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: М80-206Б-22

Студент: Ларченко А.О.

Преподаватель: Миронов Е.С.

Оценка: ____________________

Дата: 08.01.24

Москва, 2024

***Цель работы:***

Целью является приобретение практических навыков в:

Управлении серверами сообщений (№5)

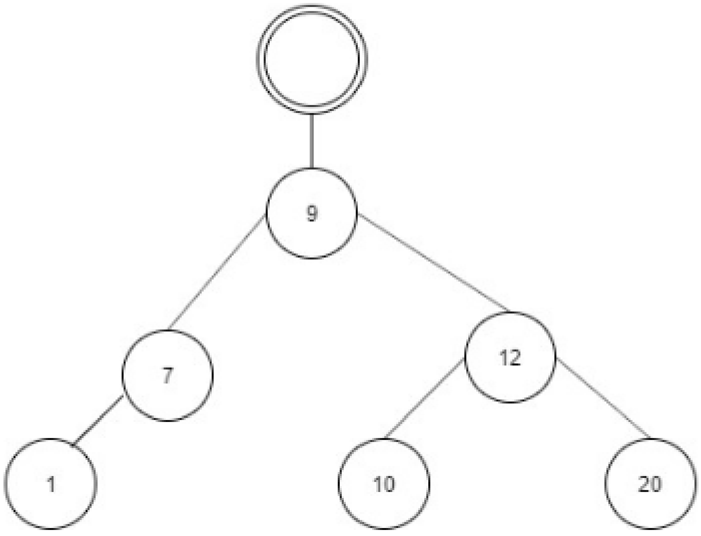
Применение отложенных вычислений (№6)

Интеграция программных систем друг с другом (№7)

***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.



**Вариант 10.**

*Топология 4.* Все вычислительные узлы хранятся в идеально сбалансированном бинарном дереве. Каждый следующий узел должен добавляться в самое наименьшее поддерево.

*Тип команд 3.* (локальный таймер) Формат команды сохранения значения: exec id subcommand

subcommand – одна из трех команд: start, stop, time.

start – запустить таймер

stop – остановить таймер

time – показать время локального таймера в миллисекундах

*Тип проверки доступности узлов 3.* Формат команды: heartbit time

Каждый узел начинает сообщать раз в time миллисекунд о том, что он работоспособен. Если от узла нет сигнала в течении 4*time миллисекунд, то должна выводится пользователю строка: «Heartbit: node id is unavailable now», где id – идентификатор недоступного вычислительного узла.

## Общий метод и алгоритм решения

Для реализации связи между управляющим узлом и исполняющими я буду использовать очередь сообщений из библиотеки *ZeroMQ*.

Очередь сообщений предоставляет гарантии, что сообщение будет доставлено независимо от того, что происходит. Очередь сообщений позволяет асинхронно взаимодействовать между слабо связанными компонентами, а также обеспечивает строгую последовательность очереди.

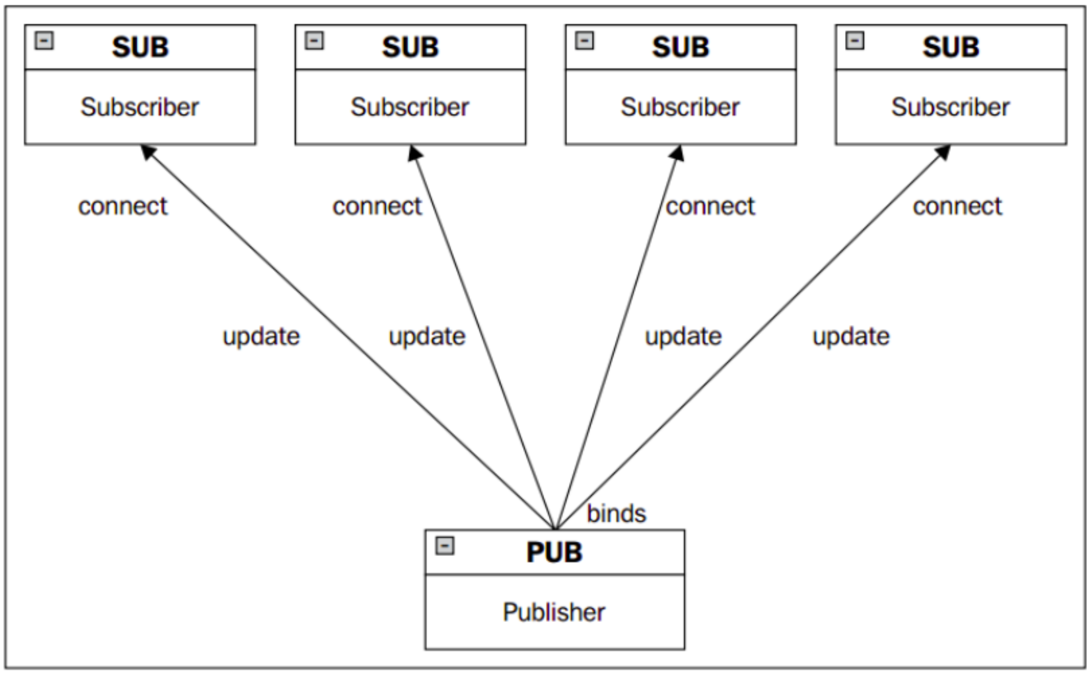
*ZeroMQ* - это не системой очередей сообщений типа WebSphereMQ, или RabbitMQ, это библиотека, которая дает нам инструменты для создания собственной системы очередей сообщений. Её еще называют сокетами на стероидах.

Для своей программы я выбрал паттерн PUB-SUB, но не обычный, управляющий узел у меня является публикатором для управляющих узлов, но при этом также является подписчиком на вычислительные узлы, чтобы получать от них обратную связь. А вычислительные узлы также являются и подписчиками и публикаторами.

PUB-сокет для управляющего узла привязан к порту, который является SUB- сокетом у вычислительных узлов.

А SUB-сокет управляющего узла привязан к другому порту, который является PUB-сокетом у вычислительных узлов.

К слову, сокет — это виртуальная конструкция из IP-адреса и номера порта, предназначенная для связи приложений или компьютеров между собой. Еще это именованный канал - FIFO, именованный pipe.



The publish-subscribe pattern

Использованные системные вызовы из библиотеки *ZeroMQ:*

socket.bind("tcp://*IP*:*port*") - установление связи socket по IP адресу и указанному порту port

- socket.set(zmq::sockopt::, "filter prefix") - настройка подписки, принимает только те сообщения, которые начинаются с префикса: "filter prefix"
- socket.connect("tcp://*IP*:*port*") - соединение с указанным портом
- socket.disconnect("tcp://*IP*:*port*") - отсоединение от порта
- socket.recv(reply, zmq::recv_flags) - принятие сообщения
- socket.send(request1, zmq::send_flags) - отправка сообщения

Использование адреса 127.0.0.1 позволяет устанавливать соединение и передавать информацию для программ-серверов, работающих на том же компьютере, что и программа-клиент, независимо от конфигурации аппаратных сетевых средств компьютера

## Код программы

***msg.h:***

```
#pragma once
#include <iostream>
#include <string>
#include <vector>

using namespace std;

enum Subcommands{
    timer,
    start,
    stop
};

enum Message_type{
    create,
    create_asw,
    exec,
    exec_asw,
    heartbit,
    heartbit_answ,
    error,
    die
};

struct Message{
    Message_type type;
    vector<int> data;
};
```

***manage_node.h:***

```
#pragma once
#include <iostream>
#include <unistd.h>
#include <signal.h>
#include <string>
#include <sys/wait.h>
#include <zmq.hpp>
#include "msg.h"
#include "awlTree.h"

using namespace std;


class Manage_node{
    public:
        zmq::context_t context;
        zmq::socket_t publisher;
        zmq::socket_t sub;

        bool send_msg(Message msg);
        void receive_msg(Message_type msg_type, AWL_tree &tree);


        Manage_node();
        ~Manage_node();

};
```

