МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 "Компьютерные науки и прикладная математика"

Кафедра 806 "Вычислительная математика и программирование"

Лабораторные работы №5-7

По курсу «Операционные системы»

Студент: Лернер Ф. Л.

Группа: М8О-208Б-23

Преподаватель: Живалев Е. А.

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**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:** 22 (бинарное дерево поиска, поиск подстроки в строке, ZeroMQ).

**Задачи работы:**

1. Реализовать распределённую систему асинхронной обработки запросов с использованием технологии очередей сообщений.
2. Создать топологию взаимодействия узлов в виде бинарного дерева поиска.
3. Предусмотреть обработку ошибок и проверку доступности узлов.
4. Реализовать команды:
   * создание нового вычислительного узла;
   * выполнение вычислений на узле;
   * проверка доступности узлов.

**Описание решения:** Программное решение реализовано на языке C++ с использованием библиотеки ZeroMQ для межпроцессного взаимодействия. Основные модули системы:

1. **Контроллер:**
   * 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:**
   * Каждое вычислительный узел создаётся в отдельном процессе с помощью системного вызова fork().
   * Обрабатывают команды на выполнение арифметических операций.
   * Реализуют команду "exec", выполняющую подсчёт суммы заданного количества чисел.
   * Отвечают на запросы "ping", подтверждая свою доступность.
3. **Процесс взаимодействия:**
   * Контроллер создаёт процесс узла, передавая ему идентификатор и порт для взаимодействия через ZeroMQ.
   * Команды, такие как "exec" и "ping", передаются через очереди сообщений ZeroMQ в формате строк, а ответы возвращаются обратно в контроллер.
   * Узлы поддерживают механизм связи с другими процессами узлов, что позволяет проверять доступность и взаимодействовать в рамках дерева поиска.
4. **Механизм проверки доступности:**
   * 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.

Пример реализации некоторых функций из программы:

#include <iostream>

#include <string>

#include "NodeManager.h"

int main() {

NodeManager manager; // testMode = false по умолчанию

std::string command;

while (true) {

std::cout << "> ";

std::getline(std::cin, command);

if (command == "exit") {

break;

}

if (command.rfind("create", 0) == 0) {

int id, parentId = -1;

sscanf(command.c\_str(), "create %d %d", &id, &parentId);

std::cout << manager.createNode(id, parentId) << std::endl;

}

else if (command.rfind("exec", 0) == 0) {

int id;

std::string text, pattern;

sscanf(command.c\_str(), "exec %d", &id);

std::cout << "Enter text string: ";

std::getline(std::cin, text);

std::cout << "Enter pattern string: ";

std::getline(std::cin, pattern);

std::cout << manager.execCommand(id, text, pattern) << std::endl;

}

else if (command.rfind("heartbit", 0) == 0) {

int interval;

sscanf(command.c\_str(), "heartbit %d", &interval);

manager.startHeartbit(interval);

std::cout << "Ok" << std::endl;

}

else if (command.rfind("ping", 0) == 0) {

int id;

sscanf(command.c\_str(), "ping %d", &id);

std::cout << manager.pingNode(id) << std::endl;

}

else {

std::cout << "Unknown command" << std::endl;

}

}

return 0;

}

**Репозиторий:** https://github.com/LernerF/labs\_os

**Исходный код:** Программа состоит из следующих файлов:

* main.cpp: точка входа, инициализация контроллера.
* controller.cpp: управление взаимодействием с пользователем и узлами.
* worker.cpp: реализация вычислительных узлов.
* tools.cpp: вспомогательные функции для работы с деревом узлов и проверкой доступности.

**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 межпроцессного взаимодействия.