

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: Матвеев Данил
Группа: М8О-207Б-21
Вариант: 14
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023

Содержание

- 1 Репозиторий
- 2 Постановка задачи
- 3 Общие сведения о программе
- 4 Общий метод и алгоритм решения
- 5 Исходный код
- 6 Демонстрация работы программы
- 7 Выводы

Репозиторий

<https://github.com/MrDenli/OsLabs>

Постановка задачи

Цель работы

Приобретение практических навыков в: управлении серверами сообщений (№6), применение отложенных вычислений (№7), интеграция программных систем друг с другом (№8)

Задание

13 вариант

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.

Топология 2: Аналогично топологии 1, но узлы находятся в дереве общего вида.

Набор команд 1 (подсчет суммы n чисел): Формат команды: `exec id n k 1 ... k n`

Команда проверки 1: Формат команды: `exec id n k1 ... kn`

`id` – целочисленный идентификатор вычислительного узла, на который отправляется команда

`n` – количество складываемых чисел (от 1 до 108)

`k1 ... kn` – складываемые числа

Пример:

```
> exec 10 3 1 2 3
```

```
Ok:10: 6
```

Общие сведения о программе

Программа выполняет следующие команды:

1. `create id parent` — создает узел `id` и родителем `parent`
2. `remove id` — удаляет узел `id`
3. `exec id n s1 ... sn` — на узле `id` считает сумму `n` чисел
4. `ping id` — проверяет доступность узла `id`

5. `exit` — завершает программу.

Общий метод и алгоритм решения

Узлы передают между собой информацию при помощи очереди сообщений ZeroMQ.

Для создания узла, мы запускаем `fork` на родителе и передаем ребенку данные для связи с ним.

При удалении мы передаем детям сигнал об удалении. Так же формируем список потомков. В итоге все узлы из этого списка удаляются

При проверки доступности мы посылаем сообщение детям, если за 3 секунды нет ответа, то считаем узел недоступный.

Для отправки сообщений мы проверяем на доступность детей, если доступны, то отправляем им сообщение. Обратно узлы отправляют сообщение или о нахождении нужного узла и о выполнении функции, или о ошибке.

Для выполнения функции подсчета суммы сообщение передается нужному узлу, после узел считает в цикле сумму и отправляет сообщение обратно

Исходный код

client.cpp

```
#include "node_tree.hpp"
#include "net_realization.hpp"
#include "set"
#include <signal.h>

int main() {
    std::set<int> all_nodes;
    std::string prog_path = "./server";
    Node task(-1);
    all_nodes.insert(-1);
    std::string command;
    while(std::cin >> command) {
        if(command == "create") {
            int id_child, id_parent;
            std::cin >> id_child >> id_parent;
            if(all_nodes.find(id_child) != all_nodes.end()) {
                std::cout << "Error: Already exists" << std::endl;
            }
            else if(all_nodes.find(id_parent) == all_nodes.end()) {
```

```

        std::cout << "Error: Parent not found" << std::endl;
    }
    else if(id_parent == task.id) {
        std::string ans = task.Create(id_child, prog_path);
        std::cout << ans << std::endl;
        all_nodes.insert(id_child);
    }
    else{
        std::string str = "create " + std::to_string(id_child);
        std::string ans = task.Send(str, id_parent);
        std::cout << ans << std::endl;
        all_nodes.insert(id_child);
    }
}

else if (command == "pingall") {
    bool flag = false;
    std::cout << "Ok: ";
    for (auto i: all_nodes) {
        std::string str = "ping " + std::to_string(i);
        std::string ans = task.Send(str, i);
        if(ans == "Error: not find" && i != -1) {
            flag = true;
            std::cout << i << ";";
        }
    }
    if (!flag) {
        std::cout << "-1\n";
    }
    else {
        std::cout << "\n";
    }
}

else if(command == "exec") {
    int id, number, cnt;
    std::cin >> id >> cnt;
    std::string msg = "exec " + std::to_string(cnt);
    for(int i = 0; i < cnt; i++) {
        std::cin >> number;

