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: Татаркин И. В.

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:** 36 (бинарное дерево поиска, локальный целочисленный словарь, pingall).

**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;
   * создание словаря 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 (Controller):**
   * 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 (Worker):**
   * Каждое вычислительный узел создаётся в отдельном процессе с помощью системного вызова fork().
   * Обрабатывают команды на создание словарей.
   * Реализуют команду "exec", выполняющую вывод нужных значений словаря
   * 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 (Ping):**
   * 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:

std::string Exec(const std::shared\_ptr<Node>& node, int id, const std::string& command\_str) {

std::string result;

try {

zmq::message\_t message(command\_str.data(), command\_str.size());

node->socket.send(message, zmq::send\_flags::none);

zmq::message\_t reply;

auto recvRes = node->socket.recv(reply, zmq::recv\_flags::none);

if (!recvRes.has\_value()) {

result = "Error:" + std::to\_string(id) + ": No response\n";

} else {

result = reply.to\_string() + "\n";

}

} catch (zmq::error\_t&) {

result = "Error:" + std::to\_string(id) + ": Node is unavailable\n";

} catch (std::exception& e) {

result = "Error:" + std::to\_string(id) + ": " + e.what() + "\n";

}

return result;

}

bool InsertNode(std::shared\_ptr<Node>& root, int id, int pid) {

if (!root) {

try {

root = std::make\_shared<Node>(id, pid);

} catch (zmq::error\_t&) {

return false;

}

return true;

}

if (id == root->id) {

return false; // уже есть

}

if (id < root->id) {

return InsertNode(root->left, id, pid);

} else {

return InsertNode(root->right, id, pid);

}

}

**Команды программы:** Программа поддерживает следующие команды:

1. **create id [parent]** — создание нового узла с указанным идентификатором.
   * Пример: create 10 5 -> "Ok: pid"
2. **exec id MyVar n** — создание словаря на указанном узле.
   * Пример: exec 10 x 4 -> "Ok:10: 6"
3. **pingall** — проверка доступности всех узлов.
   * Пример: pingall -> "Ok: -1" (все узлы доступны).

**Репозиторий:** https://github.com/IvanTatarkin/OS-Labs/blob/LW1/LW567

**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.

**Пример работы:**

./lab567

> create 5

About to fork for node 5

Parent process, child pid: 157149

Ok: 157149

> Child process for node 5 started

> create 5 6

Error: Node with id 5 already exists

> create 6 5

About to fork for node 6

Parent process, child pid: 157268

Ok: 157268

> Child process for node 6 started

> exec 5 re 3

Ok

> exec 5 re

Ok: 3

> pingall

Ok: 5;6

>

**Вывод:** В процессе выполнения работы удалось достичь всех поставленных целей. Созданная распределённая система успешно справляется с задачами асинхронной обработки запросов, поддерживает заданную структуру взаимодействия и демонстрирует устойчивость при сбоях. Программа была протестирована в среде Linux и показала стабильные результаты. В ходе работы приобретены практические навыки использования библиотеки ZeroMQ, организации взаимодействия между процессами и управления их выполнением.