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: Григорьев Т. А.

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:** 25 (дерево общего вида, локальный целочисленный словарь, heartbeat).

**Задачи работы:**

1. Реализовать распределённую систему асинхронной обработки запросов с использованием технологии очередей сообщений.
2. Создать топологию взаимодействия узлов в виде дерева общего вида.
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 организации межпроцессного взаимодействия. Основные модули системы:

1. **Клиент (Client):**
   * **Функции:**
     + Принимает команды от контроллера.
     + Создаёт новые вычислительные узлы, добавляя их в структуру дерева.
     + Обрабатывает команды на выполнение арифметических операций и проверку доступности.
     + Реализует механизмы создания и завершения процессов узлов.
   * **Структуры данных:**
     + Child: структура для хранения информации о дочерних узлах, включая идентификатор, PID и сокет ZeroMQ.
     + std::map<std::string, int>: хранение пар "имя-значение" для реализации целочисленного словаря
2. **Главный процесс (Main):**
   * **Функции:**
     + Инициализирует контроллер и управляет взаимодействием с пользователем.
     + Обрабатывает команды создания, удаления, выполнения операций, пинга и проверки состояния узлов.
     + Управляет деревом узлов через структуру Tree.
     + Реализует механизм heartbeat для проверки доступности узлов.
   * **Структуры данных:**
     + Tree: структура для хранения и управления деревом вычислительных узлов.
     + ChildNode: структура для хранения информации о дочерних узлах, аналогично клиенту.
3. **Дерево узлов (Tree):**
   * **Функции:**
     + Добавление узлов в дерево.
     + Удаление узлов из дерева.
     + Получение списка всех узлов для выполнения операций и проверки состояния.
   * **Структуры данных:**
     + Node: структура для представления узла дерева, содержащая идентификатор, список дочерних узлов и флаг нахождения.

**Процесс взаимодействия:**

* Главный процесс (контроллер) принимает команды от пользователя и выполняет соответствующие действия.
* При создании нового узла контроллер форкает новый процесс клиента, который связывается с контроллером через ZeroMQ на определённом порту.
* Команды, такие как exec, ping и heartbeat, отправляются через очереди сообщений ZeroMQ в формате строк, а ответы возвращаются обратно в контроллер.
* Узлы могут создавать дочерние узлы, формируя структуру дерева для распределённой обработки задач.

**Механизм проверки доступности (Ping и Heartbeat):**

* **Ping:**
  + Отправляет команду ping на указанный узел.
  + Получает подтверждение доступности или ошибку недоступности.
* **Heartbeat:**
  + Периодически отправляет команду heartbeat на все узлы с указанным интервалом времени.
  + Определяет доступность узлов на основе полученных ответов.

**Обработка ошибок:**

* Проверка корректности вводимых команд и их аргументов.
* Обработка ошибок при форке процессов и установлении соединений через ZeroMQ.
* Управление неудачными попытками отправки или получения сообщений.
* Обработка ситуаций, когда узел недоступен или не отвечает на команды.

**Пример реализации некоторых функций из программы:**

*client.cpp:*

bool send\_message(zmq::socket\_t& socket, const string& message\_string) {

zmq::message\_t message(message\_string.size());

memcpy(message.data(), message\_string.c\_str(), message\_string.size());

auto result = socket.send(message, zmq::send\_flags::none);

return result.has\_value();

}

string receive\_message(zmq::socket\_t& socket) {

zmq::message\_t message;

bool ok = false;

try {

auto result = socket.recv(message, zmq::recv\_flags::none);

ok = result.has\_value();

} catch (...) {

ok = false;

}

string received\_message(static\_cast<char\*>(message.data()), message.size());

if (received\_message.empty() || !ok) {

return "";

}

return received\_message;

}

*main.cpp:*

void create\_node(int id, int port) {

char\* arg0 = strdup("./client");

char\* arg1 = strdup((to\_string(id)).c\_str());

char\* arg2 = strdup((to\_string(port)).c\_str());

char\* args[] = {arg0, arg1, arg2, NULL};

execv("./client", args);

}

*tree.cpp:*

void Tree::push(int id) {

root = push(root, id);

}

Node\* Tree::push(Node\* root, int val) {

if (root == NULL) {

root = new Node;

root->id = val;

root->found = false;

return root;

} else {

root->children.push\_back(push(NULL, val));

}

return root;

}

**Пример работы функций:** Создание нового узла, выполнение команды exec, проверка доступности узлов с помощью ping и heartbeat.

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

1. create <id> — создание нового узла с указанным идентификатором.
   * **Пример:** create 123 -> "Ok"
2. exec <id> <name> <value> — выполнение команды обновления значения на указанном узле.
   * **Пример:** exec 123 sum 10 -> "Ok:123:10"
3. ping <id> — проверка доступности указанного узла.
   * **Пример:** ping 123 -> "Ok: 1"
4. heartbeat <ping\_time> — периодическая проверка доступности всех узлов с заданным интервалом времени.
   * **Пример:** heartbeat 1000 -> "Node 123 is available."
5. exit — завершение работы программы.
   * **Пример:** exit -> Завершение всех процессов и выход.

**Репозиторий:** https://github.com/timofeez/os-labs/tree/main/05-07

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

* client.cpp: реализация клиентских узлов, обработка команд и взаимодействие с контроллером.
* main.cpp: точка входа, инициализация контроллера, обработка пользовательских команд.
* tree.cpp: реализация структуры дерева для управления узлами.
* tree.hpp: определение структуры Tree и Node.

**Пример работы:**

$ ./main

Commands:

1. create (id)

2. exec (id) (name, value)

3. ping (id)

4. heartbeat (ping\_time)

5. exit

create 123

Ok

ping 123

Ok: 1

exec 123 sum 15

Ok:123:15

ping 124

Error: Not found

heartbeat 1000

Node 123 is available.

exit

**Вывод:** В ходе выполнения работы были достигнуты все поставленные цели. Реализованная распределённая система корректно обрабатывает команды асинхронной обработки запросов, поддерживает заданную иерархию взаимодействия узлов и обеспечивает устойчивость при сбоях отдельных компонентов. Программа протестирована в операционной системе Linux и продемонстрировала стабильную работу. Получены практические навыки работы с библиотекой ZeroMQ, управления процессами и организации межпроцессного взаимодействия, а также разработки структур данных для управления распределёнными системами.