Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №6-8 по курсу**

**«Операционные системы»**

Студент: Бутырев Даниил Вячеславович

Группа: М8О–206Б–20

Вариант: 21

Преподаватель: Соколов Андрей Алексеевич

Оценка:

Дата:

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Управлении серверами сообщений (№ 6)
* Применение отложенных вычислений (№ 7)
* Интеграция программных систем друг с другом (№ 8)

## Задание

Реализовать распределенную систему по асинхронной обработке запросов. В данной распределенной системе должно существовать 2 вида узлов: «управляющий» и «вычислительный». Необходимо объединить данные узлы в соответствии с той топологией, которая определена вариантом. Связь между узлами необходимо осуществить при помощи технологии очередей сообщений. Также в данной системе необходимо предусмотреть проверку доступности узлов в соответствии с вариантом. При убийстве («kill -9») любого вычислительного узла система должна пытаться максимально сохранять свою работоспособность, а именно все дочерние узлы убитого узла могут стать недоступными, но родительские узлы должны сохранить свою работоспособность.

Управляющий узел отвечает за ввод команд от пользователя и отправку этих команд на вычислительные узлы. Список основных поддерживаемых команд:

Создание нового вычислительного узла

Формат команды: create id [parent]  
id – целочисленный идентификатор нового вычислительного узла

parent – целочисленный идентификатор родительского узла. Если топологией не предусмотрено введение данного параметра, то его необходимо игнорировать (если его ввели)

Формат вывода:

«Ok: pid», где pid – идентификатор процесса для созданного вычислительного узла

«Error: Already exists» - вычислительный узел с таким идентификатором уже существует

«Error: Parent not found» - нет такого родительского узла с таким идентификатором

«Error: Parent is unavailable» - родительский узел существует, но по каким-то причинам с ним не удается связаться

«Error: [Custom error]» - любая другая обрабатываемая ошибка Пример:  
> create 10 5  
Ok: 3128

Примечания: создание нового управляющего узла осуществляется пользователем программы при помощи запуска исполняемого файла. Id и pid — это разные идентификаторы.

Удаление существующего вычислительного узла

Формат команды: remove id  
id – целочисленный идентификатор удаляемого вычислительного узла

Формат вывода:

«Ok» - успешное удаление

«Error: Not found» - вычислительный узел с таким идентификатором не найден

«Error: Node is unavailable» - по каким-то причинам не удается связаться с вычислительным узлом

«Error: [Custom error]» - любая другая обрабатываемая ошибка

Пример:

> remove 10

Ok

Примечание: при удалении узла из топологии его процесс должен быть завершен и работоспособность вычислительной сети не должна быть нарушена.

Исполнение команды на вычислительном узле

Формат команды: exec id [params]

id – целочисленный идентификатор вычислительного узла, на который отправляется команда

Формат вывода:

«Ok:id: [result]», где result – результат выполненной команды

«Error:id: Not found» - вычислительный узел с таким идентификатором не найден

«Error:id: Node is unavailable» - по каким-то причинам не удается связаться с вычислительным узлом

«Error:id: [Custom error]» - любая другая обрабатываемая ошибка  
Пример:  
Можно найти в описании конкретной команды, определенной вариантом задания.

Примечание: выполнение команд должно быть асинхронным. Т.е. пока выполняется команда на одном из вычислительных узлов, то можно отправить следующую команду на другой вычислительный узел.

Вариант 21: топология 4 - тип команд 3 — тип проверки доступности узлов 2

(3 топология: Все вычислительные узлы хранятся в бинарном дереве поиска. [parent] — является необязательным параметром.)

Топология 4 аналогично 3, но узлы находятся в идеально сбалансированном бинарном дереве. Каждый следующий узел должен добавляться в самое наименьшее поддерево.

**Набора команд 3 (локальный таймер)**

Формат команды сохранения значения: exec id subcommand

subcommand – одна из трех команд: start, stop, time.

start – запустить таймер

stop – остановить таймер

time – показать время локального таймера в миллисекундах

Пример:

> exec 10 time

Ok:10: 0

>exec 10 start

Ok:10

>exec 10 start

Ok:10

\*прошло 10 секунд\*

> exec 10 time

Ok:10: 10000

\*прошло 2 секунды\*

>exec 10 stop

Ok:10

\*прошло 2 секунды\*

>exec 10 time

Ok:10: 12000

**Команда проверки 2**

Формат команды: ping id

Команда проверяет доступность конкретного узла. Если узла нет, то необходимо выводить ошибку: «Error: Not found» Пример:

> ping 10

Ok: 1 // узел 10 доступен

> ping 17

Ok: 0 // узел 17 недоступен

**Общие сведения о программе**

В программе используется брокер сообщений RabbitMQ с помощью библиотеки SimpleAmqpClient.

Код программы:

error\_string.cpp:

#include <string>

namespace Error {

std::string forking\_process( const int pid ) {

std::string return\_str = "error: forking process: ";

return return\_str + std::to\_string( pid );

}

std::string executing\_program( const std::string& pathname ) {

std::string return\_str = "error: executing program: ";

return return\_str + pathname;

}

std::string mmap\_failed( const int mapping\_length ) {

std::string return\_str = "error: mmap memory with length: ";

return return\_str + std::to\_string( mapping\_length );

}

std::string kill\_failed( pid\_t pid, int signal ) {

std::string return\_str = "error: kill failed : pid = ";

return return\_str + std::to\_string( static\_cast< int >( pid ) ) +

" and signal = " + std::to\_string( signal );

}

std::string opening\_file( const std::string& file ) {

std::string return\_str = "error: open failed, file: ";

return return\_str + file;

}

std::string writing\_to\_file( const int fd, const std::string& file ) {

std::string return\_str = "error: writing to file with fd: ";

return return\_str + std::to\_string( fd ) + " and sentence: " + file;

}

std::string dlclose\_fail() {

return "error: dlclose fail";

}

};

error\_string.hpp:

#pragma once

#include <string>

namespace Error {

std::string forking\_process( const int pid );

std::string executing\_program( const std::string& pathname );

std::string mmap\_failed( const int mapping\_length );

std::string kill\_failed( pid\_t pid, int signal );

std::string opening\_file( const std::string& file );

std::string writing\_to\_file( const int fd, const std::string& file );

std::string dlclose\_fail();

};

exec\_arg.cpp:

#include <memory>

#include <type\_traits>

#include <cstdint>

#include <string>

#include <iostream>

#include "explicit\_enum.hpp"

#include "exec\_arg.hpp"

namespace ExecArg {

template < typename T >

void make\_execv\_arg( const T pointer, std::string& pathname,

char\*\*& exec\_arg ) {

std::allocator< char > alloc\_ch;

const int exec\_arg\_count = 4; //name, pointer, mask, null

char\* pointer\_arg = alloc\_ch.allocate( 2 \* ( sizeof( T ) + sizeof( '\0' ) ) );

const int mask\_offset = sizeof( T ) + sizeof( '\0' );

for( decltype( sizeof( T ) ) i = StandardValue::array\_begin;

i < sizeof( T ); ++i) {

pointer\_arg[ i ] = \*( reinterpret\_cast< const unsigned char\* >( &pointer ) + i );

pointer\_arg[ i + mask\_offset ] = ExecValue::here\_not\_zero;

if( pointer\_arg[ i ] == '\0' ) {

pointer\_arg[ i ] = ExecValue::not\_zero;

pointer\_arg[ i + mask\_offset ] = ExecValue::here\_zero;