***manage_node.cpp:***

```
#include "manage_node.h"


Manage_node::Manage_node(): publisher(context, zmq::socket_type::pub), sub(context,
zmq::socket_type::sub){
    publisher.bind("tcp://127.0.0.1:5555");
    sub.bind("tcp://127.0.0.1:5556");
    //фильтр для сообщений, подписываемся на все сообщения
    sub.set(zmq::sockopt::subscribe, "");
}

Manage_node::~Manage_node(){
    sub.disconnect("tcp://127.0.0.1:5556");
    publisher.disconnect("tcp://127.0.0.1:5555");
}
```

```
void Manage_node::receive_msg(Message_type msg_type, AWL_tree &tree){
    cout<<"Receiving...\n";
    switch(msg_type){
        case Message_type::heartbit_answ :{
            zmq::message_t reply;
            zmq::recv_result_t res = sub.recv(reply, zmq::recv_flags::none);
            string id_str=reply.to_string();
            int id=stoi(id_str);
            tree.change_availability(id, true);
            break;

        }
    }
}

bool Manage_node::send_msg(Message msg){
    cout<<"Sending...\n";
    string type_str=to_string(msg.type);
    switch(msg.type){
        case Message_type::create :{

            string parent_id_str=to_string(msg.data[0]);
            string new_id_str=to_string(msg.data[1]);

            zmq::message_t request1(parent_id_str);
            publisher.send(request1, zmq::send_flags::sndmore);
            zmq::message_t request2(type_str);
            publisher.send(request2, zmq::send_flags::sndmore);
            zmq::message_t request3(new_id_str);
            publisher.send(request3, zmq::send_flags::none);

            return true;
        }
        case Message_type::exec :{
            string id_str=to_string(msg.data[0]);
            string subcmd_str=to_string(msg.data[1]);

            zmq::message_t request1(id_str);
            publisher.send(request1, zmq::send_flags::sndmore);
            zmq::message_t request2(type_str);
            publisher.send(request2, zmq::send_flags::sndmore);
            zmq::message_t request3(subcmd_str);
            publisher.send(request3, zmq::send_flags::none);

            return true;

        }
        case Message_type::heartbit :{
            string id_str=to_string(msg.data[0]);
            string period_str=to_string(msg.data[1]);

            zmq::message_t request1(id_str);
            publisher.send(request1, zmq::send_flags::sndmore);
            zmq::message_t request2(type_str);
            publisher.send(request2, zmq::send_flags::sndmore);
            zmq::message_t request3(period_str);
            publisher.send(request3, zmq::send_flags::none);

            return true;



        }

        default:
            return false;
    }
    return false;

}
```

***manage_main.cpp:***

```
#include <iostream>
#include <string>
#include <zmq.hpp>
#include <unistd.h>
#include <vector>

// #include "awlTree.h"
#include "manage_node.h"
#include "timer.h"

using namespace std;

int process_creation(){
    int pid =fork();
    if(pid==-1){
        perror("Call fork was ended with erorr: ");
        exit(-1);
    }
    return pid;
}

int what_subcmd(string cmd){
    if (cmd=="time"){
        return Subcommands::timer;
    } else if(cmd=="start"){
        return Subcommands::start;
    } else if(cmd=="stop"){
        return Subcommands::stop;
    }
    return -1;
}



int main(){
    AWL_tree tree;
    Manage_node node;

    cout<<"Welcome in our programm! This is what command i can do:\n";
    cout<<" - create 'id' \n"<<" - exec 'id' 'command(start/stop/time)' \n";
    cout<<" - heartbit 'time' (in ms) \n"<<" - draw\n";
    cout<<"Or enter q or ^D to exit\n"<<"Enter you command: \n";

    while (true){
        string cmd;
        cout<<" ->";
        cin>>cmd;
        if(cmd=="create"){
            int id;
            cin>>id;
            if(id<0){
                cout<<"Error: You can only use positive number\n";
            } else if(tree.is_in_tree(id)){
                cout<<"Error: Alredy exists\n";
            } else{
                tree.insert(id);
                // tree.change_availability(id, true);
                int parent_id=tree.parent_id(id);
                if(parent_id==-1){
                    int pid=process_creation();
                    if(pid==0){
                        string id_str=to_string(id);
                        execl("./node", "./node", id_str.c_str(), NULL);
                    } else{
                        cout<<"Ok: "<<pid<<"\n";
                    }
                } else{
                    Message msg;
                    msg.type=Message_type::create;
                    msg.data.push_back(parent_id);
                    msg.data.push_back(id);
                    if(!node.send_msg(msg)){
                        cout<<"Error...\n";
                    }
                    sleep(1);
                }
            }
        } else if(cmd=="exec"){
            int id;
            string subcmd;
            cin>>id;
            cin>>subcmd;
            if(!tree.is_in_tree(id)){
                cout<<"Error: uncorrect id\n";
            } else{
                int subcmd_int=what_subcmd(subcmd);
                if(subcmd_int!=-1){
                    Message msg;
                    msg.type=Message_type::exec;
                    msg.data.push_back(id);
                    msg.data.push_back(subcmd_int);
                    if(!node.send_msg(msg)){
                        cout<<"Error...\n";
                    }
                    sleep(1);
                } else{
                    cout<<"Error: uncorrect subcommands\n";
                }
            }


        } else if(cmd=="heartbit"){
            int period;
            cin>>period;
            if(period<=0){
                cout<<"Error: uncorrect input\n";
            } else{
                Message msg;
                msg.type=Message_type::heartbit;
                msg.data.push_back(-100); // for all users
                msg.data.push_back(period);
                if(!node.send_msg(msg)){
                        cout<<"Error...\n";
                }
                Timer tm;
                tm.start();
                sleep(0.1);
                tree.bypass_reset(tree.get_root());
                while(true){
                    if(tm.times()>=4*period-150){
                        vector<int> unavailable;
                        tree.bypass(tree.get_root(), unavailable);
                        for(int i=0; i< unavailable.size(); ++i){
                            cout << "\033[1;31m";
```

```
                            cout<<"Heartbit: node "<<unavailable[i]<<" is unavailable 
now"<<"\033[0m\n";
                        }
                        if (unavailable.size()==0){
                            cout << "\033[1;32m";
                            cout<<"Heartbit: all nodes are available"<<"\033[0m\n";
                        }
                        // tm.start();
                        tm.stop();
                        break;
                    }
                    node.receive_msg(Message_type::heartbit_answ, tree);

                }


            }
        } else if(cmd=="draw"){
            tree.draw_tree();
            cout<<'\n';
        } else if(cmd=="q" or cin.eof()){
            cout<<"Break;\n";
            break;
        } else{
            cout<<"Uncorrect input. Try again:\n";
        }
    }
}
```

***__node.cpp:__***

```
#include <iostream>
#include <string>
#include <zmq.hpp>
#include <unistd.h>
#include <signal.h>
#include <sys/prctl.h>

#include "msg.h"
#include "timer.h"

using namespace std;



int main(int argc, char *argv[]){
    string my_id_str=argv[1];

    zmq::context_t context;
    zmq::socket_t publisher(context, zmq::socket_type::pub);
    publisher.connect("tcp://127.0.0.1:5556");
```

```
zmq::socket_t sub(context, zmq::socket_type::sub);
sub.connect("tcp://127.0.0.1:5555");
sub.set(zmq::sockopt::subscribe, my_id_str);
sub.set(zmq::sockopt::subscribe, "-100");

sleep(1);
int exit_status=prctl(PR_SET_PDEATHSIG, SIGKILL);

//Timer settings
Timer tm;

while(true){
    // cout<<"Child: Receiving...\n";
    zmq::message_t reply;
    zmq::recv_result_t res = sub.recv(reply, zmq::recv_flags::none);
    // if(res==-1){
    //     cout<<"FAIL\n";
    // } else{
        string req_id=reply.to_string();

    zmq::message_t cmd;
    res = sub.recv(cmd, zmq::recv_flags::none);
    string cmd_str=cmd.to_string();
    if(cmd_str==to_string(Message_type::create)){
        zmq::message_t new_id;
        res = sub.recv(new_id, zmq::recv_flags::none);
        string new_id_str=new_id.to_string();
        int pid=fork();
        if(pid==0){
            execl("./node", "./node", new_id_str.c_str(), NULL);
        } else{
            // cout<<"Child: Sending...\n";

            string pid_str=to_string(pid);
            // zmq::message_t request(pid_str);
            // publisher.send(request, zmq::send_flags::none);
            cout<<"Ok:"<<my_id_str<<": "<<pid_str<<'\n';

        }
    } else if(cmd_str==to_string(Message_type::exec)){
        zmq::message_t subcmd;
        res = sub.recv(subcmd, zmq::recv_flags::none);
        string subcmd_str=subcmd.to_string();
        cout<<"Ok:"<<my_id_str;
        if(subcmd_str==to_string(Subcommands::timer)){
            cout<<": ";
            cout<<tm.times();
            } else if(subcmd_str==to_string(Subcommands::start)){
                tm.start();

            } else{
                tm.stop();

            }
            cout<<'\n';
        } else if(cmd_str==to_string(Message_type::heartbit)){
            zmq::message_t period_m;
            res = sub.recv(period_m, zmq::recv_flags::none);
            string period_str=period_m.to_string();
            // cout<<"Ok:"<<my_id_str<<"Period: "<<period_str<<'\n';
            int period=stoi(period_str);

            Timer hb_timer;
            hb_timer.start();
            Timer delay;
            delay.start();
            while(true){
                if(delay.times()>=period*4+200){
                    hb_timer.stop();
                    delay.stop();
                    break;
                }
                sleep(0.1);
                if(hb_timer.times()>=period){
                    zmq::message_t request1(my_id_str);
                    publisher.send(request1, zmq::send_flags::none);
                    hb_timer.start();
                }
            }
        }

    }
}
```

