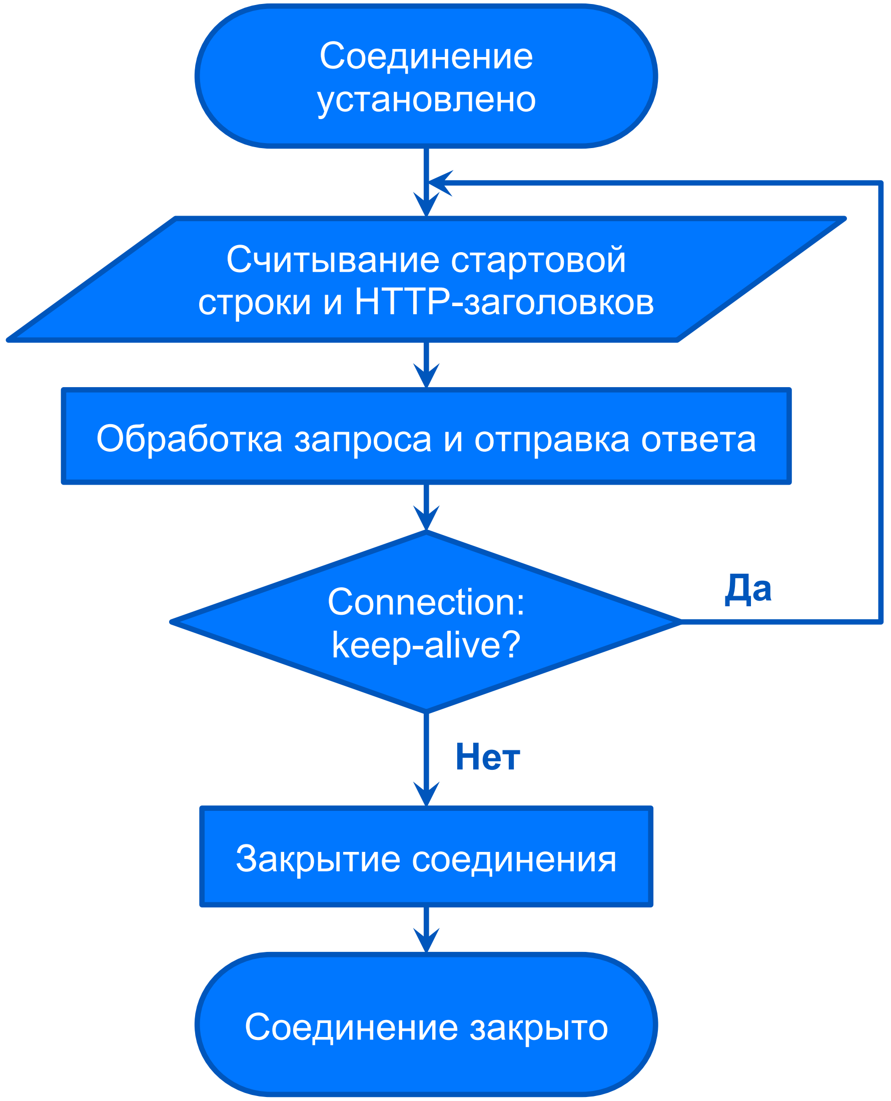
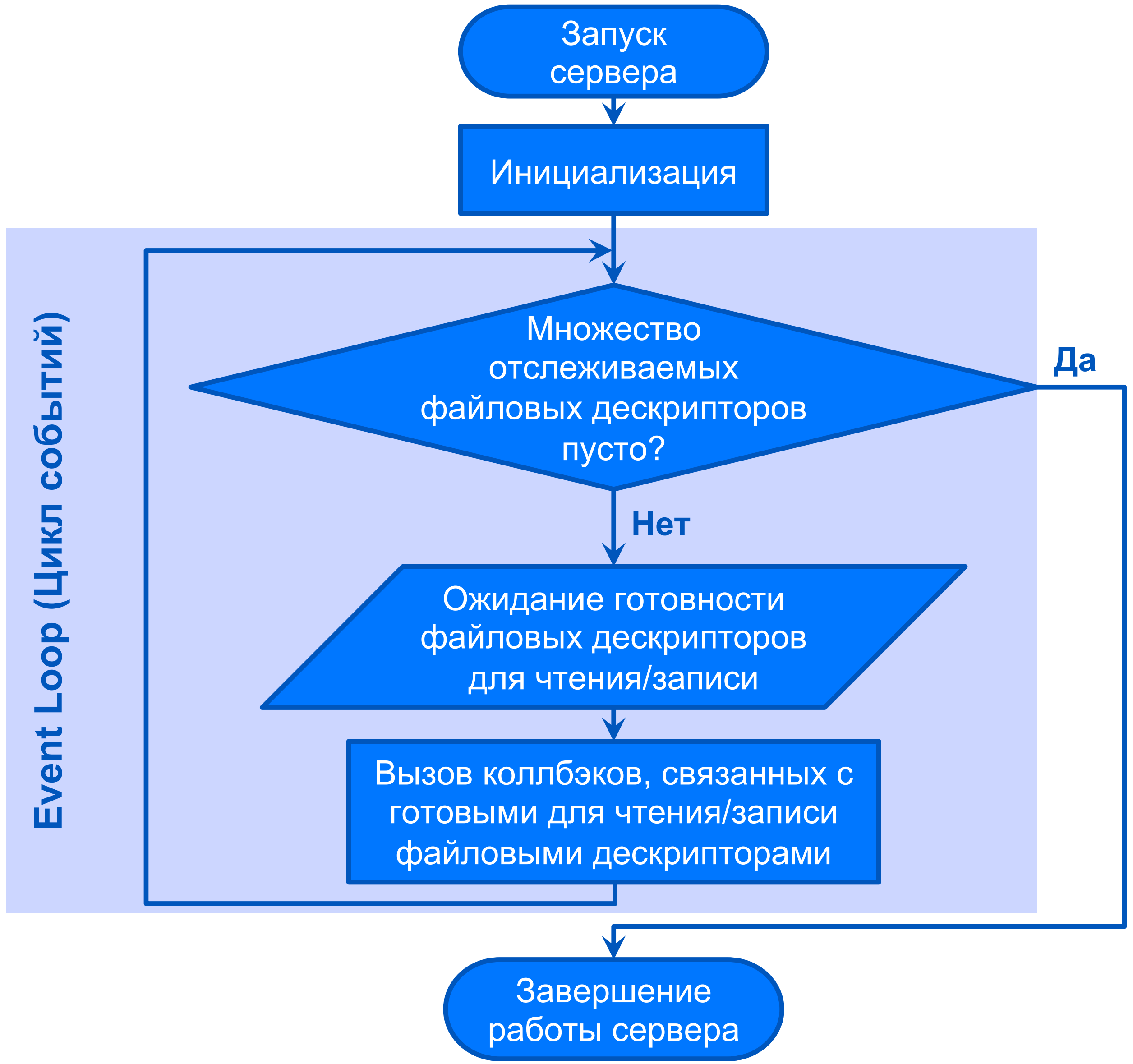
