Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторные работы №6-8 по курсу**

**«Операционные системы»**

**Управление серверами сообщений. Применение отложенных вычислений. Интеграция программных систем друг с другом.**

Студент: Мариничев И. А.

Группа: М8О-208Б-19

Преподаватель: Миронов Е. С.

Дата: 18.04.2021

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

1. **Постановка задачи. Вариант 7**

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 топологией, согласно варианту это список (т. е. от управляющего узла исходит несколько списков вычислительных узлов). Связь между узлами необходимо осуществить при помощи технологии очередей сообщений. В данной системе необходимо предусмотреть проверку доступности узлов. При убийстве 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.

1. **Общие сведения о программе**

Программа написана на языке С++ в UNIX-подобной операционной системе (Ubuntu). В программе используется очередь сообщений ZeroMQ, а также более высокоуровневая библиотека cppzmq.

Программа поддерживает следующие команды:

* create [id] [parent\_id] – создать новый узел [id], родителем которого является узел [parent\_id]. Если [parent\_id] = -1, то родительский узел – управляющий.
* kill [id] – удалить узел [id]. Все дочерние узлы будут также удалены.
* exec [id] [subcommand] – работа с локальным таймером, где [subcommand]:
  + start – запустить таймер
  + stop – остановить таймер
  + time – показать время локального таймера в миллисекундах
* pingall – проверить доступность узлов. Будет выведен список всех доступных на данный момент узлов.
* exit – выйти из программы.

1. **Общий метод и алгоритм решения**

В программе используется тип соединения Request-Response. Узлы передают информацию друг другу при помощи очереди сообщений. Все сообщения имеют следующий вид:

[id узла, которому предназначено сообщение] [команда] [аргументы]

Управляющий узел хранит структуру «список списков», в которую записывает id существующих узлов. При помощи этой структуры он определяет, в какой список нужно направить сообщение.

Вычислительный узел, получив сообщение, сравнивает свой id и id из сообщения. Если они совпадают, то узел начинает обрабатывать запрос, в противном случае узел направляет это же сообщение своему ребенку и ждет от него ответа.

Для удобства функции отправки и получения сообщений, а также функции для подключения к сокетам вынесены в отдельный заголовочный файл, который подключается к программам узлов.

Для работы с локальным таймером используется библиотека chrono (std::chrono::high\_resolution\_clock). Для проверки доступности узлов используется контейнер std::set. Управляющий узел отправляет запрос всем спискам узлов и получает в ответ строку с id всех доступных узлов списка. Все id добавляются в set, а потом выводятся на экран.

1. **Основные файлы программы**

**zmq\_server.hpp**

#include <zmq.hpp>

#include <iostream>

const int MAIN\_PORT = 4040;

void send\_message(zmq::socket\_t& socket, const std::string& msg) {

    zmq::message\_t message(msg.size());

    memcpy(message.data(), msg.c\_str(), msg.size());

    socket.send(message);

}

std::string receive\_message(zmq::socket\_t& socket) {

    zmq::message\_t message;

    int chars\_read;

    try {

        chars\_read = (int)socket.recv(&message);

    }

    catch (...) {

        chars\_read = 0;

    }

    if (chars\_read == 0) {

        return "Error: Node is not available";

    }

    std::string received\_msg(static\_cast<char\*>(message.data()), message.size());

    return received\_msg;

}

// connect client to the socket

void connect(zmq::socket\_t& socket, int id) { // note: 127.0.0.1 is an address for machine to connect to itself via network

    std::string address = "tcp://127.0.0.1:" + std::to\_string(MAIN\_PORT + id);

    socket.connect(address);

}

// disconnect client from the socket

void disconnect(zmq::socket\_t& socket, int id) {

    std::string address = "tcp://127.0.0.1:" + std::to\_string(MAIN\_PORT + id);

    socket.disconnect(address);

}

// connect socket to ip address and port and start accepting connections

void bind(zmq::socket\_t& socket, int id) {

    std::string address = "tcp://127.0.0.1:" + std::to\_string(MAIN\_PORT + id);

    socket.bind(address);

}

// disconnect socket from ip address and port and stop accepting connections

void unbind(zmq::socket\_t& socket, int id) {

    std::string address = "tcp://127.0.0.1:" + std::to\_string(MAIN\_PORT + id);

    socket.unbind(address);

