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: Верменников М.В.

Преподаватель: Миронов Е.С.

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

Дата: 12.12.24

Москва, 2024

**Постановка задачи**

**Вариант 31.**

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. 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.

Примечание: выполнение команд должно быть асинхронным. Т.е. пока выполняется команда на одном из вычислительных узлов, то можно отправить следующую команду на другой вычислительный узел.

Топология: узлы находятся в дереве общего вида.

Команда: локальный целочисленный словарь

Формат команды сохранения значения: exec id name value

id – целочисленный идентификатор вычислительного узла, на который отправляется команда

name – ключ, по которому будет сохранено значение (строка формата [A-Za-z0-9]+)

value – целочисленное значение

Формат команды загрузки значения: exec id name.

Проверка доступности: Формат команды: ping id

Команда проверяет доступность конкретного узла. Если узла нет, то необходимо выводить ошибку.

**Общий метод и алгоритм решения**

Для реализации системы очередей сообщений используем библиотеку **ZeroMQ**.

Использованные системные вызовы:

1. int select(int nfds, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, struct timeval \*timeout); Ожидает готовности файловых дескрипторов.
2. pid\_t fork(void); Создаёт новый процесс.
3. int execl(const char \*path, const char \*arg, ...); Заменяет текущий процесс новым процессом.
4. pid\_t getpid(void); Возвращает идентификатор текущего процесса.
5. int zmq\_msg\_init\_size(zmq\_msg\_t \*msg, size\_t size); Инициализирует сообщение ZeroMQ с указанным размером.
6. int zmq\_msg\_send(zmq\_msg\_t \*msg, void \*socket, int flags); Отправляет сообщение через сокет ZeroMQ.
7. void zmq\_msg\_init(zmq\_msg\_t \*msg); Инициализирует сообщение ZeroMQ.
8. int zmq\_msg\_recv(zmq\_msg\_t \*msg, void \*socket, int flags); Получает сообщение через сокет ZeroMQ.
9. void \*zmq\_msg\_data(zmq\_msg\_t \*msg); Возвращает указатель на данные сообщения ZeroMQ.
10. void \*zmq\_ctx\_new(void); Создаёт новый контекст ZeroMQ.
11. void \*zmq\_socket(void \*context, int type); Создаёт новый сокет ZeroMQ.
12. int zmq\_connect(void \*socket, const char \*addr); Подключает сокет ZeroMQ к указанному адресу.
13. int zmq\_bind(void \*socket, const char \*addr); Привязывает сокет ZeroMQ к указанному адресу.

Для реализации потребуются написать два исполняемых файла: управляющий узел и вычислительный. Ниже описана общая логика (алгоритм решения) для варианта с топологией 1, набором команд 4 и проверкой доступности 3:

1. **Управляющий узел (control)**:
   * Запускается как главный процесс.
   * Создаёт контекст ZeroMQ, биндит сокет с типом DEALER (например, на порт 5555 + (-1) для управления).
   * Считывает пользовательские команды (используя, например, select или функцию inputAvailable(), чтобы не блокировать основной поток):
     + **create** child\_id parent\_id:
       - Если parent\_id == -1, напрямую запускает новый процесс:
         1. fork() → в потомке execl(...), передавая аргументом child\_id.
         2. В родителе создаёт структуру Node для учёта нового ребёнка.
       - Иначе отправляет сообщение Create всем своим дочерним узлам, чтобы оно дошло до узла с id = parent\_id.
     + **ping** id:  
       Отправляет команду Ping узлу с номером id (через вещание всем «детям»). Если узел ответит, значит доступен.
     + **exec** id:
       - Далее читает две строки: text\_string и pattern\_string.
       - Формирует сообщение команды поиска подстроки (ExecSub), где поле st содержит text\_string + '\n' + pattern\_string. Отправляет всем дочерним, пока не дойдёт до узла id.
     + **heartbit** **time**:
       - Устанавливает интервал (в миллисекундах) «стучания» для всех вычислительных узлов. Узлы будут слать Heartbit каждые time мс.
       - Управляющий узел у себя хранит последнее время прихода heartbeat от каждого узла и если за 4 × time не было «стука», печатает сообщение о недоступности этого узла.
   * В цикле контролирует входящие сообщения:
     + Если пришёл Create с подтверждением о запуске (содержит child.id и child.pid), выводит Ok: pid.
     + При Ping от узла id — выводит, что узел доступен.
     + При ExecSub (поиск подстроки) отвечает узел, присылая позиции в m.st → выводим Ok:id:....
     + При Heartbit — обновляем время последнего «стука» от m.id.
     + Если сообщение «зависает» (нет ответа от узла > 5 с), считаем узел недоступным для этой команды.
   * Небольшая задержка (например, usleep(100000)) в конце каждой итерации, чтобы не нагружать процессор.
2. **Вычислительный узел (computing)**:
   * Запускается при create child\_id parent\_id, если parent\_id нашёлся.
   * Подключается (через zmq\_connect) к сокету родителя.
   * В цикле обрабатывает входящие сообщения:
     + Create:
       - Если m.id == my\_id, тогда fork() + execl(...), порождая нового потомка m.num.
       - Иначе (если m.id != my\_id) пробрасывает команду дальше своим детям.
     + Ping:
       - Если m.id == my\_id, отвечает сразу (отправляя Ping назад).
       - Иначе пробрасывает вниз по «дереву» (дочерним узлам).
     + ExecSub:
       - Если m.id == my\_id, разбирает поле st: первая строка — это text\_string, вторая — pattern\_string.  
         Выполняет поиск всех вхождений (можно любым алгоритмом) и возвращает родителю результат в виде: «0;3;5» или «-1», если не найдено.
       - Иначе передаёт дальше детям.
     + Heartbit:
       - Запоминает time, с которым ему нужно «стучать». После этого через каждую проверку времени (например, раз в ~100 мс) смотрит, не пора ли слать родителю команду Heartbit, my\_id, 0.
   * Точно так же узел поддерживает свой список «детей» (других процессов, которых он создал), каждому из них делает zmq\_bind или zmq\_connect (зависит от логики) и пересылает команды по аналогии с родителем.

