Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу

«Операционные системы»

Студент: Павлов Иван Дмитриевич
Группа: М8О-207Б-21
Вариант: 16
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/Pavloffff/MAI_OS/tree/main/lab6

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])
- **Исполнение команды на вычислительном узле** (Формат команды: exec id [params])
- **Проверка доступности узла** (Формат команды: ping id)
- **Удаление узла** (Формат команды remove id)

Вариант №16: топология — дерево общего вида, команда — работа с локальным словарем, проверка доступности — pingall.

Общие сведения о программе

Программа состоит из 4 файлов: client.cpp — интерфейс для общения с пользователем, apidemon.cpp — перенаправление запросов клиента на сервер и дальнейшая обработка (контрольный узел), server.cpp — исполнение команд (вычислительный узел), labtools.h — класс топологии, управление сообщениями и разные константы.

Общий метод и алгоритм решения

Связь между узлами организована с помощью zmq::socket_type::pair. Клиент асинхронно отправляет запросы в виде json-файлов на демона, который напрямую общается с сервером. При запуске программы контрольный узел уже существует на демоне, топология «дерево общего вида» лежит там же. Сам демон просто отправляет запросы на сервер. Процессов server существует столько же, сколько вычислительных узлов в топологии. При получении запроса сервер либо сам обрабатывает запрос, либо отсылает вниз по топологии (на каждом узле есть вектор сокетов для связи с дочерними узлами). Далее все команды реализованы в соответствии с ТЗ.

Исходный код

labtools.h

```
#ifndef __LABTOOLS_H__
#define __LABTOOLS_H__

#include <iostream>
#include <map>
#include <vector>
#include <set>
#include <zmq.hpp>
#include <nlohmann/json.hpp>

const int WAIT_TIME = 1000;
const int PORT = 7000;
const int DEMON_PORT = 6900;

class NTree
{
public:
```

```
    using Node = std::map<int, std::set<int>>;
    Node node;
    NTree();
    void print();
    int dfs(int child, int curChild);
    int findCheck(int parent, int child);
    int find(int parent, int child);
    std::pair<int, int> findNode(int node);
    std::vector<int> findChilds(int node);
    int insert(int parent, int child);
    int erase(int parent, int child);
    void destroyUndertree(int node);
    ~NTree();
};

NTree::NTree()
{
    std::set<int> root;
    this->node.insert({-1, root});
}

NTree::~NTree()
{
}

void NTree::print()
{
    for (auto it = this->node.begin(); it != this->node.end(); it++) {
        std::cout << it->first << ": ";
        for (auto it1 = it->second.begin(); it1 != it->second.end(); it1++) {
            std::cout << *it1 << " ";
        }
        std::cout << "\n";
    }
}

int NTree::dfs(int child, int curChild) {
    for (auto i: this->node[curChild]) {
        // std::cout << i << "\n";
        if (i == child) {
            return 1;
        }
```

```
        return NTree::dfs(child, i);
    }
    return -1;
}

int NTree::find(int parent, int child)
{
    int ans = 0;
    for (auto curChild: this->node[parent]) {
        if (curChild == child) {
            return ans;
        } else if (dfs(child, curChild) != -1) {
            return ans;
        }
        ans++;
    }
    return -1;
}

std::pair<int, int> NTree::findNode(int node)