}

**topology.hpp**

#include <list>

#include <stdexcept>

class topology {

private:

    std::list<std::list<int>> container;

public:

    void insert(int id, int parent\_id) {

        if (parent\_id == -1) { // if we were asked to create root node

            std::list<int> new\_list;

            new\_list.push\_back(id);

            container.push\_back(new\_list);

        }

        else {

            int list\_id = find(parent\_id); // trying to find parent element first

            if (list\_id == -1) { // if there is no such element with this id

                throw std::runtime\_error("Wrong parent id");

            }

            auto it1 = container.begin();

            std::advance(it1, list\_id); // set iterator to the position of parent list

            for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) { // for loop through the list

                if (\*it2 == parent\_id) { // until we find the parent

                    it1->insert(++it2, id); // make id a chid of parent\_id

                    return;

                }

            }

        }

    }

    int find(int id) {

        int cur\_list\_id = 0;

        for (auto it1 = container.begin(); it1 != container.end(); ++it1) { // for loop through all lists of root

            for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) { // for loop through all elments of current list

                if (\*it2 == id) {

                    return cur\_list\_id;

                }

            }

            ++cur\_list\_id;

        }

        return -1;

    }

    void erase(int id) {

        int list\_id = find(id);

        if (list\_id == -1) {

            throw std::runtime\_error("Wrong id");

        }

        auto it1 = container.begin();

        std::advance(it1, list\_id);

        for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {

            if (\*it2 == id) {

                it1->erase(it2, it1->end());

                if (it1->empty()) { // if we erased the head of list

                    container.erase(it1); // remove this list

                }

                return;

            }

        }

    }

    int get\_first\_id(int list\_id) { // additional method to find pointer to specified list

        auto it1 = container.begin();

        std::advance(it1, list\_id); // set iterator to the list we're looking for

        if (it1->begin() == it1->end()) { // if we reached the end it means it doesn't exist

            return -1;

        }

        return \*(it1->begin());

    }

};

**zmq\_client.cpp**

#include <unistd.h>

#include <sstream>

#include <set>

#include <chrono>

#include "zmq\_server.hpp"

#include "topology.hpp"

int main() {

    topology network;

    std::vector<zmq::socket\_t> branches;

    zmq::context\_t context;

    std::string cmd;

    while (std::cin >> cmd) {

        if (cmd == "create") {

            int node\_id, parent\_id;

            std::cin >> node\_id >> parent\_id;

            if (network.find(node\_id) != -1) {

                std::cout << "Error: already exists" << std::endl;

            }

            else if (parent\_id == -1) { //if we're creating root node

                pid\_t pid = fork(); //create two processes

                if (pid < 0) { // if parent process noticed problem with creating child process

                    perror("Can't create new process");

                    return -1;

                }

                if (pid == 0) { // child process goes here

                    // replaces the current process image with a new process image

                    execl("./computing\_node", "./computing\_node", std::to\_string(node\_id).c\_str(), NULL);

                    perror("Can't execute new process");

                    return -2;

                }

                branches.emplace\_back(context, ZMQ\_REQ);

                branches[branches.size() - 1].setsockopt(ZMQ\_SNDTIMEO, 5000);

                bind(branches[branches.size() - 1], node\_id);

                send\_message(branches[branches.size() - 1], std::to\_string(node\_id) + "pid");

                std::string reply = receive\_message(branches[branches.size() - 1]);

                std::cout << reply << std::endl;

                network.insert(node\_id, parent\_id);

            }

            else if (network.find(parent\_id) == -1) {

                std::cout << "Error: parent not found" << std::endl;

            }

            else { // if we're creating a child node from existing parent node

                int branch = network.find(parent\_id);

                send\_message(branches[branch], std::to\_string(parent\_id) + "create " + std::to\_string(node\_id));

                std::string reply = receive\_message(branches[branch]);

                std::cout << reply << std::endl;

                network.insert(node\_id, parent\_id);

            }

        }

        else if (cmd == "exec") {

            int dest\_id;

            std::string subcomand;

            std::cin >> dest\_id >> subcomand;

            int branch = network.find(dest\_id);

            if (branch == -1) {

                std::cout << "ERROR: incorrect node id" << std::endl;

            }

            else {

                if (subcomand == "start") {

                    send\_message(branches[branch], std::to\_string(dest\_id)  + "exec " + " start");

