Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №5-7 по курсу**

**«Операционные системы»**

Группа: М8О-215Б-23

Студент: Тараскаев Д.М.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 01.12.24

Москва, 2024

**Постановка задачи**

Вариант 43.

Топология узлов: общий список  
 Тип команды: подсчет суммы n чисел  
 Тип проверки доступности узлов: ping id

**Общий метод и алгоритм решения**

В программе использовались следующие системные вызовы:

* execve - запуск исполняемого файла "./client"
* eventfd2 - создание дескриптора для уведомлений между потоками
* socket - создание сокета для сетевой коммуникации
* setsockopt - настройка параметров сокета
* bind - привязка сокета к определенному адресу/порту
* listen - перевод сокета в режим прослушивания для TCP-соединений
* fcntl - настройка неблокирующего режима для файловых дескрипторов
* clone3 - создание потоков для обработки сообщений
* prctl - установка имен для потоков (ZMQbg/Reaper, ZMQbg/IO/0)
* read - чтение данных из сокетов или дескрипторов
* write - запись данных в сокеты или дескрипторы
* sendto - отправка данных через сокет с указанием адресата
* recvmsg - получение данных из сокета с дополнительными метаданными
* close - закрытие файловых дескрипторов и сокетов
* poll - проверка готовности дескрипторов к операциям ввода-вывода
* getpid - получение идентификатора процесса

**Код программы**

**client.cpp**

#include <signal.h>

#include <atomic>

#include <chrono>

#include <map>

#include <mutex>

#include <thread>

#include "net\_func.h"

#include "node.h"

#include "set"

static std::mutex nodes\_mutex;

int main() {

std::set<int> all\_nodes;

std::string prog\_path = "./worker";

Node me(-1);

all\_nodes.insert(-1);

std::string command;

while (std::cin >> command) {

if (command == "create") {

int id\_child, id\_parent;

std::cin >> id\_child >> id\_parent;

if (all\_nodes.find(id\_child) != all\_nodes.end()) {

std::cout << "Error: Already exists" << std::endl;

}

else if (all\_nodes.find(id\_parent) == all\_nodes.end()) {

std::cout << "Error: Parent not found" << std::endl;

}

else if (id\_parent == me.id) {

std::string ans = me.Create\_child(id\_child, prog\_path);

std::cout << ans << std::endl;

all\_nodes.insert(id\_child);

}

else {

std::string str = "create " + std::to\_string(id\_child);

std::string ans = me.Send(str, id\_parent);

std::cout << ans << std::endl;

all\_nodes.insert(id\_child);

}

}

else if (command == "ping") {

int id\_child;

std::cin >> id\_child;

if (all\_nodes.find(id\_child) == all\_nodes.end()) {

std::cout << "Error: Not found" << std::endl;

}

else if (me.children.find(id\_child) != me.children.end()) {

std::string ans = me.Ping\_child(id\_child);

std::cout << ans << std::endl;

}

else {

std::string str = "ping " + std::to\_string(id\_child);

std::string ans = me.Send(str, id\_child);

if (ans == "Error: not find") {

ans = "Ok: 0";

}

std::cout << ans << std::endl;

}

}

else if (command == "exec") {

int id, n;

std::cin >> id >> n;

// Формируем сообщение с количеством чисел и самими числами

std::string msg = "exec " + std::to\_string(n);

for (int i = 0; i < n; i++) {

int num;

std::cin >> num;

msg += " " + std::to\_string(num);

}

if (all\_nodes.find(id) == all\_nodes.end()) {

std::cout << "Error: Not found" << std::endl;

}

else {

std::string ping\_result =

me.Send("ping " + std::to\_string(id), id);

if (ping\_result == "Ok: 0" ||

ping\_result == "Error: not find") {

std::cout << "Error: Node is unavailable" << std::endl;

all\_nodes.erase(id);

}

else {

std::string ans = me.Send(msg, id);

std::cout << ans << std::endl;

}

}

}

}

me.Remove();

return 0;

}

**worker.cpp**

#include <signal.h>

#include <chrono>

#include <fstream>

#include <vector>

#include "net\_func.h"