```

```

        msg += " " + std::to_string(number);
    }
    if(all_nodes.find(id) == all_nodes.end()) {
        std::cout << "Error: Not found" << std::endl;
    }
    else {
        std::string ans = task.Send(msg,id);
        std::cout << ans << std::endl;
    }
}

else if(command == "remove") {
    int id;
    std::cin >> id;
    std::string msg = "remove";
    if(all_nodes.find(id) == all_nodes.end()) {
        std::cout << "Error: Not found" << std::endl;
    }
    else {
        std::string ans = task.Send(msg,id);
        if(ans != "Error: not find") {
            std::istringstream ids(ans);
            int tmp;
            while(ids >> tmp) {
                all_nodes.erase(tmp);
            }
            ans = "Ok";
            if(task.children.find(id) != task.children.end()) {
                net::unbind(task.children[id],task.children_port[id]);
                task.children[id]->close();
                task.children.erase(id);
                task.children_port.erase(id);
            }
        }
        std::cout << ans << std::endl;
    }
}

else if(command == "exit") {
    task.Remove();

```

```

        return 0;
    }
}
}

server.cpp
#include "node_tree.hpp"
#include "net_realization.hpp"
#include <fstream>
#include <signal.h>

int main(int argc, char **argv) {
    if(argc != 3) {
        return -1;
    }
    Node task(atoi(argv[1]),atoi(argv[2]));
    std::string prog_path = "./server";
    while(1) {
        std::string message;
        std::string command = " ";
        message = net::reseave_message(&(task.parent));
        std::istringstream request(message);
        request >> command;

        if(command == "create") {
            int id_child, id_parent;
            request >> id_child;
            std::string ans = task.Create(id_child, prog_path);
            net::send_message(&task.parent,ans);
        }
        else if(command == "pid") {
            std::string ans = task.Pid();
            net::send_message(&task.parent,ans);
        }
        else if(command == "ping") {
            int id_child;
            request >> id_child;
            std::string ans = task.Ping(id_child);

```

```

        net::send_message(&task.parent,ans);
    }
    else if(command == "send") {
        int id;
        request >> id;
        std::string str;
        getline(request, str);
        str.erase(0,1);
        std::string ans;
        ans = task.Send(str,id);
        net::send_message(&task.parent,ans);
    }
    else if(command == "exec") {
        int cnt, sum = 0, number;
        request >> cnt;
        for(int i = 0; i < cnt; i++) {
            request >> number;
            sum += number;
        }
        std::string to_send;
        to_send = "Ok: " + std::to_string(task.id) + ": " + std::to_string(sum);
        net::send_message(&task.parent,to_send);
    }
    else if(command == "remove") {
        std::string ans = task.Remove();
        ans = std::to_string(task.id) + " " + ans;
        net::send_message(&task.parent, ans);
        net::disconnect(&task.parent, task.parent_port);
        task.parent.close();
        break;
    }
}

return 0;
}

```

net_realization.hpp

```
#pragma once
```

```
#include <iostream>
```



```

#include <zmq.hpp>
#include <sstream>
#include <string>

namespace net {
    int bind(zmq::socket_t *socket, int id) {
        int port = 4040 + id;
        while(true) {
            std::string adress = "tcp://127.0.0.1:" + std::to_string(port);
            try {
                socket->bind(adress);
                break;
            }
            catch(...) {
                port++;
            }
        }
        return port;
    }

    void unbind(zmq::socket_t *socket, int port) {
        std::string adress = "tcp://127.0.0.1:" + std::to_string(port);
        socket->unbind(adress);
    }

    void connect(zmq::socket_t *socket, int port) {
        std::string adress = "tcp://127.0.0.1:" + std::to_string(port);
        socket->connect(adress);
    }

    void disconnect(zmq::socket_t *socket, int port) {
        std::string adress = "tcp://127.0.0.1:" + std::to_string(port);
        socket->disconnect(adress);
    }

    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());
        try{