}

}

pointer\_arg[ sizeof( T ) ] = '\0';

pointer\_arg[ mask\_offset + sizeof( T ) ] = '\0';

std::allocator< char\* > alloc\_chptr;

exec\_arg = alloc\_chptr.allocate( exec\_arg\_count );

exec\_arg[ ExecValue::program\_name ] = pathname.data();

exec\_arg[ ExecValue::pointer\_arg ] = pointer\_arg;

exec\_arg[ ExecValue::mask\_pointer\_arg ] = pointer\_arg + mask\_offset;

exec\_arg[ exec\_arg\_count - 1 ] = ExecValue\_last\_arg; //last should be null

}

template < typename T >

void get\_execv\_arg( T& pointer, const char\*\* exec\_arg ) {

char pointer\_raw[ sizeof( T ) ];

for( decltype( sizeof( T ) ) i = 0; i < sizeof( T ); ++i ) {

pointer\_raw[ i ] = exec\_arg[ ExecValue::pointer\_arg ][ i ];

if( exec\_arg[ ExecValue::mask\_pointer\_arg ][ i ] == ExecValue::here\_zero ) {

pointer\_raw[ i ] = 0;

}

}

pointer = \*reinterpret\_cast< T\* >( pointer\_raw );

}

};

template void ExecArg::make\_execv\_arg< int >( const int, std::string&, char\*\*& );

template void ExecArg::get\_execv\_arg< int >( int&, const char\*\* );

exec\_arg.hpp:

#pragma once

namespace ExecArg {

template < typename T >

void make\_execv\_arg( const T, std::string&, char\*\*& );

template <typename T>

void get\_execv\_arg( T& , const char\*\* );

}

explicit\_enum.cpp:

// ReturnValue::last\_arg = nullptr;

explicit\_enum.hpp:

#pragma once

namespace StandardValue {

enum {

bits\_in\_byte = 8,

array\_begin = 0,

};

};

namespace ReturnValue {

enum standard {

nice = 0,

};

enum exec {

exec\_fail = -1,

};

enum mmap {

mmap\_failed,

};

enum fork {

fork\_fail = -1,

fork\_child\_process = 0,

};

enum kill {

bad\_kill = -1,

};

enum open {

open\_fail = -1,

};

enum write {

write\_fail = -1,

};

enum dl {

dlclose\_success = 0,

#define ReturnValue\_dlerror\_no\_error nullptr;

};

enum error {

error\_executing\_program = 1,

error\_forking\_process,

error\_opening\_file,

error\_not\_enough\_args,

error\_writing\_to\_file,

error\_dlclose\_fail,

error\_dlsym\_fail,

error\_dlopen\_fail,

error\_unavailable\_node,

};

};

namespace MmapValue {

enum flags { //exactly

anon\_fd = -1,

anon\_offset = 0,

no\_offset = 0,

};

};

namespace ExecValue {

enum args {

here\_not\_zero = 1,

here\_zero = 2,

not\_zero = 1,

};

#define ExecValue\_last\_arg nullptr

enum pointer {

program\_name = 0,

pointer\_arg = 1,

mask\_pointer\_arg = 2,

};

};

// enum {

// read\_end=0,

// write\_end=1,

// };

// enum {

// stdin = 0,

// stdout = 1,

// stderr = 2,

// };

// error\_creating\_pipe,

// error\_child\_args,

// error\_creating\_process,

// error\_reading\_input,

// error\_writing\_pipe,

// error\_reading\_pipe,

// error\_writing\_err,

// };

makefile:

yess:

g++ -g -Wall -pedantic mlink.cpp error\_string.cpp exec\_arg.cpp timer.cpp worker.cpp -lSimpleAmqpClient -lboost\_chrono -Wl,-rpath,/usr/local/lib -o worker

g++ -g -Wall -pedantic mlink.cpp error\_string.cpp exec\_arg.cpp server.cpp -lSimpleAmqpClient -lboost\_chrono -Wl,-rpath,/usr/local/lib -o server

ld:

LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/usr/local/lib

export LD\_LIBRARY\_PATH

reset:

sudo rabbitmqctl list\_queues --vhost myvhost

sudo rabbitmqctl stop\_app

sudo rabbitmqctl reset

sudo rabbitmqctl start\_app

sudo rabbitmqctl add\_user newuser newuser

sudo rabbitmqctl add\_vhost myvhost

sudo rabbitmqctl set\_permissions -p "myvhost" "newuser" '.\*' '.\*' '.\*'

sudo rabbitmqctl list\_queues --vhost myvhost

list:

sudo rabbitmqctl list\_queues --vhost myvhost

sudo rabbitmqctl list\_consumers --vhost myvhost

mlink.cpp:

#include <signal.h>

#include <sys/types.h>

#include <unistd.h>

#include <string>

#include <iostream>

#include <sstream>

#include <utility>

#include "mlink.hpp"

#include "exec\_arg.hpp"

#include "explicit\_enum.hpp"

#include "error\_string.hpp"

/\*

\*\* instant\_ping == "[Command::instant\_ping] [id: who sends it]"

\*\* instant\_ping\_reply = "[Command::instant\_ping\_reply] [id: who should get it]"

\*\* out id = "[Command::id\_c] [id]" // I have to do that because worker can't know who will be at the top channel[ queue::up ].id[ ID::out ]

\*\* command:= "[Command] [id] "

\*\* attach == "[Command::attach\_c] [id\_whom\_to\_attach] [pid]"

\*\* exec == "[Command::exec\_c] [id] []"

\*\* ping == "[Command::ping\_c] [id]"

\*\* reply == "[Command::reply\_c] [id to whom send it] [reply]"

\*\* attach reply == "[Command::attach\_c] [id who sends message] [id who was attached] [pid of attached process] [Reply::ok]"

\*\* ping reply == "[Command::ping\_c] [id] [Reply::ok]"

\*\* exec reply == "[Command::exec\_c] [id] [ExecCommand::... [if time->time\_result]] [Reply::ok]"

\*\* errors: [Command::error\_c] [id]

\*\* [MLError::id\_unreachable] [id that is unreachable]

\*\* [MLError::id\_doesnt\_exist] [id that is not exist]

\*\* [MLError::id\_already\_exist]

\*\*

\*/

MNode::MNode()

: links\_per\_node( 3 ), queues\_per\_link( 4 ),

consumers\_per\_link( queues\_per\_link / 2 ), id( links\_per\_node + 1 /\* my \*/ ),

queue\_name( links\_per\_node, std::vector< std::string >( queues\_per\_link ) ),

consumer\_tag( links\_per\_node, std::vector< std::string >( consumers\_per\_link ) ),

routing\_key( links\_per\_node, std::vector< std::string >( queues\_per\_link - 1 ) ),

link\_exists( links\_per\_node, false ), delete\_link( links\_per\_node, false ) {}

void MNode::create\_queue( int link, MNode::queue\_e \_queue, const std::string& suffix ) {

int queue = static\_cast< int >( \_queue );

queue\_name[ link ][ queue ] = "q:";

queue\_name[ link ][ queue ] += suffix;