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

                    send\_message(branches[branch], std::to\_string(dest\_id)  +  "exec " + " stop");

                } else if (subcomand == "time") {

                    send\_message(branches[branch], std::to\_string(dest\_id)  + "exec " + " time");

                }

                std::string reply = receive\_message(branches[branch]);

                std::cout << reply << std::endl;

            }

        }

        else if (cmd == "kill") {

            int id;

            std::cin >> id;

            int branch = network.find(id);

            if (branch == -1) {

                std::cout << "ERROR: incorrect node id" << std::endl;

            }

            else {

                bool is\_first = (network.get\_first\_id(branch) == id);

                send\_message(branches[branch], std::to\_string(id) + " kill");

                std::string reply = receive\_message(branches[branch]);

                std::cout << reply << std::endl;

                network.erase(id);

                if (is\_first) { // check if id node is the first in the list

                    unbind(branches[branch], id);

                    branches.erase(branches.begin() + branch);

                }

            }

        }

        else if (cmd == "pingall") {

            std::set<int> available\_nodes;

            for (size\_t i = 0; i < branches.size(); ++i) {

                int first\_node\_id = network.get\_first\_id(i);

                send\_message(branches[i], std::to\_string(first\_node\_id) + " pingall");

                std::string received\_message = receive\_message(branches[i]);

                std::istringstream reply(received\_message);

                int node;

                while(reply >> node) {

                    available\_nodes.insert(node);

                }

            }

            std::cout << "OK: ";

            if (available\_nodes.empty()) {

                std::cout << "no available nodes" << std::endl;

            }

            else {

                for (auto v : available\_nodes) {

                    std::cout << v << " ";

                }

                std::cout << std::endl;

            }

        }

        else if (cmd == "exit") {

            for (size\_t i = 0; i < branches.size(); ++i) {

                int first\_node\_id = network.get\_first\_id(i);

                send\_message(branches[i], std::to\_string(first\_node\_id) + " kill");

                std::string reply = receive\_message(branches[i]);

                if (reply != "OK") {

                    std::cout << reply << std::endl;

                }

                else {

                    unbind(branches[i], first\_node\_id);

                }

            }

            exit(0);

        }

        else {

            std::cout << "I don't know this command" << std::endl;

        }

    }

}

**computing\_node.cpp**

#include <unistd.h>

#include <sstream>

#include <chrono>

#include "zmq\_server.hpp"

int main(int argc, char\* argv[]) {

    if (argc != 2 && argc != 3) {

        throw std::runtime\_error("Wrong amount of arguments for computing node");

    }

    int cur\_id = std::atoi(argv[1]);

    int child\_id = -1; // by default we look from the root node

    if (argc == 3) {

        child\_id = std::atoi(argv[2]);

    }

    auto start\_clock = std::chrono::high\_resolution\_clock::now();

    auto stop\_clock = std::chrono::high\_resolution\_clock::now();

    auto time\_clock = 0;

    bool flag\_clock = false;

    zmq::context\_t context;

    zmq::socket\_t parent\_socket(context, ZMQ\_REP);

    connect(parent\_socket, cur\_id);

    zmq::socket\_t child\_socket(context, ZMQ\_REQ);

    child\_socket.setsockopt(ZMQ\_SNDTIMEO, 5000);

    if (child\_id != -1) {

        bind(child\_socket, child\_id);

    }

    std::string message;

    while (true) {

        message = receive\_message(parent\_socket);

        std::istringstream request(message);

        int dest\_id;

        request >> dest\_id;

        std::string cmd;

        request >> cmd;

        if (dest\_id == cur\_id) {

            if (cmd == "pid") { //if we need to get process id for the created node

                send\_message(parent\_socket, "OK: " + std::to\_string(getpid()));

            }

            else if (cmd == "create") {

                int new\_child\_id;

                request >> new\_child\_id;

                if (child\_id != -1) {

                    unbind(child\_socket, child\_id);

                }

                bind(child\_socket, new\_child\_id);

                pid\_t pid = fork(); //create two processes

                if (pid < 0) { // if parent process noticed problem with creating child process

                    perror("Can't create new process");

                    return -1;

                }

                if (pid == 0) { // child process goes here

                    // replaces the current process image with a new process image

                    execl("./computing\_node", "./computing\_node", std::to\_string(new\_child\_id).c\_str(), std::to\_string(child\_id).c\_str(), NULL);

                    perror("Can't execute new process");

                    return -2;

