Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №6, 7 и 8 по курсу

«Операционные системы»

Всё остальное

Студент: [ Нелюбин В.С ]

Группа: М8О–206Б–19

Вариант: 18

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2021.

Постановка задачи

**Цель работы**

Целью является приобретение практических навыков в:

* Управлении серверами сообщений (№6)
* Применение отложенных вычислений (№7)
* Интеграция программных систем друг с другом (№8)

## Задание

Реализовать распределенную систему по асинхронной обработке запросов. В данной распределенной системе должно существовать 2 вида узлов: «управляющий» и «вычислительный». Необходимо объединить данные узлы в соответствии с той топологией, которая определена вариантом. Связь между узлами необходимо осуществить при помощи технологии очередей сообщений. Также в данной системе необходимо предусмотреть проверку доступности узлов в соответствии с вариантом. При убийстве («kill -9») любого вычислительного узла система должна пытаться максимально сохранять свою работоспособность, а именно все дочерние узлы убитого узла могут стать недоступными, но родительские узлы должны сохранить свою работоспособность. Управляющий узел отвечает за ввод команд от пользователя и отправку этих команд на вычислительные узлы. Список основных поддерживаемых команд:

**create NodeID ParentID**

**remove NodeID**

**exec NodeID Key Value**

**exec NodeID Key**

**heartbit PulseRate**

**ping NodeID**

Так же добавлены команды, для облегчения пользования

**exit**

**admin info**

**Общие сведения о программе**

Программа компилируется из файла interface.cpp и node2.cpp.

Также используется заголовочные файлы: iostream chrono thread string map vector cstring unistd.h/windows.h zmq.hpp. Программа интерфейса принимает со стандартного потока строки команд, и разбирает из на аргументы. Затем, в зависимости от команды, интерфейс посылает команды структуре узлов, или выполняет инструкции сам. Общение обеспечено с помощью ZeroMQ. Команды посылаются по соединениям типа REQ - REP. Сигналы heartbit посылаются по сокетам PULL – PUSH. В узле проверяется, применима ли команда к текущему узлу, к одному из дочерних узлов, или не применима вообще. В зависимости от команды и её применимости, узел может послать команду выбранным дочерним узлам, или выполнить её сам. Для обеспечения работы heartbit, на ожидание команд от родительских узлов отведён отдельный поток. Аналогично, в программе интерфейса отдельный поток принимает сигналы heartbit от всех существующих узлов. Доступность узлов проверяется с помощью проверки, к какой группе относится их PID.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Понять, как пользоваться ZeroMQ.
2. Распределить функционал между интерфейсом и узлами.
3. Сформулировать, как возможно совместить heartbit и обработку команд (которая ставится на паузу после обработки команды, и до получения следующей).

**Основные файлы программы**

**interface.cpp:**

#include <iostream>

#include <thread>

#include <time.h>

#include <string>

#include <vector>

#include <map>

#include <cstring>

#ifndef \_WIN32

#include <unistd.h>

#else

#include <windows.h>

#define sleep(n) Sleep(n)

#endif

#include <zmq.hpp>

const int CMNM=6;

const size\_t TCPE=32;

int getIntM(char\*str,int&offs){

int rez=0,sig=1;

if(str[offs]=='-'){

sig=-1;

offs++;

}

for(;((str[offs]>='0')&&(str[offs]<='9'));offs++){

rez=rez\*10+str[offs]-'0';

}

rez\*=sig;

return rez;

}

bool checkTCP(char\*ard){

int i;

for(i=0;i<6;i++){

if(ard[i]!="tcp://"[i]){

printf("%s\n",ard);for(int k=0;k<i;k++){printf(" ");}printf("^ %c\n",ard[i]);

return false;

}

}

for(int j=0;j<4;j++){

while((ard[i]>='0')&&(ard[i]<='9')){

i++;