***awlTree.h:***

```
#pragma once
#include <iostream>
#include <vector>
using namespace std;

struct node{
    int ID;
    node* left_son;
    node* right_son;
    int hight;
    bool available;
};

class AWL_tree{
    public:
        AWL_tree();

        bool is_in_tree(int ID);
        bool is_available(int ID);

        void change_availability(int ID, bool status);
        void bypass(node* cur_node, vector<int> &unavailable); //Bypassing tree and add
to vector unavailable nodes
        void bypass_reset(node* cur_node);//make all nodes unavailable

        node* get(int ID);
        node* get_root();
        int parent_id(int child_id);
        int cnt();
        void draw_tree();
        node* balancing(node* cur_node);
        bool remove(int ID);
        bool insert(int ID);
        int check_depth();



        ~AWL_tree();
    private:
        node* add(node* cur_node, node* new_node, int step);
        void draw_node(node* cur_node, int level);
        node* ll_rot(node* cur_node);
        node* lr_rot(node* cur_node);
        node* rr_rot(node* cur_node);
        node* rl_rot(node* cur_node);
        int bf(node* cur_node);
        int calHight(node* cur_node);
        int find_parent(node* cur_node, int ID);
        node* find(node* current_node ,int ID);
        int depth;
        node* root;
        int node_cnt;
};
```

***awlTree.cpp:***

```
#include "awlTree.h"



AWL_tree::AWL_tree(){
    // node* new_node= new node;
    // new_node->ID=0;
    node_cnt=0;
    depth=0;
    root=NULL;
}

int AWL_tree::check_depth(){
    return depth;
}

int AWL_tree::cnt(){
    return node_cnt;
}

int AWL_tree::calHight(node* cur_node){
    if(cur_node->left_son && cur_node->right_son){
        if(cur_node->left_son->hight<cur_node->right_son->hight){
            return cur_node->right_son->hight +1;
        } else{
            return cur_node->left_son->hight+1;
        }
    } else if(cur_node->left_son && cur_node->right_son==NULL){
        return cur_node->left_son->hight +1;
    } else if(cur_node->right_son && cur_node->left_son==NULL){
        return cur_node->right_son->hight+1;
    } else{
        return 1;
    }
}

int AWL_tree::bf(node* cur_node){
    if(cur_node->left_son && cur_node->right_son){
        return cur_node->left_son->hight-cur_node->right_son->hight;

    } else if(cur_node->left_son && cur_node->right_son==NULL){
        return cur_node->left_son->hight;
    } else if(cur_node->right_son && cur_node->left_son==NULL){
        return -cur_node->right_son->hight;
    }
    return 0;
}

node* AWL_tree::ll_rot(node* cur_node){
    node* tmp;
    node* answ;
    tmp=cur_node->left_son->right_son;
    answ=cur_node->left_son;
    cur_node->left_son=tmp;
    answ->right_son=cur_node;
    // cur_node->left_son=tmp;

    return answ;
}

node* AWL_tree::rr_rot(node* cur_node){
    node* tmp;
    node* answ;
    tmp=cur_node->right_son->left_son;
    answ=cur_node->right_son;
    cur_node->right_son=tmp;
    answ->left_son=cur_node;
    return answ;
}

node* AWL_tree::rl_rot(node* cur_node){
    cur_node->right_son=ll_rot(cur_node->right_son);
    return rr_rot(cur_node);
}

node* AWL_tree::lr_rot(node* cur_node){
    cur_node->left_son=rr_rot(cur_node->left_son);
    return ll_rot(cur_node);
}

node* AWL_tree::balancing(node* cur_node){
    if(bf(cur_node)==2 && bf(cur_node->left_son)==1){
        return ll_rot(cur_node);
    } else if(bf(cur_node)==-2 && bf(cur_node->right_son)==-1){
        return rr_rot(cur_node);
    } else if(bf(cur_node)==-2 && bf(cur_node->right_son)==1){
        return rl_rot(cur_node);
    } else if(bf(cur_node)==2 && bf(cur_node->left_son)==-1){
        return lr_rot(cur_node);
    }
    return cur_node;
}

node* AWL_tree::find(node* current_node,int ID){
    if(current_node==NULL){
        return NULL;
    }
    if(current_node->ID==ID){
        return current_node;
    } else if(ID>current_node->ID){
        return find(current_node->right_son, ID);
    }
    return find(current_node->left_son, ID);
```

```
}

bool AWL_tree::is_in_tree(int ID){
    node* cur_node=find(root, ID);
    if(cur_node==NULL){
        return false;
    }
    return true;
}

node* AWL_tree::add(node* cur_node, node* new_node, int step){
    node* answ;
    if(new_node->ID>cur_node->ID){
        if(cur_node->right_son==NULL){
            cur_node->right_son=new_node;
            // new_node->level=step+1;
            // return;
        } else{
            add(cur_node->right_son, new_node, step+1);
        }
    } else {
        if(cur_node->left_son==NULL){
            cur_node->left_son=new_node;
            // new_node->level=step+1;
            // return;
        } else{
            add(cur_node->left_son, new_node, step+1);
        }
    }
    new_node->hight=calHight(new_node);
    cur_node->hight=calHight(cur_node);
    answ= balancing(cur_node);
    cur_node->hight=calHight(cur_node);
    return answ;
}


bool AWL_tree::insert(int ID){
    if(is_in_tree(ID)){
        return false;
    }
    node* new_node= new node;
    new_node->left_son=NULL;
    new_node->right_son=NULL;
    new_node->ID=ID;
    if (node_cnt==0){
        root=new_node;
        root->hight=1;
        depth=1;
    } else{
        new_node->hight=1;
        root=add(root, new_node, 1);
        // if(depth < new_node->level){
        //     depth=new_node->level;
        // }
    }

    node_cnt++;
    depth=root->hight;
    // balancing();
    return true;
}

node* AWL_tree::get(int ID){
    node* find_node=find(root, ID);
    return find_node;
}

node* AWL_tree::get_root(){
    return root;
}

void AWL_tree::draw_node(node* cur_node, int level){
    if (cur_node==NULL){
        return;
    }
    draw_node(cur_node->right_son, level+1);
    for(int i=0; i<level; ++i){
        cout<<"|===";
    }
    cout<<cur_node->ID<<","<<cur_node->hight<<'\n';
    draw_node(cur_node->left_son, level+1);

}

void AWL_tree::draw_tree(){
    draw_node(root, 0);
}

int AWL_tree::find_parent(node* cur_node, int child_id){
    if(cur_node->ID<child_id){
        if(cur_node->right_son!=NULL){
            if(cur_node->right_son->ID==child_id){
                return cur_node->ID;
            } else{
                return find_parent(cur_node->right_son, child_id);
            }
        }
    } else if(cur_node->ID>child_id){
        if(cur_node->left_son!=NULL){
            if(cur_node->left_son->ID==child_id){
                return cur_node->ID;
            } else{
                return find_parent(cur_node->left_son, child_id);
            }
        }
    }
    return -1;

}

int AWL_tree::parent_id(int child_id){
    if(!is_in_tree(child_id)){
        return -1;
    }
    return find_parent(root, child_id);

}

bool AWL_tree::is_available(int ID){
    node* cur_node=find(root, ID);
    if(cur_node==NULL){
        return false;
    }
    return cur_node->available;
}

void AWL_tree::change_availability(int ID, bool status){
    node* cur_node=find(root, ID);
    if(cur_node!=NULL){
        cur_node->available=status;
    }
}

void AWL_tree::bypass(node* cur_node, vector<int> &unavailable){
    if(!cur_node->available){
        unavailable.push_back(cur_node->ID);
    }
    if(cur_node->left_son!=NULL){
        bypass(cur_node->left_son, unavailable);
    }
    if(cur_node->right_son!=NULL){
        bypass(cur_node->right_son, unavailable);
    }
}

void AWL_tree::bypass_reset(node* cur_node){
    cur_node->available=false;
    if(cur_node->left_son!=NULL){
        bypass_reset(cur_node->left_son);
    }
    if(cur_node->right_son!=NULL){
        bypass_reset(cur_node->right_son);
    }
}

AWL_tree::~AWL_tree(){

}



int main(){
    AWL_tree tree;
    vector<int> data;
    tree.insert(2);
    tree.insert(7);
    tree.insert(1);
    tree.insert(10);
    tree.insert(4);
    tree.insert(11);
    // tree.change_availability(2, true);
    // tree.bypass(tree.get_root(), data);
    // node* n=tree.get(4);
    // cout<<"hui\n";
    tree.draw_tree();
    // cout<<'\n';
    // cout<<tree.parent_id(2);
    // cout<<"\n"<<tree.check_depth();
    // cout<<"hui hui\n";
    // cout<<"Size = "<<data.size()<<" Tree cnt = "<<tree.cnt();
}
```

