**Отчёт по лабораторной работе №6**

**по дисциплине “Операционные системы”**

Работу выполнил(а) студент(ка) группы М8О-206Б-20

Гаврилов Артём Алексеевич, № по списку: 4

Работа сдана: 25 декабря 2021 г.

Преподаватель: Соколов А.А.

Итоговая оценка: 4.

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_.

1. **Тема.**

Сервера. Отложенные вычисления.

1. **Цель работы.**

Целью является приобретение практических навыков в:

* управлении серверами сообщений;
* применение отложенных вычислений;
* интеграция программных систем друг с другом.

1. **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.

Управляющий узел отвечает за ввод команд от пользователя и отправку этих команд на вычислительные узлы.

Список основных поддерживаемых команд:

* Создание нового вычислительного узла

Формат команды: create id [parent]

id – целочисленный идентификатор нового вычислительного узла

parent – целочисленный идентификатор родительского узла. Если топологией не предусмотрено введение данного параметра, то его необходимо игнорировать (если его ввели)

* Исполнение команды на вычислительном узле

Формат команды: exec id [params]

id – целочисленный идентификатор вычислительного узла, на который отправляется команда

**Вариант № 22.**

Топология:

**3.** Бинарное дерево поиска.

Тип команд:

**4.** Поиск подстроки в строке.

Формат команды: exec id string pattern\_string ,

id – целочисленный идентификатор вычислительного узла, на который отправляется команда,

string – строка в которой нужно найти все вхождения подстроки.

pattern\_string – заданная подстрока.

Тип проверки доступности узлов:

**3.** heartbeat time

Формат команды: heartbeat time

Каждый узел начинает сообщать раз в time миллисекунд о том, что он работоспособен. Если от узла нет сигнала в течении 4\*time миллисекунд, то должна выводится пользователю строка: «Heartbit: node id is unavailable now», где id – идентификатор недоступного вычислительного узла.

1. **Описание программы.**

Система состоит из двух программ: сервер и клиент. Используется сервер сообщений ZeroMQ. Обе программы используют заголовочный файл wrap\_zmq.h, который содержит обертки над вызовами

В программе используются следующие системные вызовы:

1. fork –– создает новый процесс, который является копией родительского процесса, за исключением разных process ID и parent process ID. В случае успеха fork() возвращает 0 для ребенка, число больше 0 для родителя – child ID, в случае ошибки возвращает -1.
2. execl — используется для выполнения другой программы. Эта другая программа, называемая процессом-потомком (child process), загружается поверх программы, содержащей вызов exec. Имя файла, содержащего процесс-потомок, задано с помощью первого аргумента. Какие-либо аргументы, передаваемые процессу-потомку, задаются либо с помощью параметров от arg0 до argN, либо с помощью массива arg[].

Также были использованы следующие вызовы из библиотеки ZMQ:

1. zmq\_ctx\_new –– создает новый контекст ZMQ.
2. zmq\_ctx\_destroy –– уничтожает контекст ZMQ.
3. zmq\_connect — создает исходящее соединение из сокет.
4. zmq\_disconnect — отсоединяет сокет от заданного endpoint’a.
5. zmq\_bind — создаёт входящее соединение на сокет.
6. zmq\_unbind — отсоединяет сокет от заданного endpoint’a.
7. zmq\_socket — создает ZMQ сокет.
8. zmq\_setsockopt — задает параметры ZMQ сокета.
9. zmq\_close — закрывает ZMQ сокет.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы с ZMQ.
2. Проработать принцип общения между клиентскими узлами и между первым клиентом и сервером и алгоритм выполнения команд клиентами.
3. Реализовать необходимые функции-обертки над вызовами функций библиотеки ZMQ.
4. Написать программу сервера и клиента
5. **Тестирование.**

./server

9874 server started correctly!

9878: Client started. Id:0

create 4

OK:9886

9886: Client started. Id:4

create 2

OK:9889

9889: Client started. Id:2

create 7

OK:9892

9892: Client started. Id:7

heartbit 10000

OK

OK

OK

OK

exec 2

abcabc

abc

OK:2:0:3

Heartbit: node 4 is unavailable now

OK

OK

OK

heartbit

OK

remove 4

unbind: ipc:///tmp/child\_pub\_left\_9886

disconnect: ipc:///tmp/parent\_pub\_9886

unbind: ipc:///tmp/child\_pub\_left\_9892

unbind: ipc:///tmp/child\_pub\_left\_9889

OK

unbind: ipc:///tmp/child\_pub\_right9886

unbind: ipc:///tmp/child\_pub\_right9892

unbind: ipc:///tmp/child\_pub\_right9889

unbind: ipc:///tmp/parent\_pub\_9886

unbind: ipc:///tmp/parent\_pub\_9892

unbind: ipc:///tmp/parent\_pub\_9889

disconnect: ipc:///tmp/child\_pub\_right9878

disconnect: ipc:///tmp/child\_pub\_right9886

disconnect: ipc:///tmp/child\_pub\_left\_9886

disconnect: ipc:///tmp/parent\_pub\_9889

disconnect: ipc:///tmp/parent\_pub\_9892

9892: Exiting child...

9889: Exiting child...

9886: Exiting child...

^C unbind: ipc:///tmp/child\_pub\_left\_9878

unbind: ipc:///tmp/child\_pub\_left\_9874

disconnect: ipc:///tmp/parent\_pub\_9878

unbind: ipc:///tmp/child\_pub\_right9878

unbind: ipc:///tmp/parent\_pub\_9878

9874 Terminated by user

disconnect: ipc:///tmp/child\_pub\_left\_9874

9878 Terminated by user

1. **Листинг программы.**

Находится в папке src.

1. **Вывод.**

В ходе выполнения данной лабораторной работы я научился писать и управлять серверами сообщений, использовать библиотеку ZMQ; понял принципы построения паттерна «Publisher – Subscriber»; закрепил навыки работы с процессами и потоками.