                }

                send\_message(child\_socket, std::to\_string(new\_child\_id) + "pid");

                child\_id = new\_child\_id;

                send\_message(parent\_socket, receive\_message(child\_socket));

            }

            else if (cmd == "exec") {

                std::string subcomand;

                request >> subcomand;

                std::string msg = "OK: " + std::to\_string(cur\_id);

                if (subcomand == "start") {

                    start\_clock = std::chrono::high\_resolution\_clock::now();

                    flag\_clock = true;

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

                    if (flag\_clock) {

                        stop\_clock = std::chrono::high\_resolution\_clock::now();

                        time\_clock += std::chrono::duration\_cast<std::chrono::milliseconds>(stop\_clock - start\_clock).count();

                        flag\_clock = false;

                    }

                } else if (subcomand == "time") {

                    if (flag\_clock == true) {

                        stop\_clock = std::chrono::high\_resolution\_clock::now();

                        time\_clock += std::chrono::duration\_cast<std::chrono::milliseconds>(stop\_clock - start\_clock).count();

                        start\_clock = stop\_clock;

                    }

                    msg += ": " + std::to\_string(time\_clock);

                }

                send\_message(parent\_socket, msg);

            }

            else if (cmd == "pingall") {

                std::string reply;

                if (child\_id != -1) {

                    send\_message(child\_socket, std::to\_string(child\_id) + " pingall");

                    std::string msg = receive\_message(child\_socket);

                    reply += " " + msg;

                }

                send\_message(parent\_socket, std::to\_string(cur\_id) + reply);

            }

            else if (cmd == "kill") {

                if (child\_id != -1) {

                    send\_message(child\_socket, std::to\_string(child\_id) + " kill");

                    std::string msg = receive\_message(child\_socket);

                    if (msg == "OK") {

                        send\_message(parent\_socket, "OK");

                    }

                    unbind(child\_socket, child\_id);

                    disconnect(parent\_socket, cur\_id);

                    break;

                }

                send\_message(parent\_socket, "OK");

                disconnect(parent\_socket, cur\_id);

                break;

            }

        }

        else if (child\_id != -1) {

            send\_message(child\_socket, message);

            send\_message(parent\_socket, receive\_message(child\_socket));

            if (child\_id == dest\_id && cmd == "kill") {

                child\_id = -1;

            }

        }

        else {

            send\_message(parent\_socket, "Error: node is unavailable");

        }

    }

}

**Makefile**

all: client computing\_node

client:

    g++ zmq\_client.cpp -lzmq -o client -Wall -pedantic

computing\_node:

    g++ computing\_node.cpp -lzmq -o computing\_node -Wall -pedantic

clean:

    rm -rf client computing\_node

1. **Демонстрация работы программы**

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab6-8$** ./client

create 1 -1

OK: 28467

create 2 1

OK: 28472

create 3 2

OK: 28475

pingall

OK: 1 2 3

create 20 10

Error: parent not found

pingall

OK: 1 2 3

create 4 2

OK: 28478

pingall

OK: 1 2 3 4

exec 4 time

OK: 4: 0

exec 4 start

OK: 4

exec 4 time

OK: 4: 7686

exec 4 time

OK: 4: 14093

exec 4 stop

OK: 4

exec 4 time

OK: 4: 20853

exec 4 time

OK: 4: 20853

kill 2

OK

pingall

OK: 1

kill 2

ERROR: incorrect node id

kill 1

OK

pingall

OK: no available nodes

exit

1. **Strace**

Утилита, отслеживающая системные вызовы, которые являются механизмом трансляции, обеспечивающим взаимодействие между процессом и операционной системой. Эти вызовы могут быть перехвачены и прочитаны, что позволяет лучше понять, какую задачу процесс пытается выполнить в заданное время. Перехватывая эти вызовы, мы можем добиться лучшего понимания поведения процессов, особенно если что-то пошло не так. Команда ОС, позволяющая отслеживать системны вызовы, называется ptrace. Strace же вызывает ptrace и читает данные о поведении процесса, а затем выводит отчет.