{
    for (auto i: this->node) {
        if (i.second.find(node) != i.second.cend()) {
            return std::make_pair(i.first, 0);
        }
    }
    return std::make_pair(-1, -1);
}

std::vector<int> NTree::findChilds(int node)
{
    std::vector<int> res;
    for (auto i: this->node[node]) {
        res.push_back(i);
    }
    return res;
}

int NTree::findCheck(int parent, int child)
{
    for (auto curChild: this->node[parent]) {
        if (curChild == child) {
```

```
            return 1;
        }
    }
    return -1;
}

int NTree::insert(int parent, int child)
{
    auto curParent = this->node.find(parent);
    if (curParent != this->node.cend()) {
        auto curChild = curParent->second.find(child);
        if (curChild != curParent->second.cend()) {
            return 0;
        }
        curParent->second.insert(child);
        std::set<int> childVec;
        this->node.insert({child, childVec});
        return 1;
    }
    return 0;
}

int NTree::erase(int parent, int child)
{
    auto curParent = this->node.find(parent);
    if (curParent != this->node.cend()) {
        auto curChild = curParent->second.find(child);
        if (curChild != curParent->second.cend()) {
            curParent->second.erase(child);
            auto p = this->node.find(child);
            this->node.erase(p);
            return 1;
        }
    }
    return 0;
}

void NTree::destroyUndertree(int node)
{
    auto curNode = this->node.find(node);
    if (curNode != this->node.cend()) {
        for (auto it: curNode->second) {
```

```
            this->destroyUndertree(it);
        }
    }
    curNode->second.clear();
    int parent = this->findNode(node).first;
    auto parentN = this->node.find(parent);
    parentN->second.erase(node);
    this->node.erase(node);
}

namespace advancedZMQ
{
    nlohmann::json sendAndRecv(nlohmann::json &request, zmq::socket_t
&socket, int debug)
    {
        std::string strFromJson = request.dump();
        if (debug) {
            std::cout << strFromJson << std::endl;
        }
        zmq::message_t msg(strFromJson.size());
        memcpy(msg.data(), strFromJson.c_str(), strFromJson.size());
        socket.send(msg);
        nlohmann::json reply;
        zmq::message_t msg2;
        socket.recv(msg2);
        std::string strToJson(static_cast<char *> (msg2.data()), msg2.size());
        if (!strToJson.empty()) {
            reply = nlohmann::json::parse(strToJson);
        } else {
            if (debug) {
                std::cout << "bad socket" << std::endl;
            }
            reply["ans"] = "error";
        }
        if (debug) {
            std::cout << reply << "\n";
        }
        return reply;
    }

    void Send(nlohmann::json &request, zmq::socket_t &socket)
    {
```

```
        std::string strFromJson = request.dump();
        zmq::message_t msg(strFromJson.size());
        memcpy(msg.data(), strFromJson.c_str(), strFromJson.size());
        socket.send(msg);
    }

    nlohmann::json Recv(zmq::socket_t &socket)
    {
        nlohmann::json reply;
        zmq::message_t msg;
        socket.recv(msg);
        std::string strToJson(static_cast<char *> (msg.data()), msg.size());
        reply = nlohmann::json::parse(strToJson);
        return reply;
    }
}

#endif
```

