Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №5-7 по курсу**

**«Операционные системы»**

Студент: Абдыкалыков Нурсултан Абдыкалыкович

Группа: М8О-206Б-23

Вариант: 1

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2025

**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. 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.

**Топология 3:**

Аналогично топологии 4, но узлы находятся в идеально сбалансированном бинарном дереве. Каждый следующий узел должен добавляться в самое наименьшее поддерево.

**Набора команд 4 (поиск подстроки в строке):**

Формат команды:

> exec id

> text\_string

> pattern\_string

[result] – номера позиций, где найден образец, разделенный точкой с запятой

text\_string — текст, в котором искать образец. Алфавит: [A-Za-z0-9]. Максимальная длина строки

108 символов

pattern\_string — образец

**Команда проверки 1**

Формат команды: pingall

Вывод всех недоступных узлов вывести разделенные через точку запятую. Пример:

> pingall

Ok: -1 // Все узлы доступны

> pingall

Ok: 7;10;15 // узлы 7, 10, 15 — недоступны

**Общий метод и алгоритм программы:**Менеджер управляет рабочими узлами, отправляет им команды (например, поиск подстроки или пинг), а рабочие узлы выполняют задачи и возвращают результаты.

**Алгоритм:**

1. **Менеджер**:
   * Запускает рабочих узлов и строит бинарное дерево для их распределения.
   * Принимает команды от пользователя: создание узлов, выполнение поиска или пинг.
   * Отправляет команды рабочим узлам через ZeroMQ.
2. **Рабочий узел**:
   * Принимает команды от менеджера.
   * Выполняет поиск подстроки или отвечает на пинг.
   * Отправляет результаты обратно менеджеру.

Используется ZeroMQ для обмена сообщениями между менеджером и рабочими.

**Исходный код:**

**common.h:**

|  |
| --- |
| **#ifndef COMMON\_H**  **#define COMMON\_H**  **#define PORT\_BASE 5550 //базовый порт**  **#define MAX\_WORKERS 128 //максимум рабочих узлов**  **#define PING\_TIMEOUT 1000 // ms**  **//команды которые можем отправить узлам**  **typedef enum {**  **CMD\_EXEC,**  **CMD\_PING,**  **CMD\_PING\_RESPONSE**  **} CommandType;**  **#endif** |

**Worker.c**

|  |
| --- |
| **#include <stdio.h>**  **#include <stdlib.h>**  **#include <string.h>**  **#include <zmq.h>**  **#include <unistd.h>**  **#include "common.h"**  **int find\_substrings(const char \**text*, const char \**pattern*, char \**result*) {**  **int n = strlen(*text*);**  **int m = strlen(*pattern*);**  **int found = 0;**  **char temp[10];**  ***result*[0] = '\0';**  **for (int i = 0; i <= n - m; i++) {**  **if (strncmp(&*text*[i], *pattern*, m) == 0) {**  **if (found) strcat(*result*, ";");**  **sprintf(temp, "%d", i);**  **strcat(*result*, temp);**  **found++;**  **}**  **}**  **return found;**  **}**  **int main(int *argc*, char \**argv*[]) {**  **if (*argc* != 2) {**  **printf("Usage: %s <id>\n", *argv*[0]);**  **exit(1);**  **}**  **int id = atoi(*argv*[1]);**  **void \*context = zmq\_ctx\_new();**  **void \*socket = zmq\_socket(context, ZMQ\_REP);**  **char endpoint[256];**  **sprintf(endpoint, "tcp://\*:%d", PORT\_BASE + id);**  **zmq\_bind(socket, endpoint);**  **printf("Worker %d started.\n", id);**  **while (1) {**  **char buffer[1024];**  **zmq\_recv(socket, buffer, sizeof(buffer), 0);**  **buffer[1023] = '\0';**  **int cmd;**  **sscanf(buffer, "%d", &cmd);**  **if (cmd == CMD\_EXEC) {**  **char \*text = strchr(buffer, ' ') + 1;**  **char \*pattern = strchr(text, ' ') + 1;**  **\*(pattern - 1) = '\0'; // разделяем строки**  **char result[256];**  **find\_substrings(text, pattern, result);**  **zmq\_send(socket, result, strlen(result), 0);**  **} else if (cmd == CMD\_PING) {**  **zmq\_send(socket, "PONG", 4, 0);**  **} else {**  **zmq\_send(socket, "Unknown command", 15, 0);**  **}**  **}**  **zmq\_close(socket);**  **zmq\_ctx\_destroy(context);**  **return 0;**  **}** |

