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»**

**Тема работы**

**“Очереди сообщений”**

Студент: Борисов Ян Артурович

Группа: М8О-208Б-20

Вариант: 30

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**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/Yannikupy/OS>

**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.  
  
Вариант 30. Команды:  
create id  
exec id text pattern (поиск подстроки)  
pingall

**Общие сведения о программе**

Для выполнения данной лабораторной работы я предварительно реализовал 2 файла с кодом:

child\_node.cpp (реализация дочерних процессов)

main\_prog.cpp (реализация основного процесса)

**Общий метод и алгоритм решения**

В makefile у нас две команды:  
all:  
 g++ -fsanitize=address main\_prog.cpp -lzmq -o main\_prog -w  
 g++ -fsanitize=address child\_node.cpp -lzmq -o child\_node -w

По сути, две работающие программы. В начале запускается main\_prog, который уже в себе вызывает child\_prog от конкретных значений child\_id, parent\_port и parent\_id. После того, как main\_prog получает сообщение, он отправляет его на child\_node, который обрабатывает это сообщение.

**Исходный код  
main\_prog.cpp**

#include <zmq.hpp>

#include <unistd.h>

#include <iostream>

#include <vector>

using namespace std;

int main()

{

zmq::context\_t context(1);

zmq::socket\_t main\_socket(context, ZMQ\_REP);

string adr = "tcp://127.0.0.1:300";

string command;

int child\_id = 0;

cout << "Commands:**\n**create id**\n**exec id text pattern**\n**pingall**\n**kill id**\n**exit**\n**";

while (1)

{

cout << "Please, enter command**\n**";

cin >> command;

if (command == "create")

{

if (child\_id == 0)

{

int id;

cin >> id;

int id\_tmp = id - 1;

main\_socket.bind(adr + to\_string(++id\_tmp));

string new\_adr = adr + to\_string(id\_tmp);

char \*adr\_ = new char[new\_adr.size() + 1];

memcpy(adr\_, new\_adr.c\_str(), new\_adr.size() + 1);

char \*id\_ = new char[to\_string(id).size() + 1];

memcpy(id\_, to\_string(id).c\_str(), to\_string(id).size() + 1);

char \*args[] = {"./child\_node", adr\_, id\_, NULL};

int id2 = fork();

if (id2 == -1)

{

std::cout << "ERROR: CALCULATING NODE WAS NOT CREATED**\n**";

id = 0;

exit(1);

}

else if (id2 == 0)

{

execv("./child\_node", args);

}

else

{

child\_id = id;

}

zmq::message\_t message;

main\_socket.recv(&message);

string recieved\_message(static\_cast<char \*>(message.data()), message.size());

cout << recieved\_message << "**\n**";

delete[] adr\_;

delete[] id\_;

}

else

{

int id;

cin >> id;

string message\_string = command + " " + to\_string(id);

zmq::message\_t message(message\_string.size());

memcpy(message.data(), message\_string.c\_str(), message\_string.size());

main\_socket.send(message);

main\_socket.recv(&message);

string recieved\_message(static\_cast<char \*>(message.data()), message.size());

cout << recieved\_message << "**\n**";

}

}

else if (command == "exec")

{

int id;

string text, pattern, s;

cin >> id >> text >> pattern;

s = text + "$" + pattern;

string message\_string = command + " " + to\_string(id) + " " + s;

zmq::message\_t message(message\_string.size());

memcpy(message.data(), message\_string.c\_str(), message\_string.size());

main\_socket.send(message);

main\_socket.recv(&message);

string recieved\_message(static\_cast<char \*>(message.data()), message.size());

cout << recieved\_message << "**\n**";

}

else if (command == "pingall")

{

if (child\_id == 0)

{

cout << "No childs to ping!**\n**";

}

else

{

string message\_string = command;

zmq::message\_t message(message\_string.size());

memcpy(message.data(), message\_string.c\_str(), message\_string.size());

// receive answer from child

//for (int j = 0; j < beat\_amount; j++)

//{

main\_socket.send(message);

main\_socket.recv(&message);

string recieved\_message(static\_cast<char \*>(message.data()), message.size());

if (recieved\_message == "OK")

{

cout << "OK: -1**\n**";

}

else

{

cout << "OK:" << recieved\_message;

}

}

}

else if (command == "kill")

{

int id;

cin >> id;

if (child\_id == 0)

{

cout << "Error: there aren't any nodes**\n**";

}

else if (child\_id == id)

{

string kill\_message = "DIE";

zmq::message\_t message(kill\_message.size());

memcpy(message.data(), kill\_message.c\_str(), kill\_message.size());

main\_socket.send(message);

cout << "Tree was deleted**\n**";

child\_id = 0;