***timer.h:***

```
#pragma once
#include <iostream>
#include <chrono>

using namespace std;

using t_t = chrono::time_point<std::chrono::system_clock>;
using Clock = chrono::high_resolution_clock;
using ms= chrono::milliseconds; // can use int64_t



class Timer{
    public:
        Timer();

        int64_t times();
        void start();
        void stop();

        ~Timer();

    private:
        bool t_work;
        t_t begin, end;
        ms oper_time{0};
};
```

***timer.cpp:***

```
#include "timer.h"
// #include<unistd.h>

Timer::Timer() {
    t_work=false;
}

Timer::~Timer(){

}



int64_t Timer::times(){
    if(t_work){
        return chrono::duration_cast<chrono::milliseconds>(Clock::now()-begin).count();

    } else{
        return oper_time.count();
    }
}

void Timer::start(){
    t_work=true;
    begin=Clock::now();
}

void Timer::stop(){
    if(t_work){
        t_work=false;
        oper_time=chrono::duration_cast<chrono::milliseconds>(Clock::now()-begin);
    }
}

// int main(){
```

```
//      Timer t;
//      cout<<"Time: "<<t.times()<<'\n';
//      t.start();
//      sleep(2);
//      cout<<"Time: "<<t.times()<<'\n';
//      t.stop();
//      sleep(2);
//      cout<<"Time: "<<t.times()<<'\n';


// }
```

## Протокол работы программы

**Тестирование:**

```
arsenii@PC-Larcha14:~/Documents/VS_code_prog/OSI/laba_5_7_clear$ strace -f
-owrite-simple.log ./manage_main
Welcome in our programm! This is what command i can do:
 - create 'id'
 - exec 'id' 'command(start/stop/time)'
 - heartbit 'time' (in ms)
 - draw
Or enter q or ^D to exit
Enter you command:
 ->create 8
Ok: 17873
 ->create 9
Sending...
Ok:8: 17889
 ->create 2
Sending...
Ok:8: 17893
 ->exec 2
time
Sending...
Ok:2: 0
 ->exec 2 start
Sending...
Ok:2
 ->exec 2 time
Sending...
Ok:2: 5656
 ->exec 2 stop
Sending...
Ok:2
```

```
 ->exec 2 time
Sending...
Ok:2: 11183
 ->exec 2 time
Sending...
Ok:2: 11183
 ->draw
|===9,1
8,2
|===2,1

 ->heartbit 3000
Sending...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
Heartbit: all nodes are available
 ->heartbit 3000
Sending...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
Heartbit: all nodes are available
 ->heartbit 3000
Sending...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
Receiving...
```

```
Heartbit: node 2 is unavailable now
 ->q
Break;
arsenii@PC-Larcha14:~/Documents/VS_code_prog/OSI/laba_5_7_clear$
```

===================================================================================

**Strace:**

```
arsenii@PC-Larcha14:~/Documents/VS_code_prog/OSI/laba_5_7_clear$ strace -f -e
trace=\!brk,clock_nanosleep,mmap,mprotect,munmap -owrite-simple3.log ./manage_main
Welcome in our programm! This is what command i can do:
 - create 'id'
 - exec 'id' 'command(start/stop/time)'
 - heartbit 'time' (in ms)
 - draw
Or enter q or ^D to exit
Enter you command:
 ->create 2
Ok: 20397
 ->create 3
Sending...
Ok:2: 20402
 ->exec 3 time
Sending...
Ok:3: 0
 ->exec 3 start
Sending...
Ok:3
ex ->exec 3 stop
Uncorrect input. Try again:
 ->Uncorrect input. Try again:
 ->Uncorrect input. Try again:
 ->exec 3 stop
Sending...
Ok:3
 ->exec 3 time
Sending...
Ok:3: 13275
 ->q
Break;
```

***write-simple3.log:***

*(отключил вывод системного вызова clock_nanosleep, т.к. занимает уж ооочень много места, прошлый лог с ним был на 200 мб и 2,5 млн строчек...)*

***Пояснения:***

sendto - отправление сообщения на сокет

recvmsg - получение сообщения с сокета

socket - создать конечную точку для связи

setsockopt() - set the socket options

bind() - bind a name to a socket

listen() - network listener daemon

getsockname() - get socket name

epoll_ctl - интерфейс управления описателями epoll (очень полезная штука, которая позволяет отложить реакцию на событие и продолжить ждать остальные события)