}

if((j<3)&&(ard[i]!='.')){

printf("%s\n",ard);for(int k=0;k<i;k++){printf(" ");}printf("^ %c\n",ard[i]);

return false;

}

if((j==3)&&(ard[i]!=':')){

printf("%s\n",ard);for(int k=0;k<i;k++){printf(" ");}printf("^ %c\n",ard[i]);

return false;

}

i++;

}

while((ard[i]>='0')&&(ard[i]<='9')){

i++;

}

if((ard[i]=='\0')||(ard[i]=='\n')){

return true;

}

printf("%s\n",ard);for(int k=0;k<i;k++){printf(" ");}printf("^ %c\n",ard[i]);

return false;

}

void killChild(std::vector<pid\_t> &pidsN , std::vector<int> &chidsN , std::vector<zmq::socket\_t> &socketsN,int idx){

socketsN[idx].close();

pidsN[idx]=0;

chidsN[idx]=0;

if(idx==pidsN.size()){

pidsN.resize(socketsN.size()-1);

chidsN.resize(socketsN.size()-1);

socketsN.resize(socketsN.size()-1);

}

}

/\*bool sockReset(zmq::socket\_t &socketN,zmq::context\_t&contextN){

char\*tmpadresN=new char[TCPE];

size\_t tmpsizeN=TCPE\*sizeof(char);

int fqN;

int pulse;

fqN=zmq\_getsockopt(socketN,ZMQ\_LAST\_ENDPOINT,tmpadresN,&tmpsizeN);

fqN=zmq\_getsockopt(socketN,ZMQ\_RCVTIMEO,&pulse,&tmpsizeN);

socketN.close();

socketN=zmq::socket\_t(contextN,ZMQ\_REQ);

socketN.bind(tmpadresN);

fqN=zmq\_setsockopt(socketN,ZMQ\_RCVTIMEO,&pulse,tmpsizeN);

return true;

}\*/

int newChildNode(std::vector<pid\_t> &pidsN , std::vector<int> &chidsN , std::vector<zmq::socket\_t> &socketsN , zmq::context\_t&contextN , char\*NodeExecN,int chidN, int\*pulse,std::string&str){

int idx;

for(idx=0;idx<socketsN.capacity();idx++){

if((pidsN[idx]==0)&&(chidsN[idx]==0)){

break;

}

}

if(socketsN.capacity()==idx){

socketsN.resize(idx+1);

pidsN.resize(idx+1);

chidsN.resize(idx+1);

pidsN[idx]=0;

chidsN[idx]=0;

}

socketsN[idx]=zmq::socket\_t(contextN,ZMQ\_REQ);

socketsN[idx].bind("tcp://\*:\*");//\*/

char\*tmpadresN=new char[TCPE];

size\_t tmpsizeN=TCPE\*sizeof(char);

int fqN;

fqN=zmq\_getsockopt(socketsN[idx],ZMQ\_LAST\_ENDPOINT,tmpadresN,&tmpsizeN);

if(fqN){

socketsN[idx].close();

delete[] tmpadresN;

printf("socket ID failure %d\n",fqN);

return 0;

}

pid\_t pidN;

pidN=fork();

if(pidN==0){

std::string vtmp=std::to\_string(chidN)+"\0\n";

char\*chidAr=new char[16];

for(int chi=0;vtmp[chi]!='\0';chi++){

chidAr[chi]=vtmp[chi];

chidAr[chi+1]='\0';

}

char \* writable = new char[str.size() + 1];

std::copy(str.begin(), str.end(), writable);

writable[str.size()] = '\0';

char\*args[]={NodeExecN,tmpadresN,chidAr,writable,NULL};

execvp(args[0],args);

\_exit (EXIT\_FAILURE);

}

else if(pidN<0){

printf("something went wrong with fork()\n");

socketsN[idx].close();

delete[] tmpadresN;

return 0;