Вся передача сообщений идёт через ZeroMQ‑сокеты типа DEALER, с флагом ZMQ\_DONTWAIT. Это даёт возможность асинхронного обмена: родитель в любой момент может читать сообщения от узлов, узлы — в любой момент принимать команды и отвечать.

**Код программы**

lib.h

#ifndefMY\_LIB\_H

#defineMY\_LIB\_H

#include<iostream>

#include<list>

#include<unordered\_set>

#include<chrono>

#include<ctime>

#include<string>

#include<cstring>

#include<unistd.h>

#include<sys/wait.h>

#include"zmq.h"

#include<sys/select.h>

#include<map>

// Проверка консольного ввода (не блокируясь).

boolinputAvailable();

// Текущее время (time\_t).

std::time\_tt\_now();

// ------------------------------------------------------------

// Перечисление команд (соответствует нашему решению)

//

// None = 0 — отсутствие команды

// Create = 1 — создать новый узел

// Ping = 2 — проверка доступности узла ("ping")

// ExecSub = 3 — поиск подстроки (вариант 4)

// Heartbit = 4 — «стук» для проверки доступности (вариант 3)

// ErrorResp = 5 — дополнительная команда для сообщений об ошибке

// ------------------------------------------------------------

enumcom: char{

None =0,

Create =1,

Ping =2,

ExecSub =3,

Heartbit =4,

ErrorResp=5

};

// Структура сообщения

classmessage{

public:

message() {}

// Конструктор без строки

message(comcommand, intid, intnum)

: command(command), id(id), num(num), sent\_time(t\_now())

{

std::memset(st, 0, sizeof(st));

}

// Конструктор со строкой

message(comcommand, intid, intnum, constchars[])

: command(command), id(id), num(num), sent\_time(t\_now())

{

std::memset(st, 0, sizeof(st));

strncpy(st, s, sizeof(st) -1);

}

booloperator==(constmessage&other) const{

return(command==other.command&&

id==other.id&&

num==other.num&&

sent\_time==other.sent\_time&&

std::strcmp(st, other.st) ==0);

}

comcommand; // Команда

intid; // ID узла (или родителя), кому адресовано / от кого

intnum; // Доп. числовое поле (PID, или время heartbeat, и пр.)

std::time\_tsent\_time;// время отправки (для таймаута)

charst[100]; // Строка (текст, паттерн и т.д.)

};

// Класс, описывающий узел

classNode{

public:

intid; // id узла

pid\_tpid; // pid процесса

void\*context;

void\*socket;

std::stringaddress;

booloperator==(constNode&other) const{

return(id==other.id&&pid==other.pid);

}

};

// Создание сокета (бинд или коннект)

NodecreateNode(intid, boolis\_child);

// Создание нового процесса (fork + execl("./computing", ...))

NodecreateProcess(intid);

// Отправка / приём сообщений (ZeroMQ)

voidsend\_mes(Node&node, constmessage&m);

messageget\_mes(Node&node);

#endif// MY\_LIB\_H

lib.cpp

#include"lib.h"

#include<sys/select.h>

#include<cstring>

boolinputAvailable() {

// Неблокирующая проверка ввода

structtimevaltv;

fd\_setfds;

tv.tv\_sec=0;

tv.tv\_usec=0;

FD\_ZERO(&fds);

FD\_SET(STDIN\_FILENO, &fds);

select(STDIN\_FILENO+1, &fds, NULL, NULL, &tv);

return(FD\_ISSET(STDIN\_FILENO, &fds));

}

std::time\_tt\_now() {

returnstd::chrono::system\_clock::to\_time\_t(std::chrono::system\_clock::now());

}

NodecreateNode(intid, boolis\_child) {

Nodenode;

node.id=id;

node.pid=getpid();

node.context=zmq\_ctx\_new();

node.socket =zmq\_socket(node.context, ZMQ\_DEALER);

// Порт: 5555 + id (пример)

node.address="tcp://127.0.0.1:"+std::to\_string(5555+id);

if(is\_child) {

// Узел-коннект

zmq\_connect(node.socket, node.address.c\_str());

} else{

// Узел-бинд

zmq\_bind(node.socket, node.address.c\_str());

}

returnnode;

}

NodecreateProcess(intid) {

pid\_tpid=fork();

if(pid==0) {

// Дочерний процесс

execl("./computing", "computing", std::to\_string(id).c\_str(), NULL);

std::cerr<<"execl failed"<<std::endl;

\_exit(1);

} elseif(pid==-1) {

// Ошибка при fork

std::cerr<<"Fork failed"<<std::endl;

\_exit(1);

}

// Родитель

Nodenode=createNode(id, false);

node.pid =pid;

returnnode;

}

voidsend\_mes(Node&node, constmessage&m) {

zmq\_msg\_trequest;

zmq\_msg\_init\_size(&request, sizeof(m));

std::memcpy(zmq\_msg\_data(&request), &m, sizeof(m));

zmq\_msg\_send(&request, node.socket, ZMQ\_DONTWAIT);

zmq\_msg\_close(&request);

}

messageget\_mes(Node&node) {

zmq\_msg\_tmsg;

zmq\_msg\_init(&msg);

intrc=zmq\_msg\_recv(&msg, node.socket, ZMQ\_DONTWAIT);

if(rc==-1) {

// Нет сообщения

zmq\_msg\_close(&msg);

returnmessage(None, -1, -1);