#include "node.h"

int my\_id = 0;

int main(int argc, char \*\*argv) {

if (argc != 3) {

return -1;

}

Node me(atoi(argv[1]), atoi(argv[2]));

my\_id = me.id;

std::string prog\_path = "./worker";

while (1) {

if (getppid() == 1) { // Если родительский процесс мертв, завершаем узел

std::cout << "[WORKER] Parent process died. Exiting...\n";

break;

}

std::string message;

std::string command = " ";

message = my\_net::reseave(&(me.parent));

std::istringstream request(message);

request >> command;

if (command == "create") {

int id\_child;

request >> id\_child;

std::string ans = me.Create\_child(id\_child, prog\_path);

my\_net::send\_message(&me.parent, ans);

}

else if (command == "pid") {

std::string ans = me.Pid();

my\_net::send\_message(&me.parent, ans);

}

else if (command == "ping") {

int id\_child;

request >> id\_child;

std::string ans = me.Ping\_child(id\_child);

my\_net::send\_message(&me.parent, ans);

}

else if (command == "send") {

int id;

request >> id;

std::string str;

getline(request, str);

str.erase(0, 1);

std::string ans;

ans = me.Send(str, id);

my\_net::send\_message(&me.parent, ans);

}

else if (command == "exec") {

int n;

request >> n;

int sum = 0;

// Считываем и суммируем n чисел

for (int i = 0; i < n; i++) {

int num;

request >> num;

sum += num;

}

std::string ans =

"Ok:" + std::to\_string(me.id) + ":" + std::to\_string(sum);

my\_net::send\_message(&me.parent, ans);

}

}

sleep(1);

return 0;

}

**net\_func.h**

#pragma once

#include <iostream>

#include <sstream>

#include <string>

#include <zmq.hpp>

namespace my\_net {

#define MY\_PORT 59000

#define MY\_IP "tcp://localhost:"

int bind(zmq::socket\_t \*socket, int id) {

int port = MY\_PORT + id;

while (true) {

std::string adress = MY\_IP + std::to\_string(port);

try {

socket->bind(adress);

break;

} catch (...) {

port++;

}

}

return port;

}

void connect(zmq::socket\_t \*socket, int port) {

std::string adress = MY\_IP + std::to\_string(port);

socket->connect(adress);

}

void unbind(zmq::socket\_t \*socket, int port) {

std::string adress = MY\_IP + std::to\_string(port);

socket->unbind(adress);

}

void disconnect(zmq::socket\_t \*socket, int port) {

std::string adress = MY\_IP + std::to\_string(port);

socket->disconnect(adress);

}

void send\_message(zmq::socket\_t \*socket, const std::string msg) {

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

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

try {

socket->send(message);

} catch (...) {

}

}

std::string reseave(zmq::socket\_t \*socket) {

zmq::message\_t message;

bool success = true;

try {

socket->recv(&message, 0);

} catch (...) {

success = false;

}

if (!success || message.size() == 0) {

throw -1;

}

std::string str(static\_cast<char \*>(message.data()), message.size());

return str;

}

} // namespace my\_net

urn a + b;

}

**node.h**

#include <iostream>

#include <sstream>

#include <unordered\_map>

#include "net\_func.h"

#include "unistd.h"

class Node {

private:

zmq::context\_t context;

public:

std::unordered\_map<int, zmq::socket\_t \*> children;

std::unordered\_map<int, int> children\_port;

zmq::socket\_t parent;

int parent\_port;

int id;

Node(int \_id, int \_parent\_port = -1)

: parent(context, ZMQ\_REP), parent\_port(\_parent\_port), id(\_id) {

if (\_id != -1) {

my\_net::connect(&parent, \_parent\_port);

}

}

std::string Ping\_child(int \_id) {

std::string ans = "Ok: 0";

if (\_id == id) {

ans = "Ok: 1";

return ans;

} else if (children.find(\_id) != children.end()) {

std::string msg = "ping " + std::to\_string(\_id);

my\_net::send\_message(children[\_id], msg);

try {

msg = my\_net::reseave(children[\_id]);

if (msg == "Ok: 1") ans = msg;

}

catch (int) {

}

return ans;