/\*queue\_broker\_name = \*/ channel->

DeclareQueue( queue\_name[ link ][ queue ], false, false, false, false );

}

void MNode::create\_consumer( int link, MNode::queue\_e \_queue, const std::string& suffix ) {

int queue = static\_cast< int >( \_queue );

if( \_queue == queue\_e::in ) {

std::cerr << "id " << id[ ID::my ] << ": creating consumer: <" << queue\_name[ link ][ queue ] << '>'

<< " in consumer\_tag[" << link << "][" << static\_cast< int >( consumer\_e::in ) << "]\n";

consumer\_tag[ link ][ static\_cast< int >( consumer\_e::in ) ] =

channel->BasicConsume( queue\_name[ link ][ queue ],

queue\_name[ link ][ queue ], true, false, true, 0 );

} else if( \_queue == queue\_e::ping\_in ) {

std::cerr << "id " << id[ ID::my ] << ": creating consumer: <" << queue\_name[ link ][ queue ] << '>'

<< " in consumer\_tag[" << link << "][" << static\_cast< int >( consumer\_e::ping\_in ) << "]\n";

consumer\_tag[ link ][ static\_cast< int >( consumer\_e::ping\_in ) ] =

channel->BasicConsume( queue\_name[ link ][ queue ],

queue\_name[ link ][ queue ], true, false, true, 100 );

} else {

std::cerr << "create\_consumer has failed\n";

}

}

void MNode::create\_publisher( int link, MNode::queue\_e queue, const std::string& suffix ) {

MNode::routing\_key\_e rk;

routing\_key[ link ][ static\_cast< int >( rk = routing\_key\_e::err ) ] = suffix;

if( queue == queue\_e::in ) {

} else if( queue == queue\_e::out ) {

routing\_key[ link ][ static\_cast< int >( rk = routing\_key\_e::out ) ] = suffix;

} else if( queue == queue\_e::ping\_out ) {

routing\_key[ link ][ static\_cast< int >( rk = routing\_key\_e::ping\_out ) ] = suffix;

} else {

std::cerr << "create\_publisher fail\n";

}

channel->BindQueue( queue\_name[ link ][ static\_cast< int >( queue ) ],

exchange\_name, routing\_key[ link ][ static\_cast< int >( rk ) ] );

}

void MNode::create( bool swap, const std::string& one, const std::string& two,

pid\_t pid, int link, MNode::queue\_e q, bool publish, bool consume ) {

std::stringstream suffix;

if( !swap )

suffix << one;

else

suffix << two;

suffix << pid;

// channel[ in ]->CheckExchangeExists( exch\_n );

create\_queue( link, q, suffix.str() );

if( publish ) create\_publisher( link, q, suffix.str() );

if( consume ) create\_consumer( link, q, suffix.str() );

}

void MNode::create( int link, pid\_t pid, bool swap ) {

Channel::OpenOpts openopts; //= Channel::OpenOpts::FromUri( "amqp://guest:guest@localhost:5672/" );

openopts.host = "localhost";

openopts.vhost = "myvhost";

openopts.port = 5672;

openopts.auth = Channel::OpenOpts::BasicAuth( "newuser", "newuser" );

channel = Channel::Open( openopts );

std::stringstream suffix;

suffix << "e:" << pid;

exchange\_name = suffix.str();

channel->DeclareExchange( exchange\_name );

create( swap, "in:", "out:", pid, link, queue\_e::in, true, true );

create( swap, "out:", "in:", pid, link, queue\_e::out, true, false );

create( swap, "ping\_in:", "ping\_out:", pid, link, queue\_e::ping\_in, false, true );

create( swap, "ping\_out:", "ping\_in:", pid, link, queue\_e::ping\_out, true, false );

link\_exists[ link ] = true;

}

void MNode::create\_link( int link, pid\_t pid, int my\_id ) {

id[ ID::my ] = my\_id;

create( link, pid, false );

id[ link ] = get\_out\_id( link );

delete\_link[ link ] = true;

}

int MNode::get\_out\_id( int link ) {

std::string message;

bool wait;

get\_message( link, message, wait = true, consumer\_e::in );

return get\_out\_id( message );

}

int MNode::get\_out\_id( const std::string& message ) {

std::stringstream mssg( message );

int answ;

int command;

mssg >> command >> answ;

return answ;

}

void MNode::get\_id( int& id, const char\*\* argv ) {

ExecArg::get\_execv\_arg< int >( id, argv );

}

// MNode& MNode::operator<<( MNode& rhs ) { //??can he

// AmqpClient::Envelope::ptr\_t message;

// const int timeout = 10;

// while( rhs.channel[ queue::in ]->BasicConsumeMessage( rhs.consumer\_tag, message, timeout ) ) {

// send\_message( message->Message()->Body() );

// }

// return \*this;

// }

bool MNode::exist( int link ) const {

return link\_exists[ link ];

}

static int \_\_time = 0;

bool MNode::get\_message( int link /\* = -1 \*/, std::string& message,

bool wait /\* = true \*/, MNode::consumer\_e \_consumer /\* = default\_arg \*/ ) {

//I could make variant with consumer != -1, but I don't need to

AmqpClient::Envelope::ptr\_t envelope;

std::vector< std::string > consumers;

const int in = static\_cast< int >( consumer\_e::in );

const int ping\_in = static\_cast< int >( consumer\_e::ping\_in );

const int consumer = static\_cast< int >( \_consumer );

if( link == -1 ) {

if( \_consumer == MNode::consumer\_e::default\_arg ) { //all available

for( int i = 0; i < links\_per\_node; ++i ) {

if( link\_exists[ i ] ) {

consumers.push\_back( consumer\_tag[ i ][ in ] );

consumers.push\_back( consumer\_tag[ i ][ ping\_in ] );

}

}

} else {

for( int i = 0; i < links\_per\_node; ++i ) {

if( link\_exists[ i ] ) {

consumers.push\_back( consumer\_tag[ i ][ consumer ] );

}

}

}

} else { //link != -1

if( \_consumer == MNode::consumer\_e::default\_arg ) { //all available

if( link\_exists[ link ] ) {

consumers.push\_back( consumer\_tag[ link ][ in ] );

consumers.push\_back( consumer\_tag[ link ][ ping\_in ] );

}

} else {

if( link\_exists[ link ] ) {

consumers.push\_back( consumer\_tag[ link ][ consumer ] );

}

}

}

// for( const auto& i : consumers ) {

// try {

// channel->BasicConsume( i, i, true, false, true, 100 );

// } catch ( ... ) {}

// }

if( consumers.empty() ) {

return false;