}

messagem;

std::memcpy(&m, zmq\_msg\_data(&msg), sizeof(message));

zmq\_msg\_close(&msg);

returnm;

}

control.cpp

#include"lib.h"

#include<sstream>

#include<limits>

#include<algorithm>

// Если хотим хранить время последнего heartbit от узлов:

staticstd::map<int, std::time\_t>last\_heartbeat;

staticintheartbit\_interval\_ms=0;// 0, если не используем heartbeat

intmain() {

// Множество всех известных id

std::unordered\_set<int>all\_id;

all\_id.insert(-1);

// Список дочерних узлов (прямые дети управляющего)

std::list<Node>children;

// Список «висящих» сообщений

std::list<message>saved\_mes;

std::stringinput\_line;

while(true) {

// 1. Приём сообщений от детей

for(auto&child: children) {

while(true) {

messagem=get\_mes(child);

if(m.command==None) {

break;// Нет новых сообщений

}

switch(m.command) {

caseCreate:

// Подтверждение "Ok: pid"

all\_id.insert(m.id);

std::cout<<"Ok: "<<m.num<<std::endl;

// Убираем из saved\_mes

{

autoit=std::find\_if(saved\_mes.begin(), saved\_mes.end(),

[&](constmessage&mm) {

returnmm.command==Create&&mm.num==m.id;

});

if(it!=saved\_mes.end()) {

saved\_mes.erase(it);

}

}

break;

casePing:

// "Ok: X is available"

std::cout<<"Ok: "<<m.id<<" is available"<<std::endl;

{

autoit=std::find\_if(saved\_mes.begin(), saved\_mes.end(),

[&](constmessage&mm) {

returnmm.command==Ping&&mm.id==m.id;

});

if(it!=saved\_mes.end()) {

saved\_mes.erase(it);

}

}

break;

caseExecSub:

// Ответ на поиск подстроки: m.st = "0;3;..." или "-1"

std::cout<<"Ok: "<<m.id<<": "<<m.st<<std::endl;

{

autoit=std::find\_if(saved\_mes.begin(), saved\_mes.end(),

[&](constmessage&mm) {

returnmm.command==ExecSub&&mm.id==m.id;

});

if(it!=saved\_mes.end()) {

saved\_mes.erase(it);

}

}

break;

caseHeartbit:

// Узел m.id стукнул

last\_heartbeat[m.id]=t\_now();

break;

caseErrorResp:

// Узел прислал ошибку

std::cerr<<"Error from node "<<m.id<<": "<<m.st<<std::endl;

break;

default:

break;

}

}

}

// 2. Проверка «просроченных» сообщений (таймаут 30с)

for(autoit=saved\_mes.begin(); it!=saved\_mes.end();) {

doublediff\_sec=std::difftime(t\_now(), it->sent\_time);

if(diff\_sec>30.0) {

// Считаем узел недоступным

switch(it->command) {

caseCreate:

std::cout<<"Error: cannot create child with parent "

<<it->id<<" (parent unavailable)"<<std::endl;

break;

casePing:

std::cout<<"Error: node "<<it->id<<" is unavailable"<<std::endl;

break;

caseExecSub:

std::cout<<"Error: node "<<it->id

<<" is unavailable (no exec response)"<<std::endl;

break;

default:

break;

}

it=saved\_mes.erase(it);

} else{

++it;

}

}

// 3. Проверка heartbeat (если установлено heartbit\_interval\_ms > 0)

if(heartbit\_interval\_ms>0) {

autonow\_t=t\_now();

for(autonid: all\_id) {

if(nid==-1) continue;// Управляющий

// Когда последний раз стучал?

if(!last\_heartbeat.count(nid)) {

continue;// Ещё ни разу не пришёл стук

}

doublems\_diff=std::difftime(now\_t, last\_heartbeat[nid]) \*1000.0;

if(ms\_diff>4.0\*heartbit\_interval\_ms) {

std::cout<<"Heartbit: node "<<nid<<" is unavailable now"<<std::endl;

// Можно убрать из last\_heartbeat, чтобы не повторять

last\_heartbeat.erase(nid);

}

}

}

// 4. Читаем пользовательский ввод (не блокируясь)

if(!inputAvailable()) {

usleep(50000);

continue;

}

// Считываем строку целиком

if(!std::getline(std::cin, input\_line)) {

// EOF или ошибка — можно сделать break или continue

continue;

}

if(input\_line.empty()) {

// Пустая строка

continue;

}

// Разбираем первую команду

std::istringstreamiss(input\_line);

std::stringcommand;

iss>>command;

if(command=="create") {

intchild\_id, parent\_id;

if(!(iss>>child\_id>>parent\_id)) {

std::cout<<"Error: invalid usage of create"<<std::endl;

continue;

}

if(all\_id.count(child\_id)) {

std::cout<<"Error: Node with id "<<child\_id<<" already exists"<<std::endl;

continue;

}

if(!all\_id.count(parent\_id)) {

std::cout<<"Error: Parent with id "<<parent\_id<<" not found"<<std::endl;

continue;

}

if(parent\_id==-1) {

// Создаём локально

Nodech=createProcess(child\_id);

children.push\_back(ch);

all\_id.insert(child\_id);

std::cout<<"Ok: "<<ch.pid<<std::endl;

} else{

// Посылаем команду Create -> parent\_id, child\_id

messagem(Create, parent\_id, child\_id);

saved\_mes.push\_back(m);

for(auto&ch: children) {

send\_mes(ch, m);

}

}

}

elseif(command=="ping") {

intid;

if(!(iss>>id)) {

std::cout<<"Error: invalid usage of ping"<<std::endl;

continue;

}

if(!all\_id.count(id)) {

std::cout<<"Error: Node with id "<<id<<" doesn't exist"<<std::endl;

continue;

