

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: Дегтярев Денис Андреевич
Группа: М8О-207Б-21
Вариант: 20
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

Репозиторий

<https://github.com/CHISH08/OCI/tree/main/6-8lab>

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 удастся 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 удастся 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.

Общие сведения о программе

Вариант 4, 1, 3

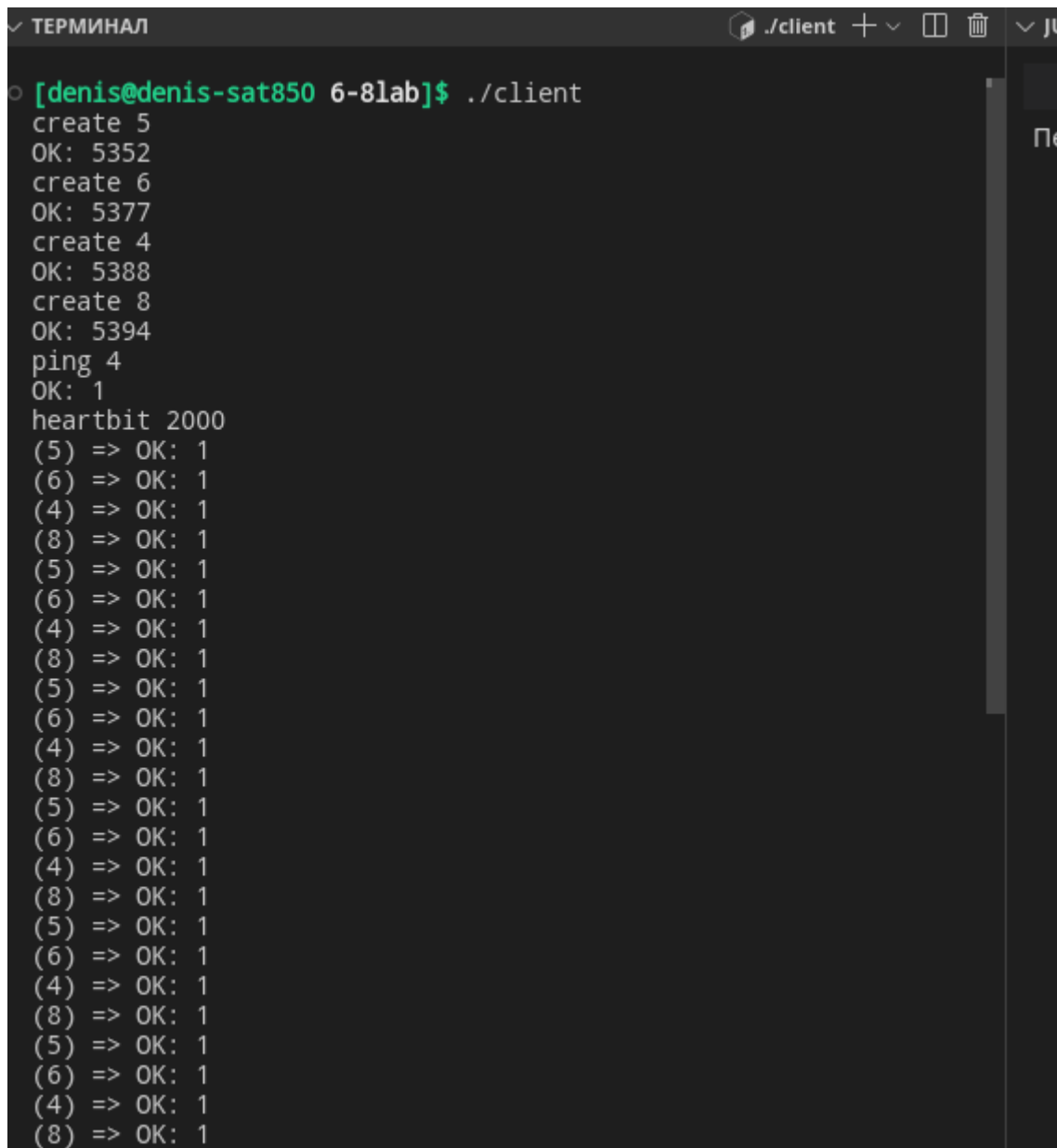
Вся библиотека `zmq.hpp`

Общий метод и алгоритм решения

Каждый узлы общаются только с соседями через очередь сообщений `zmq`. Структура – идеальное двоичное дерево. Каждый узел – отдельный процесс.

Исходный код(Расположен в репозитории)

Демонстрация работы программы



```
✓ ТЕРМИНАЛ ./.client + ▾ □ 🗑️ ▾ ju
[denis@denis-sat850 6-8lab]$ ./client
create 5
OK: 5352
create 6
OK: 5377
create 4
OK: 5388
create 8
OK: 5394
ping 4
OK: 1
heartbeat 2000
(5) => OK: 1
(6) => OK: 1
(4) => OK: 1
(8) => OK: 1
(5) => OK: 1
(6) => OK: 1
(4) => OK: 1
(8) => OK: 1
(5) => OK: 1
(6) => OK: 1
(4) => OK: 1
(8) => OK: 1
(5) => OK: 1
(6) => OK: 1
(4) => OK: 1
(8) => OK: 1
(5) => OK: 1
(6) => OK: 1
(4) => OK: 1
(8) => OK: 1
(5) => OK: 1
(6) => OK: 1
(4) => OK: 1
(8) => OK: 1
```

```
✓ ТЕРМИНАЛ
(6) => OK: 1
(4) => OK: 1
(8) => OK: 1
(5) => OK: 1
(6) => OK: 1
(4) => OK: 1
(8) => OK: 1
(5) => OK: 1
(6) => OK: 1
(4) => OK: 1
(8) => OK: 1
(5) => OK: 1
(6) => OK: 1
(4) => OK: 1
(8) => OK: 1
(5) => OK: 1
(6) => OK: 1
(4) => OK: 1
(8) => OK: 1
(5) => OK: 1
(6) => OK: 1
(4) => OK: 1
(8) => OK: 1
(5) => OK: 1
(6) => OK: 1
(4) => OK: 1
(8) => OK: 1
(5) => OK: 1
(6) => OK: 1
(4) => OK: 1
(8) => OK: 1
exec 6 2 3 4
OK: 6: 7
ping 13
Error: child is not existed!
█
```

Выводы

Создание процессов в виде двоичного дерева значительно ускоряет работу сервера, тк каждый узел – есть отдельный сервер. Zero MQ дает нам возможность при подключенном интернете создавать связи между родительским и дочерними процессами.