ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
"МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)"

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа № 6 по курсу

по курсу «Операционные системы»

Группа: М8о-207Б-18

Студент:

Токарев Никита Станиславович

Преподаватель:

Миронов Евгений Сергеевич

Оценка:

Дата:

Москва,2019

Оглавление

1.Постановка задачи.....3
2.Структура программы.....3
3.Описание программы.....3
4.Листинг программы.....4
5.Результат работы.....19
6.Вывод.....20

1.Постановка задачи

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. Управляющий узел отвечает за ввод команд от пользователя и отправку этих команд на вычислительные узлы.
Вариант № 55:

Вариант задания: 41. Топология — бинарное дерево. Тип вычислительной команды — сумма n чисел. Тип проверки узлов на доступность — пинг указанного узла.

2.Структура программы

В данной работе в моя программа состоит из 5 файлов:

1. main_node.cpp

2. CmakeLists.txt

3. child_node.cpp

4. server_functions.cpp

5. server_functions.hpp

3.Описание программы

Программ состоит из 5 объектов. CmakeLists.txt отвечает за сборку. В процесс сборки генерируется Makefile и два исполняемых объекта terminal и child_node. Соответственно все вызовы происходят в исполняемом файле

terminal(управляющий узел), a child_node вызывается с помощью системного вызова execv с передаваемыми аргументами id и port. В файле server_functions.cpp описаны такие основные функции как получение сообщения, отправка сообщения и создание вычислительного узла.

Общая логика программы:

1. Управляющий узел принимает команды, обрабатывает их и пересылает дочерним узлам(или выводит сообщение об ошибке).
2. Дочерние узлы проверяют, может ли быть команда выполнена в данном узле, если нет, то команда пересылается в один из дочерних узлов, из которого возвращается некоторое сообщение(об успехе или об ошибке), которое потом пересылается обратно по дереву.

Соответственно в порождаемом(дочернем) процессе происходит запуск вычислительного узла, а в родительском процессе происходит отправка сообщений, а также прием сообщений. Также в моей программе используются I/O потоки, создаваемые с помощью вызова context(). Также на сокеты я устанавливаю опцию, которая по истечению тайм-аута отправляет сигнал с помощью которого я определяю о доступности узла. Логика команды доступности состоит в том, что запрашивается сообщения и если они пусты или содержат error, то узел недоступен. Также у меня реализовано бинарное дерево для простоты поиска нужного id узла.