}

messagem(Ping, id, 0);

saved\_mes.push\_back(m);

for(auto&ch: children) {

send\_mes(ch, m);

}

}

elseif(command=="heartbit") {

intms;

if(!(iss>>ms)) {

std::cout<<"Error: invalid usage of heartbit"<<std::endl;

continue;

}

if(ms<=0) {

std::cout<<"Error: interval must be positive"<<std::endl;

continue;

}

heartbit\_interval\_ms=ms;

std::cout<<"Ok"<<std::endl;

// Рассылаем команду Heartbit, чтобы узлы начали стучать

messagem(Heartbit, -1, ms);

for(auto&ch: children) {

send\_mes(ch, m);

}

}

elseif(command=="exec") {

// Формат: exec <node\_id>

intnode\_id;

if(!(iss>>node\_id)) {

std::cout<<"Error: invalid usage of exec"<<std::endl;

continue;

}

if(!all\_id.count(node\_id)) {

std::cout<<"Error: Node with id "<<node\_id<<" doesn't exist"<<std::endl;

continue;

}

// Далее ждём 2 строки: текст и паттерн

std::stringtext\_line;

if(!std::getline(std::cin, text\_line) ||text\_line.empty()) {

std::cout<<"Error: no text string for exec"<<std::endl;

continue;

}

std::stringpattern\_line;

if(!std::getline(std::cin, pattern\_line) ||pattern\_line.empty()) {

std::cout<<"Error: no pattern string for exec"<<std::endl;

continue;

}

// Формируем "text\_line\npattern\_line"

std::stringcombined=text\_line+"\n"+pattern\_line;

messagem(ExecSub, node\_id, 0, combined.c\_str());

saved\_mes.push\_back(m);

for(auto&ch: children) {

send\_mes(ch, m);

}

}

else{

std::cout<<"Error: Command doesn't exist!"<<std::endl;

}

usleep(50000);

}

return0;

}

computing.cpp

#include"lib.h"

#include<vector> // нужно для std::vector

#include<string>

#include<list>

#include<algorithm>

#include<chrono>

#include<iostream>

#include<unistd.h>

#include<cstring>

// Вспомогательная функция поиска ВСЕХ вхождений pattern в text

staticstd::vector<int> findAllOccurrences(conststd::string&text,

conststd::string&pattern)

{

std::vector<int>positions;

if(pattern.empty()) {

returnpositions;

}

size\_tstart=0;

while(true) {

size\_tpos=text.find(pattern, start);

if(pos==std::string::npos) {

break;

}

positions.push\_back((int)pos);

start=pos+1;// ищем далее

}

returnpositions;

}

intmain(intargc, char\*argv[])

{

// Мой id

intmy\_id=std::atoi(argv[1]);

// Создаём узел (is\_child = true => zmq\_connect)

Nodeself=createNode(my\_id, true);

// Храним список своих детей

std::list<Node>children;

// Период heartbeat (мс), если >0, шлём стук родителю

intheartbit\_interval\_ms=0;

// Запомним время, когда в последний раз отстукивали

autolast\_hb\_time=std::chrono::steady\_clock::now();

while(true) {

// 1. Проверяем сообщения от детей

for(auto&child: children) {

while(true) {

messagecm=get\_mes(child);

if(cm.command==None) {

break;// нет сообщений

}

// Всё, что приходит от детей, пробрасываем родителю

send\_mes(self, cm);

}

}

// 2. Проверяем сообщение от родителя

while(true) {

messagem=get\_mes(self);

if(m.command==None) {

break;// нет сообщений

}

switch(m.command) {

caseCreate:

if(m.id==my\_id) {

// Создаём нового ребёнка

Nodech=createProcess(m.num);

children.push\_back(ch);

// Отправляем родителю подтверждение

messageack(Create, ch.id, ch.pid);

send\_mes(self, ack);

} else{

// Пробрасываем дальше детям

for(auto&ch: children) {

send\_mes(ch, m);

}

}

break;

casePing:

if(m.id==my\_id) {

// Ответ

send\_mes(self, m);

} else{

// Проброс

for(auto&ch: children) {

send\_mes(ch, m);

}

}

break;

caseExecSub:

if(m.id==my\_id) {

// Разбираем text + '\n' + pattern

std::stringcombined(m.st);

autopos\_nl=combined.find('\n');

if(pos\_nl==std::string::npos) {

// Неверный формат => шлём ошибку (ErrorResp)

messageerr(ErrorResp, my\_id, 0, "Bad exec format");

send\_mes(self, err);

break;

}

std::stringtext\_str =combined.substr(0, pos\_nl);

std::stringpattern\_str=combined.substr(pos\_nl+1);

// Поиск

autoocc=findAllOccurrences(text\_str, pattern\_str);

if(occ.empty()) {

// Шлём "-1"

messageans(ExecSub, my\_id, 0, "-1");

send\_mes(self, ans);

} else{

// Формируем "0;3;5" и т.д.

std::stringres;

for(size\_ti=0; i<occ.size(); i++) {

if(i>0) res+=";";

res+=std::to\_string(occ[i]);

}

messageans(ExecSub, my\_id, 0, res.c\_str());

send\_mes(self, ans);

}

} else{

// Пробрасываем детям

for(auto&ch: children) {

send\_mes(ch, m);

}

}

break;

caseHeartbit:

// Родитель сказал "стучи" каждые m.num мс

heartbit\_interval\_ms=m.num;

break;

caseErrorResp:

// Можно как-то обработать

std::cerr<<"Error from parent: "<<m.st<<std::endl;

break;

default:

// Проброс дальше детям

for(auto&ch: children) {

send\_mes(ch, m);

}

break;