apidemon.cpp

```
#include <iostream>

#include <map>

#include <vector>

#include <string>

#include <string.h>

#include <unistd.h>

#include <csignal>

#include <thread>

#include <zmq.hpp>

#include <nlohmann/json.hpp>

#include "../include/labtools.h"


using namespace advancedZMQ;


zmq::context_t context;
```

```
zmq::socket_t socket(context, zmq::socket_type::pair);
zmq::socket_t demonSocket(context, zmq::socket_type::pair);

NTree nodes;

int main(int argc, char const *argv[])
{
    demonSocket.connect(("tcp://127.0.0.1:" + std::to_string(DEMON_PORT)));
    socket.setsockopt(ZMQ_RCVTIMEO, WAIT_TIME);
    socket.setsockopt(ZMQ_SNDTIMEO, WAIT_TIME);
    socket.bind(("tcp://127.0.0.1:" + std::to_string(PORT)));
    int pid = fork();
    if (pid == 0) {
        execl("./server", "./server", std::to_string(-1).c_str(), NULL);
        return 0;
    }
    int flag = 1;
    while (flag) {
        nlohmann::json demonReply;
        demonReply = Recv(demonSocket);
        if (demonReply["type"] == "create") {
            int id = demonReply["id"];
            int parentId = demonReply["parentId"];
            if (nodes.findNode(parentId).second == -1 && parentId != -1) {
                std::cout << "Error: parent " << parentId << " not found" << std::endl;
                continue;
            }
            if (nodes.find(parentId, id) != -1) {
                std::cout << "Error: child " << id << " already exists" << std::endl;
```

```
            continue;
        }
        nlohmann::json pingRequest;
        pingRequest["type"] = "ping";
        pingRequest["id"] = parentId;
        nlohmann::json pingReply = sendAndRecv(pingRequest, socket, 0);
        if (pingReply["ans"] != "ok") {
            socket.close();
            context.close();
            std::cout << "Error: parent " << parentId << " is unavailable" <<
std::endl;
            continue;
        }
        nlohmann::json request;
        request["type"] = "create";
        request["id"] = id;
        request["parentId"] = parentId;
        nlohmann::json reply = sendAndRecv(request, socket, 0);
        if (reply["ans"] != "ok") {
            socket.close();
            context.close();
            std::cout << "Error: parent " << parentId << " is unavailable" <<
std::endl;
            continue;
        } else {
            nodes.insert(parentId, id);
        }
    } else if (demonReply["type"] == "remove") {
        int id = demonReply["id"], parentId;
        if (nodes.findNode(id).second == -1) {
```

```
            std::cout << "Error: node " << id << " not found" << std::endl;
            continue;
        }
        parentId = nodes.findNode(id).first;
        nlohmann::json pingRequest;
        pingRequest["type"] = "ping";
        pingRequest["id"] = parentId;
        nlohmann::json pingReply = sendAndRecv(pingRequest, socket, 0);
        if (pingReply["ans"] != "ok") {
            std::cout << "Error: parent " << parentId << " is unavailable" <<
std::endl;
            continue;
        }
        nlohmann::json request;
        request["type"] = "remove";
        request["id"] = id;
        request["parentId"] = parentId;
        nlohmann::json reply = sendAndRecv(request, socket, 0);
        if (reply["ans"] != "ok") {
            std::cout << "Error: node " << id << " is unavailable" << std::endl;
            continue;
        } else {
            nodes.destroyUndertree(id);
            std::cout << "ok" << std::endl;
        }
    } else if (demonReply["type"] == "print") {
        nodes.print();
    } else if (demonReply["type"] == "exec") {
        int id = demonReply["id"];
```

```
            if (id == -1) {
                std::cout << "Error: control node can't do calculation" << std::endl;
                continue;
            }
            if (nodes.findNode(id).second == -1) {
                std::cout << "Error: node " << id << " not found" << std::endl;
                continue;
            }
            nlohmann::json reply = sendAndRecv(demonReply, socket, 0);
            if (reply["ans"] != "ok") {
                std::cout << "Error: node " << id << " is unavailable" << std::endl;
                continue;
            }
        } else if (demonReply["type"] == "pingall") {
            nlohmann::json reply = sendAndRecv(demonReply, socket, 0);
            if (reply["ans"] == "ok") {
                std::cout << "ok" << std::endl;
            }
        } else if (demonReply["type"] == "close") {
            nlohmann::json destroyRequest;
            destroyRequest["type"] = "remove";
            std::vector<int> nodesRoot = nodes.findChilds(-1);
            for (int i = 0; i < nodesRoot.size(); i++) {
                destroyRequest["id"] = nodesRoot[i];
                sendAndRecv(destroyRequest, socket, 0);
            }
            flag = 0;
        }
    }
```

```
    socket.close();
    demonSocket.close();
    return 0;
}
```

