Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №5-7 по курсу

«Операционные системы»

Группа: М8О-215Б-23

Студент: Агафонов А.С.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 26.12.24

Москва, 2024

Постановка задачи

Вариант 26.

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. Список основных поддерживаемых команд:

Создание нового вычислительного узла Формат команды: create id [parent]

id – целочисленный идентификатор нового вычислительного узла

parent – целочисленный идентификатор родительского узла.

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

Топология: все вычислительные узлы хранятся в бинарном дереве поиска. [parent] — является необязательным параметром

Команда: подсчет суммы n чисел

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

Проверка доступности: Формат команды: ping id

Команда проверяет доступность конкретного узла. Если узла нет, то необходимо выводить ошибку.

Общий метод и алгоритм решения

Для реализации системы очереди сообщений используем библиотеку ZeroMQ.

Использованные системные вызовы:

1. int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout); Ожидает готовности файловых дескрипторов.
2. pid_t fork(void); Создает новый процесс.
3. int execl(const char *path, const char *arg, ...); Заменяет текущий процесс новым процессом.
4. pid_t getpid(void); Возвращает идентификатор текущего процесса.
5. void zmq_msg_init_size(zmq_msg_t *msg, size_t size); Инициализирует сообщение ZeroMQ с указанным размером.
6. int zmq_msg_send(zmq_msg_t *msg, void *socket, int flags); Отправляет сообщение через сокет ZeroMQ.
7. void zmq_msg_init(zmq_msg_t *msg); Инициализирует сообщение ZeroMQ.
8. int zmq_msg_recv(zmq_msg_t *msg, void *socket, int flags); Получает сообщение через сокет ZeroMQ.
9. void *zmq_msg_data(zmq_msg_t *msg); Возвращает указатель на данные сообщения ZeroMQ.
10. void *zmq_ctx_new(void); Создает новый контекст ZeroMQ.
11. void *zmq_socket(void *context, int type); Создает новый сокет ZeroMQ.

12. int zmq_connect(void *socket, const char *addr); Подключает сокет ZeroMQ к указанному адресу.
13. int zmq_bind(void *socket, const char *addr); Привязывает сокет ZeroMQ к указанному адресу.

Для реализации распределенной системы асинхронной обработки запросов были разработаны два исполняемых файла: **управляющий узел** (control.cpp) и **вычислительный узел** (computing.cpp). Дополнительно была создана библиотека общих функций (lib.cpp), включающая отдельные функции для отправки и приема сообщений, а также для создания дочерних процессов.

Очередь сообщений реализована с использованием библиотеки **ZeroMQ**. Сообщения передаются через сокеты типа **DEALER**, что позволяет обеспечить двустороннюю связь между управляющим и вычислительными узлами. Для обеспечения асинхронности при отправке и приеме сообщений используются флаги **ZMQ_DONTWAIT**, позволяющие выполнять операции без ожидания подтверждения.

Для стандартизации передаваемых данных создан класс message, который хранит информацию о команде (None, Ping, ExecSum), идентификаторе узла (id), числовой части данных (num), строковой части данных (st) и времени отправки сообщения (sent_time). Это позволяет эффективно управлять сообщениями и отслеживать недоставленные сообщения.

Функция createNode отвечает за создание нового вычислительного узла. Она выполняет системный вызов fork для создания дочернего процесса, а затем с помощью execl запускает исполняемый файл computing, передавая ему идентификатор нового узла. В родительском процессе инициализируется структура Node, содержащая информацию о новом узле, включая его PID и адрес сокета.

Каждый вычислительный узел хранит собственный адрес, контекст, сокет, идентификатор (id) и PID процесса. Функции send_mes и get_mes обеспечивают отправку и прием сообщений через ZeroMQ. Функция send_mes отправляет сообщение по соответствующему сокету без ожидания ответа, а функция get_mes пытается принять сообщение и возвращает его для дальнейшей обработки или сообщение с типом None, если сообщений нет.

Управляющий узел организует вычислительные узлы в **бинарное дерево поиска (BST)** для эффективного управления и поиска узлов по их идентификаторам. При отправке команды управляющий узел отправляет сообщение всем дочерним узлам и ожидает ответа хотя бы от одного из них. Каждый вычислительный узел сравнивает полученный идентификатор команды со своим id. Если id совпадает, узел выполняет соответствующую команду, иначе пересылает сообщение своим дочерним узлам.

Для проверки доступности узлов управляющий узел ведет список отправленных сообщений (saved_mes). В каждом цикле обработки управляющий узел проверяет время отправки сообщений и сравнивает его с текущим временем. Если разница превышает заданный порог (например, 5 секунд), выводится сообщение о недоступности узла. При успешном выполнении команды соответствующее сообщение удаляется из списка отправленных.