4.Листинг программы

```
//main_node.cpp
#include <iostream>
#include "zmq.hpp"
#include <string>
#include <zconf.h>
#include <vector>
#include <signal.h>
```

```
#include <sstream>
#include <set>
#include <algorithm>
#include "server_functions.h"

class IdTree {
public:
    IdTree() = default;

    ~IdTree() {
        delete_node(head_);
    }

    bool contains(int id) {
        TreeNode* temp = head_;
        while(temp != nullptr) {
            if (temp->id_ == id) {
                break;
            }
            if (id > temp->id_) {
                temp = temp->right;
            }
            if (id < temp->id_) {
                temp = temp->left;
            }
        }
        return temp != nullptr;
    }

    void insert(int id) {
        if (head_ == nullptr) {
            head_ = new TreeNode(id);
            return;
        }
        TreeNode* temp = head_;
        while(temp != nullptr) {
            if (id == temp->id_) {
                break;
```

```
        }
        if (id < temp->id_) {
            if (temp->left == nullptr) {
                temp->left = new TreeNode(id);
                break;
            }
            temp = temp->left;
        }
        if (id > temp->id_) {
            if (temp->right == nullptr) {
                temp->right = new TreeNode(id);
                break;
            }
            temp = temp->right;
        }
    }
}

void erase(int id) {
    TreeNode* prev_id = nullptr;
    TreeNode* temp = head_;
    while (temp != nullptr) {
        if (id == temp->id_) {
            if (prev_id == nullptr) {
                head_ = nullptr;
            } else {
                if (prev_id->left == temp) {
                    prev_id->left = nullptr;
                } else {
                    prev_id->right = nullptr;
                }
            }
            delete_node(temp);

        } else if (id < temp->id_) {
            prev_id = temp;
            temp = temp->left;
        } else if (id > temp->id_) {
```

```
            prev_id = temp;
            temp = temp->right;
        }
    }
}

std::vector<int> get_nodes() const {
    std::vector<int> result;
    get_nodes(head_, result);
    return result;
}

private:

struct TreeNode {
    TreeNode(int id) : id_(id) {}
    int id_;
    TreeNode* left = nullptr;
    TreeNode* right = nullptr;
};

void get_nodes(TreeNode* node, std::vector<int>& v) const {
    if (node == nullptr) {
        return;
    }
    get_nodes(node->left,v);
    v.push_back(node->id_);
    get_nodes(node->right, v);
}

void delete_node(TreeNode* node) {
    if (node == nullptr) {
        return;
    }
    delete_node(node->right);
    delete_node(node->left);
    delete node;
}
```

```
    TreeNode* head_ = nullptr;
};

int main() {
    std::string command;
    IdTree ids;
    size_t child_pid = 0;
    int child_id = 0;
    zmq::context_t context(1);
    zmq::socket_t main_socket(context, ZMQ_REQ);
    int linger = 0;
    main_socket.setsockopt(ZMQ_SNDTIMEO, 2000);
    //main_socket.setsockopt(ZMQ_RCVTIMEO, 2000);
    main_socket.setsockopt(ZMQ_LINGER, &linger, sizeof(linger));
    //main_socket.connect(get_connect_name(30000));
    int port = bind_socket(main_socket);

    while (true) {
        std::cin >> command;
        if (command == "create") {
            size_t node_id;
            std::string result;
            std::cin >> node_id;
            if (child_pid == 0) {
                child_pid = fork();
                if (child_pid == -1) {
                    std::cout << "Unable to create first worker node\n";
                    child_pid = 0;
                    exit(1);
                } else if (child_pid == 0) {
                    create_node(node_id, port);
                } else {
                    child_id = node_id;
                    send_message(main_socket,"pid");
                    result = recieve_message(main_socket);

                }
```

```
        } else {
//              if (child_id == node_id) {
//                  std::cout << "Error: Already exists";
//              }
            std::ostringstream msg_stream;
            msg_stream << "create " << node_id;
            send_message(main_socket, msg_stream.str());
            result = recieve_message(main_socket);
        }

        if (result.substr(0,2) == "Ok") {
            ids.insert(node_id);
        }
        std::cout << result << "\n";

    } else if (command == "remove") {