poll - input/output multiplexing (мультиплекси́рование — уплотнение канала, то есть передача нескольких потоков данных с меньшей скоростью по одному каналу)

```
20387 execve("./manage_main", ["./manage_main"], 0x7ffc80cae7e0 /* 56 vars */) = 0

20387 arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe4ad486d0) = -1 EINVAL (Invalid argument)

20387 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

20387 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=81715, ...}, AT_EMPTY_PATH) = 0

20387 close(3)                    = 0

20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libzmq.so.5", O_RDONLY|O_CLOEXEC) = 3

20387 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240\233\1\0\0\0\0\0"..., 832) = 832

20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=634936, ...}, AT_EMPTY_PATH) = 0

20387 close(3)                    = 0

20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = 3

20387 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

20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2260296, ...}, AT_EMPTY_PATH) = 0

20387 close(3)                    = 0

20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
```

```
20387 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
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=125488, ...}, AT_EMPTY_PATH) = 0
20387 close(3)                    = 0
20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
20387 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832
20387 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
20387 pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) =
48
20387 pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0
 =\340\2563\265?\356\25x\261\27\313A#\350"..., 68, 896) = 68
20387 newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2216304, ...}, AT_EMPTY_PATH) = 0
20387 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
20387 close(3)                    = 0
20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libbsd.so.0", O_RDONLY|O_CLOEXEC) = 3
20387 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
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=89096, ...}, AT_EMPTY_PATH) = 0
20387 close(3)                    = 0
20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libsodium.so.23",
 O_RDONLY|O_CLOEXEC) = 3
20387 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
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=355040, ...}, AT_EMPTY_PATH) = 0
20387 close(3)                    = 0
20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpgm-5.3.so.0",
 O_RDONLY|O_CLOEXEC) = 3
20387 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\340L\0\0\0\0\0\0"..., 832) = 832
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=310264, ...}, AT_EMPTY_PATH) = 0
20387 close(3)                    = 0
20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libnorm.so.1", O_RDONLY|O_CLOEXEC) =
3
```

```
20387 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 \255\0\0\0\0\0\0"..., 832) = 832
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=497824, ...}, AT_EMPTY_PATH) = 0
20387 close(3)                    = 0
20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libgssapi_krb5.so.2",
 O_RDONLY|O_CLOEXEC) = 3
20387 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
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=338648, ...}, AT_EMPTY_PATH) = 0
20387 close(3)                    = 0
20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
20387 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
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=940560, ...}, AT_EMPTY_PATH) = 0
20387 close(3)                    = 0
20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libmd.so.0", O_RDONLY|O_CLOEXEC) = 3
20387 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
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=47472, ...}, AT_EMPTY_PATH) = 0
20387 close(3)                    = 0
20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC)
= 3
20387 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
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=21448, ...}, AT_EMPTY_PATH) = 0
20387 close(3)                    = 0
20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libkrb5.so.3", O_RDONLY|O_CLOEXEC) =
3
20387 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
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=827936, ...}, AT_EMPTY_PATH) = 0
20387 close(3)                    = 0
20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libk5crypto.so.3",
 O_RDONLY|O_CLOEXEC) = 3
20387 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
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=182864, ...}, AT_EMPTY_PATH) = 0
20387 close(3)                    = 0
```

```
    20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libcom_err.so.2",
     O_RDONLY|O_CLOEXEC) = 3
    20387 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
    20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=18504, ...}, AT_EMPTY_PATH) = 0
    20387 close(3)                    = 0
    20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libkrb5support.so.0",
     O_RDONLY|O_CLOEXEC) = 3
    20387 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
    20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=52016, ...}, AT_EMPTY_PATH) = 0
    20387 close(3)                    = 0
    20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libkeyutils.so.1", O_RDONLY|O_CLOEXEC)
= 3
    20387 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
    20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=22600, ...}, AT_EMPTY_PATH) = 0
    20387 close(3)                    = 0
    20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libresolv.so.2", O_RDONLY|O_CLOEXEC) =
3
    20387 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
    20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=68552, ...}, AT_EMPTY_PATH) = 0
    20387 close(3)                    = 0
    20387 arch_prctl(ARCH_SET_FS, 0x7f57c38339c0) = 0
    20387 set_tid_address(0x7f57c3833c90)   = 20387
    20387 set_robust_list(0x7f57c3833ca0, 24) = 0
    20387 rseq(0x7f57c3834360, 0x20, 0, 0x53053053) = 0
    20387 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
     rlim_max=RLIM64_INFINITY}) = 0
    20387 getrandom("\x2c\x01\xb6\x7f\xd4\xc3\xed\xa3", 8, GRND_NONBLOCK) = 8
    20387 futex(0x7f57c382977c, FUTEX_WAKE_PRIVATE, 2147483647) = 0
    20387 openat(AT_FDCWD, "/sys/devices/system/cpu/online", O_RDONLY|O_CLOEXEC) = 3
    20387 read(3, "0-11\n", 1024)          = 5
    20387 close(3)                    = 0
```

```
20387 openat(AT_FDCWD, "/sys/devices/system/cpu",
 O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 3