}

else

{

string kill\_message = command + " " + to\_string(id);

zmq::message\_t message(kill\_message.size());

memcpy(message.data(), kill\_message.c\_str(), kill\_message.size());

main\_socket.send(message);

main\_socket.recv(&message);

string received\_message(static\_cast<char \*>(message.data()), message.size());

cout << received\_message << "**\n**";

}

}

else if (command == "exit")

{

if (child\_id)

{

string kill\_message = "DIE";

zmq::message\_t message(kill\_message.size());

memcpy(message.data(), kill\_message.c\_str(), kill\_message.size());

main\_socket.send(message);

cout << "Tree was deleted**\n**";

child\_id = 0;

}

main\_socket.close();

context.close();

return 0;

}

else

{

cout << "Error: incorrect command**\n**";

}

}

}

**child\_node.cpp**

#include <zmq.hpp>

#include <iostream>

#include <unistd.h>

using namespace std;

void send\_message(std::string message\_string, zmq::socket\_t &socket)

{

zmq::message\_t message\_back(message\_string.size());

memcpy(message\_back.data(), message\_string.c\_str(), message\_string.size());

if (!socket.send(message\_back))

{

std::cout << "Error: can't send message from node with pid " << getpid() << "**\n**";

}

}

int main(int argc, char \*argv[])

{

string adr = argv[1];

zmq::context\_t context(1);

zmq::socket\_t main\_socket(context, ZMQ\_REQ);

main\_socket.connect(argv[1]);

send\_message("OK: " + to\_string(getpid()), main\_socket);

int id = stoi(argv[2]);

int left\_id = 0;

int right\_id = 0;

zmq::context\_t context\_l(1);

zmq::context\_t context\_r(1);

zmq::socket\_t left\_socket(context\_l, ZMQ\_REP);

string adr\_left = "tcp://127.0.0.1:300";

zmq::socket\_t right\_socket(context\_r, ZMQ\_REP);

string adr\_right = "tcp://127.0.0.1:300";

while (1)

{

zmq::message\_t message\_main;

main\_socket.recv(&message\_main);

string recieved\_message(static\_cast<char \*>(message\_main.data()), message\_main.size());

string command;

for (int i = 0; i < recieved\_message.size(); ++i)

{

if (recieved\_message[i] != ' ')

{

command += recieved\_message[i];

}

else

{

break;

}

}

if (command == "exec")

{ //----------EXEC----------

int id\_proc;

string id\_proc\_, value\_;

string key;

int pos;

for (int i = 5; i < recieved\_message.size(); ++i)

{

if (recieved\_message[i] != ' ')

{

id\_proc\_ += recieved\_message[i];

}

else

{

pos = i;

break;

}

}

id\_proc = stoi(id\_proc\_);

if (id\_proc == id)

{ // id == proc\_id, this node must execute

int j = pos;

string text, pattern;

string res = "OK:" + to\_string(id) + ":";

while (recieved\_message[j] != '$')

{

text.push\_back(recieved\_message[j]);

++j;

}

++j;

while (j != recieved\_message.size())

{

pattern.push\_back(recieved\_message[j]); //tttt t

++j;

}

int flag = 0;

if (pattern.size() <= text.size())

{

//cout << text << " " << pattern << "\n";

for (int i = 0; i <= text.size() - pattern.size(); ++i)

{

//cout << "TEST";

string pat;

for (int z = 0; z < pattern.size(); ++z)

{

if (text[i + z] == pattern[z])

pat.push\_back(text[i + z]);

}

if (pat == pattern)

{

res = res + to\_string(i) + ";";

flag = 1;

}

pat.clear();

}

}

if (flag == 0)

{

res = res + "-1";

}

zmq::message\_t message(res.size());

memcpy(message.data(), res.c\_str(), res.size());

if (!main\_socket.send(message))

{

cout << "Error: can't send message to main node from node with pid: " << getpid() << "**\n**";

}

}

else

{ // id != prod\_id

if (id > id\_proc)

{ //go to left

if (left\_id == 0)

{ // if node not exists

string message\_string = "Error:id: Not found";

zmq::message\_t message(message\_string.size());

memcpy(message.data(), message\_string.c\_str(), message\_string.size());

if (!main\_socket.send(message))

{

cout << "Error: can't send message to main node from node with pid: " << getpid() << "**\n**";

}

}

else

{

zmq::message\_t message(recieved\_message.size());

memcpy(message.data(), recieved\_message.c\_str(), recieved\_message.size());

if (!left\_socket.send(message))

{

cout << "Error: can't send message to left node from node with pid: " << getpid() << "**\n**";

}

// catch and send to parent

if (!left\_socket.recv(&message))

{

cout << "Error: can't receive message from left node in node with pid: " << getpid() << "**\n**";

}

if (!main\_socket.send(message))

{

cout << "Error: can't send message to main node from node with pid: " << getpid() << "**\n**";