client.cpp

```
#include <iostream>
#include <map>
#include <vector>
#include <string>
#include <string.h>
#include <unistd.h>
#include <csignal>
#include <thread>
#include <zmq.hpp>
#include <nlohmann/json.hpp>
#include "../include/labtools.h"

using namespace advancedZMQ;

zmq::context_t context;
zmq::socket_t demonSocket(context, zmq::socket_type::pair);

int main(int argc, char const *argv[])
{
    int demonPid = fork();
    if (demonPid == 0) {
        execl("./apidemon", "./apidemon", NULL);
        return 0;
    } else {
```

```
    demonSocket.setsockopt(ZMQ_RCVTIMEO, WAIT_TIME);
    demonSocket.setsockopt(ZMQ_SNDTIMEO, WAIT_TIME);
    demonSocket.bind(("tcp://127.0.0.1:" + std::to_string(DEMON_PORT)));
    std::cout << "\nWrite:\n command [arg1] ... [argn]\n";
    std::cout << "\ncreate [id] [parentId] to create calculation node\n";
    std::cout << "\nremove [id] [parentId] to remove calculation node\n";
    std::cout << "\nexec [id] [key:string] [value:int] to add [key:value] to map on calculation node\n";
    std::cout << "\nexec [id] [key:string] to show the value in map on calculation node\n";
    std::cout << "\npingall to show all the unreachible nodes\n";
    std::cout << "\nprint to show NTree topology\n";
    std::cout << "\nhelp to see commands again\n\n";
    std::string command;
    while (std::cin >> command) {
        nlohmann::json demonRequest;
        if (command == "create") {
            int id, parentId;
            std::cin >> id >> parentId;
            demonRequest["type"] = "create";
            demonRequest["id"] = id;
            demonRequest["parentId"] = parentId;
            Send(demonRequest, demonSocket);
        } else if (command == "remove") {
            int id;
            std::cin >> id;
            demonRequest["type"] = "remove";
            demonRequest["id"] = id;
            Send(demonRequest, demonSocket);
        } else if (command == "print") {
```

```
            demonRequest["type"] = "print";
            Send(demonRequest, demonSocket);
        } else if (command == "exec") {
            std::string key;
            int id;
            char c;
            std::cin >> id >> key;
            c = getchar();
            if (c == ' ') {
                int value;
                std::cin >> value;
                demonRequest["type"] = "exec";
                demonRequest["action"] = "add";
                demonRequest["id"] = id;
                demonRequest["key"] = key;
                demonRequest["value"] = value;
            } else {
                demonRequest["type"] = "exec";
                demonRequest["action"] = "check";
                demonRequest["id"] = id;
                demonRequest["key"] = key;
            }
            Send(demonRequest, demonSocket);
        } else if (command == "pingall") {
            demonRequest["type"] = "pingall";
            Send(demonRequest, demonSocket);
        } else if (command == "help") {
            std::cout << "\nWrite:\n command [arg1] ... [argn]\n";
            std::cout << "\ncreate [id] [parentId] to create calculation node\n";
```

```
            std::cout << "\nremove [id] [parentId] to remove calculation node\n";
            std::cout << "\nexec [id] [key:string] [value:int] to add [key:value] to map on calculation node\n";
            std::cout << "\nexec [id] [key:string] to show the value in map on calculation node\n";
            std::cout << "\npingall to show all the unreachible nodes\n";
            std::cout << "\nprint to show NTree topology\n";
            std::cout << "\nhelp to see commands again\n\n";
        }
    }
    if (std::cin.eof()) {
        nlohmann::json demonRequest;
        demonRequest["type"] = "close";
        Send(demonRequest, demonSocket);
    }
    demonSocket.close();
  }
  return 0;
}
```