**Manager.c**

|  |
| --- |
| **#include <stdio.h>**  **#include <stdlib.h>**  **#include <string.h>**  **#include <zmq.h>**  **#include <unistd.h>**  **#include <sys/wait.h>**  **#include "common.h"**  **typedef struct Node {**  **int id;**  **struct Node \*left;**  **struct Node \*right;**  **} Node;**  **Node \*root = NULL;**  **Node\* nodes[MAX\_WORKERS] = {0};**  **void \*context;**  **int get\_size(Node\* *node*) {**  **if (!*node*) return 0;**  **return 1 + get\_size(*node*->left) + get\_size(*node*->right);**  **}**  **void add\_node(int *id*) {**  **Node\* new\_node = (Node\*)malloc(sizeof(Node));**  **new\_node->id = *id*;**  **new\_node->left = NULL;**  **new\_node->right = NULL;**  **nodes[*id*] = new\_node;**  **if (!root) {**  **root = new\_node;**  **return;**  **}**  **Node\* current = root;**  **while (1) {**  **int left\_size = get\_size(current->left);**  **int right\_size = get\_size(current->right);**  **if (left\_size <= right\_size) {**  **if (!current->left) {**  **current->left = new\_node;**  **break;**  **}**  **current = current->left;**  **} else {**  **if (!current->right) {**  **current->right = new\_node;**  **break;**  **}**  **current = current->right;**  **}**  **}**  **}**  **void start\_worker(int *id*) {**  **pid\_t pid = fork();**  **if (pid == 0) {**  **char id\_str[10];**  **sprintf(id\_str, "%d", *id*);**  **execl("./worker", "./worker", id\_str, NULL);**  **perror("execl");**  **exit(1);**  **}**  **}**  **void print\_tree(Node\* *node*, int *level*) {**  **if (!*node*) return;**  **print\_tree(*node*->right, *level* + 1);**  **for (int i = 0; i < *level*; i++) printf("    ");**  **printf("%d\n", *node*->id);**  **print\_tree(*node*->left, *level* + 1);**  **}**  **void send\_command(int *id*, CommandType *cmd*, const char \**payload*) {**  **void \*socket = zmq\_socket(context, ZMQ\_REQ);**  **char endpoint[256];**  **sprintf(endpoint, "tcp://localhost:%d", PORT\_BASE + *id*);**  **zmq\_connect(socket, endpoint);**  **int timeout = 1000; // 1 секунда**  **zmq\_setsockopt(socket, ZMQ\_RCVTIMEO, &timeout, sizeof(timeout));**  **char message[512];**  **if (*payload*)**  **sprintf(message, "%d %s", *cmd*, *payload*);**  **else**  **sprintf(message, "%d", *cmd*);**  **zmq\_send(socket, message, strlen(message), 0);**  **char buffer[1024];**  **int rc = zmq\_recv(socket, buffer, 1024, 0);**  **if (rc == -1) {**  **printf("Node %d is unavailable.\n", *id*);**  **} else {**  **buffer[rc] = '\0';**  **printf("Ok:%d: %s\n", *id*, buffer);**  **}**  **zmq\_close(socket);**  **}**  **void pingall() {**  **int unavailable[MAX\_WORKERS] = {0};**  **int failed\_count = 0;**  **for (int id = 1; id < MAX\_WORKERS; id++) {**  **if (nodes[id]) {**  **void \*socket = zmq\_socket(context, ZMQ\_REQ);**  **char endpoint[256];**  **sprintf(endpoint, "tcp://localhost:%d", PORT\_BASE + id);**  **zmq\_connect(socket, endpoint);**  **int timeout = 500;**  **zmq\_setsockopt(socket, ZMQ\_RCVTIMEO, &timeout, sizeof(timeout));**  **zmq\_send(socket, "1", 1, 0); // CMD\_PING**  **char buffer[256];**  **int rc = zmq\_recv(socket, buffer, 256, 0);**  **if (rc == -1) {**  **unavailable[failed\_count++] = id;**  **}**  **zmq\_close(socket);**  **}**  **}**  **if (failed\_count == 0) {**  **printf("Ok: -1\n");**  **} else {**  **printf("Ok: ");**  **for (int i = 0; i < failed\_count; i++) {**  **printf("%d", unavailable[i]);**  **if (i < failed\_count - 1)**  **printf(";");**  **}**  **printf("\n");**  **}**  **}**  **int main() {**  **context = zmq\_ctx\_new();**  **printf("Manager started.\n");**  **char command[1024];**  **while (1) {**  **printf("> ");**  **fflush(stdout);**  **if (!fgets(command, sizeof(command), stdin))**  **break;**  **command[strcspn(command, "\n")] = 0;**  **if (strncmp(command, "create", 6) == 0) {**  **int id;**  **if (sscanf(command + 7, "%d", &id) == 1) {**  **add\_node(id);**  **printf("Tree structure:\n");**  **print\_tree(root, 0);**  **start\_worker(id);**  **printf("Node %d created.\n", id);**  **} else {**  **printf("Invalid create command.\n");**  **}**  **} else if (strncmp(command, "exec", 4) == 0) {**  **int id;**  **char text[128], pattern[128];**  **if (sscanf(command + 5, "%d %s %s", &id, text, pattern) == 3) {**  **char payload[300];**  **snprintf(payload, sizeof(payload), "%s %s", text, pattern);**  **send\_command(id, CMD\_EXEC, payload);**  **} else {**  **printf("Usage: exec <id> <text> <pattern>\n");**  **}**  **} else if (strncmp(command, "pingall", 7) == 0) {**  **pingall();**  **} else if (strncmp(command, "exit", 4) == 0) {**  **break;**  **} else {**  **printf("Unknown command.\n");**  **}**  **}**  **zmq\_ctx\_destroy(context);**  **return 0;**  **}** |

**Вывод:**

В ходе выполнения данной лабораторной работы я познакомился с библиотекой ZeroMQ, которая является эффективным инструментом для организации взаимодействия между различными компонентами приложения через систему очередей сообщений. Я изучил основные принципы работы с очередями, а также освоил методы отправки и получения сообщений, что позволило мне лучше понять механизмы асинхронного взаимодействия в распределённых системах.