}

if( \_\_time == 0 ) {

std::cerr << "id " << id[ ID::my ] << ": consumers";

for( auto i : consumers ) std::cerr << " id " << id[ ID::my ] << ": <" << i << '>';

std::cerr << '\n';

\_\_time = 1;

}

try {

if( wait ) {

envelope = channel->BasicConsumeMessage( consumers ); //error? what if consumer\_tag[ i ] == ""?

channel->BasicAck( envelope );

message = envelope->Message()->Body();

\_\_time = 0;

return true;

} else {

const int timeout = 100;

if ( channel->BasicConsumeMessage( consumers, envelope, timeout ) ) {

channel->BasicAck( envelope );

message = envelope->Message()->Body();

\_\_time = 0;

return true;

} else {

return false;

}

}

} catch ( AmqpClient::ConsumerTagNotFoundException exc ) {

std::cerr << "id " << id[ ID::my ] << ": " << exc.what() << '\n';

return false;

}

}

// void MNode::ping\_reply() {

// AmqpClient::Envelope::ptr\_t message;

// while( channel[ queue::ping\_in ]->BasicGet( message, queue\_name[ queue::ping\_in ] ) ) {

// channel[ queue::ping\_out ]->BasicPublish( exchange\_name[ queue::ping\_out ],

// routing\_key, AmqpClient::BasicMessage::Create( MNode::instant\_ping\_reply() ) );

// }

// }

// bool MNode::instant\_ping( const std::string& message ) {

// return message == "p::";

// }

void MNode::send\_instant\_ping\_reply( const std::string& command ) {

std::stringstream mssg( command );

int useless;

int \_id;

bool check\_connection;

mssg >> useless >> \_id;

std::stringstream send\_mssg; // why can't I just use mssg( "" )??

send\_mssg << static\_cast< int >( Command::instant\_ping\_reply ) << ' ' << \_id;

for( int link = 0; link < links\_per\_node; ++link ) {

if( link\_exists[ link ] && ( id[ link ] == \_id ) ) {

send\_message( link, send\_mssg.str(),

check\_connection = false, routing\_key\_e::ping\_out );

return;

}

}

std::cerr << "instant\_ping\_reply failed\n";

}

// std::string MNode::instant\_ping\_reply( const std::string& message ) {

// std::stringstream mssg;

// mssg << static\_cast< int >( Command::reply\_c ) << ' ' << message;

// return mssg.str();

// }

MNode::Command MNode::get\_command( const std::string& message ) {

std::stringstream mssg;

mssg.str( message );

int command;

mssg >> command;

return static\_cast< MNode::Command >( command );

}

int MNode::get\_id( const std::string& message ) {

std::stringstream mssg( message );

int command;

int id;

mssg >> command;

mssg >> id;

return id;

}

void MNode::send\_message( int link, const std::string& message,

bool check\_connection, MNode::routing\_key\_e \_rout /\* =routing\_key\_e::out \*/ ) { //if don't have id

const int rout = static\_cast< int >( \_rout );

std::cerr << "id " << id[ static\_cast< int >( ID::my ) ]

<< ": sending message to id " << id[ link ] << ":\"" << message << "\"\n";

if( !check\_connection ) {

channel->BasicPublish( exchange\_name,

routing\_key[ link ][ rout ],

AmqpClient::BasicMessage::Create( message ) );

} else if( ping( link ) ) {

channel->BasicPublish( exchange\_name, routing\_key[ link ][ rout ],

AmqpClient::BasicMessage::Create( message ) );

} else {

std::string error\_message = MNode::error\_id\_unreachable( id[ link ], id[ ID::my ]);

//throw message somewhere

const int err = static\_cast< int >( routing\_key\_e::err );

channel->BasicPublish( exchange\_name, routing\_key[ link ][ err ],

AmqpClient::BasicMessage::Create( error\_message ) );

}

}

bool MNode::ping( int link ) {

bool check\_connection;

send\_message( link, instant\_ping\_message(),

check\_connection = false, routing\_key\_e::ping\_out );

// channel->BasicPublish( exchange\_name, routing\_key[ link ][ routkeyping\_out ],

// AmqpClient::BasicMessage::Create( MNode::instant\_ping\_message() ) );

AmqpClient::Envelope::ptr\_t message; //maybe wait?

// sleep( 5 );

bool wait;

std::string \_message;

return get\_message( link, \_message, wait = false, consumer\_e::ping\_in );

// return channel[ queue::ping\_in ]->BasicGet( message, queue\_name[ queue::ping\_in ] );

//i donno if this would work

//1) if time between sending and receiving would be too big

//2) if I would already have some ping mesages at queue

}

std::string MNode::instant\_ping\_message() {

std::stringstream mssg;

mssg << static\_cast< int >( Command::instant\_ping ) << ' ' << id[ ID::my ];

return mssg.str();

}

// bool MNode::instant\_ping\_reply( const std::string& message ) {

// return message == "p::1";

// }

std::string MNode::error\_id\_unreachable( int id\_out, int id\_my ) {

std::stringstream mssg;

mssg << static\_cast< int >( Command::error\_c ) << ' ' <<

id\_my << ' ' << static\_cast< int >( MLError::id\_unreachable ) << ' ' << id\_out;

return mssg.str();

}

void MNode::attach\_worker( int link, const std::string& command ) {

std::stringstream mssg( command );

int useless;

int \_id;

mssg >> useless; //command

mssg >> \_id;

int pid; // size( pid\_t ) > size( int ) ?

mssg >> pid;

attach\_worker( link, \_id, pid );

}

void MNode::attach\_worker( int link, int out\_id, pid\_t pid ) {

id[ link ] = out\_id;

bool swap;

create( link, pid, swap = true );

delete\_link[ link ] = false;

std::string mssg\_id = MNode::id\_out\_message( id[ static\_cast< int >( ID::my ) ] );

bool check\_connection;

send\_message( link, mssg\_id, check\_connection = false );

}

std::string MNode::id\_out\_message( int id ) {

std::stringstream mssg;

mssg << static\_cast< int >( Command::id\_c ) << ' ' << id;

return mssg.str();

}

std::string MNode::attach\_reply\_message( const std::string& message, int my\_id ) {

std::stringstream mssg( message );

std::stringstream replymssg;

int another\_id;

int pid;

int command;

mssg >> command >> another\_id >> pid;

replymssg << static\_cast< int >( Command::reply\_c ) << ' '<< static\_cast< int >( Command::attach\_c )

<< ' ' << my\_id << ' ' << another\_id << ' ' << pid << ' ' << static\_cast< int >( Reply::ok );

return replymssg.str();

}

std::string MNode::error\_id\_doesnt\_exist( int id\_out, int id\_my ) {

std::stringstream mssg;

mssg << static\_cast< int >( Command::error\_c ) << ' '

<< id\_my << ' ' << static\_cast< int >( MLError::id\_doesnt\_exist ) << ' ' << id\_out;

return mssg.str();

}

std::string MNode::error\_id\_already\_exist( int id ) {

std::stringstream mssg;

mssg << static\_cast< int >( Command::error\_c ) << ' ' << id << static\_cast< int >( MLError::id\_already\_exist );

return mssg.str();

