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

Лабораторные работы №5-7 по курсу «Операционные системы»

Студент: Тарасов Егор Дмитриевич
Группа: М8О-209Б-23
Вариант: 19
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

Репозиторий

https://github.com/EgorTarasov1/mai-os-labs

Постановка задачи

Цель работы

Целью является приобретение практических навыков в:

- Управлении серверами сообщений (№5)
- Применение отложенных вычислений (№6)
- Интеграция программных систем друг с другом (№7)

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])

Исполнение команды на вычислительном узле

(Формат команды: exec id [params])

Проверка доступности узла

(Формат команды: ping id)

Удаление узла

(Формат команды kill id)

Вариант №19: топология — бинарное дерево поиска, команда — работа с локальным таймером, проверка доступности — ping id.

Общие сведения о программе

Связь между вычислительными узлами и менеджером узлов реализована с использованием ZeroMQ и сокетов типа ZMQ_PAIR. Это позволяет обеспечить двусторонний обмен сообщениями между узлами и менеджером. В текущей реализации не установлены таймауты (ZMQ_SNDTIMEO и ZMQ_RECVTIMEO), что может привести к зависанию программы, если дочерний узел был убит или недоступен. Для повышения надёжности системы рекомендуется добавить обработку таймаутов. Каждый узел обрабатывает сообщения, полученные от менеджера. Если идентификатор сообщения совпадает с идентификатором узла, он выполняет команду и отправляет результат обратно. Если идентификатор не совпадает, узел передаёт сообщение дальше по иерархии (если такая возможность будет реализована в будущем). В текущей версии иерархическая передача сообщений не поддерживается, но это может быть добавлено для расширения функциональности.

Общий метод и алгоритм решения

Используемые методы системные вызовы:

zmq_ctx_new()	Создает новый контекст ZeroMQ. Используется в конструкторе NodeManager для инициализации контекста.
zmq_socket(void *context, int type)	Создает сокет ZeroMQ. В вашем коде используется тип сокета ZMQ_PAIR для двустороннего обмена сообщениями между узлами и менеджером.
zmq_bind(void *socket, const char *endpoint)	Привязывает сокет к указанному адресу (например, tcp://*:5555). Используется для настройки сокета в конструкторе NodeManager.
zmq_send(void *socket, const void *buf, size_t len, int flags)	Отправляет сообщение через сокет. В вашем коде используется в методе sendMessage для отправки строковых сообщений.
zmq_recv(void *socket, void *buf, size_t len, int flags)	Получает сообщение из сокета. Используется в методе receiveMessage для получения строковых сообщений.
zmq_close(void *socket)	Закрывает сокет ZeroMQ. Вызывается в деструкторе NodeManager для освобождения ресурсов.
zmq_ctx_destroy(void *context)size);	Уничтожает контекст ZeroMQ.

Исходный код

ComputerNode.h:

```
#ifndef COMPUTE_NODE_H
#define COMPUTE_NODE_H

#include <chrono>
#include <string>

class ComputeNode {
private:
    int id;
    bool timerRunning;
    bool isAvailable;
    std::chrono::steady_clock::time_point startTime;
    long long elapsedTime;

public:
    ComputeNode(int nodeId);

    std::string execute(const std::string& command);
    void killNode();
    bool checkAvailability();
};

#endif
```

ComputerNode.cpp

```
#include "ComputeNode.h"

ComputeNode::ComputeNode(int nodeId)
    : id(nodeId), timerRunning(false), isAvailable(true), elapsedTime(0) {}

std::string ComputeNode::execute(const std::string& command) {
    if (!isAvailable) {
        return "Error:" + std::to_string(id) + ": Node is unavailable";
    }

    if (command == "start") {
        if (!timerRunning) {
            timerRunning = true;
            startTime = std::chrono::steady_clock::now();
```

```
        }
        return "Ok:" + std::to_string(id);
    } else if (command == "stop") {
        if (timerRunning) {
            elapsedTime += std::chrono::duration_cast<std::chrono::milliseconds>(
                std::chrono::steady_clock::now() - startTime).count();
            timerRunning = false;
        }
        return "Ok:" + std::to_string(id);
    } else if (command == "time") {
        long long currentTime = elapsedTime;
        if (timerRunning) {
            currentTime += std::chrono::duration_cast<std::chrono::milliseconds>(
                std::chrono::steady_clock::now() - startTime).count();
        }
        return "Ok:" + std::to_string(id) + ": " + std::to_string(currentTime);
    } else if (command == "ping") {
        return isAvailable ? "Ok: 1" : "Ok: 0";
    }

    return "Error:" + std::to_string(id) + ": Unknown command";
}

void ComputeNode::killNode() {
    isAvailable = false;
}

bool ComputeNode::checkAvailability() {
    return isAvailable;
}
```