server.cpp

```
#include <iostream>
#include <map>
#include <vector>
#include <string>
#include <string.h>
#include <unistd.h>
#include <csignal>
#include <thread>
#include <zmq.hpp>
#include <nlohmann/json.hpp>
```

```
#include "../include/labtools.h"

using namespace advancedZMQ;

std::vector<zmq::socket_t> childSockets;
std::vector<int> childIds;

int main(int argc, char const *argv[])
{
    int curId = std::stoi(std::string(argv[1])), flag = 1;

    zmq::context_t parentContext(1);
    zmq::socket_t parentSocket(parentContext, zmq::socket_type::pair);
    parentSocket.connect(("tcp://127.0.0.1:" + std::to_string(PORT + curId + 1)));

    std::map<std::string, int> localDict;
    void *arg = NULL;
    while (flag) {
        nlohmann::json reply = Recv(parentSocket);
        nlohmann::json request;
        request["type"] = reply["type"];
        request["id"] = curId;
        request["pid"] = 0;
        request["ans"] = "error";
        if (reply["type"] == "ping") {
            if (reply["id"] == curId) {
                request["ans"] = "ok";
                request["id"] = curId;
                request["pid"] = getpid();
```

```
        } else {
            for (int i = 0; i < childSockets.size(); i++) {
                nlohmann::json pingRequest;
                pingRequest["type"] = "ping";
                pingRequest["id"] = reply["id"];
                nlohmann::json pingReply = sendAndRecv(pingRequest,
childSockets[i], 0);
                if (pingReply["ans"] == "ok") {
                    request["ans"] = "ok";
                    request["id"] = curId;
                    request["pid"] = getpid();
                    break;
                }
            }
        }
    } else if (reply["type"] == "create") {
        if (reply["parentId"] == curId) {
            int i = reply["id"];
            zmq::socket_t childSocket(parentContext, zmq::socket_type::pair);
            childSocket.setsockopt(ZMQ_RCVTIMEO, WAIT_TIME);
            childSocket.setsockopt(ZMQ_SNDTIMEO, WAIT_TIME);
            childSocket.bind(("tcp://*:" + std::to_string(PORT + i + 1)).c_str());
            int pid = fork();
            if (pid == 0) {
                int i = reply["id"];
                execl("./server", "./server", std::to_string(i).c_str(), NULL);
                return 0;
            } else {
                nlohmann::json pingRequest;
```

```
                pingRequest["type"] = "ping";
                pingRequest["id"] = reply["id"];
                nlohmann::json pingReply = sendAndRecv(pingRequest, childSocket,
0);
                if (pingReply["ans"] == "ok") {
                    std::cout << "OK: " << pingReply["pid"] << std::endl;
                    int i = reply["id"];
                    childSockets.push_back(std::move(childSocket));
                    childIds.push_back(reply["id"]);
                    request["ans"] = "ok";
                    request["parentId"] = reply["parentId"];
                }
            }
        } else {
            nlohmann::json newRequest = reply;
            for (int i = 0; i < childSockets.size(); i++) {
                nlohmann::json newReply = sendAndRecv(newRequest,
childSockets[i], 0);
                if (newReply["ans"] == "ok") {
                    request["ans"] = "ok";
                    break;
                }
            }
        }
    } else if (reply["type"] == "remove") {
        int c = 0;
        for (int i = 0; i < childSockets.size(); i++) {
            int childId = childIds[i];
            nlohmann::json destroyRequest;
            destroyRequest["type"] = "destroy";
```

```
            if (childId == reply["id"]) {
                Send(destroyRequest, childSockets[i]);
                int k = reply["id"];
                childSockets.erase(std::next(childSockets.begin() + i));
                childIds.erase(childIds.begin() + i);
                c++;
                break;
            }
        }
        if (c > 0) {
            request["ans"] = "ok";
            c = 0;
        }
        for (int i = 0; i < childSockets.size(); i++) {
            nlohmann::json newReply = sendAndRecv(reply, childSockets[i], 0);
            if (newReply["ans"] == "ok") {
                request["ans"] = "ok";
            }
        }
    } else if (reply["type"] == "destroy") {
        nlohmann::json newRequest = reply;
        for (int i = 0; i < childSockets.size(); i++) {
            int childId = childIds[i];
            Send(newRequest, childSockets[i]);
        }
        flag = 0;
    } else if (reply["type"] == "exec") {
        if (reply["action"] == "add") {
            if (reply["id"] == curId) {
```

```
                std::string key = reply["key"];
                int value = reply["value"];
                localDict.insert({key, value});
                std::cout << "ok:" << curId << ":" << key << ":" << value <<
std::endl;
                request["ans"] = "ok";
            } else {
                nlohmann::json newRequest = reply;
                for (int i = 0; i < childSockets.size(); i++) {
                    nlohmann::json newReply = sendAndRecv(newRequest,
childSockets[i], 0);
                    if (newReply["ans"] == "ok") {
                        request["ans"] = "ok";
                        break;
                    }
                }
            }
        } else {
            if (reply["id"] == curId) {
                std::string key = reply["key"];
                if (localDict.find(key) != localDict.cend()) {
                    std::cout << "ok:" << curId << ":" << localDict[key] << std::endl;
                    request["ans"] = "ok";
                } else {
                    std::cout << "Error: key " << key << " not found" << std::endl;
                    request["ans"] = "ok";
                }
            } else {
                nlohmann::json newRequest = reply;
                for (int i = 0; i < childSockets.size(); i++) {
```

```
                    nlohmann::json newReply = sendAndRecv(newRequest,
childSockets[i], 0);
                    if (newReply["ans"] == "ok") {
                        request["ans"] = "ok";
                        break;
                    }
                }
            }
        }
    } else if (reply["type"] == "pingall") {
        nlohmann::json newRequest;
        newRequest["type"] = "pingall";
        for (int i = 0; i < childSockets.size(); i++) {
            nlohmann::json newReply = sendAndRecv(newRequest, childSockets[i],
0);
            if (newReply["ans"] == "ok") {
                request["ans"] = "ok";
            } else {
                std::cout << childIds[i] << std::endl;
            }
        }
        request["ans"] = "ok";
    }
    Send(request, parentSocket);
}
for (int i = 0; i < childSockets.size(); i++) {
    childSockets[i].close();
}
parentSocket.close();
return 0;
```

