Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №6,7,8 по курсу «Операционные системы»

Студент: Хомяков Иван Андреевич
Группа: М8О-207Б-21
Вариант: 44
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/EbumbaE/OS_LAB/lab6-8

Постановка задачи

Цель работы

Целью является приобретение практических навыков в:

1. Управлении серверами сообщений (№6)
2. Применение отложенных вычислений (№7)
3. 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

- **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

- **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

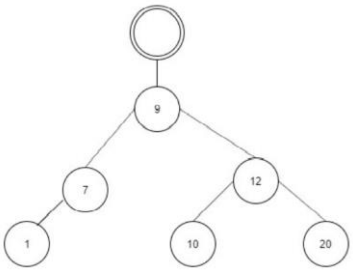
«Error:id: Not found» - вычислительный узел с таким идентификатором не найден

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

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

Топология 4:

Узлы находятся в идеально сбалансированном бинарном дереве. Каждый следующий узел должен добавляться в самое наименьшее левое поддерево. Все вычислительные узлы хранятся в бинарном дереве поиска. [parent] — является необязательным параметром.



Набора команд 3 (локальный таймер):

Формат команды сохранения значения: exec id subcommand. subcommand – одна из трех команд: start, stop, time. start – запустить таймер, stop – остановить таймер, time – показать время локального таймера в миллисекундах.

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

В main пользователь вводит команды, он отсылает их orchestra (создается из main). orchestra их принимает и обрабатывает. Структура orchestra представляет из себя двусвязный список корней деревьев. orchestra расчитывает путь до ребенка по дереву для доставки сообщения и передает его в message. orchestra использует также самописную библиотеку tree для работы со сбалансированным деревом. При помощи программы test_tree можно проверить работоспсобность дерева. Т.к. используется перебалансировка, ребенок должен знать, является ли он корнем. Корень не может исполнять команды exec. Также, если удалить корень как ребенка (при помощи команды remove child), его титул корня перейдет другому, кого заребалансирует на позицию корня. orchestra следит за этим и только в этом случае отсылает напрямую сообщение о смене роли. Если удалить корень при помощи команды remove parent - удалится дерево.

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. execv() — 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[]

Использованы следующие вызовы из библиотеки ZMQ:

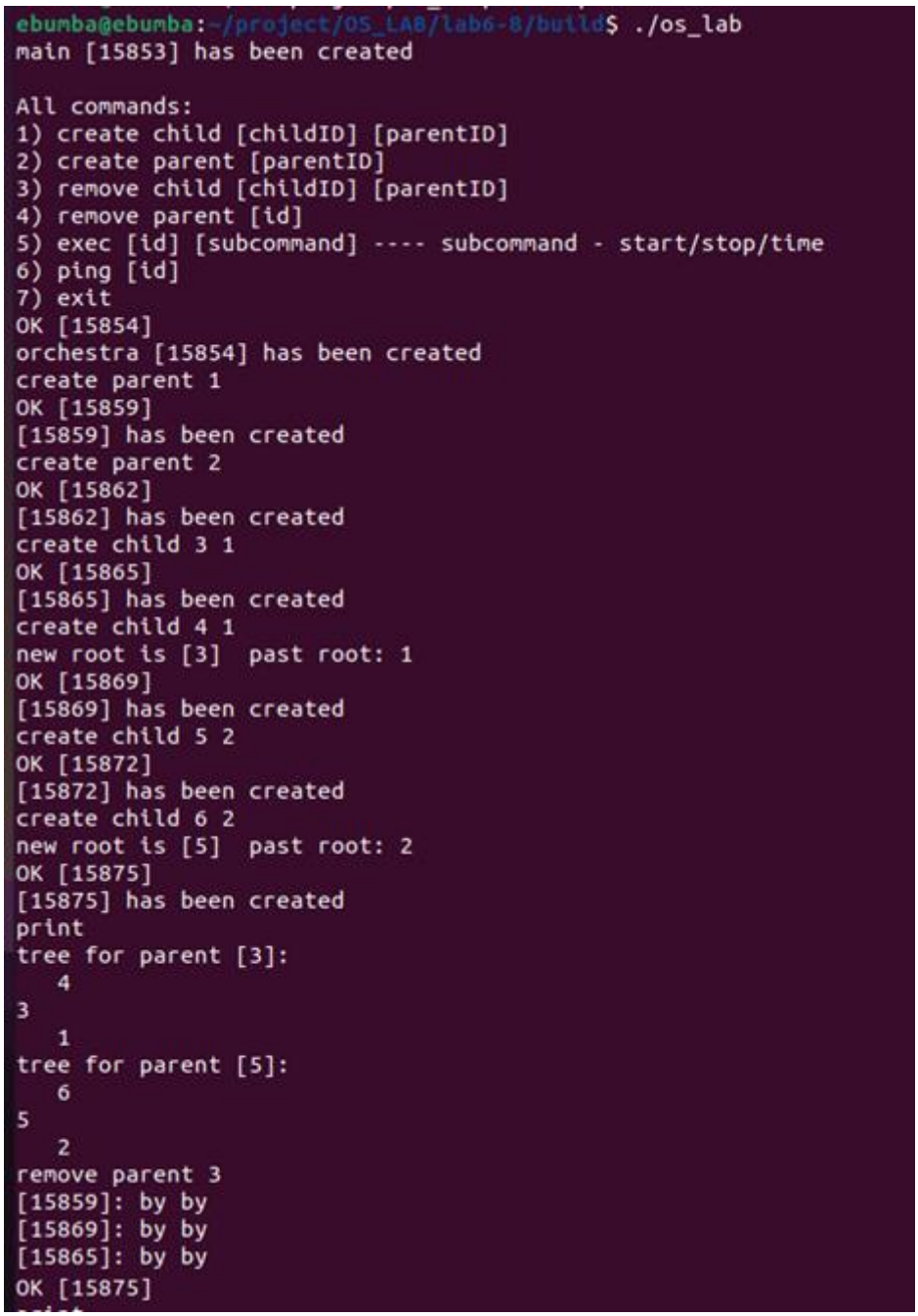
1. zmq_ctx_new — создает новый контекст ZMQ.
2. zmq_connect — создает входящее соединение на сокет.
3. zmq_disconnect — отсоединяет сокет от заданного endpoint’a.
4. zmq_socket — создает ZMQ сокет.

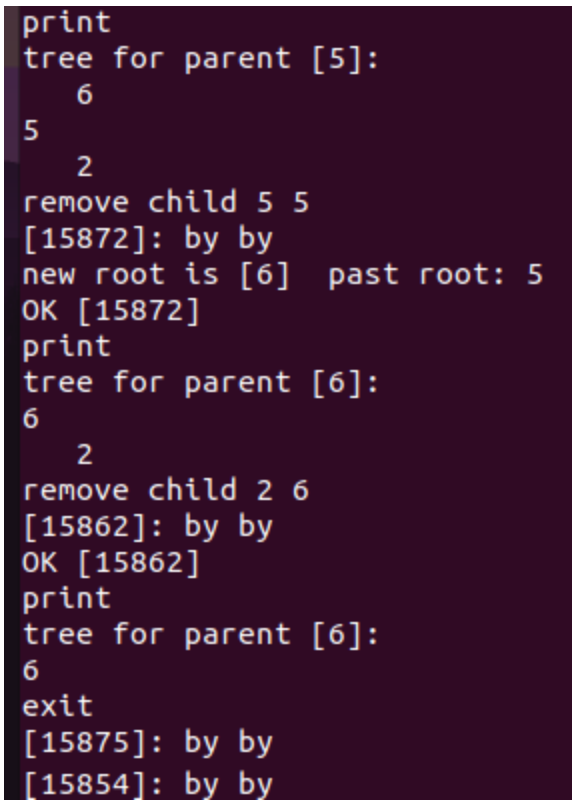
5. zmq_close — закрывает ZMQ сокет.
6. zmq_ctx_destroy — уничтожает контекст ZMQ.

Исходный код

В репозитории.

Демонстрация работы программы





Выводы

В ходе выполнения лабораторной работы изучил основы работы с очередями сообщений ZMQ и реализовал программу с использованием этой библиотеки. Реализовал распределенную систему по асинхронной обработке запросов, придумал и продумал ее структуру.