}

}

}

else

{ // go to right

if (right\_id == 0)

{ // if node not exists

string message\_string = "Error:id: Not found";

zmq::message\_t message(message\_string.size());

memcpy(message.data(), message\_string.c\_str(), message\_string.size());

if (!main\_socket.send(message))

{

cout << "Error: can't send message to main node from node with pid: " << getpid() << "**\n**";

}

}

else

{

zmq::message\_t message(recieved\_message.size());

memcpy(message.data(), recieved\_message.c\_str(), recieved\_message.size());

if (!right\_socket.send(message))

{

cout << "Error: can't send message to right node from node with pid: " << getpid() << "**\n**";

}

// catch and send to parent

if (!right\_socket.recv(&message))

{

cout << "Error: can't receive message from left node in node with pid: " << getpid() << "**\n**";

}

if (!main\_socket.send(message))

{

cout << "Error: can't send message to main node from node with pid: " << getpid() << "**\n**";

}

}

}

}

}

else if (command == "create")

{

int id\_proc; // id of node for creating

string id\_proc\_;

for (int i = 7; i < recieved\_message.size(); ++i)

{

if (recieved\_message[i] != ' ')

{

id\_proc\_ += recieved\_message[i];

}

else

{

break;

}

}

id\_proc = stoi(id\_proc\_);

if (id\_proc == id)

{

send\_message("Error: Already exists", main\_socket);

}

else if (id\_proc > id)

{

if (right\_id == 0)

{ // there is not right node

right\_id = id\_proc;

int right\_id\_tmp = right\_id - 1;

while (1)

{

try

{

right\_socket.bind(adr\_right + to\_string(++right\_id\_tmp));

break;

}

catch (...)

{

}

}

adr\_right += to\_string(right\_id\_tmp);

char \*adr\_right\_ = new char[adr\_right.size() + 1];

memcpy(adr\_right\_, adr\_right.c\_str(), adr\_right.size() + 1);

char \*right\_id\_ = new char[to\_string(right\_id).size() + 1];

memcpy(right\_id\_, to\_string(right\_id).c\_str(), to\_string(right\_id).size() + 1);

char \*args[] = {"./child\_node", adr\_right\_, right\_id\_, NULL};

int f = fork();

if (f == 0)

{

execv("./child\_node", args);

}

else if (f == -1)

{

cout << "Error in forking in node with pid: " << getpid() << "**\n**";

}

else

{

zmq::message\_t message\_from\_node;

if (!right\_socket.recv(&message\_from\_node))

{

cout << "Error: can't receive message from right node in node with pid:" << getpid() << "**\n**";

}

string recieved\_message\_from\_node(static\_cast<char \*>(message\_from\_node.data()), message\_from\_node.size());

if (!main\_socket.send(message\_from\_node))

{

cout << "Error: can't send message to main node from node with pid:" << getpid() << "**\n**";

}

}

delete[] adr\_right\_;

delete[] right\_id\_;

}

else

{ // send task to right node

send\_message(recieved\_message, right\_socket);

// catch and send to parent

zmq::message\_t message;

if (!right\_socket.recv(&message))

{

cout << "Error: can't receive message from left node in node with pid: " << getpid() << "**\n**";

}

if (!main\_socket.send(message))

{

cout << "Error: can't send message to main node from node with pid: " << getpid() << "**\n**";

}

}

}

else

{

if (left\_id == 0)

{ // there is not left node

left\_id = id\_proc;

int left\_id\_tmp = left\_id - 1;

while (1)

{

try

{

left\_socket.bind(adr\_left + to\_string(++left\_id\_tmp));

break;

}

catch (...)

{

}

}

adr\_left += to\_string(left\_id\_tmp);

char \*adr\_left\_ = new char[adr\_left.size() + 1];

memcpy(adr\_left\_, adr\_left.c\_str(), adr\_left.size() + 1);

char \*left\_id\_ = new char[to\_string(left\_id).size() + 1];

memcpy(left\_id\_, to\_string(left\_id).c\_str(), to\_string(left\_id).size() + 1);

char \*args[] = {"./child\_node", adr\_left\_, left\_id\_, NULL};

int f = fork();

if (f == 0)

{

execv("./child\_node", args);

}

else if (f == -1)

{

cout << "Error in forking in node with pid: " << getpid() << "**\n**";

}

else

{

// catch message from new node

zmq::message\_t message\_from\_node;

if (!left\_socket.recv(&message\_from\_node))

{

cout << "Error: can't receive message from left node in node with pid:" << getpid() << "**\n**";

}

string recieved\_message\_from\_node(static\_cast<char \*>(message\_from\_node.data()), message\_from\_node.size());

// send message to main node

if (!main\_socket.send(message\_from\_node))

{

cout << "Error: can't send message to main node from node with pid:" << getpid() << "**\n**";