        if (child_pid == 0) {
            std::cout << "Error:Not found\n";
            continue;
        }
        size_t node_id;
        std::cin >> node_id;
        if (node_id == child_id) {
            kill(child_pid, SIGTERM);
            kill(child_pid, SIGKILL);
            child_id = 0;
            child_pid = 0;
            std::cout << "Ok\n";
            ids.erase(node_id);
            continue;
        }
        std::string message_string = "remove " + std::to_string(node_id);
        send_message(main_socket, message_string);
        std::string recieved_message = recieve_message(main_socket);
        if (recieved_message.substr(0, std::min<int>(recieved_message.size(), 2)) == "Ok") {
            ids.erase(node_id);
        }
```

```
    std::cout << recieved_message << "\n";

} else if (command == "exec") {
    int id, n;
    std::cin >> id >> n;
    std::vector<int> numbers(n);
    for (int i = 0; i < n; ++i) {
        std::cin >> numbers[i];
    }

    std::string message_string = "exec " + std::to_string(id) + " " + std::to_string(n);
    for (int i = 0; i < n; ++i) {
        message_string += " " + std::to_string(numbers[i]);
    }

    send_message(main_socket, message_string);
    std::string recieved_message = recieve_message(main_socket);
    std::cout << recieved_message << "\n";

} else if (command == "ping") {
    int id;
    std::cin >> id;
    send_message(main_socket, "ping");
    std::string recieved = recieve_message(main_socket);
    std::istringstream is;
    std::vector from_tree = ids.get_nodes();
    int r = 0;

    for (auto it = from_tree.begin(); it != from_tree.end(); ++it) {
        if(*it == id) {
            r = 1;
        }
    }
    if(r == 0) {
        std::cout << "Error: Not found\n";
    } else {
        if (recieved.substr(0,std::min<int>(recieved.size(), 5)) == "Error") {
            std::cout << "Ok:0\n";
```

```
            } else {
                std::cout << "Ok:1\n";
            }
        }
    } else if (command == "exit") {
        break;
    }

}

return 0;
}
//CmakeLists.txt
cmake_minimum_required(VERSION 3.10)
project(os_lab_06)

set(CMAKE_CXX_STANDARD 17)

add_executable(terminal main_node.cpp)
set_target_properties(
    terminal PROPERTIES
        COMPILE_OPTIONS "-g;-Wall;-Wextra;-Wpedantic;"
)
add_executable(child_node child_node.cpp)
add_library(functions server_functions.cpp server_functions.h)

target_link_libraries(functions zmq)
target_link_libraries(terminal zmq functions)
target_link_libraries(child_node zmq functions)

//child_node.cpp
#include <iostream>
#include "zmq.hpp"
#include <string>
#include <sstream>
#include <zconf.h>
#include <exception>
#include <signal.h>
```

```
#include "server_functions.h"


int main(int argc, char** argv) { //аргументы - айди и номер порта, к которому нужно подключиться
//    zmq::context_t context (1);

//    zmq::message_t msg(strlen(argv[1]));
//    zmq::message_t msg_2(strlen(argv[2]));
//    zmq::message_t rcv;
//    memcpy(msg.data(), argv[1],strlen(argv[1]));
//    memcpy(msg_2.data(), argv[2],strlen(argv[2]));
//    socket.send(msg);
//    socket.recv(&rcv);
//    socket.send(msg_2);
    int id = std::stoi(argv[1]);
    int parent_port = std::stoi(argv[2]);
    zmq::context_t context(3);
    zmq::socket_t parent_socket(context, ZMQ_REP);

//    zmq::socket_t socket (context, ZMQ_REQ);
//    socket.connect ("tcp://127.0.0.1:5555");
    parent_socket.connect(get_port_name(parent_port));

    int left_pid = 0;
    int right_pid = 0;
    int left_id = 0;
    int right_id = 0;


    zmq::socket_t left_socket(context, ZMQ_REQ);
    zmq::socket_t right_socket(context, ZMQ_REQ);
    int linger = 0;
    left_socket.setsockopt(ZMQ_SNDTIMEO, 2000);
    //left_socket.setsockopt(ZMQ_RCVTIMEO, 2000);
    left_socket.setsockopt(ZMQ_LINGER, &linger, sizeof(linger));
    right_socket.setsockopt(ZMQ_SNDTIMEO, 2000);
    //right_socket.setsockopt(ZMQ_RCVTIMEO, 2000);
    right_socket.setsockopt(ZMQ_LINGER, &linger, sizeof(linger));
```

```
    int left_port = bind_socket(left_socket);
    int right_port = bind_socket(right_socket);

    while (true) {
        std::string request_string;

        request_string = recieve_message(parent_socket);



//        std::ostringstream stream;
//        stream << "Worker: id:" << id << "\n"
//               << "pid:" << getpid() << "\n"
//               << "parent port:" << parent_port << "\n"