Код программы

Control.cpp

```
#include "lib.h"
#include <sstream>
#include <vector>

// Узлы в виде BST
struct BSTNode {
    int id;
    Node node;
    BSTNode *left;
    BSTNode *right;
    BSTNode(int _id, Node _n) : id(_id), node(_n), left(nullptr), right(nullptr) {}
};

BSTNode* insertBST(BSTNode* root, int id, Node node) {
    if (!root) return new BSTNode(id, node);
    if (id < root->id) root->left = insertBST(root->left, id, node);
    else if (id > root->id) root->right = insertBST(root->right, id, node);
    return root;
}

BSTNode* findBST(BSTNode* root, int id) {
    if (!root) return nullptr;
    if (id == root->id) return root;
    else if (id < root->id) return findBST(root->left, id);
    else return findBST(root->right, id);
}
```

```
std::vector<message> saved_mes;
std::vector<Node> all_nodes;

int main() {
    BSTNode *root = nullptr;

    std::string command;
    while (true) {
        // Проверка ответов от узлов
        for (auto &nd : all_nodes) {
            message m = get_mes(nd);
            if (m.command == None) continue;
            for (auto it = saved_mes.begin(); it != saved_mes.end(); ++it) {
                if (it->command == m.command && it->id == m.id) {
                    // Нашли связанное сообщение
                    if (m.command == Ping) {
                        // Узел доступен
                        std::cout << "Ok: 1" << std::endl;
                    } else if (m.command == ExecSum) {
                        // Результат суммы
                        std::cout << "Ok:" << m.id << ": " << m.num << std::endl;
                    }
                    saved_mes.erase(it);
                    break;
                }
            }
        }

        // Проверка таймаутов
        for (auto it = saved_mes.begin(); it != saved_mes.end();) {
```

```
    double diff = std::difftime(t_now(), it->sent_time);
    if (diff > 5) {
        // Таймаут
        if (it->command == Ping) {
            // Узел не ответил на Ping
            std::cout << "Ok: 0" << std::endl;
        } else if (it->command == ExecSum) {
            // Узел не ответил на Exec
            std::cout << "Error:" << it->id << ": Node is unavailable" << std::endl;
        }
        it = saved_mes.erase(it);
    } else {
        ++it;
    }
}

// Обработка команд пользователя
if (!inputAvailable()) {
    usleep(100000);
    continue;
}

std::cin >> command;
if (command == "create") {
    int id, parent_id = -1;
    std::cin >> id;
    if (std::cin.peek() != '\n') {
        std::cin >> parent_id;
    }
```

```
    if (findBST(root, id)) {
        std::cout << "Error: Already exists" << std::endl;
        continue;
    }

    // Создаём узел
    Node child = createProcess(id);
    all_nodes.push_back(child);
    root = insertBST(root, id, child);
    std::cout << "Ok: " << child.pid << std::endl;

} else if (command == "exec") {
    int id, n;
    std::cin >> id >> n;

    BSTNode* node_ptr = findBST(root, id);
    if (!node_ptr) {
        std::cout << "Error:" << id << ": Not found" << std::endl;
        // Считываем оставшиеся числа, чтобы очистить ввод
        for (int i=0; i<n; i++){int tmp; std::cin>>tmp;}
        continue;
    }

    std::ostringstream oss;
    for (int i=0; i<n; i++) {
        long long val;
        std::cin >> val;
        oss << val << " ";
    }
    std::string nums_str = oss.str();
```

```
            char buf[30];
            memset(buf,0,sizeof(buf));
            strncpy(buf, nums_str.c_str(), 29);

            message m(ExecSum, id, n, buf);
            saved_mes.push_back(m);
            send_mes(node_ptr->node, m);

        } else if (command == "ping") {
            int id;
            std::cin >> id;
            BSTNode* node_ptr = findBST(root, id);
            if (!node_ptr) {
                std::cout << "Error: Not found" << std::endl;
                continue;
            }
            message m(Ping, id, 0);
            saved_mes.push_back(m);
            send_mes(node_ptr->node, m);
        } else {
            std::cout << "Error: Command doesn't exist!" << std::endl;
        }
        usleep(100000);
    }
    return 0;
}
```

