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: Попов А. Д.

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

**Задачи работы:**

1. Разработать распределённую систему для асинхронной обработки запросов с применением очередей сообщений.
2. Организовать взаимодействие узлов в виде бинарного дерева поиска.
3. Обеспечить обработку ошибок и проверку доступности узлов.
4. Реализовать следующие команды:

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

**Описание 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.

Пример реализации некоторых функций из программы:

TreeNode\* find\_node(TreeNode\* root, int node\_id) {

if (root == NULL || root->id == node\_id) {

return root;

}

//printf("Searching for node %d in subtree of node %d\n", node\_id, root->id);

if (node\_id < root->id) {

return find\_node(root->left, node\_id);

}

return find\_node(root->right, node\_id);

}

TreeNode\* insert\_node(TreeNode\* root, int node\_id, pid\_t process\_id, const char\* endpoint) {

if (root == NULL) {

//printf("Creating new root node with id %d\n", node\_id);

return create\_node(node\_id, process\_id, endpoint);

}

if (node\_id < root->id) {

//printf("Inserting node %d to the left of node %d\n", node\_id, root->id);

root->left = insert\_node(root->left, node\_id, process\_id, endpoint);

} else if (node\_id > root->id) {

//printf("Inserting node %d to the right of node %d\n", node\_id, root->id);

root->right = insert\_node(root->right, node\_id, process\_id, endpoint);

}

return root;

}

Пример работы функций: создание нового узла и поиск узла в бинарном.

**Команды программы:** 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

1. **pingall** — проверка доступности всех узлов.
   * Пример:

> pingall

Ok: -1 (все узлы доступны)

Ok: 5 (узел 5 недоступен)

**Репозиторий:** https://github.com/aldpopov/OS\_labs/tree/master/LW5-7

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

* main.c: Точка входа, инициализация менеджера и обработка команд пользователя.
* manage\_node.c: Управление взаимодействием с пользователем и узлами.
* calc\_node.c: Реализация вычислительных узлов.
* tree.c: Реализация бинарного дерева поиска.
* message.c: Реализует функции для отправки и получения сообщений между узлами через ZeroMQ.
* upcoming\_operations.c: Управляет очередью предстоящих операций, включая их добавление, очистку и проверку ответов от узлов.

**Пример работы:**

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.