}

}

// 3. Отправляем heartbeat (если нужно)

if(heartbit\_interval\_ms>0) {

autonow=std::chrono::steady\_clock::now();

autodiff=std::chrono::duration\_cast<std::chrono::milliseconds>(now-last\_hb\_time).count();

if(diff>=heartbit\_interval\_ms) {

messagehb(Heartbit, my\_id, 0);

send\_mes(self, hb);

last\_hb\_time=std::chrono::steady\_clock::now();

}

}

usleep(50000);

}

return0;

}

**Протокол работы программы**

**Тестирование:**

**Тест 1.**

root@55217656a245:/workspaces/MAI\_OS\_Labs/lab5-7# ./control

create 1 -1

Ok: 14046

create 2 -1

Ok: 14057

create 3 2

Ok: 14071

create 4 3

Ok: 14082

pig 1

Error: Command doesn't exist!

ping 1

Ok: 1 is available

**Тест 2.**

root@55217656a245:/workspaces/MAI\_OS\_Labs/lab5-7# strace -f -o straced.txt ./control

create 1 -1

Ok: 18074

create 2 -1

Ok: 18101

create 22 2

Ok: 18124

exec 22

abracadabra

abra

Ok: 22: 0;7

pig 22

Error: Command doesn't exist!

ping 22

Ok: 22 is available

hearbit 2000

Error: Command doesn't exist!

heartbit 2000

Ok

exec 22

abracadabra

cad

Ok: 22: 4

**Strace:**

**18039 execve("./control", ["./control"], 0x7ffde6015368 /\* 26 vars \*/) = 0**

18039 brk(NULL) = 0x6b3a000

18039 mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f1d0466d000

18039 access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

18039 openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

18039 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=25786, ...}, AT\_EMPTY\_PATH) = 0

18039 mmap(NULL, 25786, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f1d04666000

18039 close(3) = 0

18039 openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libzmq.so.5", O\_RDONLY|O\_CLOEXEC) = 3

18039 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

18039 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=774184, ...}, AT\_EMPTY\_PATH) = 0

18039 mmap(NULL, 772088, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f1d045a9000

18039 mmap(0x7f1d045c2000, 487424, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19000) = 0x7f1d045c2000

18039 mmap(0x7f1d04639000, 147456, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x90000) = 0x7f1d04639000

18039 mmap(0x7f1d0465d000, 36864, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xb4000) = 0x7f1d0465d000

18039 close(3) = 0

18039 openat(AT\_FDCWD, "/usr/local/lib64/libstdc++.so.6", O\_RDONLY|O\_CLOEXEC) = 3

18039 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

18039 newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2530008, ...}, AT\_EMPTY\_PATH) = 0

18039 mmap(NULL, 2543808, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f1d0433b000

18039 mmap(0x7f1d043e0000, 1216512, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xa5000) = 0x7f1d043e0000

18039 mmap(0x7f1d04509000, 581632, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ce000) = 0x7f1d04509000

18039 mmap(0x7f1d04597000, 57344, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25c000) = 0x7f1d04597000

18039 mmap(0x7f1d045a5000, 12480, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f1d045a5000

18039 close(3) = 0

18039 openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libm.so.6", O\_RDONLY|O\_CLOEXEC) = 3

18039 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

18039 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=907784, ...}, AT\_EMPTY\_PATH) = 0

18039 mmap(NULL, 909560, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f1d0425c000

18039 mmap(0x7f1d0426c000, 471040, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x10000) = 0x7f1d0426c000

18039 mmap(0x7f1d042df000, 368640, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x83000) = 0x7f1d042df000

18039 mmap(0x7f1d04339000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xdc000) = 0x7f1d04339000

18039 close(3) = 0

18039 openat(AT\_FDCWD, "/usr/local/lib64/libgcc\_s.so.1", O\_RDONLY|O\_CLOEXEC) = 3

18039 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

18039 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=906528, ...}, AT\_EMPTY\_PATH) = 0

18039 mmap(NULL, 181160, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f1d0422f000

18039 mmap(0x7f1d04233000, 143360, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x4000) = 0x7f1d04233000

18039 mmap(0x7f1d04256000, 16384, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x27000) = 0x7f1d04256000

18039 mmap(0x7f1d0425a000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2b000) = 0x7f1d0425a000

18039 close(3) = 0

18039 openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

18039 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20t\2\0\0\0\0\0"..., 832) = 832

18039 pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

18039 newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=1922136, ...}, AT\_EMPTY\_PATH) = 0

18039 pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

18039 mmap(NULL, 1970000, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f1d0404e000

18039 mmap(0x7f1d04074000, 1396736, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x26000) = 0x7f1d04074000

18039 mmap(0x7f1d041c9000, 339968, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x17b000) = 0x7f1d041c9000

18039 mmap(0x7f1d0421c000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ce000) = 0x7f1d0421c000

18039 mmap(0x7f1d04222000, 53072, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f1d04222000

18039 close(3) = 0

18039 openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libbsd.so.0", O\_RDONLY|O\_CLOEXEC) = 3

18039 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

18039 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=84840, ...}, AT\_EMPTY\_PATH) = 0

18039 mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f1d0404c000

18039 mmap(NULL, 86224, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f1d04036000

18039 mmap(0x7f1d0403a000, 49152, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x4000) = 0x7f1d0403a000

18039 mmap(0x7f1d04046000, 12288, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x10000) = 0x7f1d04046000

18039 mmap(0x7f1d04049000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x13000) = 0x7f1d04049000

18039 mmap(0x7f1d0404b000, 208, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f1d0404b000

18039 close(3) = 0

18039 openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libsodium.so.23", O\_RDONLY|O\_CLOEXEC) = 3

18039 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 \314\0\0\0\0\0\0"..., 832) = 832