20387 newfstatat(3, "", {st_mode=S_IFDIR|0755, st_size=0, ...}, AT_EMPTY_PATH) = 0
20387 getdents64(3, 0x55f459f1bee0 /* 30 entries */, 32768) = 864
20387 getdents64(3, 0x55f459f1bee0 /* 0 entries */, 32768) = 0
20387 close(3)                    = 0
20387 getpid()                    = 20387
20387 sched_getaffinity(20387, 128, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]) = 8
20387 newfstatat(AT_FDCWD, "/etc/nsswitch.conf", {st_mode=S_IFREG|0644, st_size=542, ...},
0) = 0
20387 newfstatat(AT_FDCWD, "/", {st_mode=S_IFDIR|0755, st_size=4096, ...}, 0) = 0
20387 openat(AT_FDCWD, "/etc/nsswitch.conf", O_RDONLY|O_CLOEXEC) = 3
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=542, ...}, AT_EMPTY_PATH) = 0
20387 read(3, "# /etc/nsswitch.conf\n#\n# Example"..., 4096) = 542
20387 read(3, "", 4096)           = 0
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=542, ...}, AT_EMPTY_PATH) = 0
20387 close(3)                    = 0
20387 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=81715, ...}, AT_EMPTY_PATH) = 0
20387 close(3)                    = 0
20387 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/glibc-hwcaps/x86-64-v4/libnss_db.so.2",
 O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
20387 newfstatat(AT_FDCWD, "/lib/x86_64-linux-gnu/glibc-hwcaps/x86-64-v4",
 0x7ffe4ad456c0, 0) = -1 ENOENT (No such file or directory)

                              ...openat+newfstatat…

20387 openat(AT_FDCWD, "/etc/protocols", O_RDONLY|O_CLOEXEC) = 3
20387 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2932, ...}, AT_EMPTY_PATH) = 0
20387 lseek(3, 0, SEEK_SET)       = 0
```

```
    20387 read(3, "# Internet (IP) protocols\n#\n# Up"..., 4096) = 2932

    20387 read(3, "", 4096)               = 0

    20387 close(3)                        = 0

    20387 eventfd2(0, EFD_CLOEXEC)        = 3

    20387 fcntl(3, F_GETFL)               = 0x2 (flags O_RDWR)

    20387 fcntl(3, F_SETFL, O_RDWR|O_NONBLOCK) = 0

    20387 fcntl(3, F_GETFL)               = 0x802 (flags O_RDWR|O_NONBLOCK)

    20387 fcntl(3, F_SETFL, O_RDWR|O_NONBLOCK) = 0

    20387 getpid()                        = 20387

    20387 getpid()                        = 20387

    20387 getrandom("\x44\x22\x00\x61\x70\xa2\x3a\xe2\xe4\x3b\x22\x87\xa3\x11\xb3\xff", 16, 0) =
16

    20387 getrandom("\x0d\x1d\xd1\x8b\xc8\x9a\x35\xc7\x37\x4b\x70\x6c\x57\xdd\xe8\xad", 16, 0)
= 16

    20387 eventfd2(0, EFD_CLOEXEC)        = 4

    20387 fcntl(4, F_GETFL)               = 0x2 (flags O_RDWR)

    20387 fcntl(4, F_SETFL, O_RDWR|O_NONBLOCK) = 0

    20387 fcntl(4, F_GETFL)               = 0x802 (flags O_RDWR|O_NONBLOCK)

    20387 fcntl(4, F_SETFL, O_RDWR|O_NONBLOCK) = 0

    20387 getpid()                        = 20387

    20387 epoll_create1(EPOLL_CLOEXEC)    = 5

    20387 epoll_ctl(5, EPOLL_CTL_ADD, 4, {events=0, data={u32=1509016160,

     u64=94507969397344}}) = 0

    20387 epoll_ctl(5, EPOLL_CTL_MOD, 4, {events=EPOLLIN, data={u32=1509016160,

     u64=94507969397344}}) = 0

    20387 getpid()                        = 20387

    20387 rt_sigaction(SIGRT_1, {sa_handler=0x7f57c3291870, sa_mask=[],

     sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,

     sa_restorer=0x7f57c3242520}, NULL, 8) = 0

    20387 rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

    20387 rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
```

```
20387clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_T\
HREAD|CL
ONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
 child_tid=0x7f57c2fb6910, parent_tid=0x7f57c2fb6910, exit_signal=0, stack=0x7f57c27b6000,
 stack_size=0x7ffc80, tls=0x7f57c2fb6640} => {parent_tid=[20389]}, 88) = 20389
20389 rseq(0x7f57c2fb6fe0, 0x20, 0, 0x53053053 <unfinished ...>
20387 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
20389 <... rseq resumed>)             = 0
20387 <... rt_sigprocmask resumed>NULL, 8) = 0
20389 set_robust_list(0x7f57c2fb6920, 24 <unfinished ...>
20387 eventfd2(0, EFD_CLOEXEC <unfinished ...>
20389 <... set_robust_list resumed>)    = 0
20387 <... eventfd2 resumed>)          = 6
20389 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
20387 fcntl(6, F_GETFL <unfinished ...>
20389 <... rt_sigprocmask resumed>NULL, 8) = 0
20387 <... fcntl resumed>)             = 0x2 (flags O_RDWR)
20387 fcntl(6, F_SETFL, O_RDWR|O_NONBLOCK) = 0
20389 rt_sigprocmask(SIG_BLOCK, ~[RTMIN RT_1],  <unfinished ...>
20387 fcntl(6, F_GETFL <unfinished ...>
20389 <... rt_sigprocmask resumed>NULL, 8) = 0
20387 <... fcntl resumed>)             = 0x802 (flags O_RDWR|O_NONBLOCK)
20387 fcntl(6, F_SETFL, O_RDWR|O_NONBLOCK <unfinished ...>
20389 sched_getparam(20389,  <unfinished ...>
20387 <... fcntl resumed>)             = 0
20389 <... sched_getparam resumed>[0])  = 0
20387 getpid( <unfinished ...>
20389 sched_getscheduler(20389 <unfinished ...>
20387 <... getpid resumed>)            = 20387
20389 <... sched_getscheduler resumed>) = 0 (SCHED_OTHER)
```

```
20387 epoll_create1(EPOLL_CLOEXEC <unfinished ...>
20389 sched_setscheduler(20389, SCHED_OTHER, [0] <unfinished ...>
20387 <... epoll_create1 resumed>)      = 7
20387 epoll_ctl(7, EPOLL_CTL_ADD, 6, {events=0, data={u32=1509037216,
 u64=94507969418400}} <unfinished ...>
20389 <... sched_setscheduler resumed>) = 0
20387 <... epoll_ctl resumed>)         = 0
20387 epoll_ctl(7, EPOLL_CTL_MOD, 6, {events=EPOLLIN, data={u32=1509037216,
 u64=94507969418400}} <unfinished ...>
20389 prctl(PR_SET_NAME, "ZMQbg/Reaper" <unfinished ...>
20387 <... epoll_ctl resumed>)         = 0
20389 <... prctl resumed>)             = 0
20389 epoll_wait(5,  <unfinished ...>
20387 rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
20387

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CL
ONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
 child_tid=0x7f57c27b5910, parent_tid=0x7f57c27b5910, exit_signal=0, stack=0x7f57c1fb5000,
 stack_size=0x7ffc80, tls=0x7f57c27b5640} => {parent_tid=[20390]}, 88) = 20390
20387 rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
20387 eventfd2(0, EFD_CLOEXEC)         = 8
20387 fcntl(8, F_GETFL)                = 0x2 (flags O_RDWR)
20387 fcntl(8, F_SETFL, O_RDWR|O_NONBLOCK) = 0
20387 fcntl(8, F_GETFL <unfinished ...>
20390 rseq(0x7f57c27b5fe0, 0x20, 0, 0x53053053 <unfinished ...>
20387 <... fcntl resumed>)             = 0x802 (flags O_RDWR|O_NONBLOCK)
20387 fcntl(8, F_SETFL, O_RDWR|O_NONBLOCK) = 0
20387 getpid()                         = 20387
20387 eventfd2(0, EFD_CLOEXEC)         = 9
20387 fcntl(9, F_GETFL)                = 0x2 (flags O_RDWR)
20390 <... rseq resumed>)              = 0
20387 fcntl(9, F_SETFL, O_RDWR|O_NONBLOCK) = 0
20390 set_robust_list(0x7f57c27b5920, 24 <unfinished ...>
20387 fcntl(9, F_GETFL)                = 0x802 (flags O_RDWR|O_NONBLOCK)
20390 <... set_robust_list resumed>)   = 0