}

else{

pidsN[idx]=pidN;

chidsN[idx]=chidN;

zmq::message\_t requestN (3);

memcpy (requestN.data (), "R?\0", 3);

socketsN[idx].send (requestN,zmq::send\_flags::none);

zmq::message\_t replyN;

socketsN[idx].recv(replyN,zmq::recv\_flags::none);

return pidN;

}

printf("something failed\n");

return false;

}

int tryToAdd(std::vector<pid\_t> &pids , std::vector<int> &chids , std::vector<zmq::socket\_t> &sockets , zmq::context\_t&context , char\*NodeExec,int chid, int\*pulse,std::string&heartbitListener){

int success=1;

for(int che=0;che<chids.size();che++){

if(chid==chids[che]){

success=0;

}

}

if(success){

success=newChildNode(pids , chids , sockets , context , NodeExec , chid, pulse, heartbitListener);

if(!success){

printf("could not create new node\n");

}

else{

}

}

return success;

}

std::string sendToAll(std::vector<pid\_t> &pids , std::vector<int> &chids , std::vector<zmq::socket\_t> &sockets , int i, int arg1, int arg2, std::string command, int CurId){

bool found=false;

int bdr1,bdr2,idx=0;

if(i==0){

for(idx=0;idx<chids.capacity();idx++){

if((arg2==chids[idx])&&(pids[idx]!=0)){

found=true;

break;

}

}

}

if((i!=0)&&(i!=4)&&(i!=6)){

for(idx=0;idx<chids.capacity();idx++){

if((arg1==chids[idx])&&(pids[idx]!=0)){

found=true;

break;

}

}

}

if(found){

bdr1=idx;

bdr2=idx+1;

}

else{

bdr1=0;

bdr2=sockets.capacity();

}

bool success=false;

bool pidAva=true;

std::string rez="N\0";

for(idx=bdr1;idx<bdr2;idx++){

if((pids[idx]!=0)&&(chids[idx]!=0)){

if(getpgid(pids[idx]) >= 0){

zmq::message\_t request (command.size());

memcpy (request.data (), (command+"\0").c\_str(), command.size()+1);

sockets[idx].send (request,zmq::send\_flags::none);

}

}

}

zmq::message\_t reply;

for(idx=bdr1;idx<bdr2;idx++){

if((pids[idx]!=0)&&(chids[idx]!=0)){

if(getpgid(pids[idx]) >= 0){

sockets[idx].recv(reply,zmq::recv\_flags::none);

if((((char\*)reply.data())[0]=='Y')&&(rez=="N\0")){

rez="Y\0";

success=true;

}

if(((char\*)reply.data())[0]=='K'){

rez[0]='K';

rez+=" ";

for(int ij=2;((char\*)reply.data())[ij]!='\0';ij++){

rez+=((char\*)reply.data())[ij];

}

success=true;

for(int r=2;r<reply.size();r++){

if(getIntM((char\*)reply.data(),r)==chids[idx]){

killChild(pids,chids,sockets,idx);

break;

}

}

}

if(((char\*)reply.data())[0]=='V'){

int val=0;

for(int ij=2;((char\*)reply.data())[ij]!='\0';ij++){

val=val\*10+((char\*)reply.data())[ij]-'0';

}

rez="V"+std::to\_string(val)+"\0";

}

}

else{

printf("Node %d is gone\n",chids[idx]);

if(rez[0]=='N'){

rez[0]='U';

}

}

}

}

return rez;

}

bool addWord(std::map<std::string,int>&vocab,std::string&arg3,int arg2){

bool success;

if(vocab.count(arg3)){

success=false;

}

else{

vocab[arg3]=arg2;

success=true;

}

return success;

}

bool findWord(std::map<std::string,int>&vocab,std::string&arg3,int&result){

bool success;

if(vocab.count(arg3)){

success=true;

result=vocab[arg3];

}

else{

success=false;

result=0;

}

return success;

}

void waitForMessage(zmq::socket\_t &socket, zmq::message\_t &reply, int &flag){

flag=1;

socket.recv(reply,zmq::recv\_flags::none);

flag=2;

