Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №6-8 по курсу «Операционные системы»

Студент: Черкашин Андрей Викторович
Группа: М8О–206Б–20
Вариант: 45
Преподаватель: Соколов Андрей Алексеевич
Оценка: 5
Дата: 25 декабря 2021 г.
Подпись: _____

Москва, 2021

Постановка задачи

Цель работы

Целью является приобретение практических навыков в:

- Управлении серверами сообщений (№ 6)
- Применение отложенных вычислений (№ 7)
- Интеграция программных систем друг с другом (№ 8)

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)

Формат вывода:

«Ok: pid», где pid – идентификатор процесса для созданного вычислительного узла

«Error: Already exists» - вычислительный узел с таким идентификатором уже существует

«Error: Parent not found» - нет такого родительского узла с таким идентификатором

«Error: Parent is unavailable» - родительский узел существует, но по каким-то причинам с ним не удается связаться

«Error: [Custom error]» - любая другая обрабатываемая ошибка Пример:

> create 10 5

Ok: 3128

Примечания: создание нового управляющего узла осуществляется пользователем программы при помощи запуска исполняемого файла. Id и pid — это разные идентификаторы.

Удаление существующего вычислительного узла

Формат команды: remove id

id – целочисленный идентификатор удаляемого вычислительного узла

Формат вывода:

«Ok» - успешное удаление

«Error: Not found» - вычислительный узел с таким идентификатором не найден

«Error: Node is unavailable» - по каким-то причинам не удается связаться с вычислительным узлом

«Error: [Custom error]» - любая другая обрабатываемая ошибка

Пример:

> remove 10

Ok

Примечание: при удалении узла из топологии его процесс должен быть завершен и работоспособность вычислительной сети не должна быть нарушена.

Исполнение команды на вычислительном узле

Формат команды: exec id [params]

id – целочисленный идентификатор вычислительного узла, на который отправляется команда

Формат вывода:

«Ok:id: [result]», где result – результат выполненной команды

«Error:id: Not found» - вычислительный узел с таким идентификатором не найден

«Error:id: Node is unavailable» - по каким-то причинам не удается связаться с вычислительным узлом

«Error:id: [Custom error]» - любая другая обрабатываемая ошибка

Пример:

Можно найти в описании конкретной команды, определенной вариантом задания.

Примечание: выполнение команд должно быть асинхронным. Т.е. пока выполняется команда на одном из вычислительных узлов, то можно отправить следующую команду на другой вычислительный узел.

Вариант 45.

Топология 4(3): все вычислительные узлы хранятся в бинарном дереве поиска.

Набора команд 2 (локальный целочисленный словарь)

Формат команды сохранения значения: exec id name value

id – целочисленный идентификатор вычислительного узла, на который отправляется команда

name – ключ, по которому будет сохранено значение (строка формата [A-Za-z0-9]+)

value – целочисленное значение

Формат команды загрузки значения: exec id name

Тип проверки доступности узлов

Команда проверки 2

Формат команды: ping id

Команда проверяет доступность конкретного узла. Если узла нет, то необходимо выводить

ошибку: «Error: Not found»

Общие сведения о программе

Система состоит из двух программ: сервер и клиент. Используется сервер сообщений ZeroMQ.

Программа сервер компилируется из файла server.c. Программа клиент из файла client.c.

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 (pid ребенка) для родителя – child ID, в случае ошибки возвращает -1.
2. **execl** — используется для замены процесса на выполнения другой программы. Имя файла, содержащего процесс-потомок, задано с помощью первого аргумента. Какие-либо аргументы, передаваемые процессу-потомку задаются после первого аргумента, конец аргументов означает NULL.

Также были использованы следующие вызовы из библиотеки ZMQ:

1. **zmq_ctx_new —** создает новый контекст ZMQ.
2. **zmq_connect** — создает входящее соединение на сокет.
3. **zmq_disconnect** — отсоединяет сокет от заданного endpoint’a.
4. **zmq_socket** — создает ZMQ сокет.
5. **zmq_setsockopt** — задает параметры ZMQ сокета.
6. **zmq_close —** закрывает ZMQ сокет.
7. **zmq_ctx_destroy —** уничтожает контекст ZMQ.

Описание программы.

Программа строится на паттерне publisher-subscriber. Это означает, что есть сокет, созданный с помощью контекста ZMQ. Сокет можно подвязать под publisher (того, кто будет писать сообщения в сокет). Далее к этому сокету может быть присоединен subscriber (тот, кто будет слушать сообщения, переданные publisher-ом). Дерево состоит из ID клиентов, которые принимают сообщения от сервера и исполняют заданные действия. При добавлении новой ноды происходит fork для создания нового процесса, после этого нода добавляется в дерево и связывается с другими нодами с помощью сокетов. Передача чообщения от родителя до определенного ребенка происходит последовательно, т.е. от родителя (сервера) к корню дерева клиентов, а от корня по веткам, пока не дойдет до нужной ноды. Создание первого клиента происходит сразу при запуске сервера. Команда проверки узлов ping реализована таким образом: сервер посылает сигнал на нужную ноду, а клиент должен его вернуть, если сигнал не приходит в заданное время → нода не доступна. Хранение целочисленных данных в клиенте происходит с помощью map.

Листинг программы

Листинг программы приведен в репозитории github (папка src).

Пример работы

```
slash@avm:~/Desktop/lab6_updated$ ./server
11157 server started correctly!
11161: Client started. Id:0
exec 0 l 12
OK:0:l:12
exec 0 m 13
OK:0:m:13
exec 0 m
OK:0:m:13
create 2
OK:11188
11188: Client started. Id:2
exec 2 l 11
OK:2:l:11
exec 0 l
OK:0:l:12
exec 2 l
OK:2:l:11
ping 1
Error:1:Node with that number doesn't exist.
```

```
ping 2
OK
ping 0
OK
```

Вывод

В ходе выполенния данной лабораторной работы я познакомился с ZMQ и работой с сокетами в паттерне publisher-subscriber. Были укреплены знания по разделению процессов с помощью fork, а также понят принцип межпроцессорного взаимодействия по протоколу TPC. Также был получен опыт задания параметров сокета и опыт создания endpoint-ов, что может помочь в будущем при написании серверов или других программ с межпроцессным взаимодействием.ё