}

MNode::ExecCommand MNode::get\_exec\_command( const std::string& message ) {

std::stringstream mssg( message );

int useless;

int command;

mssg >> useless; //command

mssg >> useless; //id

mssg >> command;

return static\_cast< ExecCommand >( command );

}

std::string MNode::exec\_reply\_message( int time, int id, ExecCommand command ) {

std::stringstream mssg;

mssg << static\_cast< int >( Command::reply\_c ) << ' '

<< static\_cast< int >( Command::exec\_c ) << ' '

<< id << ' ' << static\_cast< int >( command ) << ' '

<< time << ' ' << static\_cast< int >( Reply::ok );

return mssg.str();

}

std::string MNode::exec\_reply\_message( int id, ExecCommand command ) {

std::stringstream mssg;

mssg << static\_cast< int >( Command::reply\_c ) << ' '

<< static\_cast< int >( Command::exec\_c ) << ' '

<< id << ' ' << static\_cast< int >( command ) << ' '

<< static\_cast< int >( Reply::ok );

return mssg.str();

}

std::string MNode::ping\_reply\_message( int id ) {

std::stringstream mssg;

mssg << static\_cast< int >( Command::reply\_c ) << ' '

<< static\_cast< int >( Command::ping\_c ) << ' '

<< id << ' ' << static\_cast< int >( Reply::ok );

return mssg.str();

}

std::string MNode::make\_reply( const std::string& reply ) {

std::stringstream mssg;

mssg << static\_cast< int >( Command::reply\_c ) << ' ' << reply;

return mssg.str();

}

void MNode::command\_close( int link, int \_id ) {

std::stringstream mssg;

mssg << static\_cast< int >( Command::close\_c ) << ' ' << id[ link ];

bool check\_connection;

send\_message( link, mssg.str(), check\_connection = false );

}

pid\_t MNode::create\_worker( int \_id ) {

pid\_t fork\_res = fork();

if( fork\_res == ReturnValue::fork\_fail ) {

std::cerr << Error::forking\_process( getpid() );

//exit( ReturnValue::error\_fork\_fail );

} else if( fork\_res == ReturnValue::fork\_child\_process ) {

std::string prog\_name = "worker";

char\*\* exec\_arg;

ExecArg::make\_execv\_arg< int >( \_id, prog\_name, exec\_arg );

if( execv( prog\_name.c\_str(), exec\_arg ) == ReturnValue::exec\_fail ) {

std::cerr << Error::executing\_program( prog\_name );

}

} else {

return fork\_res;

}

return -1;

}

void MNode::command\_attach\_worker( int link, int id, pid\_t pid ) {

std::stringstream mssg;

mssg << static\_cast< int >( Command::attach\_c ) << ' ' << id << ' ' << pid;

bool check\_connection;

send\_message( link, mssg.str(), check\_connection = true );

}

void MNode::command\_exec( int link, int id, const std::string& command ) {

std::stringstream mssg;

mssg << static\_cast< int >( Command::exec\_c ) << ' ' << id << ' ';

if( command == "time" ) {

mssg << static\_cast< int >( ExecCommand::time );

} else if( command == "stop" ) {

mssg << static\_cast< int >( ExecCommand::stop );

} else if( command == "start" ) {

mssg << static\_cast< int >( ExecCommand::start );

} else {

std::cerr << "error command\_exec";

return;

}

bool check\_connection;

send\_message( link, mssg.str(), check\_connection = true );

}

void MNode::command\_ping( int link, int id ) {

std::stringstream mssg;

mssg << static\_cast< int >( Command::ping\_c ) << ' ' << id;

bool check\_connection;

send\_message( link, mssg.str(), check\_connection = true );

}

MNode::~MNode() {

for( int link = 0; link < links\_per\_node; ++link ) {

if( link\_exists[ link ] ) {

if( !delete\_link[ link ] ) {

command\_close( link );

} else {

// std::cerr << "id " << id[ static\_cast< int >( ID::my ) ] << "->"

// << id[ link ] << " start cancelling...";

for( int consumer = 0; consumer < consumers\_per\_link /\* = 3 \*/; ++consumer )

channel->BasicCancel( consumer\_tag[ link ][ consumer ] );

// std::cerr << "success; " << std::endl;

//unbind?

channel->DeleteExchange( exchange\_name );

for( int i = 0; i < queues\_per\_link; ++i ) {

std::cerr << "id " << id[ static\_cast< int >( ID::my ) ] << "->"

<< id[ link ]<< " start deleting queue...";

channel->DeleteQueue( queue\_name[ link ][ i ] );

std::cerr << "success; " << std::endl;

}

}

}

}

}

std::string MNode::parse\_reply( const std::string& message ) {

int command\_reply, command\_i, id, other\_id;

ExecCommand exec\_command;

std::stringstream mssg( message );

std::stringstream return\_mssg;

mssg >> command\_reply;

if( command\_reply != static\_cast< int >( Command::reply\_c ) ) {

std::cerr << "parse\_reply fail\n";

return "";

}

mssg >> command\_i >> id;

Command command = static\_cast< Command >( command\_i );

int error\_type\_i;

MLError error\_type;

switch( command ) {

case Command::attach\_c:

int pid;

mssg >> other\_id >> pid;

return\_mssg << "Ok: added worker with id = " << other\_id << ", pid = " << pid << " to worker with id = " << id << '\n';

break;

case Command::exec\_c:

int exec\_command\_i;

mssg >> exec\_command\_i;

exec\_command = static\_cast< ExecCommand >( exec\_command\_i );

if( exec\_command == ExecCommand::time ) {

int time;

mssg >> time;

return\_mssg << "Ok: " << id << " time = " << time << '\n';

} else if( exec\_command == ExecCommand::stop ){

return\_mssg << "Ok: " << id << " timer stopped\n";

} else if( exec\_command == ExecCommand::start ){

return\_mssg << "Ok: " << id << " timer started\n";

} else {

return\_mssg << "Error: operator<<\n";

}

break;

case Command::ping\_c:

return\_mssg << "Ok: worker with id = " << id << " is available\n";

break;

case Command::error\_c:

mssg >> error\_type\_i;

error\_type = static\_cast< MLError >( error\_type\_i );

if( error\_type == MLError::id\_unreachable ) {

mssg >> other\_id;

return\_mssg << "Error: " << id << ": id = " << other\_id << " is unreachable\n";

} else if( error\_type == MLError::id\_doesnt\_exist ) {

mssg >> other\_id;

return\_mssg << "Error: " << id << ": id = " << other\_id << " doesn't exist\n";

} else if( error\_type == MLError::id\_already\_exist ) {

return\_mssg << "Error: id = " << id << " already exist";

}

break;

default:

std::cerr << "parse\_reply fail, unknown reply\n";

break;

}

return return\_mssg.str();

}

void MNode::kill\_process( const std::string& message ) {

int command, id, pid; //attach

std::stringstream mssg( message );

mssg >> command >> id >> pid;

kill( pid, SIGKILL );

}

void MNode::set\_my\_id( int \_id ) {

id[ ID::my ] = \_id;

}

mlink.hpp:

#pragma once

#include <iostream>

#include <SimpleAmqpClient/SimpleAmqpClient.h>

#include <string>

#include <sys/types.h>

#include <unistd.h>

#include <ostream>

namespace Link {

enum Link {

left = 0,

right = 1,

up = 2,

};

};

namespace ID {

enum ID {

left = Link::left,

right = Link::right,

up = Link::up,

my = 3,

};

};

class MNode;

std::ostream& operator<<( std::ostream&, MNode& );

class MNode { //amount of links, here 1 link == 4 queue

using Channel = AmqpClient::Channel;

public:

enum class routing\_key\_e {

out = 0,

ping\_out = 1,

err = 2,

};

enum class consumer\_e {

default\_arg = -1,

in = 0,

ping\_in = 1,

};

enum class queue\_e {

in = 0,

out = 1,

ping\_in = 2,

ping\_out = 3,

};

enum class Command {

id\_c,

attach\_c,

exec\_c,

ping\_c,

close\_c,

error\_c,

reply\_c,

instant\_ping,

instant\_ping\_reply,

undefined\_c,

};

enum class ExecCommand {

stop,

time,

start,

undefined\_ec,

};

enum class MLError {

id\_unreachable,

id\_doesnt\_exist,

id\_already\_exist,

};

MNode();

void create\_link( int, pid\_t, int );

// void send\_message( std::string&, int );

bool exist( int ) const;

bool get\_message( int, std::string&, bool = false,

MNode::consumer\_e = MNode::consumer\_e::default\_arg );

void send\_message( int, const std::string&, bool = true,

MNode::routing\_key\_e = MNode::routing\_key\_e::out );

bool ping( int );

// bool worker\_reachable( int );

// MNode& operator<<( MNode& );

void attach\_worker( int, const std::string& );

void attach\_worker( int, int, pid\_t );

void command\_attach\_worker( int, int, pid\_t );

void command\_exec( int, int, const std::string& );

void command\_ping( int, int );

void command\_close( int, int = ID::my );

void command\_close();

void ping\_reply();

void set\_my\_id( int );

std::string instant\_ping\_message();

void send\_instant\_ping\_reply( const std::string& );

~MNode();

friend std::ostream& operator<<( std::ostream&, MNode& );

// static std::string instant\_ping\_reply();

static Command get\_command( const std::string& );

static int get\_id( const std::string& );

static std::string id\_out\_message( int );

static void get\_id( int&, const char\*\* );

static ExecCommand get\_exec\_command( const std::string& );

static std::string exec\_reply\_message( int, int, ExecCommand );

static std::string exec\_reply\_message( int, ExecCommand );

static std::string ping\_reply\_message( int );

static std::string attach\_reply\_message( const std::string&, int );

static std::string error\_id\_doesnt\_exist( int, int );

static std::string error\_id\_unreachable( int, int );

static std::string error\_id\_already\_exist( int );

static std::string parse\_reply( const std::string& );

static std::string make\_reply( const std::string& reply );

// static bool instant\_ping( const std::string& message );

// static bool instant\_ping\_reply( const std::string& message);

static pid\_t create\_worker( int );

static void kill\_process( const std::string& );

private:

enum class Reply {

ok = 0,

instant\_reply = 1,

};

const int links\_per\_node;

const int queues\_per\_link;

const int consumers\_per\_link;

std::vector< int > id;//[ 4 ];

Channel::ptr\_t channel;

// std::string prefix

std::vector< std::vector < std::string > > queue\_name;//[ 3 ][ 4 ];

// std::string queue\_broker\_name[ 3 ][ 4 ];

std::vector< std::vector< std::string > > consumer\_tag; //[ 3 ][ 2 ]

//so I can't use just vector

//because it could have "" in it,

//so I should copy all not-empty

//tags

std::string exchange\_name;

std::vector< std::vector< std::string > > routing\_key;//[ 3 ][ 2 ];

std::vector< bool > link\_exists;//[ 3 ];

std::vector< bool > delete\_link;//[ 3 ];

void create\_queue( int, MNode::queue\_e, const std::string& );

void create\_consumer( int, MNode::queue\_e, const std::string& );

void create\_publisher( int, MNode::queue\_e, const std::string& );

void create( bool, const std::string&, const std::string&,

pid\_t, int, MNode::queue\_e, bool, bool );

void create( int, pid\_t, bool = false );

int get\_out\_id( int );

int get\_out\_id( const std::string& message );

};

queue\_func.cpp:

#include <iostream>

#include <SimpleAmqpClient/SimpleAmqpClient.h>

#include <string>

#include <sys/types.h>

#include <unistd.h>

#include <ostream>

class MLink;

std::ostream& operator<<( std::ostream&, MLink& );

class MLink {

using Channel = AmqpClient::Channel;

public:

enum Link {

up = 0,

left = 1,

right = 2,

};

enum class Command {

attach\_c,

exec\_c,

ping\_c,

close\_c,

error\_c,

undefined\_c,

};

enum class ExecCommand {

stop,

time,

start,

undefined\_ec,

};

enum class Error {

id\_unreachable,

id\_doesnt\_exist,

id\_already\_exist,

};

MLink();

void create( pid\_t, int, bool = true );

void get\_message( std::string& );

// void send\_message( std::string&, int );

void send\_message( const std::string& );

bool exist() const;

bool worker\_reachable( int );

MLink& operator<<( MLink& );

void attach\_worker( int, pid\_t );

void attach\_worker( int, const std::string& );

void command\_attach\_worker( int, pid\_t );

void command\_exec( int, std::string& );

void command\_ping( int );

void command\_close( int );

void command\_close();

void ping\_reply();

~MLink();

friend std::ostream& operator<<( std::ostream&, MLink& );

static Command get\_command( std::string& );

static void get\_id( int&, char\*\* );

static int get\_id( std::string& );

static ExecCommand get\_exec\_command( std::string& );

static std::string exec\_reply\_message( int, int, ExecCommand );

static std::string exec\_reply\_message( int, ExecCommand );

static std::string ping\_reply\_message( int );

static std::string attach\_reply\_message( const std::string&, int );

static std::string error\_id\_doesnt\_exist( Command, int );

static std::string error\_id\_unreachable( int, int );

static std::string error\_id\_already\_exist( int );

// static bool instant\_ping( const std::string& message );

// static bool instant\_ping\_reply( const std::string& message);

static std::string instant\_ping\_message();

static std::string instant\_ping\_reply();

static pid\_t create\_worker( int );

static void kill\_process( const std::string& );

private:

enum class ch {

in = 0,

out = 1,

ping\_in = 2,

ping\_out = 3,

};

enum class ID {

my = 0,

out = 1,

};

enum class Reply {

ok = 0,

};

int id[ 2 ];

Channel::ptr\_t channel[ 4 ];

// std::string prefix

std::string queue\_name[ 4 ];

std::string queue\_broker\_name[ 4 ];

std::string exchange\_name[ 4 ];

std::string consumer\_tag;

bool connection\_exists;

std::string routing\_key;

void create\_queue( MLink::ch direction, const std::string& suffix );

int get\_out\_id();

int get\_out\_id( std::string& message );

