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»**

**УПРАВЛЕНИЕ СЕРВЕРАМИ СООБЩЕНИЙ, ПРИМЕНЕНИЕ ОТЛОЖЕННЫХ ВЫЧИСЛЕНИЙ, ИНТЕГРАЦИЯ ПРОГРАММНЫХ СИСТЕМ ДРУГ С ДРУГОМ**

Студент: Куценко Максим Дмитриевич

Группа: М8О–212Б-22

Вариант: 31

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Управлении серверами сообщений (No5)
* Применение отложенных вычислений (No6)
* Интеграция программных систем друг с другом (No7)

## Задание

Реализовать распределенную систему по асинхронной обработке запросов. В данной распределенной системе должно существовать 2 вида узлов: «управляющий» и «вычислительный». Необходимо объединить данные узлы в соответствии с той топологией, которая определена вариантом. Связь между узлами необходимо осуществить при помощи технологии очередей сообщений. Также в данной системе необходимо предусмотреть проверку доступности узлов в соответствии с вариантом. При убийстве («kill -9») любого вычислительного узла система должна пытаться максимально сохранять свою работоспособность, а именно все дочерние узлы убитого узла могут стать недоступными, но родительские узлы должны сохранить свою работоспособность.

**Вариант 31:**

Топология 1: Узлы находятся в списке.

Набор команд 4: Поиск подстроки в строке.

Команда проверки 3: heartbit time.

**Общие сведения о программе**

Основной файл программы (управляющий узел) - control.cpp, calc.cpp – вычислительный узел, передающий программы от управляющего и выполняющий команды. Также используется заголовочные файлы: iostream, vector, zmq.hpp, zmq\_addon.hpp, list.

В программе используются следующие системные вызовы:

1. **fork** — создание дочернего процесса.
2. **execl** — запуск исполняемого файла.

В программе работа с очередью сообщений происходит с помощью:

1. **context** – потокобезопасный класс представляющий внутренний контекст ZMQ.
2. **socket** – класс, осуществляющий связь между программами и обмен данными между ними.
3. **socket.bind()** – связывается с портом.
4. **socket.unbind()** – прерывает связь с портом.
5. **socket.connect()** – подключается к порту.
6. **socket.disconnect() –** отключается от порта.

В программе вместо стандартных zmq::message\_t используются zmq::multipart\_t — композитное сообщение, в которое можно поместить различные данные. Соответственно, пересылка такого сообщения происходит через msg.send(zmq::socket\_ref socket) и smg.rcv(zmq::socket\_ref socket).

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы с ZeroMQ.
2. Написать контейнер для id узлов.
3. Написать код для управляющего узла, который будет принимать от пользователя данные и отправлять их дочерним узлам по цепочке.
4. Написать код для дочерних узлов, убедиться что сообщения правильно передаются между узлами.

**Результаты работы**

> create 100 -1

OK:13445

> create 101 100

OK:13456

> create 102 100

Node already has a child!

> kill 101

OK

> create 102 100

OK:13616

> heartbit 2000

OK

> kill 100

OK

> heartbit 2000

> create 100 -1

OK:13856

> create 110 -1

OK:13865

> create 120 -1

OK:13869

> create 130 -1

OK:13872

> create 140 -1

OK:13875

> create 150 -1

OK:13879

> heartbit 200

OK

OK

OK

OK

OK

OK

> create 150 100

Error: Already exists

> create 102 102

Error: Parent not found

> create 111 110

OK:14356

> create 112 111

OK:14378

> create 113 112

OK:14407

> exec 113

> aaaaabaab

> ab

OK:113:4;7;

> exec 115

Error: id 115 not found

> kill 112

OK

> create 113 111

OK:14886

> exit

[Maxim@HONOR-MB15 src]$

**Вывод**

Научился работать с очередями сообщений на основе ZeroMQ для C++. Наибольшую сложность представило обеспечивание правильной пересылки сообщений по цепочкам от управляющего узла к вычислительному и обратно. После обеспечения правильного создания узлов по цепочке оставшаяся часть лабораторной работы не представила особой сложности.

Структура zmq::multipart\_t оказалась крайне удобной в применении и без неё выполнение лабораторной работы было бы сложнее.