20387 fcntl(9, F_SETFL, O_RDWR|O_NONBLOCK) = 0
20390 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
20387 getpid()                         = 20387
20390 <... rt_sigprocmask resumed>NULL, 8) = 0
20387 getpid( <unfinished ...>
20390 rt_sigprocmask(SIG_BLOCK, ~[RTMIN RT_1],  <unfinished ...>
20387 <... getpid resumed>)            = 20387
20390 <... rt_sigprocmask resumed>NULL, 8) = 0
20387 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>
20390 sched_getparam(20390,  <unfinished ...>
20387 <... poll resumed>)              = 0 (Timeout)
20390 <... sched_getparam resumed>[0])  = 0
20390 sched_getscheduler(20390 <unfinished ...>
20387 socket(AF_NETLINK, SOCK_RAW|SOCK_CLOEXEC, NETLINK_ROUTE <unfinished
...>
20390 <... sched_getscheduler resumed>) = 0 (SCHED_OTHER)
20387 <... socket resumed>)            = 10
20390 sched_setscheduler(20390, SCHED_OTHER, [0] <unfinished ...>
20387 bind(10, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12 <unfinished ...>
20390 <... sched_setscheduler resumed>) = 0
20387 <... bind resumed>)              = 0
20390 prctl(PR_SET_NAME, "ZMQbg/IO/0" <unfinished ...>
20387 getsockname(10, {sa_family=AF_NETLINK, nl_pid=20387, nl_groups=00000000}, [12])
= 0
20390 <... prctl resumed>)             = 0
```

```
20387 sendto(10, [{nlmsg_len=20, nlmsg_type=RTM_GETLINK,
nlmsg_flags=NLM_F_REQUEST|NLM_F_DUMP, nlmsg_seq=1704658042, nlmsg_pid=0},
{ifi_family=AF_UNSPEC, ...}], 20, 0, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12
<unfinished ...>

20390 epoll_wait(7,  <unfinished ...>

20387 <... sendto resumed>)            = 20

20387 recvmsg(10, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000},
msg_namelen=12, msg_iov=[{iov_base=[[{nlmsg_len=1404, nlmsg_type=RTM_NEWLINK,
nlmsg_flags=NLM_F_MULTI, nlmsg_seq=1704658042, nlmsg_pid=20387},
{ifi_family=AF_UNSPEC, ifi_type=ARPHRD_LOOPBACK, ifi_index=if_nametoindex("lo"),
ifi_flags=IFF_UP|IFF_LOOPBACK|IFF_RUNNING|IFF_LOWER_UP, ifi_change=0}, [[{nla_len=7,
nla_type=IFLA_IFNAME}, "lo"], [{nla_len=8, nla_type=IFLA_TXQLEN}, 1000], [{nla_len=5,
nla_type=IFLA_OPERSTATE}, 0], [{nla_len=5, nla_type=IFLA_LINKMODE}, 0], [{nla_len=8,
nla_type=IFLA_MTU}, 65536], [{nla_len=8, nla_type=IFLA_MIN_MTU}, 0], [{nla_len=8,
nla_type=IFLA_MAX_MTU}, 0], [{nla_len=8, nla_type=IFLA_GROUP}, 0], [{nla_len=8,
nla_type=IFLA_PROMISCUITY}, 0], [{nla_len=8, nla_type=0x3d /* IFLA_??? */},
"\x00\x00\x00\x00"], [{nla_len=8, nla_type=IFLA_NUM_TX_QUEUES}, 1], [{nla_len=8,
nla_type=IFLA_GSO_MAX_SEGS}, 65535], [{nla_len=8, nla_type=IFLA_GSO_MAX_SIZE},
65536], [{nla_len=8, nla_type=0x3a /* IFLA_??? */}, "\x00\x00\x01\x00"], [{nla_len=8, nla_type=0x3b
/* IFLA_??? */}, "\xf8\xff\x07\x00"], [{nla_len=8, nla_type=0x3c /* IFLA_??? */}, "\xff\xff\x00\x00"],
[{nla_len=8, nla_type=IFLA_NUM_RX_QUEUES}, 1], [{nla_len=5, nla_type=IFLA_CARRIER}, 1],
[{nla_len=12, nla_type=IFLA_QDISC}, "noqueue"], [{nla_len=8,
nla_type=IFLA_CARRIER_CHANGES}, 0], [{nla_len=8, nla_type=IFLA_CARRIER_UP_COUNT},
0], [{nla_len=8, nla_type=IFLA_CARRIER_DOWN_COUNT}, 0], [{nla_len=5,
nla_type=IFLA_PROTO_DOWN}, 0], [{nla_len=36, nla_type=IFLA_MAP}, {mem_start=0,
mem_end=0, base_addr=0, irq=0, dma=0, port=0}], [{nla_len=10, nla_type=IFLA_ADDRESS},
00:00:00:00:00:00], [{nla_len=10, nla_type=IFLA_BROADCAST}, 00:00:00:00:00:00], [{nla_len=204,
nla_type=IFLA_STATS64}, {rx_packets=18620, tx_packets=18620, rx_bytes=1990342,
tx_bytes=1990342, rx_errors=0, tx_errors=0, rx_dropped=0, tx_dropped=0, multicast=0, collisions=0,
rx_length_errors=0, rx_over_errors=0, rx_crc_errors=0, rx_frame_errors=0, rx_fifo_errors=0,
rx_missed_errors=0, tx_aborted_errors=0, tx_carrier_errors=0, tx_fifo_errors=0, tx_heartbeat_errors=0,
tx_window_errors=0, rx_compressed=0, tx_compressed=0, rx_nohandler=0}], [{nla_len=100,
nla_type=IFLA_STATS}, {rx_packets=18620, tx_packets=18620, rx_bytes=1990342,
tx_bytes=1990342, rx_errors=0, tx_errors=0, rx_dropped=0, tx_dropped=0, multicast=0, collisions=0,
rx_length_errors=0, rx_over_errors=0, rx_crc_errors=0, rx_frame_errors=0, rx_fifo_errors=0,
rx_missed_errors=0, tx_aborted_errors=0, tx_carrier_errors=0, tx_fifo_errors=0, tx_heartbeat_errors=0,
tx_window_errors=0, rx_compressed=0, tx_compressed=0, rx_nohandler=0}], [{nla_len=12,
nla_type=IFLA_XDP}, [{nla_len=5, nla_type=IFLA_XDP_ATTACHED},
XDP_ATTACHED_NONE]], [{nla_len=804, nla_type=IFLA_AF_SPEC}, [[{nla_len=12,
nla_type=AF_MCTP}, [{nla_len=8, nla_type=IFLA_MCTP_NET}, 1]], [{nla_len=140,
nla_type=AF_INET}, [{nla_len=136, nla_type=IFLA_INET_CONF},
[[IPV4_DEVCONF_FORWARDING-1] = 0, [IPV4_DEVCONF_MC_FORWARDING-1] = 0,
[IPV4_DEVCONF_PROXY_ARP-1] = 0, [IPV4_DEVCONF_ACCEPT_REDIRECTS-1] = 1,
[IPV4_DEVCONF_SECURE_REDIRECTS-1] = 1, [IPV4_DEVCONF_SEND_REDIRECTS-1] = 1,
[IPV4_DEVCONF_SHARED_MEDIA-1] = 1, [IPV4_DEVCONF_RP_FILTER-1] = 2,
[IPV4_DEVCONF_ACCEPT_SOURCE_ROUTE-1] = 0, [IPV4_DEVCONF_BOOTP_RELAY-1] = 0,
[IPV4_DEVCONF_LOG_MARTIANS-1] = 0, [IPV4_DEVCONF_TAG-1] = 0,
[IPV4_DEVCONF_ARPFILTER-1] = 0, [IPV4_DEVCONF_MEDIUM_ID-1] = 0,
[IPV4_DEVCONF_NOXFRM-1] = 1, [IPV4_DEVCONF_NOPOLICY-1] = 1,
[IPV4_DEVCONF_FORCE_IGMP_VERSION-1] = 0, [IPV4_DEVCONF_ARP_ANNOUNCE-1] = 0,
[IPV4_DEVCONF_ARP_IGNORE-1] = 0, [IPV4_DEVCONF_PROMOTE_SECONDARIES-1] = 1,
[IPV4_DEVCONF_ARP_ACCEPT-1] = 0, [IPV4_DEVCONF_ARP_NOTIFY-1] = 0,
[IPV4_DEVCONF_ACCEPT_LOCAL-1] = 0, [IPV4_DEVCONF_SRC_VMARK-1] = 0,
[IPV4_DEVCONF_PROXY_ARP_PVLAN-1] = 0, [IPV4_DEVCONF_ROUTE_LOCALNET-1] = 0,
[IPV4_DEVCONF_IGMPV2_UNSOLICITED_REPORT_INTERVAL-1] = 10000,
[IPV4_DEVCONF_IGMPV3_UNSOLICITED_REPORT_INTERVAL-1] = 1000,
[IPV4_DEVCONF_IGNORE_ROUTES_WITH_LINKDOWN-1] = 0,
[IPV4_DEVCONF_DROP_UNICAST_IN_L2_MULTICAST-1] = 0,
[IPV4_DEVCONF_DROP_GRATUITOUS_ARP-1] = 0, [IPV4_DEVCONF_BC_FORWARDING-1] =
0, ...]]], [{nla_len=648, nla_type=AF_INET6}, [[{nla_len=8, nla_type=IFLA_INET6_FLAGS},
IF_READY], [{nla_len=20, nla_type=IFLA_INET6_CACHEINFO}, {max_reasm_len=65535,
tstamp=170, reachable_time=21796, retrans_time=1000}], [{nla_len=236,
nla_type=IFLA_INET6_CONF}, [[DEVCONF_FORWARDING] = 0, [DEVCONF_HOPLIMIT] = 64,
[DEVCONF_MTU6] = 65536, [DEVCONF_ACCEPT_RA] = 1, [DEVCONF_ACCEPT_REDIRECTS]
= 1, [DEVCONF_AUTOCONF] = 1, [DEVCONF_DAD_TRANSMITS] = 1,
[DEVCONF_RTR_SOLICITS] = -1, [DEVCONF_RTR_SOLICIT_INTERVAL] = 4000,
[DEVCONF_RTR_SOLICIT_DELAY] = 1000, [DEVCONF_USE_TEMPADDR] = -1,
[DEVCONF_TEMP_VALID_LFT] = 604800, [DEVCONF_TEMP_PREFERED_LFT] = 86400,
[DEVCONF_REGEN_MAX_RETRY] = 3, [DEVCONF_MAX_DESYNC_FACTOR] = 600,
[DEVCONF_MAX_ADDRESSES] = 16, [DEVCONF_FORCE_MLD_VERSION] = 0,
[DEVCONF_ACCEPT_RA_DEFRTR] = 1, [DEVCONF_ACCEPT_RA_PINFO] = 1,
[DEVCONF_ACCEPT_RA_RTR_PREF] = 1, [DEVCONF_RTR_PROBE_INTERVAL] = 60000,
[DEVCONF_ACCEPT_RA_RT_INFO_MAX_PLEN] = 0, [DEVCONF_PROXY_NDP] = 0,
[DEVCONF_OPTIMISTIC_DAD] = 0, [DEVCONF_ACCEPT_SOURCE_ROUTE] = 0,