}

void f\_ill(int i){

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

printf(" ");

}

}

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

zmq::context\_t context (1);

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

psocket.connect (argv[1]);

zmq::message\_t reply;

psocket.recv(reply,zmq::recv\_flags::none);

int CurId=0;

while(argv[0][CurId]!='\0'){CurId++;}

char\*NodeExec=new char(CurId+1);

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

NodeExec[i]=argv[0][i];

NodeExec[i+1]='\0';

}

CurId=0;

if(argv[2][0]=='M'){CurId=-1;}

else{

for(int i=(argv[2][0]=='-');argv[2][i]!='\0';i++){

CurId=CurId\*10+argv[2][i]-'0';

}

if(argv[2][0]=='-'){

CurId\*=-1;

}

}

zmq::socket\_t socketT (context, ZMQ\_PUSH);

socketT.connect (argv[3]);

std::string heartbitListener=argv[3];

zmq::message\_t request (3);

memcpy (request.data (), "R!\0", 3);

psocket.send(request,zmq::send\_flags::none);

std::vector<pid\_t> pids={0};

std::vector<int> chids={0};

int num=255,i=-1,j,ln,tln;

std::map<std::string,int>vocab{};

int type,arg1=-8,arg2=-8,idx;

std::string command,answer,arg3="\0dummy\0";

bool found,success;

int bdr1,bdr2;

std::vector<zmq::socket\_t> sockets={};

int heartbit=0;

int haveMessage=0;

std::thread waitMes;

std::string heartbet;

double heartBitT=0;

while(1){

if(heartbit){

if(((double)clock()-heartBitT) > (double)(heartbit/1000 \* CLOCKS\_PER\_SEC)){

heartBitT+=(double)(heartbit/1000 \* CLOCKS\_PER\_SEC);

heartbet="HBT "+std::to\_string(CurId)+"\0\0";

zmq::message\_t request (heartbet.length()+1);

memcpy (request.data (), heartbet.c\_str(), heartbet.length()+1);

socketT.send(request,zmq::send\_flags::dontwait);

}

}

if(haveMessage==0){

haveMessage=1;

waitMes=std::thread(waitForMessage,std::ref(psocket),std::ref(reply),std::ref(haveMessage));

}

if(haveMessage==2){

haveMessage=0;

waitMes.join();

((char\*)reply.data())[reply.size()]='\0';

command=(char\*)reply.data();

i=1;

arg1=0;

arg2=0;

arg3="+";

while(command[i]==' '){i++;}

if(command[i]=='-'){arg3="-";i++;}

while(command[i]!=' '){

arg1=arg1\*10+((char)command[i])-'0';

i++;

}

if(arg3=="-"){arg1\*=-1;arg3="+";}

while(command[i]==' '){i++;}

if(command[i]=='-'){arg3="-";i++;}

while(command[i]!=' '){

arg2=arg2\*10+((char)command[i])-'0';

i++;

}

if(arg3=="-"){arg2\*=-1;}

arg3="";

while(command[i]==' '){i++;}

while(command[i]!='\0'){

arg3=arg3+((char)command[i]);

i++;

}

i=((char)command[0])-'0';

success=true;

if(i==0){

if(arg2==CurId){

int tmpa=tryToAdd(pids ,chids ,sockets , context ,NodeExec,arg1, &heartbit,heartbitListener);

answer="V"+std::to\_string(tmpa);

}

else{

answer=sendToAll(pids ,chids ,sockets,i,arg1,arg2,command,CurId);

}

}

if((i>0)&&(i<4)){

if(arg1==CurId){

if(i==1){

break;//exit while loop. do all finalization there

}

if(i==2){

success=addWord(vocab,arg3,arg2);

answer="Y\n";

}

if(i==3){

int rezu=0;

success=findWord(vocab,arg3,rezu);

if(success){

answer="V "+std::to\_string(rezu)+"\0";