//               << "left port:" << left_port << "\n"
//               << "right port:" << right_port << "\n"
//               << "left child: id:" << left_id << " pid:" << left_pid << "\n"
//               << "right child: id:" << right_id << " pid:" << right_pid << "\n"
//               << "request:" << request_string << "\n\n";
//        send_message(socket, stream.str());
//        recieve_message(socket);

        std::istringstream command_stream(request_string);
        std::string command;
        command_stream >> command;

        if (command == "id") {
            std::string parent_string = "Ok:" + std::to_string(id);
            send_message(parent_socket, parent_string);
        } else if (command == "pid") {
            std::string parent_string = "Ok:" + std::to_string(getpid());
            send_message(parent_socket, parent_string);
        } else  if (command == "create") {
            int id_to_create;
            command_stream >> id_to_create;
            // управляюший узел сообщает id нового узла и порт, к которому его надо подключить
            if (id_to_create == id) {
```

```
    // если id равен данному, значит узел уже существует, посылаем ответ с ошибкой
    std::string message_string = "Error: Already exists";
    send_message(parent_socket, message_string);
} else if (id_to_create < id) {
    if (left_pid == 0) {
        left_pid = fork();
        if (left_pid == -1) {
            send_message(parent_socket, "Error: Cannot fork");
            left_pid = 0;
        } else if (left_pid == 0) {
            create_node(id_to_create,left_port);
        } else {
            left_id = id_to_create;
            send_message(left_socket, "pid");
            send_message(parent_socket, recieve_message(left_socket));
        }
    } else {
        send_message(left_socket, request_string);
        send_message(parent_socket, recieve_message(left_socket));
    }
} else {
    if (right_pid == 0) {
        right_pid = fork();
        if (right_pid == -1) {
            send_message(parent_socket, "Error: Cannot fork");
            right_pid = 0;
        } else if (right_pid == 0) {
            create_node(id_to_create,right_port);
        } else {
            right_id = id_to_create;
            send_message(right_socket, "pid");
            send_message(parent_socket, recieve_message(right_socket));
        }
    } else {
        send_message(right_socket, request_string);
        send_message(parent_socket, recieve_message(right_socket));
    }
}
```

```
} else if (command == "remove") {
    int id_to_delete;
    command_stream >> id_to_delete;
    if (id_to_delete < id) {
        if (left_id == 0) {
            send_message(parent_socket, "Error: Not found");
        } else if (left_id == id_to_delete) {
            send_message(left_socket, "kill_children");
            recieve_message(left_socket);
            kill(left_pid,SIGTERM);
            kill(left_pid,SIGKILL);
            left_id = 0;
            left_pid = 0;
            send_message(parent_socket, "Ok");

        } else {
            send_message(left_socket, request_string);
            send_message(parent_socket, recieve_message(left_socket));
        }
    } else {
        if (right_id == 0) {
            send_message(parent_socket, "Error: Not found");
        } else if (right_id == id_to_delete) {
            send_message(right_socket, "kill_children");
            recieve_message(right_socket);
            kill(right_pid,SIGTERM);
            kill(right_pid,SIGKILL);
            right_id = 0;
            right_pid = 0;
            send_message(parent_socket, "Ok");
        } else {
            send_message(right_socket, request_string);
            send_message(parent_socket, recieve_message(right_socket));
        }
    }
} else if (command == "exec") {
    int exec_id;
```

```
    command_stream >> exec_id;
    if (exec_id == id) {
        int n;
        command_stream >> n;
        int sum = 0;
        for (int i = 0; i < n; ++i) {
            int cur_num;
            command_stream >> cur_num;
            sum += cur_num;
        }
        std::string recieve_message = "Ok:" + std::to_string(id) + ":" + std::to_string(sum);
        send_message(parent_socket, recieve_message);

    } else if (exec_id < id) {
        if (left_pid == 0) {
            std::string recieve_message = "Error:" + std::to_string(exec_id) + ": Not found";