Computing.cpp

```
#include "lib.h"
#include <sstream>

int main(int argc, char *argv[])
{
    Node I = createNode(atoi(argv[1]), true);

    while (true) {
        message m = get_mes(I);
        if (m.command == None) {
            usleep(100000);
            continue;
        }

        switch (m.command) {
        case Ping:
            if (m.id == I.id) {
                send_mes(I, {Ping, I.id, 1});
            }
            break;
        case ExecSum:
        {
            if (m.id == I.id) {
                std::istringstream iss(m.st);
                int n = m.num;
                int sum = 0;
                for (int i = 0; i < n; i++) {
                    int val;
                    if (!(iss >> val)) {
                        // Предполагаем корректный ввод.
```

```
                    break;
                }
                sum += val;
            }
            // Возвращаем результат
            send_mes(I, {ExecSum, I.id, sum});
        }
    }
        break;
    default:
        break;
    }
    usleep(100000);
  }
  return 0;
}
```

Lib.cpp

```
#include "lib.h"

#include <algorithm>
#include <sys/time.h>

/*
 * Проверяем, есть ли данные в stdin (не блокируемся).
 * Возвращаем true, если можно читать из stdin без блокировок.
 */
bool inputAvailable() {
  struct timeval tv;
  tv.tv_sec = 0;
```

```
    tv.tv_usec = 0;

    fd_set read_fds;
    FD_ZERO(&read_fds);
    FD_SET(STDIN_FILENO, &read_fds);

    select(STDIN_FILENO + 1, &read_fds, nullptr, nullptr, &tv);
    return (FD_ISSET(STDIN_FILENO, &read_fds) != 0);
}

/*
 * Возвращает текущее время (time_t).
 */
std::time_t t_now() {
    return std::chrono::system_clock::to_time_t(std::chrono::system_clock::now());
}

/*
 * Создаем и настраиваем Node в текущем процессе:
 *  - Устанавливаем id, pid, is_child
 *  - Инициализируем ZMQ-сокет
 *  - Выполняем zmq_bind (если is_child=false) или zmq_connect (если is_child=true)
 */
Node createNode(int id, bool is_child) {
    Node resultNode;
    resultNode.id = id;
    resultNode.pid = getpid();
    resultNode.is_child = is_child;

    resultNode.context = zmq_ctx_new();
```

```
    resultNode.socket = zmq_socket(resultNode.context, ZMQ_DEALER);

    // Адрес для подключения/привязки
    resultNode.address = "tcp://127.0.0.1:" + std::to_string(5555 + id);

    if (is_child) {
        zmq_connect(resultNode.socket, resultNode.address.c_str());
    } else {
        zmq_bind(resultNode.socket, resultNode.address.c_str());
    }

    return resultNode;
}

/*
 * Создает новый процесс (дочерний) и в нем запускает "computing".
 * В родительском процессе инициализируем и возвращаем структуру Node (с bind).
 */
Node createProcess(int id) {
    pid_t childPid = fork();
    if (childPid == 0) {
        // Мы в дочернем процессе
        execl("./computing", "computing", std::to_string(id).c_str(), nullptr);
        // Если execl не сработал:
        std::cerr << "execl failed" << std::endl;
        _exit(1);
    } else if (childPid == -1) {
        // Не удалось вызвать fork
        std::cerr << "Fork failed" << std::endl;
        _exit(1);
```

```
    }
    // Родительский процесс
    Node newNode = createNode(id, false);
    newNode.pid = childPid;
    return newNode;
}

/*
 * Отправка сообщения m через сокет (не блокируя).
 */
void send_mes(Node &node, message m) {
    zmq_msg_t tmpMsg;
    zmq_msg_init_size(&tmpMsg, sizeof(m));
    std::memcpy(zmq_msg_data(&tmpMsg), &m, sizeof(m));

    zmq_msg_send(&tmpMsg, node.socket, ZMQ_DONTWAIT);
    zmq_msg_close(&tmpMsg);
}

/*
 * Попытка чтения сообщения из сокета (не блокируя).
 * Если нет доступных сообщений, возвращаем message(None, -1, -1).
 */
message get_mes(Node &node) {
    zmq_msg_t msgBuffer;
    zmq_msg_init(&msgBuffer);

    // Пробуем прочитать сообщение
    int msgBytes = zmq_msg_recv(&msgBuffer, node.socket, ZMQ_DONTWAIT);
    if (msgBytes == -1) {
```

```
        zmq_msg_close(&msgBuffer);
        return message(None, -1, -1);
    }

    message receivedMsg;
    std::memcpy(&receivedMsg, zmq_msg_data(&msgBuffer), sizeof(receivedMsg));

    zmq_msg_close(&msgBuffer);
    return receivedMsg;
}
```