18039 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=363208, ...}, AT\_EMPTY\_PATH) = 0

18039 mmap(NULL, 365576, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f1d03fdc000

18039 mprotect(0x7f1d03fe8000, 311296, PROT\_NONE) = 0

18039 mmap(0x7f1d03fe8000, 233472, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xc000) = 0x7f1d03fe8000

18039 mmap(0x7f1d04021000, 73728, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x45000) = 0x7f1d04021000

18039 mmap(0x7f1d04034000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x57000) = 0x7f1d04034000

18039 close(3) = 0

18039 openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libpgm-5.3.so.0", O\_RDONLY|O\_CLOEXEC) = 3

18039 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220I\0\0\0\0\0\0"..., 832) = 832

18039 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=306400, ...}, AT\_EMPTY\_PATH) = 0

18039 mmap(NULL, 326096, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f1d03f8c000

18039 mmap(0x7f1d03f90000, 167936, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x4000) = 0x7f1d03f90000

18039 mmap(0x7f1d03fb9000, 118784, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2d000) = 0x7f1d03fb9000

18039 mmap(0x7f1d03fd6000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x49000) = 0x7f1d03fd6000

18039 mmap(0x7f1d03fd8000, 14800, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f1d03fd8000

18039 close(3) = 0

18039 openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libnorm.so.1", O\_RDONLY|O\_CLOEXEC) = 3

18039 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\300\246\0\0\0\0\0\0"..., 832) = 832

18039 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=489480, ...}, AT\_EMPTY\_PATH) = 0

18039 mmap(NULL, 1214976, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f1d03e63000

18039 mprotect(0x7f1d03e6d000, 438272, PROT\_NONE) = 0

18039 mmap(0x7f1d03e6d000, 278528, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xa000) = 0x7f1d03e6d000

18039 mmap(0x7f1d03eb1000, 155648, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x4e000) = 0x7f1d03eb1000

18039 mmap(0x7f1d03ed8000, 16384, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x74000) = 0x7f1d03ed8000

18039 mmap(0x7f1d03edc000, 719360, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f1d03edc000

18039 close(3) = 0

18039 openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libgssapi\_krb5.so.2", O\_RDONLY|O\_CLOEXEC) = 3

18039 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

18039 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=338680, ...}, AT\_EMPTY\_PATH) = 0

18039 mmap(NULL, 337152, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f1d03e10000

18039 mmap(0x7f1d03e1c000, 221184, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xc000) = 0x7f1d03e1c000

18039 mmap(0x7f1d03e52000, 53248, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x42000) = 0x7f1d03e52000

18039 mmap(0x7f1d03e5f000, 16384, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x4f000) = 0x7f1d03e5f000

18039 close(3) = 0

18039 openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libmd.so.0", O\_RDONLY|O\_CLOEXEC) = 3

18039 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

18039 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=47312, ...}, AT\_EMPTY\_PATH) = 0

18039 mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f1d03e0e000

18039 mmap(NULL, 49384, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f1d03e01000

18039 mmap(0x7f1d03e03000, 28672, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7f1d03e03000

18039 mmap(0x7f1d03e0a000, 8192, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x9000) = 0x7f1d03e0a000

18039 mmap(0x7f1d03e0c000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xa000) = 0x7f1d03e0c000

18039 close(3) = 0

18039 openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libpthread.so.0", O\_RDONLY|O\_CLOEXEC) = 3

18039 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

18039 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=14480, ...}, AT\_EMPTY\_PATH) = 0

18101 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18124 <... clock\_nanosleep resumed>NULL) = 0

18124 getpid() = 18124

18124 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18124 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

**18039 <... pselect6 resumed>) = 0 (Timeout)**

18101 getpid( <unfinished ...>

18039 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18101 <... getpid resumed>) = 18101

18101 poll([{fd=15, events=POLLIN}], 1, 0) = 0 (Timeout)

18101 getpid() = 18101

18101 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18101 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18124 <... clock\_nanosleep resumed>NULL) = 0

18124 getpid() = 18124

18124 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18124 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18074 <... clock\_nanosleep resumed>NULL) = 0

18074 getpid() = 18074

18074 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18074 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18039 <... clock\_nanosleep resumed>NULL) = 0

18039 getpid() = 18039

18039 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18039 getpid() = 18039

18039 poll([{fd=16, events=POLLIN}], 1, 0 <unfinished ...>

18101 <... clock\_nanosleep resumed>NULL) = 0

18039 <... poll resumed>) = 0 (Timeout)

18101 getpid( <unfinished ...>

**18039 pselect6(1, [0], NULL, NULL, {tv\_sec=0, tv\_nsec=0}, NULL <unfinished ...>**

18101 <... getpid resumed>) = 18101

**18039 <... pselect6 resumed>) = 0 (Timeout)**

18101 poll([{fd=15, events=POLLIN}], 1, 0 <unfinished ...>

18039 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18101 <... poll resumed>) = 0 (Timeout)

18124 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18124 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18074 <... clock\_nanosleep resumed>NULL) = 0

18074 getpid() = 18074

18074 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18074 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18101 <... clock\_nanosleep resumed>NULL) = 0

18101 getpid() = 18101

18101 poll([{fd=15, events=POLLIN}], 1, 0) = 0 (Timeout)

18101 getpid() = 18101

18101 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

18039 <... clock\_nanosleep resumed>NULL) = 0

18101 <... poll resumed>) = 0 (Timeout)

18039 getpid( <unfinished ...>

18101 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18039 <... getpid resumed>) = 18039

18039 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18039 getpid() = 18039

18039 poll([{fd=16, events=POLLIN}], 1, 0) = 0 (Timeout)

**18039 pselect6(1, [0], NULL, NULL, {tv\_sec=0, tv\_nsec=0}, NULL) = 0 (Timeout)**

18039 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18124 <... clock\_nanosleep resumed>NULL) = 0

