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: Постнов Александр Вячеславович
Группа: М8О-201Б-21
Вариант: 12
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

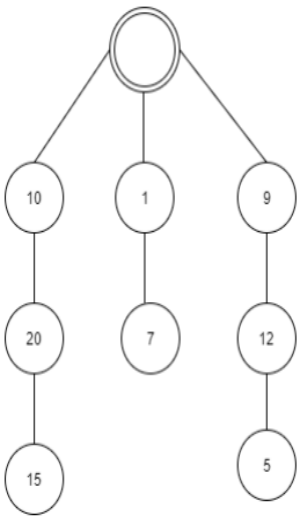
https://github.com/kappaprideonly/mai-os-labs

Постановка задачи

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.

Типы топологий

Топология 1



Все вычислительные узлы находятся в списке. Есть только один управляющий узел. Чтобы добавить новый вычислительный узел к управляющему, то необходимо выполнить команду: create id -1.

Набора команд 4 (поиск подстроки в строке)

Формат команды:

> exec id
> text_string
> pattern_string
[result] – номера позиций, где найден образец, разделенный точкой с запятой

Команда проверки 3

Формат команды: heartbit 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.

Общие сведения о программе

CMakeLists.txt - описание сборки проекта

topology.h - объявление класса топологии

topology.cpp - реализация функции класса топологии

calcnode.hpp - реализация класса вычислительного узла

handlernode.hpp - реализация класса управляющего узла

zmqf.h - объявление функций для комфортной работы с библиотекой zmq

zmqf.cpp - реализация

search.h - объявление функций для поиска подстроки в строке

search.cpp - реализация

client.cpp - в этой программе создаются вычислительные узлы / обработка команд пользователя

server.cpp - в этой программе отдельным процессом работают вычислительные узлы

Общий метод и алгоритм решения

В лабораторной работе я использовал ZeroMQ

ZeroMQ — высокопроизводительная асинхронная библиотека обмена сообщениями, ориентированная на использование в распределённых и параллельных вычислениях. Библиотека реализует очередь сообщений, которая может функционировать без выделенного брокера сообщений.

Использовал паттерны pub-sub и rep-req

Исходный код

CMakeLists.txt

```
add_library(topology STATIC src/topology.cpp include/topology.h)
add_library(zmfq STATIC src/zmqf.cpp include/zmqf.h)
add_library(search STATIC include/search.h src/search.cpp)

target_include_directories(topology PRIVATE include)
target_include_directories(zmfq PRIVATE include)
target_include_directories(search PRIVATE include)


target_link_libraries(topology PRIVATE zmq)
target_link_libraries(zmfq PRIVATE zmq)

add_executable(client client.cpp include/handlernode.hpp
include/calcnode.hpp)
target_link_libraries(client PRIVATE topology)
target_link_libraries(client PRIVATE zmfq)
target_link_libraries(client PRIVATE search)


add_executable(server server.cpp include/handlernode.hpp
include/calcnode.hpp)
target_link_libraries(server PRIVATE topology)
target_link_libraries(server PRIVATE zmfq)
target_link_libraries(server PRIVATE search)


target_include_directories(client PRIVATE include)
target_include_directories(server PRIVATE include)
```

client.cpp

```
#include <iostream>

#include "handlernode.hpp"
#include "calcnode.hpp"
```

```
#include "topology.h"
#include "zmqf.h"

int main() {
    THandlerNode handlerNode;
    handlerNode.Start();
    TTopology topology;
    std::cout << "Commands:\n";
    std::cout << "create child parent\n";
    std::cout << "kill child\n";
    std::cout << "exec id line pattern\n";
    std::cout << "exit\n";
    std::string command;
    while (std::cin >> command) {
        if (command == "create") {
            int child;
            int parent;
            std::cin >> child;
            std::cin >> parent;
            if (topology.Find(child) != -1) {
                std::cout << "Child already exists!\n";
                continue;
            }
            if (parent == -1) {
                topology.Insert(child);
                handlerNode.CreateCalcNode(child);
            } else {
                if (!topology.Insert(parent, child)) {
                    std::cout << "Parent doesn't exist!\n";
                } else {
                    handlerNode.CreateCalcNode(child);
                }
            }
        }
        if (command == "exec") {
            int child;
            std::cin >> child;
            if (topology.Find(child) == -1) {
                std::cout << "Child doesn't exist\n";
                continue;
            }
            std::string line;
            std::cin >> line;
            std::string pattern;
            std::cin >> pattern;
```

```
            std::string message = "exec " + line + " " + pattern;
            handlerNode.SendMessageChild(child, message);
        }
        if (command == "kill") {
            int child;
            std::cin >> child;
            if (topology.Find(child) == -1) {
                std::cout << "Child doesn't exist\n";
                continue;
            }
            handlerNode.SendMessageChild(child, "kill");
            topology.Erase(child);
        }

        if (command == "heartbit") {
            int time;
            std::cin >> time;
            std::cout << time << "\n";
            handlerNode.synhSocket.set(zmq::sockopt::rcvtimeo, time * 4);
            for (int i = 0; i < 5; ++i) {
                for (const auto& list : topology.container) {
                    for (const auto& elem : list) {
                        handlerNode.SendMessageChild(elem, "ping");
                    }
                }
            sleep((unsigned)(time/1000));
            }
            handlerNode.synhSocket.set(zmq::sockopt::rcvtimeo, 5000);
            continue;
        }

        if (command == "exit") {
            for (const auto& list : topology.container) {
                for (const auto& elem : list) {
                    handlerNode.SendMessageChild(elem, "kill");
                }
            }
            // handlerNode.Kill();
            break;
        }
    }
    handlerNode.Kill();
    return 0;
}
```