Lib.h

```
#pragma once

#include <iostream>
#include <chrono>
#include <ctime>
#include <string>
#include <cstring>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/select.h>
#include "zmq.h"
#include <vector>

/*
 * Проверка готовности ввода с консоли (не блокирующая).
 * return true, если есть данные для чтения; иначе false.
 */
bool inputAvailable();
```

```
/*
 * Возвращает текущее время в формате time_t.
 */
std::time_t t_now();

/*
 * Возможные команды для взаимодействия между управляющим и вычислительными узлами
 */
enum com : char {
    None = 0,   // Пустое сообщение (отсутствие команды)
    Ping = 1,   // Проверка доступности узла
    ExecSum = 2 // Вычисление суммы чисел
};

/*
 * Класс для хранения сообщения, передаваемого между узлами
 */
class message {
public:
    message()
        : command(None), id(-1), num(0), sent_time(0) {
        std::memset(st, 0, sizeof(st));
    }

    message(com _cmd, int _id, int _num)
        : command(_cmd), id(_id), num(_num), sent_time(t_now()) {
        std::memset(st, 0, sizeof(st));
    }
```

```
    message(com _cmd, int _id, int _num, const char* s)
        : command(_cmd), id(_id), num(_num), sent_time(t_now()) {
        std::memset(st, 0, sizeof(st));
        std::strncpy(st, s, 29);
    }

    com command;          // Тип команды
    int id;               // Идентификатор узла, к которому обращаемся
    int num;              // Числовое поле (исп. для количества чисел или результата)
    std::time_t sent_time; // Время отправки сообщения (для таймаута)
    char st[30];          // Строковое поле для передачи набора чисел
};

/*
 * Класс, описывающий "узел" с точки зрения ZMQ-связи:
 *  - id узла,
 *  - pid процесса,
 *  - сокет и контекст ZeroMQ,
 *  - флаг is_child (дочерний/вычислительный или управляющий).
 */
class Node {
public:
    int id;           // ID узла
    pid_t pid;        // PID процесса
    void *context;    // контекст ZeroMQ
    void *socket;     // сокет ZeroMQ
    bool is_child;    // true, если это вычислительный узел
    std::string address; // адрес (tcp://127.0.0.1:порт)

    bool operator==(const Node &other) const {
```

```
        return (id == other.id && address == other.address);
    }
};

/*
 * Создать структуру Node в текущем процессе (с учетом is_child)
 * и настроить bind/connect на порт (5555 + id).
 */
Node createNode(int id, bool is_child);

/*
 * Создает новый процесс через fork() + execl("./computing", ...).
 * В родительском процессе возвращает Node (с bind), а в дочернем
 * происходит замена образа процесса на computing.
 */
Node createProcess(int id);

/*
 * Отправить сообщение m через сокет node.socket (ZMQ_DEALER, неблокирующе).
 */
void send_mes(Node &node, message m);

/*
 * Получить сообщение из сокета node.socket (ZMQ_DEALER, неблокирующе).
 * Возвращает message с command=None, если сообщений нет.
 */
message get_mes(Node &node);
```

Протокол работы программы

Тестирование:

Тест 1.

root@983c3166cd08:/workspaces/os_base/lab5/src# ./control
create 5
Ok: 1828
create 2
Ok: 1860
create 3
Ok: 1892
create 1
Ok: 1931
create 8
Ok: 1969
exec 8 3 1 2 3
Ok:8: 6
ping 8
Ok: 1
ping 5
Ok: 1
ping 10
Error: Not found

Тест 2.

root@983c3166cd08:/workspaces/os_base/lab5/src# ./control
create 5
Ok: 2354
create 2
Ok: 2383
ping 2
kill -9 2383
Ok: 0
ping 5
Ok: 1

Strace:

19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)

```
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 1 (in [0], left {tv_sec=0, tv_nsec=0})
19856 socket(AF_NETLINK, SOCK_RAW|SOCK_CLOEXEC, NETLINK_ROUTE) = 9
19856 bind(9, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12) = 0
19856 sendto(9, [{nlmsg_len=20, nlmsg_type=RTM_GETLINK, nlmsg_flags=NLM_F_REQUEST|NLM_F_DUMP, nlmsg_seq=1735248578, nlmsg_pid=0}, {ifi_family=AF_UNSPEC, ...}], 20, 0, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12) = 20
19856 sendto(9, [{nlmsg_len=20, nlmsg_type=RTM_GETADDR, nlmsg_flags=NLM_F_REQUEST|NLM_F_DUMP, nlmsg_seq=1735248579, nlmsg_pid=0}, {ifa_family=AF_UNSPEC, ...}], 20, 0, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12) = 20
19856 socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC, IPPROTO_TCP) = 9
19856 bind(9, {sa_family=AF_INET, sin_port=htons(5560), sin_addr=inet_addr("127.0.0.1")}, 16) = 0
19875 socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC, IPPROTO_TCP) = 9
19875 connect(9, {sa_family=AF_INET, sin_port=htons(5560), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operation now in progress)
19873 recvfrom(10, 0x7f33cc0012f8, 12, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)
19873 sendto(10, "\377\0\0\0\0\0\0\0\1\177", 10, 0, NULL, 0 <unfinished ...>
19875 recvfrom(9, 0x7f5a90001be8, 12, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)
19873 <... sendto resumed>)           = 10
19875 sendto(9, "\377\0\0\0\0\0\0\0\1\177", 10, 0, NULL, 0) = 10
19873 recvfrom(10,  <unfinished ...>
19875 recvfrom(9,  <unfinished ...>
19873 <... recvfrom resumed>"\377\0\0\0\0\0\0\0\1\177", 12, 0, NULL, NULL) = 10
19875 <... recvfrom resumed>"\377\0\0\0\0\0\0\0\1\177", 12, 0, NULL, NULL) = 10
19873 recvfrom(10,  <unfinished ...>
19875 recvfrom(9,  <unfinished ...>
19873 <... recvfrom resumed>0x7f33cc001302, 2, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)
19875 <... recvfrom resumed>0x7f5a90001bf2, 2, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)
19873 sendto(10, "\3", 1, 0, NULL, 0 <unfinished ...>
19875 sendto(9, "\3", 1, 0, NULL, 0 <unfinished ...>
19873 <... sendto resumed>)           = 1
19875 <... sendto resumed>)           = 1
19875 recvfrom(9,  <unfinished ...>
19873 recvfrom(10,  <unfinished ...>
19875 <... recvfrom resumed>"\3", 2, 0, NULL, NULL) = 1
19873 <... recvfrom resumed>"\3", 2, 0, NULL, NULL) = 1
19875 recvfrom(9, 0x7f5a90001bf3, 53, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)
```

19873 recvfrom(10, 0x7f33cc001303, 53, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)
19875 sendto(9, "\1NULL\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 53, 0, NULL, 0) = 53
19873 sendto(10, "\1NULL\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 53, 0, NULL, 0) = 53
19875 recvfrom(9, "\1NULL\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 53, 0, NULL, NULL) = 53
19873 recvfrom(10, <unfinished ...>
19875 recvfrom(9, <unfinished ...>
19873 <... recvfrom resumed>"\1NULL\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 53, 0, NULL, NULL) = 53
19875 <... recvfrom resumed>0x7f5a90003d18, 8192, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)
19873 recvfrom(10, <unfinished ...>
19875 sendto(9, "\4)\5READY\vSocket-Type\0\0\0\6DEALER\10I"..., 43, 0, NULL, 0 <unfinished ...>
19873 <... recvfrom resumed>0x7f33cc003b48, 8192, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)
19875 <... sendto resumed>)   = 43
19873 sendto(10, "\4)\5READY\vSocket-Type\0\0\0\6DEALER\10I"..., 43, 0, NULL, 0) = 43
19873 recvfrom(10, <unfinished ...>
19875 recvfrom(9, <unfinished ...>
19873 <... recvfrom resumed>"\4)\5READY\vSocket-Type\0\0\0\6DEALER\10I"..., 8192, 0, NULL, NULL) = 43
19875 <... recvfrom resumed>"\4)\5READY\vSocket-Type\0\0\0\6DEALER\10I"..., 8192, 0, NULL, NULL) = 43
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 pselect6(1, [0], NULL, NULL, {tv_sec=0, tv_nsec=0}, NULL) = 0 (Timeout)
19856 --- SIGINT {si_signo=SIGINT, si_code=SI_KERNEL} ---
19871 --- SIGINT {si_signo=SIGINT, si_code=SI_KERNEL} ---
19872 +++ killed by SIGINT +++
19873 +++ killed by SIGINT +++
19856 +++ killed by SIGINT +++
19874 +++ killed by SIGINT +++
19875 +++ killed by SIGINT +++
19871 +++ killed by SIGINT +++

Вывод

Очень понравилось выполнять данную лабораторную работу. Было крайне интересно изучить очереди сообщений и потратить большое число времени на отладку отправки и приему сообщений через неё. Очередь сообщений является эффективным инструментом для межпроцессорного взаимодействия, так как позволяет легко асинхронно обмениваться сообщениями и масштабировать систему, а так же синхронизировать процессы, выполняющие действия с разными скоростями, что очень важно в клиент-серверной архитектуре. ZeroMQ является удобной и эффективной библиотекой для создания пользовательских очередей сообщений.