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab6-8$** strace -o strace\_log.txt ./client

create 1 -1

OK: 28484

exit

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab6-8$** cat strace\_log.txt

execve("./client", ["./client"], 0x7ffff166da70 /\* 28 vars \*/) = 0

brk(NULL) = 0x7ffff3fda000

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=61725, ...}) = 0

mmap(NULL, 61725, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f9627e7a000

close(3) = 0

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/lib/libzmq.so.5", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\276\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=6400680, ...}) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f9627e70000

mmap(NULL, 2483240, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f96279a1000

mprotect(0x7f96279fb000, 2097152, PROT\_NONE) = 0

mmap(0x7f9627bfb000, 20480, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x5a000) = 0x7f9627bfb000

close(3) = 0

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/lib/x86\_64-linux-gnu/libstdc++.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\304\10\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=1594864, ...}) = 0

mmap(NULL, 3702848, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f9627610000

mprotect(0x7f9627789000, 2097152, PROT\_NONE) = 0

mmap(0x7f9627989000, 49152, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x179000) = 0x7f9627989000

mmap(0x7f9627995000, 12352, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f9627995000

close(3) = 0

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libgcc\_s.so.1", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\300\*\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=96616, ...}) = 0

mmap(NULL, 2192432, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f96273f0000

mprotect(0x7f9627407000, 2093056, PROT\_NONE) = 0

mmap(0x7f9627606000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x16000) = 0x7f9627606000

close(3) = 0

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20\35\2\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2030928, ...}) = 0

mmap(NULL, 4131552, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f9626ff0000

mprotect(0x7f96271d7000, 2097152, PROT\_NONE) = 0

mmap(0x7f96273d7000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7f96273d7000

mmap(0x7f96273dd000, 15072, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f96273dd000

close(3) = 0

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/librt.so.1", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\"\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=31680, ...}) = 0

mmap(NULL, 2128864, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f9626de0000

mprotect(0x7f9626de7000, 2093056, PROT\_NONE) = 0

mmap(0x7f9626fe6000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x6000) = 0x7f9626fe6000

close(3) = 0

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libpthread.so.0", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000b\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=144976, ...}) = 0

mmap(NULL, 2221184, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f9626bc0000

mprotect(0x7f9626bda000, 2093056, PROT\_NONE) = 0

mmap(0x7f9626dd9000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19000) = 0x7f9626dd9000

mmap(0x7f9626ddb000, 13440, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f9626ddb000

close(3) = 0

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libm.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\200\272\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=1700792, ...}) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f9627e60000

mmap(NULL, 3789144, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f9626820000

mprotect(0x7f96269bd000, 2093056, PROT\_NONE) = 0

mmap(0x7f9626bbc000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19c000) = 0x7f9626bbc000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f9627e50000

arch\_prctl(ARCH\_SET\_FS, 0x7f9627e50740) = 0

mprotect(0x7f96273d7000, 16384, PROT\_READ) = 0

mprotect(0x7f9626bbc000, 4096, PROT\_READ) = 0

mprotect(0x7f9626dd9000, 4096, PROT\_READ) = 0

mprotect(0x7f9626fe6000, 4096, PROT\_READ) = 0

mprotect(0x7f9627606000, 4096, PROT\_READ) = 0

mprotect(0x7f9627989000, 40960, PROT\_READ) = 0

mprotect(0x7f9627bfb000, 16384, PROT\_READ) = 0

mprotect(0x7f962820d000, 4096, PROT\_READ) = 0

mprotect(0x7f9627e29000, 4096, PROT\_READ) = 0

munmap(0x7f9627e7a000, 61725) = 0

set\_tid\_address(0x7f9627e50a10) = 28483

set\_robust\_list(0x7f9627e50a20, 24) = 0

rt\_sigaction(SIGRTMIN, {sa\_handler=0x7f9626bc5cb0, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_SIGINFO, sa\_restorer=0x7f9626bd2980}, NULL, 8) = 0

rt\_sigaction(SIGRT\_1, {sa\_handler=0x7f9626bc5d50, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_RESTART|SA\_SIGINFO, sa\_restorer=0x7f9626bd2980}, NULL, 8) = 0

rt\_sigprocmask(SIG\_UNBLOCK, [RTMIN RT\_1], NULL, 8) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=8192\*1024}) = 0

brk(NULL) = 0x7ffff3fda000

brk(0x7ffff3ffb000) = 0x7ffff3ffb000

futex(0x7f962799609c, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0

futex(0x7f96279960a8, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0

eventfd2(0, 0) = 3

fcntl(3, F\_GETFL) = 0x2 (flags O\_RDWR)