}

else{

answer="N\n";

}

}

}

else{

answer=sendToAll(pids ,chids ,sockets,i,arg1,arg2,command,CurId);

}

}

if(i==4){

heartbit=arg1;

answer=sendToAll(pids ,chids ,sockets,i,arg1,arg2,command,CurId);

}

if(i==5){

printf("NODE WAS ASKED FOR PING. USE THE CHEATY HEARTBEAT COLLECTOR\n");

}

if(i==6){

break;

}

if(i==7){

f\_ill(arg1);printf("┌────────────────────────\n");

f\_ill(arg1);printf("| ");printf("%d: heartbit = %d\n",CurId,heartbit);

f\_ill(arg1);printf("| ");printf("pids= [");for(int r=0;r<sockets.capacity();r++){printf(" %d",pids[r]);}printf(" ]\n");

f\_ill(arg1);printf("| ");printf("chids= [");for(int r=0;r<sockets.capacity();r++){printf(" %d",chids[r]);}printf(" ]\n");

f\_ill(arg1);printf("| ");for (const auto& p : vocab ) {std::cout <<"["<<p.first<<"]"<<p.second<<"\t";}printf("\n");

f\_ill(arg1);printf("└────────────────────────\n");

arg1+=2;

success=true;

answer=sendToAll(pids ,chids ,sockets,i,arg1,arg2,command,CurId);

}

if((answer[0]!='V')&&(answer[0]!='K')){

if(success){

answer="Y\0";

}

else{

answer="N\0";

}

}

zmq::message\_t request (answer.length()+1);

memcpy (request.data (), answer.c\_str(), answer.length()+1);

psocket.send(request,zmq::send\_flags::none);

}

}

answer=sendToAll(pids ,chids ,sockets,6,0,0,"6 0 0 0",CurId);

if(answer[0]=='K'){

answer=answer+" "+std::to\_string(CurId);

}

else{

answer="K "+std::to\_string(CurId);

}

answer+="\0";

request=zmq::message\_t(answer.size()+1);

memcpy (request.data (), answer.c\_str(), answer.size()+1);

psocket.send(request,zmq::send\_flags::none);

psocket.close();

socketT.close();

context.close();

return 0;

}

**interface.cpp:**

#include <iostream>

#include <chrono>

#include <thread>

#include <string>

#include <map>

#include <vector>

#include <cstring>

#ifndef \_WIN32

#include <unistd.h>

#else

#include <windows.h>

#define sleep(n) Sleep(n)

#endif

#include <zmq.hpp>

int getIntM(char\*str,int&offs){

int rez=0,sig=1;

if(str[offs]=='-'){

sig=-1;

offs++;

}

for(;((str[offs]>='0')&&(str[offs]<='9'));offs++){

rez=rez\*10+str[offs]-'0';

}

rez\*=sig;

return rez;

}

std::string getStrM(char\*str,int&offs){

std::string rez="";

for(;((str[offs]!='\0')&&(str[offs]!=' ')&&(str[offs]!='\n'));offs++){

rez=rez+str[offs];

}

return rez;

}

double getDubM(char\*str,int&offs){

double rez=0;

int sig=1;

if(str[offs]=='-'){

sig=-1;

offs++;

}

for(;((str[offs]>='0')&&(str[offs]<='9'));offs++){

rez=rez\*10+str[offs]-'0';

}

if(str[offs]!='.'){

rez\*=sig;

return rez;

}

int tenpow=0;

offs++;

for(;((str[offs]>='0')&&(str[offs]<='9'));offs++){

rez=rez\*10+str[offs]-'0';

tenpow++;

}

while(tenpow){

rez\*=0.1;

tenpow--;

}

rez\*=sig;

return rez;