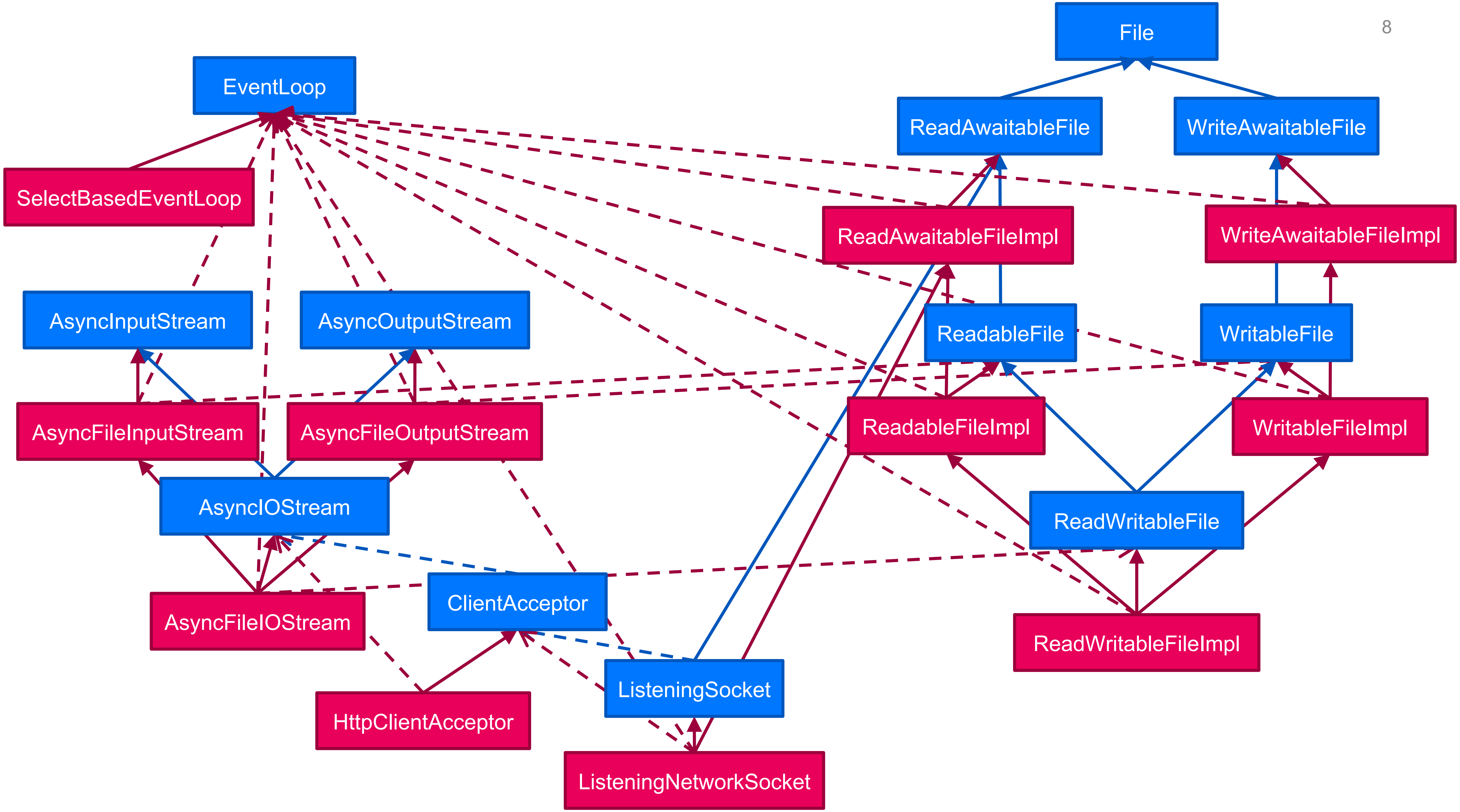