}

Сборка программы

```
ggame@ggame:~/OS/ready/lab6$ ./make.sh
-- The CXX compiler identification is GNU 11.3.0
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ggame/OS/ready/lab6/build
[ 16%] Building CXX object CmakeFiles/client.dir/src/client.cpp.o
[ 33%] Linking CXX executable client
[ 33%] Built target client
[ 50%] Building CXX object CmakeFiles/server.dir/src/server.cpp.o
[ 66%] Linking CXX executable server
[ 66%] Built target server
[ 83%] Building CXX object CmakeFiles/apidemon.dir/src/apidemon.cpp.o
[100%] Linking CXX executable apidemon
[100%] Built target apidemon
```

Демонстрация работы программы

Терминал работы программы:
Write:
command [arg1] ... [argn]

create [id] [parentId] to create calculation node

remove [id] [parentId] to remove calculation node

exec [id] [key:string] [value:int] to add [key:value] to map on calculation node

exec [id] [key:string] to show the value in map on calculation node

pingall to show all the unreachible nodes

print to show NTree topology

help to see commands again

```
create 0 -1
OK: 14022
crwate 4 0
create 4 0
OK: 14028
create 3 4
OK: 14031
create 5 4
OK: 14034
print
-1: 0
0: 4
3:
4: 3 5
5:
exec 3 4 5
ok:3:4:5
exec 3 4
ok:3:5
exec 3 5
Error: key 5 not found
exec 5 6 7
ok:5:6:7
exec 5 6
ok:5:7
exec 5 4
Error: key 4 not found
pingall
ok
remove 3
ok
print
-1: 0
0: 4
4: 5
5:
exec 0 1 3
ok:0:1:3
```

exec 0 1
ok:0:3

Втророй терминал:
ggame@ggame:~/OS/ready/lab6$ ps -a
  PID TTY   TIME CMD
 3586 tty2   00:02:17 Xorg
 3679 tty2   00:00:00 gnome-session-b
13815 pts/0   00:00:00 make.sh
13987 pts/0   00:00:00 client
13990 pts/0   00:00:00 apidemon
13993 pts/0   00:00:00 server
14022 pts/0   00:00:00 server
14028 pts/0   00:00:00 server
14031 pts/0   00:00:00 server <defunct>
14034 pts/0   00:00:00 server
14138 pts/1   00:00:00 ps
ggame@ggame:~/OS/ready/lab6$ kill -9 14034
ggame@ggame:~/OS/ready/lab6$ ps -a
  PID TTY   TIME CMD
 3586 tty2   00:02:17 Xorg
 3679 tty2   00:00:00 gnome-session-b
13815 pts/0   00:00:00 make.sh
13987 pts/0   00:00:00 client
13990 pts/0   00:00:00 apidemon
13993 pts/0   00:00:00 server
14022 pts/0   00:00:00 server
14028 pts/0   00:00:00 server
14031 pts/0   00:00:00 server <defunct>
14034 pts/0   00:00:00 server <defunct>
14204 pts/1   00:00:00 ps

Терминал работы программы:
exec 5 4 3
Error: node 5 is unavailable
pingall
0
4
5

Выводы

В ходе выполнения данной ЛР я познакомился с сокетами ZMQ и с отличной

библиотекой nlohmann/json.hpp, также узнал как писать асинхронный код на Си, используя системные вызовы UNIX.