}

const int CMNM=7;

const size\_t TCPE=32;

void updateTimers(std::map<int,double>&timers,zmq::socket\_t &socket,bool &running){

zmq::message\_t reply (10);

int chid,i;

int pulse=0;

while(running){

std::this\_thread::sleep\_for(std::chrono::milliseconds(500));

socket.recv(reply,zmq::recv\_flags::none);

i=4;

chid=getIntM(((char\*)reply.data()),i);

timers[chid]=(double)clock();

}

}

const bool TCP4HRT=true;

int main(){

std::map<int,double>timers;

std::vector<int>chids={-1};

char\*NodeExec=new char(10);

for(int i=0;i<10;i++){

NodeExec[i]="./nodo.out\0"[i];

}

int num=255,i,j,ln,tln;

int type,arg1=-8,arg2=-8,idx;

std::string command,arg3="\0dummy\0";

bool found,success;

int bdr1,bdr2;

const char\*cmds[CMNM]={"create ","remove ","exec ","heartbit ","ping ","exit","admin info"};

char\*inpt=(char\*)malloc(sizeof(char)\*num);

zmq::context\_t context (1);

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

socket.bind ("tcp://\*:\*");//\*/

char\*tmpadresN=new char[TCPE];

size\_t tmpsizeN=TCPE\*sizeof(char);

char\*heartbitListener=new char[TCPE];

for(i=0;i<TCPE-1;i++){

heartbitListener[i]=i%26+'A';

heartbitListener[i+1]='\0';

}

zmq::socket\_t sucketT (context,ZMQ\_PULL);

sucketT.bind ("tcp://\*:\*");//\*/

if(zmq\_getsockopt(sucketT,ZMQ\_LAST\_ENDPOINT,heartbitListener,&tmpsizeN)){

sucketT.close();

delete[] tmpadresN;

printf("socket T ID failure1\n");

return -1;

}

int deftime=1000;

if(zmq\_setsockopt(sucketT,ZMQ\_RCVTIMEO,&deftime,sizeof(int))){

sucketT.close();

delete[] tmpadresN;

printf("socket T ID failure2\n");

return -1;

}

tmpsizeN=TCPE\*sizeof(char);

int fqN;

fqN=zmq\_getsockopt(socket,ZMQ\_LAST\_ENDPOINT,tmpadresN,&tmpsizeN);

if(fqN){

socket.close();

delete[] tmpadresN;

printf("socket C ID failure %d\n",fqN);

return -1;

}

pid\_t pidN;

pidN=fork();

if(pidN==0){

char\*chidAr=new char[2];

chidAr[0]='M';

chidAr[1]='\0';

char\*args[]={NodeExec,tmpadresN,chidAr,heartbitListener,NULL};

execvp(args[0],args);

\_exit (EXIT\_FAILURE);

}

else if(pidN<0){

printf("something went wrong with fork()\n");

socket.close();

delete[] tmpadresN;

return false;

}

else{

}

zmq::message\_t requestN (5);

memcpy (requestN.data (), "Ready", 5);

socket.send (requestN,zmq::send\_flags::none);

zmq::message\_t replyN;

socket.recv(replyN,zmq::recv\_flags::none);

bool snd,needEnd=true;

std::thread lifeChecker;

int heartbit=0;

bool running=true;

if(TCP4HRT){

lifeChecker=std::thread(updateTimers,std::ref(timers),std::ref(sucketT),std::ref(running));

}

while(NULL!=fgets (inpt, num, stdin)){

needEnd=false;

ln=(int)strlen(inpt);

if(inpt[0]=='\n'){

break;

}

for(i=0;i<CMNM;i++){

found=true;

tln=(int)strlen(cmds[i]);

if(ln<tln){continue;}

for(j=0;j<tln;j++){

if(cmds[i][j]!=inpt[j]){

found=false;

break;

}

}

if(found){

break;

}

found=false;

}

if(!found){

printf("Error: command not found\n");

continue;

}

if(i<5){

arg1=getIntM(inpt,j);

if(i>2){i++;}

if(i==0){

j++;

arg2=getIntM(inpt,j);