fcntl(3, F\_SETFL, O\_RDWR|O\_NONBLOCK) = 0

fcntl(3, F\_GETFL) = 0x802 (flags O\_RDWR|O\_NONBLOCK)

fcntl(3, F\_SETFL, O\_RDWR|O\_NONBLOCK) = 0

fstat(0, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(136, 1), ...}) = 0

read(0, "create 1 -1\n", 1024) = 12

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f9627e50a10) = 28484

eventfd2(0, 0) = 4

fcntl(4, F\_GETFL) = 0x2 (flags O\_RDWR)

fcntl(4, F\_SETFL, O\_RDWR|O\_NONBLOCK) = 0

fcntl(4, F\_GETFL) = 0x802 (flags O\_RDWR|O\_NONBLOCK)

fcntl(4, F\_SETFL, O\_RDWR|O\_NONBLOCK) = 0

clock\_gettime(CLOCK\_MONOTONIC, {tv\_sec=104208, tv\_nsec=341386100}) = 0

epoll\_create(1) = 5

epoll\_ctl(5, EPOLL\_CTL\_ADD, 4, {0, {u32=4093569408, u64=140737286957440}}) = 0

epoll\_ctl(5, EPOLL\_CTL\_MOD, 4, {EPOLLIN, {u32=4093569408, u64=140737286957440}}) = 0

mmap(NULL, 8392704, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_STACK, -1, 0) = 0x7f9626010000

mprotect(0x7f9626011000, 8388608, PROT\_READ|PROT\_WRITE) = 0

clone(child\_stack=0x7f962680ffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7f96268109d0, tls=0x7f9626810700, child\_tidptr=0x7f96268109d0) = 28485

sched\_getparam(28485, [0]) = 0

sched\_getscheduler(28485) = 0 (SCHED\_OTHER)

sched\_setscheduler(28485, SCHED\_OTHER, [0]) = 0

eventfd2(0, 0) = 6

fcntl(6, F\_GETFL) = 0x2 (flags O\_RDWR)

fcntl(6, F\_SETFL, O\_RDWR|O\_NONBLOCK) = 0

fcntl(6, F\_GETFL) = 0x802 (flags O\_RDWR|O\_NONBLOCK)

fcntl(6, F\_SETFL, O\_RDWR|O\_NONBLOCK) = 0

clock\_gettime(CLOCK\_MONOTONIC, {tv\_sec=104208, tv\_nsec=343305400}) = 0

epoll\_create(1) = 7

epoll\_ctl(7, EPOLL\_CTL\_ADD, 6, {0, {u32=4093570528, u64=140737286958560}}) = 0

epoll\_ctl(7, EPOLL\_CTL\_MOD, 6, {EPOLLIN, {u32=4093570528, u64=140737286958560}}) = 0

mmap(NULL, 8392704, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_STACK, -1, 0) = 0x7f9625800000

mprotect(0x7f9625801000, 8388608, PROT\_READ|PROT\_WRITE) = 0

clone(child\_stack=0x7f9625ffffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7f96260009d0, tls=0x7f9626000700, child\_tidptr=0x7f96260009d0) = 28486

sched\_getparam(28486, [0]) = 0

sched\_getscheduler(28486) = 0 (SCHED\_OTHER)

sched\_setscheduler(28486, SCHED\_OTHER, [0]) = 0

eventfd2(0, 0) = 8

fcntl(8, F\_GETFL) = 0x2 (flags O\_RDWR)

fcntl(8, F\_SETFL, O\_RDWR|O\_NONBLOCK) = 0

fcntl(8, F\_GETFL) = 0x802 (flags O\_RDWR|O\_NONBLOCK)