            send_message(parent_socket, recieve_message);
        } else {
            send_message(left_socket, request_string);
            send_message(parent_socket, recieve_message(left_socket));
        }
    } else {
        if (right_pid == 0) {
            std::string recieve_message = "Error:" + std::to_string(exec_id) + ": Not found";
            send_message(parent_socket, recieve_message);
        } else {
            send_message(right_socket, request_string);
            send_message(parent_socket, recieve_message(right_socket));
        }
    }

} else if (command == "ping") {
    std::ostringstream res;
    std::string left_res;
    std::string right_res;
    if (left_pid != 0) {
        send_message(left_socket, "ping");
        left_res = recieve_message(left_socket);
```

```
        }
        if (right_pid != 0) {
            send_message(right_socket, "ping");
            right_res = recieve_message(right_socket);
        }
        if (!left_res.empty() && left_res.substr(std::min<int>(left_res.size(),5)) != "Error") {
            res << left_res;
        }


        if (!right_res.empty() && right_res.substr(std::min<int>(right_res.size(),5)) != "Error") {
            res << right_res;
        }
        send_message(parent_socket, res.str());
    } else if (command == "kill_children") {  // УБИТЬ ВСЕХ ДЕТЕЙ
        if (left_pid == 0 && right_pid == 0) {
            send_message(parent_socket, "Ok");
        } else {
            if (left_pid != 0) {
                send_message(left_socket, "kill_children");
                recieve_message(left_socket);
                kill(left_pid,SIGTERM);
                kill(left_pid,SIGKILL);
            }
            if (right_pid != 0) {
                send_message(right_socket, "kill_children");
                recieve_message(right_socket);
                kill(right_pid,SIGTERM);
                kill(right_pid,SIGKILL);
            }
            send_message(parent_socket, "Ok");


        }
    }
    if (parent_port == 0) {
        break;
    }
}
```

```
}
//server_functions.h
#pragma once
#include <string>
#include <zconf.h>
#include "zmq.hpp"

bool send_message(zmq::socket_t& socket, const std::string& message_string);

std::string recieve_message(zmq::socket_t& socket);

std::string get_port_name(int port);

int bind_socket(zmq::socket_t& socket);

void create_node(int id, int port);
//server_functions.cpp
#include "server_functions.h"


bool send_message(zmq::socket_t& socket, const std::string& message_string) {
    zmq::message_t message(message_string.size());
    memcpy(message.data(), message_string.c_str(), message_string.size());
    return socket.send(message);
}

std::string recieve_message(zmq::socket_t& socket) {
    zmq::message_t message;
    bool ok;
    try {
        ok = socket.recv(&message);
    } catch (...) {
        ok = false;
    }
    std::string recieved_message(static_cast<char*>(message.data()), message.size());
    if (recieved_message.empty() || !ok) {
        return "Error: Node is not available";
```

```
    }
    return recieved_message;
}

std::string get_port_name(int port) {
    return "tcp://127.0.0.1:" + std::to_string(port);
}
int bind_socket(zmq::socket_t& socket) {
    int port = 30000;
    while (true) {
        try {
            socket.bind(get_port_name(port));
            break;
        } catch(...) {
            port++;
        }
    }
    return port;
}

void create_node(int id, int port) {
    char* arg1 = strdup((std::to_string(id)).c_str());
    char* arg2 = strdup((std::to_string(port)).c_str());
    char* args[] = {"./child_node", arg1, arg2, NULL};
    execv("./child_node", args);
}
```

5.Результат работы

```
nikita@nikita-HP:~/oc/lr6v55/work$ ./terminal
create 2
Ok:8101
create 8
Ok:8106
create 4
Ok:8111
create 0
```

Ok:8116

ping 13

Error: Not found

ping 4

Ok:0

exec 8 2 3 4 5 6 7

Ok:8:7

remove 11

Error: Not found

remove 8

Ok

ping 8

Error: Not found

exec 1 5 1 1 1 1 1

Error:1: Not found

exec 2 5 1 1 1 1 1

Ok:2:5

exit

6.Вывод

В ходе данной работы я получил некие навыки работы с библиотекой zmq. Данная библиотека позволяет организовать асинхронный обмен сообщений. Также хотелось бы отметить высокую сложность данной работы. В ходе данной работы я получил опыт работы с многопроцессорными программами. Хотелось бы отметить, что асинхронная обработка реализуется благодаря свойствам библиотеки zmq: асинхронный обмен сообщений.