}

if(i==2){

j++;

arg3=getStrM(inpt,j);

if(inpt[j]==' '){

j++;

arg2=getIntM(inpt,j);

}

else{i++;}

}

}else{i++;}

snd=true;

if(i==0){

for(tln=0;tln<chids.size();tln++){

if(chids[tln]==arg1){

printf("Error: Already exists\n");

snd=false;

break;

}

if(chids[tln]==arg2){

snd=true;

break;

}

}

if(tln==chids.size()){

printf("Error: Parent not found\n");

snd=false;

}

}

if(((i>0)&&(i<4))||(i==5)){

for(tln=0;tln<chids.size();tln++){

if((chids[tln]==arg1)){

snd=true;

break;

}

}

if(tln==chids.size()){

printf("Error: Not found\n");

snd=false;

}

else if(i==5){

snd=false;

if(heartbit>0){

if(timers.count(arg1)){

if(((double)clock()-timers[arg1]) > 4\*(double)(heartbit/1000 \* CLOCKS\_PER\_SEC)){

printf("Ok: 0\n");

}

else{

printf("Ok: 1\n");

}

}

else{

printf("Error: Node Heartbit has not been set\n");

}

}

else{

printf("Error: Heartbit is set to 0\n");

}

}

}

if(i==7){

for (const auto& p : timers ) {

std::cout <<"["<<p.first<<"]"<<(p.second-(double)clock())<<std::endl;

}printf("\n");

snd=true;

arg1=1;

arg2=0;

arg3="beans\0";

}

if(snd){

command=std::to\_string(i)+" "+std::to\_string(arg1)+" "+std::to\_string(arg2)+" "+arg3+"\0";

zmq::message\_t request (command.size());

memcpy (request.data (), command.c\_str(), command.size());

socket.send(request,zmq::send\_flags::none);

zmq::message\_t reply;

socket.recv(reply,zmq::recv\_flags::none);

((char\*)reply.data())[reply.size()]='\0';

if( ((char\*)reply.data())[0]=='U' ){

printf("Error: Node not avaliable\n");

}

if(i==0){

if( ((char\*)reply.data())[0]=='V' ){

chids.push\_back(arg1);

int ofs=1;

int ji=getIntM((char\*)reply.data(),ofs);

printf("Ok: %d\n",ji);

}

else{

printf("Error: could not create new node\n");

}

}

else if((i==1)&&(((char\*)reply.data())[0]=='Y')){

int ii;

for(ii=0;chids[ii]!=arg1;ii++);

for(ii;ii<chids.size()-1;ii++){

chids[ii]=chids[ii+1];

}

chids.resize(chids.size()-1);

printf("Ok\n");

}

else if((i==4)&&(((char\*)reply.data())[0]=='Y')){

heartbit = arg1;

}

else if(((char\*)reply.data())[0]=='K'){

int ii=1,jj=1,ji=1;

char tpm;

while(jj<reply.size()){

jj++;

tpm=((char\*)reply.data())[jj];

if((tpm>'9')||(tpm<'0')){break;}

ji=getIntM((char\*)reply.data(),jj);

for(ii=0;chids[ii]!=ji;ii++);

for(ii;ii<chids.size()-1;ii++){

chids[ii]=chids[ii+1];

}

chids.resize(chids.size()-1);

}

}

else{

if(((char\*)reply.data())[0]=='Y'){

printf("Ok\n");

}

if(((char\*)reply.data())[0]=='V'){

int ofs=1+(((char\*)reply.data())[1]==' ');

int ji=getIntM((char\*)reply.data(),ofs);

printf("Ok: %d\n",ji);

}

if(((char\*)reply.data())[0]=='N'){

printf("Not Ok\n");

}

}

}

if(i==6){

break;

}

if((i==1)&&(arg1==-1)){

break;

}

needEnd=true;

