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»**

Группа:М8О-215Б-23

Студент: Венгер Ирина Витальевна

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2024.

# Содержание

1. Постановка задачи.
2. Общие сведения о программе.
3. Общий метод и алгоритм решения.
4. Код программы.
5. Демонстрация работы программы.
6. Вывод.

## Постановка задачи

**Тема:** Управление серверами сообщений и организация распределённых вычислений

**Цель работы:** Целью лабораторной работы являлось приобретение практических навыков в:

* управлении серверами сообщений;
* применении отложенных вычислений;
* интеграции программных систем друг с другом.

**Вариант:** 30 (бинарное дерево поиска, поиск подстроки, pingall).

## Общие сведения о программе

В данной лабораторной работе я разработала распределённую систему для асинхронной обработки запросов с применением очередей сообщений, организовала взаимодействие узлов в виде бинарного дерева поиска, обеспечила обработку ошибок и проверку доступности узлов и реализовала следующие команды:

* добавление нового вычислительного узла;
* выполнение вычислений на узле (поиск подстроки в строке);
* проверка доступности узлов.

## Общий метод и алгоритм решения

Программа была разработана на языке C с использованием библиотеки ZeroMQ для организации взаимодействия между процессами. Система состоит из следующих основных модулей:

1. **Менеджер (manage\_node):**
   * Принимает команды от пользователя.
   * Создаёт новые вычислительные узлы, добавляя их в бинарное дерево поиска.
   * Отправляет команды узлам и обрабатывает ответы.
   * Реализует асинхронное выполнение команд.
2. **Вычислительные узлы (calc\_node):**
   * Каждый вычислительный узел создаётся в отдельном процессе с помощью системного вызова fork().
   * Обрабатывают команды на поиск подстроки в строке.
   * Отвечают на команду "exec", выполняющую поиск подстроки в строке.
   * Отвечают на запросы "ping", подтверждая свою доступность.
3. **Процесс взаимодействия:**
   * Менеджер создаёт процесс узла, передавая ему идентификатор и порт для взаимодействия через ZeroMQ.
   * Команды, такие как "exec" и "ping", передаются через очереди сообщений, а ответы возвращаются менеджеру.
   * Узлы поддерживают механизм связи с другими процессами узлов, что позволяет проверять доступность и взаимодействовать в рамках дерева поиска.
4. **Механизм проверки доступности (pingall):**
   * Рекурсивно проверяет все узлы дерева.
   * Выводит список недоступных узлов.
5. **Обработка ошибок:**
   * Проверка существования узлов, доступности родительских узлов, корректности входных данных.
   * Обработка сбоев связи между узлами и контроллером.

Программа поддерживает следующие команды:

1. **create id [parent]** — создание нового узла с указанным идентификатором. Ввиду использования бинарного дерева в качестве топологии параметр parent является необязательным.
   * Пример:

> create 6

Ok: 1234

1. **exec id text pattern** — выполнение команды поиска подстроки в некоторой строке на указанном узле.
   * Пример:

> exec 6

abracadabra

abra

Ok: 6: 0;7

## Код программы

Код программы main.c смотрите в приложении 1.

## Демонстрация работы программы

make run\_5

> create 2

Ok: 4748

> create 5 2

Ok: 4752

> create 6

Ok: 4756

> pingall

Ok: -1

> exec 6

> abracadabra

> abra

> Ok: 6: 0;7

> exit

## Вывод

В ходе выполнения работы были достигнуты все поставленные цели. Реализованная распределённая система корректно выполняет задачи асинхронной обработки запросов, поддерживает заданную топологию взаимодействия и обеспечивает устойчивость при сбоях. Программа протестирована в операционной системе Linux и показала стабильную работу. Получены практические навыки работы с библиотекой ZeroMQ, управления процессами и организации взаимодействия между процессами.

## Приложения

Приложение 1 – код программы:

main.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <signal.h>

#include <unistd.h>

#include "include/calc\_node.h"

#include "include/manage\_node.h"

#include "include/tree.h"

#include "include/message.h"

#include "include/upcoming\_operations.h"

#include <sys/wait.h>

#define MAX\_CMD\_LENGTH 1024

//TreeNode\* root = NULL;

//UpcomingOperation\* upcoming\_operations = NULL;

void handle\_signal(int signal) {

if (root != NULL) {

kill\_tree(root);

free\_tree(root);

}

exit(0);

}

int main() {

signal(SIGINT, handle\_signal);

signal(SIGTERM, handle\_signal);

char command[MAX\_CMD\_LENGTH];

char text[MAX\_TEXT\_LENGTH];

char pattern[MAX\_TEXT\_LENGTH];

int node\_id, parent\_id;

pid\_t process\_id;

zmq\_pollitem\_t poll\_items[1];

poll\_items[0].socket = NULL;

poll\_items[0].fd = STDIN\_FILENO;

poll\_items[0].events = ZMQ\_POLLIN;

printf("> ");

fflush(stdout);

while (1) {

int poll\_result = zmq\_poll(poll\_items, 1, 100);

cleanup\_operations();

check\_responses();

check\_process\_status(root);

if (poll\_result == 0) {

continue;

}

if (poll\_items[0].revents & ZMQ\_POLLIN) {

if (fgets(command, MAX\_CMD\_LENGTH, stdin) == NULL) {

break;

}

command[strcspn(command, "\n")] = 0;

if (sscanf(command, "create %d %d", &node\_id, &parent\_id) == 2) {

create\_calc\_node(node\_id, parent\_id);

} else if (sscanf(command, "create %d", &node\_id) == 1) {

create\_calc\_node(node\_id, -1);

} else if (strcmp(command, "print") == 0) {

print\_tree(root);

} else if (strcmp(command, "pingall") == 0) {

ping\_all();

} else if (sscanf(command, "exec %d", &node\_id) == 1) {

printf("> ");

if (fgets(text, MAX\_TEXT\_LENGTH, stdin) == NULL) {

break;

}

text[strcspn(text, "\n")] = 0;

printf("> ");

if (fgets(pattern, MAX\_TEXT\_LENGTH, stdin) == NULL) {

break;

}

pattern[strcspn(pattern, "\n")] = 0;

exec(node\_id, text, pattern);

} else if (sscanf(command, "kill %d", &process\_id) == 1) {

if (kill(process\_id, SIGTERM) == 0) {

printf("Ok: Process %d killed\n", process\_id);

TreeNode\* node = find\_node\_by\_pid(root, process\_id);

if (node) {

mark\_node\_unavailable(node->id);

} else {

printf("Error: Node with PID %d not found\n", process\_id);

}

} else {

printf("Error: Failed to kill process %d\n", process\_id);

}

} else if (strcmp(command, "exit") == 0) {

break;

} else {

printf("Error: Unknown command\n");

}

printf("> ");

fflush(stdout);

}

}

return 0;

}