fcntl(8, F\_SETFL, O\_RDWR|O\_NONBLOCK) = 0

clock\_gettime(CLOCK\_MONOTONIC, {tv\_sec=104208, tv\_nsec=345453300}) = 0

poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

**socket(AF\_NETLINK, SOCK\_RAW|SOCK\_CLOEXEC, NETLINK\_ROUTE) = 9**

**bind(9, {sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, 12) = 0**

**getsockname(9, {sa\_family=AF\_NETLINK, nl\_pid=28483, nl\_groups=00000000}, [12]) = 0**

**time(NULL) = 1618761332 (2021-04-18T18:55:32+0300)**

**sendto(9, "\24\0\0\0\22\0\1\3tV|`\0\0\0\0\0\0\0\0", 20, 0, {sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, 12) = 20**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=80, type=0x10 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761332, pid=28483}, "\x00\x00\x01\x00\x08\x00\x00\x00\x40\x00\x00\x00\x00\x00\x00\x00\x0a\x00\x01\x00\x04\xea\x56\x54\x04\xec\x00\x00\x08\x00\x1b\x00"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 80**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=76, type=0x10 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761332, pid=28483}, "\x00\x00\x04\x03\x01\x00\x00\x00\x49\x00\x00\x00\x00\x00\x00\x00\x0a\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x08\x00\x1b\x00"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 76**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=80, type=0x10 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761332, pid=28483}, "\x00\x00\x21\x03\x14\x00\x00\x00\x43\x10\x00\x00\x00\x00\x00\x00\x0a\x00\x01\x00\x04\xea\x56\x54\x04\xe8\x00\x00\x08\x00\x1b\x00"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 80**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=80, type=0x10 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761332, pid=28483}, "\x00\x00\x21\x03\x0e\x00\x00\x00\x40\x00\x00\x00\x00\x00\x00\x00\x0a\x00\x01\x00\x04\xea\x56\x54\x04\xe9\x00\x00\x08\x00\x1b\x00"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 80**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=80, type=0x10 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761332, pid=28483}, "\x00\x00\x21\x03\x12\x00\x00\x00\x40\x00\x00\x00\x00\x00\x00\x00\x0a\x00\x01\x00\x06\xea\x56\x54\x04\xe8\x00\x00\x08\x00\x1b\x00"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 80**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=20, type=NLMSG\_DONE, flags=NLM\_F\_MULTI, seq=1618761332, pid=28483}, 0}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 20**

**sendto(9, "\24\0\0\0\26\0\1\3uV|`\0\0\0\0\0\0\0\0", 20, 0, {sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, 12) = 20**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=60, type=0x14 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761333, pid=28483}, "\x02\x10\x00\x00\x08\x00\x00\x00\x08\x00\x01\x00\xa9\xfe\x97\x1c\x08\x00\x04\x00\xa9\xfe\xff\xff\x14\x00\x06\x00\xff\xff\xff\xff"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 60**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=64, type=0x14 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761333, pid=28483}, "\x0a\x40\x00\xfd\x08\x00\x00\x00\x14\x00\x01\x00\xfe\x80\x00\x00\x00\x00\x00\x00\xe9\x3c\x94\x41\x9e\x24\x97\x1c\x14\x00\x06\x00"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 64**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=60, type=0x14 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761333, pid=28483}, "\x02\x08\x00\x00\x01\x00\x00\x00\x08\x00\x01\x00\x7f\x00\x00\x01\x08\x00\x04\x00\x7f\xff\xff\xff\x14\x00\x06\x00\xff\xff\xff\xff"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 60**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=64, type=0x14 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761333, pid=28483}, "\x0a\x80\x00\xfe\x01\x00\x00\x00\x14\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x14\x00\x06\x00"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 64**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=60, type=0x14 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761333, pid=28483}, "\x02\x18\x00\x00\x14\x00\x00\x00\x08\x00\x01\x00\xc0\xa8\x64\x07\x08\x00\x04\x00\xc0\xa8\x64\xff\x14\x00\x06\x00\x51\xca\x03\x00"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 60**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=64, type=0x14 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761333, pid=28483}, "\x0a\x40\x00\xfd\x14\x00\x00\x00\x14\x00\x01\x00\xfe\x80\x00\x00\x00\x00\x00\x00\x25\x20\xf5\x7e\xe3\x5d\x64\x5b\x14\x00\x06\x00"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 64**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=60, type=0x14 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761333, pid=28483}, "\x02\x10\x00\x00\x0e\x00\x00\x00\x08\x00\x01\x00\xa9\xfe\xee\x85\x08\x00\x04\x00\xa9\xfe\xff\xff\x14\x00\x06\x00\xff\xff\xff\xff"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 60**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=64, type=0x14 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761333, pid=28483}, "\x0a\x40\x00\xfd\x0e\x00\x00\x00\x14\x00\x01\x00\xfe\x80\x00\x00\x00\x00\x00\x00\x99\x07\xe3\x03\xc8\xb1\xee\x85\x14\x00\x06\x00"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 64**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=60, type=0x14 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761333, pid=28483}, "\x02\x10\x00\x00\x12\x00\x00\x00\x08\x00\x01\x00\xa9\xfe\xc3\x47\x08\x00\x04\x00\xa9\xfe\xff\xff\x14\x00\x06\x00\xff\xff\xff\xff"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 60**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=64, type=0x14 /\* NLMSG\_??? \*/, flags=NLM\_F\_MULTI, seq=1618761333, pid=28483}, "\x0a\x40\x00\xfd\x12\x00\x00\x00\x14\x00\x01\x00\xfe\x80\x00\x00\x00\x00\x00\x00\x61\x44\xae\x0e\x49\x63\xc3\x47\x14\x00\x06\x00"...}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 64**

**recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base={{len=20, type=NLMSG\_DONE, flags=NLM\_F\_MULTI, seq=1618761333, pid=28483}, 0}, iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 20**

**close(9) = 0**

**socket(AF\_INET, SOCK\_STREAM|SOCK\_CLOEXEC, IPPROTO\_TCP) = 9**

**setsockopt(9, SOL\_SOCKET, SO\_REUSEADDR, [1], 4) = 0**

**bind(9, {sa\_family=AF\_INET, sin\_port=htons(4041), sin\_addr=inet\_addr("127.0.0.1")}, 16) = 0**

**listen(9, 100) = 0**

**getsockname(9, {sa\_family=AF\_INET, sin\_port=htons(4041), sin\_addr=inet\_addr("127.0.0.1")}, [128->16]) = 0**

write(6, "\1\0\0\0\0\0\0\0", 8) = 8

write(8, "\1\0\0\0\0\0\0\0", 8) = 8

poll([{fd=8, events=POLLIN}], 1, 0) = 1 ([{fd=8, revents=POLLIN}])

read(8, "\1\0\0\0\0\0\0\0", 8) = 8

poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

clock\_gettime(CLOCK\_MONOTONIC, {tv\_sec=104208, tv\_nsec=354235100}) = 0

poll([{fd=8, events=POLLIN}], 1, 5000) = 1 ([{fd=8, revents=POLLIN}])

read(8, "\1\0\0\0\0\0\0\0", 8) = 8

poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

write(6, "\1\0\0\0\0\0\0\0", 8) = 8

poll([{fd=8, events=POLLIN}], 1, -1) = 1 ([{fd=8, revents=POLLIN}])

read(8, "\1\0\0\0\0\0\0\0", 8) = 8

poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

write(6, "\1\0\0\0\0\0\0\0", 8) = 8

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(136, 1), ...}) = 0

write(1, "OK: 28484\n", 10) = 10

read(0, "exit\n", 1024) = 5

poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

write(6, "\1\0\0\0\0\0\0\0", 8) = 8

poll([{fd=8, events=POLLIN}], 1, -1) = 1 ([{fd=8, revents=POLLIN}])

read(8, "\1\0\0\0\0\0\0\0", 8) = 8

poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

poll([{fd=8, events=POLLIN}], 1, 0) = ? ERESTART\_RESTARTBLOCK (Interrupted by signal)

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=28484, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

restart\_syscall(<... resuming interrupted poll ...>) = 0

write(6, "\1\0\0\0\0\0\0\0", 8) = 8

lseek(0, -1, SEEK\_CUR) = -1 ESPIPE (Illegal seek)

exit\_group(0) = ?

+++ exited with 0 +++

1. **Выводы**

В ходе данной лабораторной работы я изучил технологию очереди сообщений, на основе которой построил сеть с заданной топологией.

Наряду с каналами и отображаемыми файлами, очереди сообщений являются достаточно удобным способом для взаимодействия между процессами. ZeroMQ предоставляет достаточно простой интерфейс для передачи сообщений, а также поддерживает все возможные типы соединений. В совокупности с библиотекой cppzmq, позволяющей упростить процесс написания кода, является отличной альтернативой другим серверам сообщений (MSMQ, RabbitMQ, Nats).

Навык работы с очередями сообщений является актуальным и необходимым, т. к. данная технология довольно часто встречается на практике при работе с мультипроцессорными программами, в частности при создании мессенджеров или, например, чатов в многопользовательских играх.