}

running=false;

if(TCP4HRT){lifeChecker.join();}

sucketT.close();

if(needEnd){

command="6 0 0 0\0\0";

zmq::message\_t request (command.size());

memcpy (request.data (), command.c\_str(), command.size());

socket.send(request,zmq::send\_flags::none);

}

socket.close();

context.close();

return 0;

}

//0 "create childID parentID"

//1 "remove childID"

//2 "exec childID name value"

//3 "exec childID name"

//4 "heartbit TIME"

//5 "ping childID"

//6 "exit"

//7 "admin info"

**Пример работы**

create 12 -1

Ok: 206303

create 13 -1

Ok: 206308

create 16 12

Ok: 6313

create 2 3

Error: Parent not found

create 1992 12

Ok: 6319

admin info

[0]-93

┌────────────────────────

| -1: heartbit = 0

| pids= [ 206303 206308 ]

| chids= [ 12 13 ]

|

└────────────────────────

┌────────────────────────

| 12: heartbit = 0

| pids= [ 206313 206319 ]

| chids= [ 16 1992 ]

|

└────────────────────────

┌────────────────────────

| 16: heartbit = 0

| pids= [ ]

| chids= [ ]

|

└────────────────────────

┌────────────────────────

| 13: heartbit = 0

| pids= [ ]

| chids= [ ]

|

└────────────────────────

┌────────────────────────

| 1992: heartbit = 0

| pids= [ ]

| chids= [ ]

|

└────────────────────────

Ok

create -5 12

Ok: 6333

exec 13 bruh 228

Ok

exec 13 bruh

Ok: 228

exec 16 bruh

Ok

admin info

[0]-70

┌────────────────────────

| -1: heartbit = 0

| pids= [ 206303 206308 ]

| chids= [ 12 13 ]

|

└────────────────────────

┌────────────────────────

| 13: heartbit = 0

| pids= [ ]

| chids= [ ]

| [bruh]228

└────────────────────────

┌────────────────────────

| 12: heartbit = 0

| pids= [ 206313 206319 206333 0 ]

| chids= [ 16 1992 -5 0 ]

|

└────────────────────────

┌────────────────────────

| 1992: heartbit = 0

| pids= [ ]

| chids= [ ]

|

└────────────────────────

┌────────────────────────

| -5: heartbit = 0

| pids= [ ]

| chids= [ ]

|

└────────────────────────

┌────────────────────────

| 16: heartbit = 0

| pids= [ ]

| chids= [ ]

|

└────────────────────────

Ok

remove 12

admin info

[0]-70

┌────────────────────────

| -1: heartbit = 0

| pids= [ 0 206308 ]

| chids= [ 0 13 ]

|

└────────────────────────

┌────────────────────────

| 13: heartbit = 0

| pids= [ ]

| chids= [ ]

| [bruh]228

└────────────────────────

Ok

heartbit 1000

ping 13

Error: Node Heartbit has not been set

admin info

[-1]-32

[0]-1845

┌────────────────────────

| -1: heartbit = 1000

| pids= [ 0 206308 ]

| chids= [ 0 13 ]

|

└────────────────────────

┌────────────────────────

| 13: heartbit = 1000

| pids= [ ]

| chids= [ ]

| [bruh]228

└────────────────────────

Ok

ping 13

Error: Node Heartbit has not been set

ping -1

Ok: 1

ping 13

Ok: 1

exit

aaaaa@TRASHBox:~/Desktop/OC/laba6$

Программа говорит, что у узла 13 не установлен heartbit, поскольку на момент запроса узел 13 ещё не успел отправить ни одного сигнала (система, на которой проводилась работа, очень медленная)

**Вывод**

Грандиозная работа. В процессе работы над ней было рассмотрено большое количество вариантов взаимодействия процессов. Например, пока система PUSH-PULL не работала (из-за незаметной синтаксической ошибки), таймеры heartbit планировалось хранить в файле отображённом в память (с реаллокацией).