};

server.cpp:

#include <SimpleAmqpClient/SimpleAmqpClient.h>

#include <string>

#include <unistd.h>

#include <iostream>

#include <functional>

#include <thread>

#include "explicit\_enum.hpp"

#include "mlink.hpp"

void manage\_reply( const std::string& reply, MNode& node, int link );

bool wrong\_input( std::basic\_istream< char >& cin = std::cin ) {

if( cin.fail() ) {

// if( std::cin.eof() ) {

// exit( ReturnValue::nice );

std::cerr << "wrong input\n";

cin.clear();

return true;

}

return false;

}

int main() {

bool all\_good = true;

MNode node;

int my\_id = -1;

node.set\_my\_id( my\_id );

//fork-exec another working process with name of queue as execve-argument

int worker\_id = 0;

pid\_t pid = MNode::create\_worker( worker\_id ); //what will happend if I fork or exec in another thread?

int link = Link::right;

node.attach\_worker( link, worker\_id, pid );

std::string message;

// std::function< void() > ping\_reply = [ &stop\_thread, &node, link ](){

// while( !stop\_thread ) {

// std::string message;

// bool wait;

// node.get\_message( link, message, wait = true, consumer::ping\_in );

// if( MNode::instant\_ping( message ) ) {

// bool check\_connection;

// node.send\_message( link, MNode::instant\_ping\_reply(), check\_connection = false, routing\_key::ping\_out );

// } else {

// std::cerr << "something went wrong with ping\_reply\n";

// }

// }

// };

bool stop\_thread = false;

bool got\_message = false;

std::function< void() > get\_message = [ &message, &got\_message, &stop\_thread ](){

while( true ) {

while( got\_message && !stop\_thread )

sleep( 1 );

if( stop\_thread ) return;

std::getline( std::cin, message, '\n' );

got\_message = true;

}

};

std::thread message\_handling( get\_message );

std::cerr << "You can use:\n"

<< "\t->\tcreate [id]\n"

<< "\t->\texec [id] [command]\n"

<< "\t->\tping [id]\n";

std::string command;

int id;

while( all\_good ) {

command.clear();

id = -1;

bool wait;

std::string reply;

bool got\_reply = true;

while( !got\_message || got\_reply ) {

got\_reply = node.get\_message( link, reply, wait = false );

if( got\_reply ) {

manage\_reply( reply, node, link );

reply.clear();

}

}

//I should remove ping when sending

//from the first worker or make ping replier in another thread

// std::cerr << NLink::parse\_message( message );

std::stringstream mssg( message );

if( wrong\_input() ) {

got\_message = false;

continue;

}

mssg >> command;

if( command == "exit" ) {

stop\_thread = true;

message\_handling.join();

break;

} else if( command == "w" ) { //wait for messages

continue;

}

mssg >> id;

if( wrong\_input( mssg ) ) {

got\_message = false;

continue;

}

if( command == "create" ) {

pid\_t pid = MNode::create\_worker( id );

node.command\_attach\_worker( link, id, pid );

} else if ( command == "exec" ) {

std::string time\_param;

mssg >> time\_param;

if( wrong\_input( mssg ) ) {

continue;

}

node.command\_exec( link, id, time\_param );

} else if ( command == "ping" ) {

node.command\_ping( link, id ); //even first could be broken tho'

} else {

std::cerr << "wrong command" << std::endl;

}

got\_message = false;

std::cerr << "\t> ";

}

//while-loop: create [id]; exec [id] [start|stop|time]; ping [id]; exit

//create -> create process and message working node with: create id pid ----------------new node and balance

//exec -> message working node with: exec id -----------------taking into account id

//ping ->

return 0;

}

void manage\_reply( const std::string& reply, MNode& node, int link ) {

MNode::Command command = MNode::get\_command( reply );

if( command == MNode::Command::instant\_ping ) {

node.send\_instant\_ping\_reply( reply );

} else if( command == MNode::Command::error\_c ) {

std::cerr << "initial worker is anavailable";

return;

} else if( command == MNode::Command::reply\_c ) {

std::cerr << MNode::parse\_reply( reply );

} else {

std::cerr << "received wrong message:" << reply << '\n';

}

}

timer.cpp:

#include "timer.hpp"

#include <ctime>

void Timer::start() {

if( timer\_is\_on ) return;

::time( &start\_time );

timer\_is\_on = true;

}

void Timer::stop() {

if( !timer\_is\_on ) return;

time\_t end\_time;

::time( &end\_time );

add\_time += difftime( end\_time, start\_time );

timer\_is\_on = false;

}

int Timer::time() const {

if( !timer\_is\_on ) return add\_time;

time\_t end\_time;

::time( &end\_time );

return add\_time + difftime( end\_time, start\_time );

}

timer.hpp:

#pragma once

#include <ctime>

class Timer {

public:

Timer() = default;

int time() const;

void stop();

void start();

private:

time\_t start\_time;

int add\_time = 0;

bool timer\_is\_on = false;

};

worker.cpp:

#include <sstream>

#include <sys/types.h>

#include <unistd.h>

#include <string>

#include "mlink.hpp"

#include "explicit\_enum.hpp"

#include "timer.hpp"

void manage\_command( MNode&, int, const std::string&, MNode::Command, int, int );

int main( int argc, const char\*\* argv ) {

int my\_id; // my left right

pid\_t pid = getpid();

Timer timer;

MNode node; //up, left, rigth

MNode::get\_id( my\_id, argv );

std::cerr << "worker " << pid << ": got id = " << my\_id << std::endl;

node.create\_link( Link::up, pid, my\_id );

bool all\_good = true;

std::string message;

while( all\_good ) {

node.get\_message( -1, message, true );

std::cerr << "worker " << pid << ": got message: " << message << std::endl;

MNode::Command command = MNode::get\_command( message );

if( command == MNode::Command::instant\_ping ) {

std::cerr << "worker " << pid << ": and it is an instant ping" << std::endl;

node.send\_instant\_ping\_reply( message );

continue;

} else if( command == MNode::Command::error\_c ) {

// I can go straight to the worker, which id would fit for attaching or find out, that sended id already exists

std::cerr << "worker " << pid << ": and it is an error" << std::endl;

std::cerr << "worker " << pid << my\_id << ": can't send message\n";

return ReturnValue::error\_unavailable\_node;

} else if( command == MNode::Command::reply\_c ) {

node.send\_message( Link::up, message );

continue;

}

int another\_id = MNode::get\_id( message );

if( another\_id > my\_id ) {

std::cerr << "worker " << pid << ": and it's to id = " << another\_id << std::endl;

manage\_command( node, Link::right, message, command, another\_id, my\_id );

} else if ( another\_id < my\_id ) {

std::cerr << "worker " << pid << ": and it's to id = " << another\_id << std::endl;

manage\_command( node, Link::left, message, command, another\_id, my\_id );

} else {

MNode::ExecCommand ecommand;

std::string reply;

switch( command ) {

case MNode::Command::attach\_c:

std::cerr << "and its attach with duplication of my id" << std::endl;