Схема работы сервера


Запуск сервера
Инициализация
Event Loop (Цикл событий)
Множество отслеживаемых файловых дескрипторов пусто?
Да
Нет
Ожидание готовности файловых дескрипторов для чтения/записи
Вызов коллбэков, связанных с готовыми для чтения/записи файловыми дескрипторами
Завершение работы сервера


File
EventLoop
ReadAwaitableFile
WriteAwaitableFile
SelectBasedEventLoop
ReadAwaitableFileImpl
WriteAwaitableFileImpl
AsyncInputStream
AsyncOutputStream
ReadableFile
WritableFile
AsyncFileInputStream
AsyncFileOutputStream
ReadableFileImpl
WritableFileImpl
AsyncIOStream
ReadWritableFile
AsyncFileIOStream
ClientAcceptor
ReadWritableFileImpl
ListeningSocket
HttpClientAcceptor
ListeningNetworkSocket

File

```
class File {
    int fd;

public:
    File(int fd): fd{ fd } {}
    virtual ~File() { ::close(fd); }
    int get_fd() const { return fd; }
};
```

```
class ReadAwaitableFile: virtual public File {
public:
    using Cb = std::function<void(bool success)>;
    virtual void on_readable(Cb) const = 0;
};
```

```
class WriteAwaitableFile: virtual public File {
public:
    using Cb = std::function<void(bool success)>;
    virtual void on_writable(Cb) const = 0;
};
```

```
class ReadableFile: virtual public ReadAwaitableFile {
public:
    using Cb = std::function<void(ssize_t read)>;
    virtual void read(char* buf, size_t buf_size, Cb) = 0;
};
```

```
class WritableFile: virtual public WriteAwaitableFile {
public:
    using Cb = std::function<void(ssize_t written)>;
    virtual void write(const char* buf, size_t buf_size, Cb) = 0;
};
```

```
class ReadWritableFile: virtual public ReadableFile, virtual public WritableFile {
};
```

AsyncStream

```
class AsyncInputStream {
public:
    using Cb = std::function<void(bool success, const char* buf, size_t size)>;

    virtual ~AsyncInputStream() = default;

    virtual void read_till(const char* delimiter, size_t del_size, Cb) = 0;
    virtual void read(size_t count, Cb) = 0;
};


class AsyncOutputStream {
public:
    using Cb = std::function<void(bool success)>;

    virtual ~AsyncOutputStream() = default;

    virtual void write(const char* buf, size_t count, Cb) = 0;
};


class AsyncIOStream:
    virtual public AsyncInputStream,
    virtual public AsyncOutputStream
{
};
```

ClientAcceptor

```
class ClientAcceptor {
public:
    virtual ~ClientAcceptor() = default;
    virtual void accept(AsyncIOStream*) = 0;
};
```

ListeningSocket

```
class ListeningSocket: virtual public ReadAwaitableFile
{
public:
    virtual ~ListeningSocket() = default;
    virtual void start_listening(ClientAcceptor&) = 0;
};
```

EventLoop

```
#include <functional>

class EventLoop {
public:
    using ResCb = std::function<void(bool success)>;
    using Cb = std::function<void()>;

    virtual ~EventLoop() = default;

    virtual void on_readable(int fd, ResCb) = 0;
    virtual void on_writable(int fd, ResCb) = 0;
    virtual void schedule(Cb) = 0;
};
```

Спасибо за внимание!

E-mail: i.anferov@corp.mail.ru
Telegram: igor_anferov
GitHub: igor-anferov

Игорь Анфёров

образование