NodeManager.h:

```
#ifndef NODE_MANAGER_H
#define NODE_MANAGER_H

#include "ComputeNode.h"
#include <map>
#include <set>
#include <string>
#include <zmq.h>

class NodeManager {
private:
    std::map<int, ComputeNode*> nodes;
    std::map<int, std::set<int>> children;

    void* context;
    void* socket;
```

```
    void processCommand(int id, const std::string& command);
    void killNodeRecursive(int id);

public:
    NodeManager();
    ~NodeManager();

    std::string createNode(int id, int parent = -1);
    std::string execCommand(int id, const std::string& command);
    std::string killNode(int id);
    std::string pingNode(int id);

    void sendMessage(const std::string& message);
    std::string receiveMessage();
};

#endif
```

NodeManager.cpp

```
#include "NodeManager.h"
#include <iostream>
#include <sstream>

NodeManager::NodeManager() {
    context = zmq_ctx_new();
    socket = zmq_socket(context, ZMQ_PAIR);
    zmq_bind(socket, "tcp://*:5555");
}

NodeManager::~NodeManager() {
    for (auto& pair : nodes) {
        delete pair.second;
    }
    zmq_close(socket);
    zmq_ctx_destroy(context);
}

std::string NodeManager::createNode(int id, int parent) {
    if (nodes.find(id) != nodes.end()) {
        return "Error: Already exists";
    }

    if (parent != -1) {
        if (nodes.find(parent) == nodes.end()) {
            return "Error: Parent not found";
        }
        children[parent].insert(id);
    }
```

```
    nodes[id] = new ComputeNode(id);
    return "Ok:" + std::to_string(id);
}

void NodeManager::processCommand(int id, const std::string& command) {
    if (nodes.find(id) != nodes.end()) {
        std::string result = nodes[id]->execute(command);
        sendMessage(result);
    } else {
        sendMessage("Error: Node not found");
    }
}

std::string NodeManager::execCommand(int id, const std::string& command) {
    if (nodes.find(id) == nodes.end()) {
        return "Error: Not found";
    }

    std::ostringstream oss;
    oss << id << " " << command;
    sendMessage(oss.str());

    return receiveMessage();
}

void NodeManager::killNodeRecursive(int id) {
    if (nodes.find(id) != nodes.end()) {
        nodes[id]->killNode();
    }

    if (children.find(id) != children.end()) {
        for (int childId : children[id]) {
            killNodeRecursive(childId);
        }
    }
}

std::string NodeManager::killNode(int id) {
    if (nodes.find(id) == nodes.end()) {
        return "Error: Not found";
    }

    killNodeRecursive(id);
    return "Ok: Node " + std::to_string(id) + " killed";
}

std::string NodeManager::pingNode(int id) {
    if (nodes.find(id) == nodes.end()) {
        return "Ok: 0";
    }
```

```
    std::string result = nodes[id]->execute("ping");
    return result;
}

void NodeManager::sendMessage(const std::string& message) {
    zmq_send(socket, message.c_str(), message.size(), 0);
}

std::string NodeManager::receiveMessage() {
    char buffer[512] = {0};
    zmq_recv(socket, buffer, sizeof(buffer), 0);
    return std::string(buffer);
}
```

Сборка программы

```
[main] Building folder: c:/Users/Xiaomi/Desktop/ll5-7/build
[build] Starting build
[proc] Executing command: "C:\Program Files\CMake\bin\cmake.EXE" --build c:/Us-
ers/Xiaomi/Desktop/ll5-7/build --config Debug --target all -j 10 --
[build] [ 25%] Building CXX object CMakeFiles/DistributedSys-
tem.dir/NodeManager.cpp.obj
[build] [ 50%] Linking CXX executable DistributedSystem.exe
[build] [100%] Built target DistributedSystem
[driver] Build completed: 00:00:05.617
[build] Build finished with exit code 0
```

Демонстрация работы программы

S C:\Users\Xiaomi\Desktop\ll5-7\build> ./DistributedSystem

Distributed System Controller

> create 9

Ok:9

> create 7 0

Error: Parent not found

> create 7 9

Ok:7

> create 1 7

Ok:1

> create 12 9

Ok:12

> create 10 12

Ok:10

> create 20 12

Ok:20

> ping 9

Ok: 1

> ping 21

Ok: 0

> exec 9 start

Ok:9

Command sent to node 9 asynchronously

> exec 9 time

Command sent to node 9 asynchronously

> Ok:9: 8296

exec 9 time

Command sent to node 9 asynchronously

> Ok:9: 12651

exec 9 stop

Command sent to node 9 asynchronously

Ok:9

> kill 9

Ok: Node 9 killed

> ping 9

Error:9: Node is unavailable

> exit

Выводы

В ходе проекта я изучил основы работы с очередями сообщений ZeroMQ и разработал систему управления вычислительными узлами, используя сокет типа ZMQ_PAIR для надёжной двусторонней связи между `NodeManager` и `ComputeNode`. Наиболее сложными задачами были удаление узлов из сети и добавление новых между существующими, требовавшие переподключения сокетов для сохранения стабильной связи. Реализованная система позволяет

эффективно распределять команды и контролировать состояние узлов, обеспечивая отказоустойчивость. Этот опыт с ZeroMQ будет полезен при настройке собственных систем распределённых вычислений, позволяя масштабировать и поддерживать устойчивость сети.