}

}

delete[] adr\_left\_;

delete[] left\_id\_;

}

else

{ // send task to left node

send\_message(recieved\_message, left\_socket);

// catch and send to parent

zmq::message\_t message;

if (!left\_socket.recv(&message))

{

cout << "Error: can't receive message from left node in node with pid: " << getpid() << "**\n**";

}

if (!main\_socket.send(message))

{

cout << "Error: can't send message to main node from node with pid: " << getpid() << "**\n**";

}

}

}

}

else if (command == "pingall")

{

if (left\_id != 0)

send\_message(recieved\_message, left\_socket);

if (right\_id != 0)

send\_message(recieved\_message, right\_socket);

string l = "OK";

string r = "OK";

if (left\_id != 0)

{

zmq::message\_t message\_left;

if (left\_socket.recv(&message\_left))

{

string left(static\_cast<char \*>(message\_left.data()), message\_left.size());

l = left;

}

else

{

l = to\_string(left\_id);

}

}

if (right\_id != 0)

{

zmq::message\_t message\_right;

if (right\_socket.recv(&message\_right))

{

string right(static\_cast<char \*>(message\_right.data()), message\_right.size());

r = right;

}

else

{

r = to\_string(right\_id);

}

}

if (l == r && l == "OK")

{

send\_message("OK", main\_socket);

}

else

{

if (l != "OK" && r != "OK")

{

send\_message(l + "; " + r + "; ", main\_socket);

}

else if (l != "OK")

{

send\_message(l + "; ", main\_socket);

}

else

{

send\_message(r + "; ", main\_socket);

}

}

}

else if (command == "kill")

{

int id\_proc;

string id\_proc\_;

for (int i = 5; i < recieved\_message.size(); ++i)

{

if (recieved\_message[i] != ' ')

{

id\_proc\_ += recieved\_message[i];

}

else

{

break;

}

}

id\_proc = stoi(id\_proc\_);

if (id\_proc > id)

{

if (right\_id == 0)

{

send\_message("Error: there isn`t node with this id", main\_socket);

}

else

{

if (right\_id == id\_proc)

{

send\_message("Ok: " + to\_string(right\_id), main\_socket);

send\_message("DIE", right\_socket);

right\_socket.unbind(adr\_right);

adr\_right = "tcp://127.0.0.1:300";

right\_id = 0;

}

else

{

right\_socket.send(message\_main);

zmq::message\_t message;

right\_socket.recv(&message);

main\_socket.send(message);

}

}

}

else if (id\_proc < id)

{

if (left\_id == 0)

{

send\_message("Error: there isn`t node with this id", main\_socket);

}

else

{

if (left\_id == id\_proc)

{

send\_message("OK: " + to\_string(left\_id), main\_socket);

send\_message("DIE", left\_socket);

left\_socket.unbind(adr\_left);

adr\_left = "tcp://127.0.0.1:300";

left\_id = 0;

}

else

{

left\_socket.send(message\_main);

zmq::message\_t message;

left\_socket.recv(&message);

main\_socket.send(message);

}

}

}

}

else if (command == "DIE")

{

if (left\_id)

{

send\_message("DIE", left\_socket);

left\_socket.unbind(adr\_left);

adr\_left = "tcp://127.0.0.1:300";

left\_id = 0;

}

if (right\_id)

{

send\_message("DIE", right\_socket);

right\_socket.unbind(adr\_right);

adr\_right = "tcp://127.0.0.1:300";

right\_id = 0;

}

main\_socket.unbind(adr);

return 0;

}

}

}

**Демонстрация работы программы**

g++ -fsanitize=address main\_prog.cpp -lzmq -o main\_prog -w

g++ -fsanitize=address child\_node.cpp -lzmq -o child\_node -w

yanniku@DESKTOP-US1A6DR:/mnt/c/Users/boris/CLionProjects/OCs/OS/os\_lab6-8/src$ ./main\_prog

Commands:

create id

exec id text pattern

pingall

kill id

exit

Please, enter command

create 2

OK: 2373

Please, enter command

create 5

OK: 2380

Please, enter command

create 1

OK: 2387

Please, enter command

exec 1 manofman man

OK:1:1;6;

Please, enter command

pingall

OK: -1

Please, enter command

kill 1

OK: 1

Please, enter command

pingall

OK: -1

Please, enter command

create 6

OK: 2395

Please, enter command

pingall

OK: -1

Please, enter command

kill 5

Ok: 5

Please, enter command

pingall

OK: -1

Please, enter command

kill 2

Tree was deleted

Please, enter command

pingall

No childs to ping!

Please, enter command

exit

**Выводы**Данная лабораторная работа была очень сложной для меня. Во время ее выполнения я полностью осознал концепцию очередей сообщений на основе zero message queue, научился читать документацию.