}

else {

return ans;

}

}

std::string Create\_child(int child\_id, std::string program\_path) {

std::string program\_name =

program\_path.substr(program\_path.find\_last\_of("/") + 1);

children[child\_id] = new zmq::socket\_t(context, ZMQ\_REQ);

int new\_port = my\_net::bind(children[child\_id], child\_id);

children\_port[child\_id] = new\_port;

int pid = fork();

if (pid == 0) {

execl(program\_path.c\_str(), program\_name.c\_str(),

std::to\_string(child\_id).c\_str(),

std::to\_string(new\_port).c\_str(), (char \*)NULL);

} else {

std::string child\_pid;

try {

children[child\_id]->setsockopt(ZMQ\_SNDTIMEO, 3000);

my\_net::send\_message(children[child\_id], "pid");

child\_pid = my\_net::reseave(children[child\_id]);

}

catch (int) {

child\_pid = "Error: can't connect to child";

}

return "Ok: " + child\_pid;

}

}

std::string Pid() { return std::to\_string(getpid()); }

std::string Send(std::string str, int \_id) {

if (\_id == id) {

return "Ok: 1";

}

if (children.find(\_id) != children.end()) {

if (Ping\_child(\_id) == "Ok: 1") {

my\_net::send\_message(children[\_id], str);

try {

return my\_net::reseave(children[\_id]);

}

catch (int) {

return "Error: not find";

}

}

}

for (auto &child : children) {

std::string msg = "send " + std::to\_string(\_id) + " " + str;

my\_net::send\_message(children[child.first], msg);

try {

std::string response = my\_net::reseave(children[child.first]);

if (response != "Error: not find") {

return response;

}

}

catch (int) {

}

}

return "Error: not find";

}

std::string Remove() {

std::string ans;

if (children.size() > 0) {

for (auto &child : children) {

if (Ping\_child(child.first) == "Ok: 1") {

std::string msg = "remove";

my\_net::send\_message(children[child.first], msg);

try {

msg = my\_net::reseave(children[child.first]);

if (ans.size() > 0) ans = ans + " " + msg;

else ans = msg;

}

catch (int) { }

}

my\_net::unbind(children[child.first],

children\_port[child.first]);

children[child.first]->close();

}

children.clear();

children\_port.clear();

}

return ans;

}

};

**Протокол работы программы**

**Тестирование**

$ ./client

create 1 -1

Ok: 13019

create 2 1

Ok: 13026

create 3 2

Ok: 13029

exec 3 2 1 3

Ok:3:4

ping 2

Ok: 1

(В другом терминале $ kill -9 13026)

ping 2

Ok: 0

exit

**strace:**

$ strace ./client

14163 eventfd2(0, EFD\_CLOEXEC) = 3

14163 fcntl(3, F\_GETFL) = 0x2 (flags O\_RDWR)

14163 fcntl(3, F\_SETFL, O\_RDWR|O\_NONBLOCK) = 0

# ZeroMQ использует eventfd для асинхронного оповещения о событиях между потоками.  
 14163 socket(AF\_INET, SOCK\_STREAM|SOCK\_CLOEXEC, IPPROTO\_TCP) = 10

14163 setsockopt(10, SOL\_SOCKET, SO\_REUSEADDR, [1], 4) = 0

14163 bind(10, {sa\_family=AF\_INET, sin\_port=htons(59001), sin\_addr=inet\_addr("127.0.0.1")}, 16) = 0

14163 listen(10, 100) = 0

#Здесь программа создает TCP-сокет и привязывает его к порту 59001 на локальном интерфейсе.  
 14163 clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, ...

14164 prctl(PR\_SET\_NAME, "ZMQbg/Reaper") = 0

...

14165 prctl(PR\_SET\_NAME, "ZMQbg/IO/0") = 0

# потоки для обработки сообщений (Reaper для управления ресурсами и IO для операций ввода-вывода).

14163 clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, ...

14185 execve("./worker", ["worker", "1", "59001"], 0x7ffcb02eb0c8 /\* 51 vars \*/)

**Вывод**

Программа успешно демонстрирует работу с очередью сообщений.