18124 getpid() = 18124

18124 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18124 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18074 <... clock\_nanosleep resumed>NULL) = 0

18074 getpid() = 18074

18074 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18074 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18101 <... clock\_nanosleep resumed>NULL) = 0

18118 getpid( <unfinished ...>

18039 <... getpid resumed>) = 18039

18101 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18039 poll([{fd=16, events=POLLIN}], 1, 0 <unfinished ...>

18118 <... getpid resumed>) = 18101

18039 <... poll resumed>) = 0 (Timeout)

18118 poll([{fd=6, events=POLLIN}], 1, 0 <unfinished ...>

**18039 pselect6(1, [0], NULL, NULL, {tv\_sec=0, tv\_nsec=0}, NULL <unfinished ...>**

18118 <... poll resumed>) = 1 ([{fd=6, revents=POLLIN}])

**18039 <... pselect6 resumed>) = 0 (Timeout)**

18118 getpid( <unfinished ...>

18039 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18124 <... clock\_nanosleep resumed>NULL) = 0

18118 <... getpid resumed>) = 18101

18124 getpid( <unfinished ...>

18118 read(6, <unfinished ...>

18124 <... getpid resumed>) = 18124

18118 <... read resumed>"\1\0\0\0\0\0\0\0", 8) = 8

18124 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

18118 epoll\_ctl(7, EPOLL\_CTL\_MOD, 9, {events=EPOLLIN|EPOLLOUT, data={u32=1275073584, u64=140468180489264}} <unfinished ...>

18124 <... poll resumed>) = 0 (Timeout)

18118 <... epoll\_ctl resumed>) = 0

18124 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

**18118 sendto(9, "\0\200\4\0\0\0\2\0\0\0\0\0\0\0\376\177\0\0\244Omg\0\0\0\0\0\0\0\0\0\0"..., 130, 0, NULL, 0 <unfinished ...>**

18074 <... clock\_nanosleep resumed>NULL) = 0

**18118 <... sendto resumed>) = 130**

18103 <... epoll\_wait resumed>[{events=EPOLLIN, data={u32=4093643728, u64=139762329455568}}], 256, -1) = 1

18118 getpid( <unfinished ...>

18074 getpid( <unfinished ...>

18118 <... getpid resumed>) = 18101

**18103 recvfrom(18, <unfinished ...>**

18118 poll([{fd=6, events=POLLIN}], 1, 0 <unfinished ...>

18074 <... getpid resumed>) = 18074

18118 <... poll resumed>) = 0 (Timeout)

**18103 <... recvfrom resumed>"\0\200\4\0\0\0\2\0\0\0\0\0\0\0\376\177\0\0\244Omg\0\0\0\0\0\0\0\0\0\0"..., 8192, 0, NULL, NULL) = 130**

18118 epoll\_wait(7, <unfinished ...>

18074 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

18118 <... epoll\_wait resumed>[{events=EPOLLOUT, data={u32=1275073584, u64=140468180489264}}], 256, -1) = 1

18103 getpid( <unfinished ...>

18118 epoll\_ctl(7, EPOLL\_CTL\_MOD, 9, {events=EPOLLIN, data={u32=1275073584, u64=140468180489264}} <unfinished ...>

18074 <... poll resumed>) = 0 (Timeout)

18118 <... epoll\_ctl resumed>) = 0

18103 <... getpid resumed>) = 18039

18118 epoll\_wait(7, <unfinished ...>

18074 getpid( <unfinished ...>

18103 write(16, "\1\0\0\0\0\0\0\0", 8 <unfinished ...>

18074 <... getpid resumed>) = 18074

18103 <... write resumed>) = 8

18074 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

18103 epoll\_wait(15, <unfinished ...>

18074 <... poll resumed>) = 0 (Timeout)

18074 getpid() = 18074

18074 write(6, "\1\0\0\0\0\0\0\0", 8 <unfinished ...>

18082 <... epoll\_wait resumed>[{events=EPOLLIN, data={u32=975308208, u64=975308208}}], 256, -1) = 1

18074 <... write resumed>) = 8

18039 getpid() = 18039

18039 poll([{fd=16, events=POLLIN}], 1, 0) = 0 (Timeout)

18039 getpid() = 18039

18039 poll([{fd=16, events=POLLIN}], 1, 0 <unfinished ...>

18074 <... clock\_nanosleep resumed>NULL) = 0

18039 <... poll resumed>) = 0 (Timeout)

18074 getpid( <unfinished ...>

**18039 pselect6(1, [0], NULL, NULL, {tv\_sec=0, tv\_nsec=0}, NULL <unfinished ...>**

18074 <... getpid resumed>) = 18074

**18039 <... pselect6 resumed>) = 0 (Timeout)**

18074 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

18039 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18074 <... poll resumed>) = 0 (Timeout)

18074 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18101 <... clock\_nanosleep resumed>NULL) = 0

18101 getpid() = 18101

18101 poll([{fd=15, events=POLLIN}], 1, 0) = 0 (Timeout)

18101 getpid() = 18101

18101 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18101 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18124 <... clock\_nanosleep resumed>NULL) = 0

18124 getpid() = 18124

18124 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18124 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18039 <... clock\_nanosleep resumed>NULL) = 0

18039 getpid( <unfinished ...>

18074 <... clock\_nanosleep resumed>NULL) = 0

18039 <... getpid resumed>) = 18039

18074 getpid( <unfinished ...>

18039 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

18074 <... getpid resumed>) = 18074

18039 <... poll resumed>) = 0 (Timeout)

18074 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

18039 getpid() = 18039

18074 <... poll resumed>) = 0 (Timeout)