server.cpp

```
#include <cstdlib>
#include <iostream>
#include <string>
#include <unistd.h>

#include "zmqf.h"
#include "calcnode.hpp"
#include "handlernode.hpp"

int main(int argc, char* argv[]) {
    if (argc != 4) {
        std::cout << "argc server error!\n";
        exit(EXIT_FAILURE);
    }
    TCalcNode calc(atoi(argv[1]), atoi(argv[2]), argv[3]);
    calc.Start();
    std::cout << "Calc node is starting! pid: " << getpid() << "\n";
    while (true) {
        std::string message = calc.Receive();
        if (message == "kill") {
            calc.Kill();
            std::cout << "Ok! Kill " << calc.filter << "!\n";
            break;
        }
        if (message == "ping") {
            std::cout << "Ok! The node " << calc.filter << " is
available!\n";
            continue;
        }
        if (message.rfind("exec", 0) == 0) {
            auto answers = calc.Exec(message);
            if (answers.empty()) {
                std::cout << -1;
            }
            for (auto const &elem : answers) {
                std::cout << elem << " ";
            }
            std::cout << "\n";
            continue;
        }
    }
}
```

calcnode.hpp

```
#ifndef CALCNODE_H
#define CALCNODE_H

#include <cstdlib>
#include <string>
#include <iostream>
#include <unistd.h>
#include <utility>
#include <zmq.hpp>

#include "zmqf.h"
#include "search.h"


class TCalcNode {
   private:
        zmq::context_t contextHandler;
        zmq::context_t contextSynh;
   public:
        zmq::socket_t handlerSocket;
        zmq::socket_t synhSocket;
        int handlerPort;
        int synhPort;
        std::string filter;
        TCalcNode(int handlerPort, int synhPort, std::string filter):
            handlerSocket(contextHandler, ZMQ_SUB),
            synhSocket(contextSynh, ZMQ_REQ),
            handlerPort(handlerPort),
            synhPort(synhPort),
            filter(std::move(filter))
        {};

        int Start() {
            Connect(handlerSocket, handlerPort);
            Connect(synhSocket, synhPort);
            handlerSocket.set(zmq::sockopt::subscribe, filter);
            synhSocket.set(zmq::sockopt::rcvtimeo, 5000);
            synhSocket.set(zmq::sockopt::sndtimeo, 5000);
            return 0;
        }

        int Kill() {
            handlerSocket.close();
            synhSocket.close();
            contextHandler.close();
```

```
            contextSynh.close();
            return 0;
        }

        std::string Receive() {
            std::string adress = ReceiveMessage(handlerSocket);
            std::string content = ReceiveMessage(handlerSocket);
            SendMessage(synhSocket, "Ok!");
            std::string answer = ReceiveMessage(synhSocket);
            if (answer == "Error: Node is unavailable") {
                    std::cout << answer << "\n";
            }
            return content;
        }


        std::vector<unsigned int> Exec(const std::string& message) {
            auto components = StringToVectorStrings(message, ' ');
            std::string pattern = components[2];
            std::string text = components[1];
            return KMP(pattern, text);
        }
};

#endif
```