```

```

        socket->send(message);
    }catch(...) {}
}

std::string reaseve_message(zmq::socket_t *socket) {
    zmq::message_t message;
    bool success = true;
    try {
        socket->recv(&message,0);
    }
    catch(...) {
        success = false;
    }
    if(!success || message.size() == 0) {;
        throw -1;
    }
    std::string str(static_cast<char*>(message.data()), message.size());
    return str;
}

}

node_tree.hpp
#include <iostream>
#include "net_realization.hpp"
#include <sstream>
#include <unordered_map>
#include "unistd.h"

class Node {
private:
    zmq::context_t context;
public:
    std::unordered_map<int,zmq::socket_t*> children;
    std::unordered_map<int,int> children_port;
    zmq::socket_t parent;
    int parent_port;
    int id;
    Node(int _id , int _parent_port = -1):id(_id),parent(context,ZMQ_REP),parent_port(_parent_port) {

```

```

        if(_id != -1) {
            net::connect(&parent,_parent_port);
        }
    }

    std::string Ping(int _id) {
        std::string ans = "Ok: 0";
        if(_id == id) {
            ans = "Ok: 1";
            return ans;
        }
        else if (children.find(_id) != children.end()) {
            std::string msg = "ping " + std::to_string(_id);
            net::send_message(children[_id],msg);
            try{
                msg = net::reseave_message(children[_id]);
                if(msg == "Ok: 1")
                    ans = msg;
            }
            catch(int) {}
            return ans;
        }
        else {
            return ans;
        }
    }

    std::string Create(int child_id,std::string program_path) {
        std::string program_name = program_path.substr(program_path.find_last_of("/") + 1);
        children[child_id] = new zmq::socket_t(context,ZMQ_REQ);
        int new_port = net::bind(children[child_id],child_id);
        children_port[child_id] = new_port;
        int pid = fork();
        if(pid == 0) {
            execl(program_path.c_str(), program_name.c_str(), std::to_string(child_id).c_str() ,
std::to_string(new_port).c_str() ,(char*)NULL);
        }
        else {
            std::string child_pid;
            try{

```

```

        children[child_id]->setsockopt(ZMQ_SNDTIMEO,3000);
        net::send_message(children[child_id],"pid");
        child_pid = net::reseave_message(children[child_id]);
    } catch(int) {
        child_pid = "Error: can't connect to child";
    }
    return "Ok: " + child_pid;
}
}

std::string Pid() {
    return std::to_string(getpid());
}

std::string Send(std::string str,int _id) {
    if(children.size() == 0) {
        return "Error: not find";
    }
    else if(children.find(_id) != children.end()) {
        if(Ping(_id) == "Ok: 1") {
            net::send_message(children[_id],str);
            std::string ans;
            try{
                ans = net::reseave_message(children[_id]);
            } catch(int) {
                ans = "Error: not find";
            }
            return ans;
        }
    }
    else {
        std::string ans = "Error: not find";
        for(auto& child: children ) {
            if(Ping(child.first) == "Ok: 1") {
                std::string msg = "send " + std::to_string(_id) + " " + str;
                net::send_message(children[child.first],msg);
                try {
                    msg = net::reseave_message(children[child.first]);
                }
                catch(int) {