MNode::kill\_process( message );

node.send\_message( Link::up, MNode::make\_reply( MNode::error\_id\_already\_exist( my\_id ) ) );

break;

case MNode::Command::exec\_c:

std::cerr << "and its exec" << std::endl;

ecommand = MNode::get\_exec\_command( message );

if( ecommand == MNode::ExecCommand::time ) {

reply = MNode::exec\_reply\_message( timer.time(), my\_id, ecommand );

} else {

reply = MNode::exec\_reply\_message( my\_id, ecommand );

if( ecommand == MNode::ExecCommand::stop ) {

timer.stop();

} else if( ecommand == MNode::ExecCommand::start ) {

timer.start();

}

}

node.send\_message( Link::up, reply );

break;

case MNode::Command::ping\_c:

std::cerr << "and its ping" << std::endl;

reply = MNode::ping\_reply\_message( my\_id );

node.send\_message( Link::up, reply );

break;

case MNode::Command::close\_c:

std::cerr << "and its close" << std::endl;

all\_good = false;

break;

default:

std::cerr << "worker's command handler failed\n";

break;

}

}

}

return ReturnValue::nice;

}

void manage\_command( MNode& node, int link, const std::string& message, MNode::Command command, int another\_id, int my\_id ) {

if( node.exist( link ) ) {

std::cerr << "and I'm sending it to this id" << std::endl;

node.send\_message( link, message ); // check when sending is better overall

} else if( command == MNode::Command::attach\_c ) {

std::cerr << "and it's attach, so I will attach it" << std::endl;

node.attach\_worker( link, message ); //errors?

std::string reply = MNode::attach\_reply\_message( message, my\_id );

node.send\_message( Link::up, reply );

} else {

std::cerr << "worker id=" << my\_id << ": and it was an error, no one is attached and connection doesn't exist" << std::endl;

std::string error\_reply = MNode::make\_reply( MNode::error\_id\_doesnt\_exist( another\_id, my\_id ) );

node.send\_message( Link::up, error\_reply ); //errors?

}

}

**Пример работы**

steep@gg:~/education\_disk/instit/os/6lab/src$ ./server

worker 33551: got id = 0

id -1: creating consumer: <q:out:33551> in consumer\_tag[1][0]

id 0: creating consumer: <q:in:33551> in consumer\_tag[2][0]

id 0: creating consumer: <q:ping\_in:33551> in consumer\_tag[2][1]

id -1: creating consumer: <q:ping\_out:33551> in consumer\_tag[1][1]

id 0: consumers id 0: <q:in:33551>

id -1: sending message to id 0:"0 -1"

id 0: consumers id 0: <q:in:33551> id 0: <q:ping\_in:33551>

You can use:

-> create [id]

-> exec [id] [command]

-> ping [id]

id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

exec 0 time

id -1: sending message to id 0:"2 0 1"

id -1: sending message to id 0:"7 -1"

worker 33551: got message: 7 -1

worker 33551: and it is an instant ping

id 0: sending message to id -1:"8 -1"

id 0: consumers id 0: <q:in:33551> id 0: <q:ping\_in:33551>

> id worker -1: consumers33551 id : got message: -12 0 1: <q:out:33551

> id -1: <q:ping\_out:33551>

and its exec

id 0: sending message to id -1:"6 2 0 1 0 0"

id 0: sending message to id -1:"7 0"

id id -10: sending message to id : consumers0 id :"08 0: <"

q:ping\_in:33551>

id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

id 0: consumers id 0: <q:in:33551> id 0: <q:ping\_in:33551>

Ok: 0 time = 0

id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

exec 0 start

id -1: sending message to id 0:"2 0 2"

id -1: sending message to id 0:"7 -1"

worker 33551: got message: 7 -1

worker 33551: and it is an instant ping

id 0: sending message to id -1:"8 -1"

> id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

id 0: consumers id 0: <q:in:33551> id 0: <q:ping\_in:33551>

worker 33551: got message: 2 0 2

and its exec

id 0: sending message to id -1:"6 2 0 2 0"

id 0: sending message to id -1:"7 0"

id 0: consumers id 0: <q:ping\_in:33551>

id -1: sending message to id 0:"8 0"

id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

id 0: consumers id 0: <q:in:33551> id 0: <q:ping\_in:33551>

Ok: 0 timer started

id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

exec 0 time

id -1: sending message to id 0:"2 0 1"

id -1: sending message to id 0:"7 -1"

worker 33551: got message: 7 -1

worker 33551: and it is an instant ping

id 0: sending message to id -1:"8 -1"

id 0: consumers id 0: <q:in:33551> id 0: <q:ping\_in:33551>

worker 33551: got message: 2 0 1

and its exec

id 0: sending message to id -1:"6 2 0 1 8 0"

id 0: sending message to id -1:"7 0"

> id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

id 0: consumers id 0: <q:ping\_in:33551>

id -1: sending message to id 0:"8 0"

id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

id 0: consumers id 0: <q:in:33551> id 0: <q:ping\_in:33551>

Ok: 0 time = 8

id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

exec 0 stop

id -1: sending message to id 0:"2 0 0"

id -1: sending message to id 0:"7 -1"

worker 33551: got message: 7 -1

worker 33551: and it is an instant ping

id 0: sending message to id -1:"8 -1"

id 0: consumers id 0: <q:in:33551> id 0: <q:ping\_in:33551>

worker 33551: got message: 2 0 0

and its exec

id 0: sending message to id -1:"6 2 0 0 0"

id 0: sending message to id -1:"7 0"

> id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

id 0: consumers id 0: <q:ping\_in:33551>

id -1: sending message to id 0:"8 0"

id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

id 0: consumers id 0: <q:in:33551> id 0: <q:ping\_in:33551>

Ok: 0 timer stopped

id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

ping 0

id -1: sending message to id 0:"3 0"

id -1: sending message to id 0:"7 -1"

worker 33551: got message: 7 -1

worker 33551: and it is an instant ping

id 0: sending message to id -1:"8 -1"

id 0: consumers id 0: <q:in:33551> id 0: <q:ping\_in:33551>

worker 33551: got message: 3 0

and its ping

id 0: sending message to id -1:"6 3 0 0"

id 0: sending message to id -1:"7 0"

> id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

id 0: consumers id 0: <q:ping\_in:33551>

id -1: sending message to id 0:"8 0"

id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

Ok: worker with id = 0 is available

id -1: consumers id -1: <q:out:33551> id -1: <q:ping\_out:33551>

id 0: consumers id 0: <q:in:33551> id 0: <q:ping\_in:33551>

exit

id -1: sending message to id 0:"4 0"

worker 33551: got message: 4 0

and its close

steep@gg:~/education\_disk/instit/os/6lab/src$ id 0->-1 start deleting queue...success;

id 0->-1 start deleting queue...success;

id 0->-1 start deleting queue...success;

id 0->-1 start deleting queue...success;

**Вывод**

Для реализации взаимодействия между процессами можно использовать технологию message queue. Она является очень гибкой и простой в использовании, её лёгко встроить в приложение, а для работы нужен лишь запущенный сервер сообщений.