handlernode.hpp

```
#ifndef HANDLERNODE_H
#define HANDLERNODE_H


#include <cstdlib>
#include <ratio>
#include <string>
#include <iostream>
#include <unistd.h>

#include "zmqf.h"


class THandlerNode {
   private:
        zmq::context_t contextHandler;
        zmq::context_t contextSynh;
   public:
```

```
        zmq::socket_t handlerSocket;
        zmq::socket_t synhSocket;
        int handlerPort;
        int synhPort;
        THandlerNode():
            handlerSocket(contextHandler, ZMQ_PUB),
            synhSocket(contextSynh, ZMQ_REP)

        {};

        int Start() {
            handlerPort = BindSocket(handlerSocket);
            synhPort = BindSocket(synhSocket);
            synhSocket.set(zmq::sockopt::rcvtimeo, 5000);
            synhSocket.set(zmq::sockopt::sndtimeo, 5000);
            return 0;
        }

        int Kill() {
            handlerSocket.close();
            synhSocket.close();
            contextHandler.close();
            contextSynh.close();
            return 0;
        }

        int CreateCalcNode(int calcNodeId){
            int calcpid = fork();
            if (calcpid == -1) {
                std::cout << "Fork error\n";
                return EXIT_FAILURE;
            }
            if (calcpid == 0) {
                if(execl("server", "server",
                std::to_string(handlerPort).data(),
std::to_string(synhPort).data(),
                std::to_string(calcNodeId).data(), nullptr) == -1) {
                    std::cout << "Failed to exec\n";
                    exit(EXIT_FAILURE);
                }
            }
            return 0;
        }
        int SendMessageChild(int id, const std::string& line) {
```

```
            handlerSocket.send(zmq::buffer(std::to_string(id)),
zmq::send_flags::sndmore);

            SendMessage(handlerSocket, line);
            auto message = ReceiveMessage(synhSocket);
            if (message != "Error: Node is unavailable") {
                SendMessage(synhSocket, "Ok!");
            } else {
                std::cout << message << "\n";
            }

            return 0;
        }
};


#endif
```

search.h

```
#ifndef SEARCH_H
#define SEARCH_H

#include <string>
#include <vector>
#include <algorithm>


std::vector<unsigned int> ZFunction(const std::string & str);
std::vector<unsigned int> PrefixFunction(const std::string & str);
std::vector<unsigned int> KMP(const std::string & pattern, const
std::string & text);
std::vector<std::string> StringToVectorStrings(std::string const&
stringNumbers, char separator);


#endif /* SEARCH_H */
```

search.cpp

```
#include "search.h"



std::vector<unsigned int> ZFunction(const std::string & s) {
```

```
    unsigned int n = s.size();
    std::vector<unsigned int> z(n);
    unsigned int l = 0;
    unsigned int r = 0;
    for (unsigned int i = 1; i < n; ++i) {
        if (i <= r) {
            z[i] = std::min(z[i - l], r - i);
        }
        while (i + z[i] < n and s[i + z[i]] == s[z[i]]) {
            ++z[i];
        }
        if (i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }
    return z;
}

std::vector<unsigned int> PrefixFunction(const std::string & s) {
    std::vector<unsigned int> z = ZFunction(s);
    unsigned int n = s.size();
    std::vector<unsigned int> sp(n);
    for (unsigned int i = n - 1; i > 0; --i) {
        sp[i + z[i] - 1] = z[i];
    }
    return sp;
}

std::vector<unsigned int> KMP(const std::string & pattern, const
std::string & text) {
    std::vector<unsigned int> p = PrefixFunction(pattern);
    unsigned int m = pattern.size();
    unsigned int n = text.size();
    unsigned int i = 0;
    std::vector<unsigned int> ans;
    if (m > n) {
        return ans;
    }
    while (i < n - m + 1) {
        unsigned int j = 0;
        while (j < m and pattern[j] == text[i + j]) {
            ++j;
        }
        if (j == m) {
```

```
            ans.push_back(i);
        } else {
            if (j > 0 and j > p[j - 1]) {
                i = i + j - p[j - 1] - 1;
            }
        }
        ++i;
    }
    return ans;
}

std::vector<std::string> StringToVectorStrings(std::string const&
stringNumbers, char separator=' ') {
    std::vector<std::string> results;
    auto start = stringNumbers.begin();
    auto end = stringNumbers.end();
    auto next = std::find(start, end, separator);
    while (next != end) {
        results.emplace_back(start, next);
        start = next + 1;
        next = std::find(start, end, separator);
    }
    results.emplace_back(start, next);
    return results;
}
```

topology.h

```
#ifndef TOPOLOGY_H
#define TOPOLOGY_H

#include <list>
#include <algorithm>
#include <cmath>
#include <ostream>


class TTopology {
private:
    using TListType = std::list< std::list<int> >;
    using TIterator = typename std::list<int>::iterator;
    using TListIterator = typename TListType::iterator;

    // TListType container;
```

```
    // size_t containerSize{};
public:
    TListType container;
    size_t containerSize{};
    bool Erase(const int & elem);

    int Find(const int & elem);

    bool Insert(const int & parent, const int & elem);

    void Insert(const int & elem);

    size_t Size();

};
#endif /* TOPOLOGY_H */
```