```

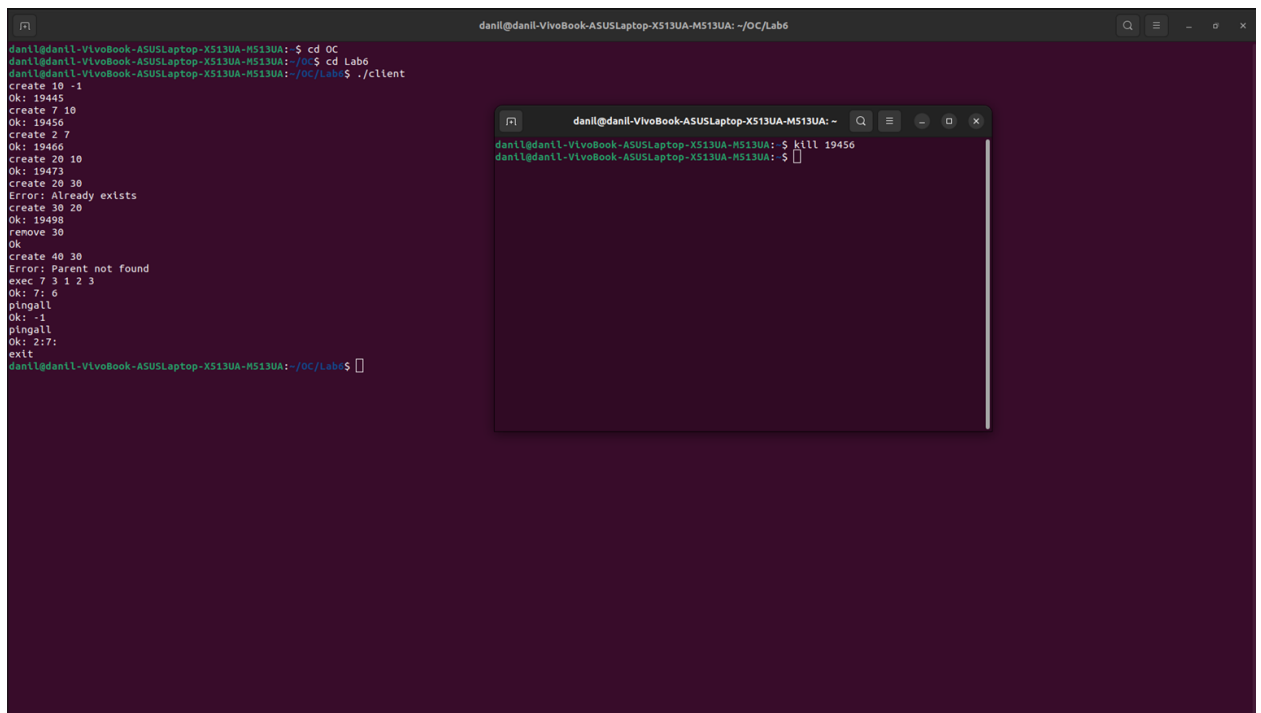
```

        msg = "Error: not find";
    }
    if(msg != "Error: not find") {
        ans = msg;
    }
}
}
return ans;
}
}

std::string Remove() {
    std::string ans;
    if(children.size() > 0) {
        for(auto& child: children ) {
            if(Ping(child.first) == "Ok: 1") {
                std::string msg = "remove";
                net::send_message(children[child.first],msg);
                try{
                    msg = net::reseave_message(children[child.first]);
                    if(ans.size() > 0)
                        ans = ans + " " + msg;
                    else
                        ans = msg;
                } catch(int) {}
            }
            net::unbind(children[child.first], children_port[child.first]);
            children[child.first]->close();
        }
        children.clear();
        children_port.clear();
    }
    return ans;
}
};

```

Демонстрация работы программы



```
danil@danil-VivoBook-ASUSLaptop-X513UA-M513UA: ~$ cd OC
danil@danil-VivoBook-ASUSLaptop-X513UA-M513UA: ~/OC$ cd Lab6
danil@danil-VivoBook-ASUSLaptop-X513UA-M513UA: ~/OC/Lab6$ ./client
create 10 -1
Ok: 19445
create 7 10
Ok: 19456
create 2 7
Ok: 19466
create 20 10
Ok: 19473
create 20 30
Error: Already exists
create 30 20
Ok: 19498
remove 30
Ok
create 40 30
Error: Parent not found
exec 7 3 1 2 3
Ok: 7: 6
pingall
Ok: -1
pingall
Ok: 2:7:
exit
danil@danil-VivoBook-ASUSLaptop-X513UA-M513UA: ~/OC/Lab6$
```

```
danil@danil-VivoBook-ASUSLaptop-X513UA-M513UA: ~$ kill 19456
danil@danil-VivoBook-ASUSLaptop-X513UA-M513UA: ~$
```

Выводы

Мне понравилась данная лабораторная работа, при выполнении работы я получил навыки работы с серверами, очередью сообщений, применения отложенных вычислений, интеграции программных систем друг с другом, так же я повторил деревья и способы взаимодействия с ними. Данные знания помогут мне в будущем.