18039 poll([{fd=16, events=POLLIN}], 1, 0 <unfinished ...>

18074 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18039 <... poll resumed>) = 0 (Timeout)

**18039 pselect6(1, [0], NULL, NULL, {tv\_sec=0, tv\_nsec=0}, NULL) = 0 (Timeout)**

18039 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18101 <... clock\_nanosleep resumed>NULL) = 0

18101 getpid() = 18101

18101 poll([{fd=15, events=POLLIN}], 1, 0) = 0 (Timeout)

18101 getpid() = 18101

18101 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

18124 <... clock\_nanosleep resumed>NULL) = 0

18101 <... poll resumed>) = 0 (Timeout)

18124 getpid( <unfinished ...>

18101 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18124 <... getpid resumed>) = 18124

18124 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18124 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18074 <... clock\_nanosleep resumed>NULL) = 0

18074 getpid() = 18074

18074 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

18039 <... clock\_nanosleep resumed>NULL) = 0

18074 <... poll resumed>) = 0 (Timeout)

18039 getpid( <unfinished ...>

18074 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18039 <... getpid resumed>) = 18039

18074 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18101 <... clock\_nanosleep resumed>NULL) = 0

18101 getpid() = 18101

18101 poll([{fd=15, events=POLLIN}], 1, 0 <unfinished ...>

18039 <... clock\_nanosleep resumed>NULL) = 0

18101 <... poll resumed>) = 0 (Timeout)

18039 getpid( <unfinished ...>

18101 getpid( <unfinished ...>

18039 <... getpid resumed>) = 18039

18101 <... getpid resumed>) = 18101

18039 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

18101 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

18039 <... poll resumed>) = 0 (Timeout)

18101 <... poll resumed>) = 0 (Timeout)

18039 getpid( <unfinished ...>

18101 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18039 <... getpid resumed>) = 18039

18039 poll([{fd=16, events=POLLIN}], 1, 0) = 0 (Timeout)

**18039 pselect6(1, [0], NULL, NULL, {tv\_sec=0, tv\_nsec=0}, NULL) = 0 (Timeout)**

18039 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18124 <... clock\_nanosleep resumed>NULL) = 0

18124 getpid() = 18124

18124 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18124 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18074 <... clock\_nanosleep resumed>NULL) = 0

18074 getpid() = 18074

18074 poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

18074 clock\_nanosleep(CLOCK\_REALTIME, 0, {tv\_sec=0, tv\_nsec=50000000}, <unfinished ...>

18101 <... clock\_nanosleep resumed>NULL) = 0

18101 getpid() = 18101

18101 poll([{fd=15, events=POLLIN}], 1, 0 <unfinished ...>

18039 <... clock\_nanosleep resumed>NULL) = 0

18101 <... poll resumed>) = 0 (Timeout)

18039 getpid( <unfinished ...>

18101 getpid( <unfinished ...>

18039 <... getpid resumed>) = 18039

18101 <... getpid resumed>) = 18101

18039 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

18124 <... clock\_nanosleep resumed>NULL) = 0

18101 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

18039 <... poll resumed>) = 0 (Timeout)

18124 getpid( <unfinished ...>

18074 <... clock\_nanosleep resumed>NULL) = ? ERESTART\_RESTARTBLOCK (Interrupted by signal)

18039 --- SIGINT {si\_signo=SIGINT, si\_code=SI\_KERNEL} ---

18124 <... getpid resumed>) = 18124

18101 <... poll resumed>) = ? ERESTART\_RESTARTBLOCK (Interrupted by signal)

18124 --- SIGINT {si\_signo=SIGINT, si\_code=SI\_KERNEL} ---

18103 <... epoll\_wait resumed> <unfinished ...>) = ?

18102 <... epoll\_wait resumed> <unfinished ...>) = ?

18076 <... epoll\_wait resumed> <unfinished ...>) = ?

18075 <... epoll\_wait resumed> <unfinished ...>) = ?

18074 --- SIGINT {si\_signo=SIGINT, si\_code=SI\_KERNEL} ---

18128 <... epoll\_wait resumed> <unfinished ...>) = ?

18127 <... epoll\_wait resumed> <unfinished ...>) = ?

18103 +++ killed by SIGINT +++

18102 +++ killed by SIGINT +++

18101 --- SIGINT {si\_signo=SIGINT, si\_code=SI\_KERNEL} ---

18076 +++ killed by SIGINT +++

18128 +++ killed by SIGINT +++

18127 +++ killed by SIGINT +++

18126 <... epoll\_wait resumed> <unfinished ...>) = ?

18125 <... epoll\_wait resumed> <unfinished ...>) = ?

18124 +++ killed by SIGINT +++

18118 <... epoll\_wait resumed> <unfinished ...>) = ?

18117 <... epoll\_wait resumed> <unfinished ...>) = ?

18082 <... epoll\_wait resumed> <unfinished ...>) = ?

18080 <... epoll\_wait resumed> <unfinished ...>) = ?

18075 +++ killed by SIGINT +++

18039 +++ killed by SIGINT +++

18126 +++ killed by SIGINT +++

18125 +++ killed by SIGINT +++

18118 +++ killed by SIGINT +++

18117 +++ killed by SIGINT +++

18101 +++ killed by SIGINT +++

18082 +++ killed by SIGINT +++

18080 +++ killed by SIGINT +++

18074 +++ killed by SIGINT +++

**Вывод**

Лабораторная работа мне крайне понравилась! Очередь сообщений является эффективным инструментом для межпроцессорного взаимодействия, так как позволяет легко асинхронно обмениваться сообщениями и масштабировать систему, а так же синхронизировать процессы, выполняющие действия с разными скоростями, что очень важно в клиент-серверной архитектуре. ZeroMQ является удобной и эффективной библиотекой для создания пользовательских очередей сообщений.