topology.cpp

```
#include "topology.h"



bool TTopology::Erase(const int & elem) {
    for (auto it1 = container.begin(); it1 != container.end(); ++it1) {
        for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {
            if (*it2 == elem) {
                if (it1->size() > 1) {
                    it1->erase(it2);
                } else {
                    container.erase(it1);
                }
                --containerSize;
                return true;
            }
        }
    }
    return false;
}

int TTopology::Find(const int & elem) {
    int ind = 0;
    for (auto & it1 : container) {
```

```
        for (int & it2 : it1) {
            if (it2 == elem) {
                return ind;
            }
        }
        ++ind;
    }
    return -1;
}

bool TTopology::Insert(const int & parent, const int & elem) {
    for (auto & it1 : container) {
        for (auto it2 = it1.begin(); it2 != it1.end(); ++it2) {
            if (*it2 == parent) {
                it1.insert(++it2, elem);
                ++containerSize;
                return true;
            }
        }
    }
    return false;
}

void TTopology::Insert(const int & elem) {
    std::list<int> newList;
    newList.push_back(elem);
    ++containerSize;
    container.push_back(newList);
}

size_t TTopology::Size(){
    return containerSize;
}
```

zmqf.h

```
#ifndef ZMQF_H
#define ZMQF_H

#include <zmq.hpp>
const int MAIN_PORT = 4040;

bool SendMessage(zmq::socket_t& socket, const std::string& message);

std::string ReceiveMessage(zmq::socket_t& socket);
```

```
int BindSocket(zmq::socket_t& socket);

void Connect(zmq::socket_t &socket, int port);

void Disconnect(zmq::socket_t &socket, int port);

int Bind(zmq::socket_t &socket, int id);

void Unbind(zmq::socket_t &socket, int port);

#endif
```

zmqf.cpp

```
#include <iostream>

#include "zmqf.h"


bool SendMessage(zmq::socket_t& socket, const std::string& message) {
    try {
        socket.send(zmq::buffer(message), zmq::send_flags::none);
        return true;
    } catch(...) {
        return false;
    }
}

std::string ReceiveMessage(zmq::socket_t& socket) {
    zmq::message_t message;
    bool messageReceived;
    try {
        messageReceived = (bool)socket.recv(message,
zmq::recv_flags::none);
    } catch(...) {
        messageReceived = false;
    }
    std::string received(static_cast<char*>(message.data()),
message.size());
    if(!messageReceived || received.empty()) {
        return "Error: Node is unavailable";
    }
```

```
    return received;
}

int BindSocket(zmq::socket_t& socket) {
    int port = 3000;
    std::string portTemplate = "tcp://*:";
    while(true) {
        try {
            socket.bind(portTemplate + std::to_string(port));
            break;
        } catch(...) {
            port++;
        }
    }
    return port;
}

void Connect(zmq::socket_t &socket, int port) {
    std::string address = "tcp://localhost:" + std::to_string(port);
    socket.connect(address);
}

void Disconnect(zmq::socket_t &socket, int port) {
    std::string address = "tcp://localhost:" + std::to_string(port);
    socket.disconnect(address);
}

int Bind(zmq::socket_t &socket, int id) {
    int port = MAIN_PORT + id;
    std::string address = "tcp://*:" + std::to_string(port);
    while(true){
        try{
            socket.bind(address);
            break;
        }
        catch(...){
            port++;
        }
    }
    return port;
}

void Unbind(zmq::socket_t &socket, int port) {
    std::string address = "tcp://*:" + std::to_string(port);
    socket.unbind(address);
```

}

Демонстрация работы программы

➜ lab6-8 git:(main) ./client
Commands:
create child parent
kill child
exec id line pattern
exit
create 5 -1
Calc node is starting! pid: 50499
create 4 5
Calc node is starting! pid: 50505
create 6 5
Calc node is starting! pid: 50513
create 2 -1
Calc node is starting! pid: 50518
kill 5
Ok! Kill 5!
exec 6 abacaba aba
0 4
heartbit 2000
2000
Ok! The node 6 is available!
Ok! The node 4 is available!

Ok! The node 2 is available!

Ok! The node 6 is available!

Ok! The node 4 is available!

Ok! The node 2 is available!

Ok! The node 6 is available!

Ok! The node 4 is available!

Ok! The node 2 is available!

Ok! The node 6 is available!

Ok! The node 4 is available!

Ok! The node 2 is available!

Ok! The node 6 is available!

Ok! The node 4 is available!

Ok! The node 2 is available!

exit

Ok! Kill 6!

Ok! Kill 4!

Ok! Kill 2!

Выводы:

В ходе лабораторной работы познакомился с сокетами, очередями сообщений, с высокопроизводительной библиотекой zmq, ознакомился с клиент-серверной архитектурой, ознакомился с некоторыми паттернами.