[DEVCONF_MC_FORWARDING] = 0, [DEVCONF_DISABLE_IPV6] = 0,
[DEVCONF_ACCEPT_DAD] = -1, [DEVCONF_FORCE_TLLAO] = 0,
[DEVCONF_NDISC_NOTIFY] = 0, [DEVCONF_MLDV1_UNSOLICITED_REPORT_INTERVAL] =
10000, [DEVCONF_MLDV2_UNSOLICITED_REPORT_INTERVAL] = 1000, ...]], [{nla_len=300,
nla_type=IFLA_INET6_STATS}, [[IPSTATS_MIB_NUM] = 37, [IPSTATS_MIB_INPKTS] = 6,
[IPSTATS_MIB_INOCTETS] = 432, [IPSTATS_MIB_INDELIVERS] = 6,
[IPSTATS_MIB_OUTFORWDATAGRAMS] = 0, [IPSTATS_MIB_OUTPKTS] = 6,
[IPSTATS_MIB_OUTOCTETS] = 432, [IPSTATS_MIB_INHDRERRORS] = 0,
[IPSTATS_MIB_INTOOBIGERRORS] = 0, [IPSTATS_MIB_INNOROUTES] = 0,
[IPSTATS_MIB_INADDRERRORS] = 0, [IPSTATS_MIB_INUNKNOWNPROTOS] = 0,
[IPSTATS_MIB_INTRUNCATEDPKTS] = 0, [IPSTATS_MIB_INDISCARDS] = 0,
[IPSTATS_MIB_OUTDISCARDS] = 0, [IPSTATS_MIB_OUTNOROUTES] = 0,
[IPSTATS_MIB_REASMTIMEOUT] = 0, [IPSTATS_MIB_REASMREQDS] = 0,
[IPSTATS_MIB_REASMOKS] = 0, [IPSTATS_MIB_REASMFAILS] = 0,
[IPSTATS_MIB_FRAGOKS] = 0, [IPSTATS_MIB_FRAGFAILS] = 0,
[IPSTATS_MIB_FRAGCREATES] = 0, [IPSTATS_MIB_INMCASTPKTS] = 0,
[IPSTATS_MIB_OUTMCASTPKTS] = 2, [IPSTATS_MIB_INBCASTPKTS] = 0,
[IPSTATS_MIB_OUTBCASTPKTS] = 0, [IPSTATS_MIB_INMCASTOCTETS] = 0,
[IPSTATS_MIB_OUTMCASTOCTETS] = 152, [IPSTATS_MIB_INBCASTOCTETS] = 0,
[IPSTATS_MIB_OUTBCASTOCTETS] = 0, [IPSTATS_MIB_CSUMERRORS] = 0, ...]], [{nla_len=52,
nla_type=IFLA_INET6_ICMP6STATS}, [[ICMP6_MIB_NUM] = 6, [ICMP6_MIB_INMSGS] = 2,
[ICMP6_MIB_INERRORS] = 0, [ICMP6_MIB_OUTMSGS] = 2, [ICMP6_MIB_OUTERRORS] = 0,
[ICMP6_MIB_CSUMERRORS] = 0]], [{nla_len=20, nla_type=IFLA_INET6_TOKEN},
inet_pton(AF_INET6, "::")], [{nla_len=5, nla_type=IFLA_INET6_ADDR_GEN_MODE},
IN6_ADDR_GEN_MODE_EUI64]]]]], {nla_len=4, nla_type=NLA_F_NESTED|0x3e /* IFLA_???
*/}]], [{nlmsg_len=1440, nlmsg_type=RTM_NEWLINK, nlmsg_flags=NLM_F_MULTI,
nlmsg_seq=1704658042, nlmsg_pid=20387}, {ifi_family=AF_UNSPEC, ifi_type=ARPHRD_ETHER,
ifi_index=if_nametoindex("enp3s0"), ifi_flags=IFF_UP|IFF_BROADCAST|IFF_MULTICAST,
ifi_change=0}, [[{nla_len=11, nla_type=IFLA_IFNAME}, "enp3s0"], [{nla_len=8,
nla_type=IFLA_TXQLEN}, 1000], [{nla_len=5, nla_type=IFLA_OPERSTATE}, 2], [{nla_len=5,
nla_type=IFLA_LINKMODE}, 0], [{nla_len=8, nla_type=IFLA_MTU}, 1500], [{nla_len=8,
nla_type=IFLA_MIN_MTU}, 68], [{nla_len=8, nla_type=IFLA_MAX_MTU}, 9194], [{nla_len=8,
nla_type=IFLA_GROUP}, 0], [{nla_len=8, nla_type=IFLA_PROMISCUITY}, 0], [{nla_len=8,
nla_type=0x3d /* IFLA_??? */}, "\x00\x00\x00\x00"], [{nla_len=8,
nla_type=IFLA_NUM_TX_QUEUES}, 1], [{nla_len=8, nla_type=IFLA_GSO_MAX_SEGS}, 64],
[{nla_len=8, nla_type=IFLA_GSO_MAX_SIZE}, 64000], [{nla_len=8, nla_type=0x3a /* IFLA_??? */},
"\x00\x00\x01\x00"], [{nla_len=8, nla_type=0x3b /* IFLA_??? */}, "\x00\xfa\x00\x00"], [{nla_len=8,
nla_type=0x3c /* IFLA_??? */}, "\x40\x00\x00\x00"], [{nla_len=8,
nla_type=IFLA_NUM_RX_QUEUES}, 1], [{nla_len=5, nla_type=IFLA_CARRIER}, 0], [{nla_len=13,
nla_type=IFLA_QDISC}, "fq_codel"], [{nla_len=8, nla_type=IFLA_CARRIER_CHANGES}, 1],
[{nla_len=8, nla_type=IFLA_CARRIER_UP_COUNT}, 0], [{nla_len=8,
nla_type=IFLA_CARRIER_DOWN_COUNT}, 1], [{nla_len=5, nla_type=IFLA_PROTO_DOWN}, 0],
[{nla_len=36, nla_type=IFLA_MAP}, {mem_start=0, mem_end=0, base_addr=0, irq=0, dma=0,
port=0}], [{nla_len=10, nla_type=IFLA_ADDRESS}, 04:7c:16:31:22:f5], [{nla_len=10,
nla_type=IFLA_BROADCAST}, ff:ff:ff:ff:ff:ff], [{nla_len=204, nla_type=IFLA_STATS64},
{rx_packets=0, tx_packets=0, rx_bytes=0, tx_bytes=0, rx_errors=0, tx_errors=0, rx_dropped=0,
tx_dropped=0, multicast=0, collisions=0, rx_length_errors=0, rx_over_errors=0, rx_crc_errors=0,
rx_frame_errors=0, rx_fifo_errors=0, rx_missed_errors=0, tx_aborted_errors=0, tx_carrier_errors=0,
tx_fifo_errors=0, tx_heartbeat_errors=0, tx_window_errors=0, rx_compressed=0, tx_compressed=0,
rx_nohandler=0}], [{nla_len=100, nla_type=IFLA_STATS}, {rx_packets=0, tx_packets=0, rx_bytes=0,
tx_bytes=0, rx_errors=0, tx_errors=0, rx_dropped=0, tx_dropped=0, multicast=0, collisions=0,
rx_length_errors=0, rx_over_errors=0, rx_crc_errors=0, rx_frame_errors=0, rx_fifo_errors=0,
rx_missed_errors=0, tx_aborted_errors=0, tx_carrier_errors=0, tx_fifo_errors=0, tx_heartbeat_errors=0,
tx_window_errors=0, rx_compressed=0, tx_compressed=0, rx_nohandler=0}], [{nla_len=12,
nla_type=IFLA_XDP}, [{nla_len=5, nla_type=IFLA_XDP_ATTACHED},
XDP_ATTACHED_NONE]], [{nla_len=10, nla_type=IFLA_PERM_ADDRESS}, 04:7c:16:31:22:f5],
[{nla_len=792, nla_type=IFLA_AF_SPEC}, [[{nla_len=140, nla_type=AF_INET}, [{nla_len=136,
nla_type=IFLA_INET_CONF}, [[IPV4_DEVCONF_FORWARDING-1] = 0,
[IPV4_DEVCONF_MC_FORWARDING-1] = 0, [IPV4_DEVCONF_PROXY_ARP-1] = 0,
[IPV4_DEVCONF_ACCEPT_REDIRECTS-1] = 1, [IPV4_DEVCONF_SECURE_REDIRECTS-1] = 1,
[IPV4_DEVCONF_SEND_REDIRECTS-1] = 1, [IPV4_DEVCONF_SHARED_MEDIA-1] = 1,
[IPV4_DEVCONF_RP_FILTER-1] = 2, [IPV4_DEVCONF_ACCEPT_SOURCE_ROUTE-1] = 0,
[IPV4_DEVCONF_BOOTP_RELAY-1] = 0, [IPV4_DEVCONF_LOG_MARTIANS-1] = 0,
[IPV4_DEVCONF_TAG-1] = 0, [IPV4_DEVCONF_ARPFILTER-1] = 0,
[IPV4_DEVCONF_MEDIUM_ID-1] = 0, [IPV4_DEVCONF_NOXFRM-1] = 0,
[IPV4_DEVCONF_NOPOLICY-1] = 0, [IPV4_DEVCONF_FORCE_IGMP_VERSION-1] = 0,
[IPV4_DEVCONF_ARP_ANNOUNCE-1] = 0, [IPV4_DEVCONF_ARP_IGNORE-1] = 0,
[IPV4_DEVCONF_PROMOTE_SECONDARIES-1] = 1, [IPV4_DEVCONF_ARP_ACCEPT-1] = 0,
[IPV4_DEVCONF_ARP_NOTIFY-1] = 0, [IPV4_DEVCONF_ACCEPT_LOCAL-1] = 0,
[IPV4_DEVCONF_SRC_VMARK-1] = 0, [IPV4_DEVCONF_PROXY_ARP_PVLAN-1] = 0,
[IPV4_DEVCONF_ROUTE_LOCALNET-1] = 0,
[IPV4_DEVCONF_IGMPV2_UNSOLICITED_REPORT_INTERVAL-1] = 10000,
[IPV4_DEVCONF_IGMPV3_UNSOLICITED_REPORT_INTERVAL-1] = 1000,
[IPV4_DEVCONF_IGNORE_ROUTES_WITH_LINKDOWN-1] = 0,
[IPV4_DEVCONF_DROP_UNICAST_IN_L2_MULTICAST-1] = 0,
[IPV4_DEVCONF_DROP_GRATUITOUS_ARP-1] = 0, [IPV4_DEVCONF_BC_FORWARDING-1] =
0, ...]]], [{nla_len=648, nla_type=AF_INET6}, [[{nla_len=8, nla_type=IFLA_INET6_FLAGS}, 0],
[{nla_len=20, nla_type=IFLA_INET6_CACHEINFO}, {max_reasm_len=65535, tstamp=1058776,
reachable_time=17116, retrans_time=1000}], [{nla_len=236, nla_type=IFLA_INET6_CONF},
[[DEVCONF_FORWARDING] = 0, [DEVCONF_HOPLIMIT] = 64, [DEVCONF_MTU6] = 1500,
[DEVCONF_ACCEPT_RA] = 0, [DEVCONF_ACCEPT_REDIRECTS] = 1,
[DEVCONF_AUTOCONF] = 1, [DEVCONF_DAD_TRANSMITS] = 1,
[DEVCONF_RTR_SOLICITS] = -1, [DEVCONF_RTR_SOLICIT_INTERVAL] = 4000,
[DEVCONF_RTR_SOLICIT_DELAY] = 1000, [DEVCONF_USE_TEMPADDR] = 0,
[DEVCONF_TEMP_VALID_LFT] = 604800, [DEVCONF_TEMP_PREFERED_LFT] = 86400,
[DEVCONF_REGEN_MAX_RETRY] = 3, [DEVCONF_MAX_DESYNC_FACTOR] = 600,
[DEVCONF_MAX_ADDRESSES] = 16, [DEVCONF_FORCE_MLD_VERSION] = 0,
[DEVCONF_ACCEPT_RA_DEFRTR] = 1, [DEVCONF_ACCEPT_RA_PINFO] = 1,
[DEVCONF_ACCEPT_RA_RTR_PREF] = 1, [DEVCONF_RTR_PROBE_INTERVAL] = 60000,
[DEVCONF_ACCEPT_RA_RT_INFO_MAX_PLEN] = 0, [DEVCONF_PROXY_NDP] = 0,
[DEVCONF_OPTIMISTIC_DAD] = 0, [DEVCONF_ACCEPT_SOURCE_ROUTE] = 0,
[DEVCONF_MC_FORWARDING] = 0, [DEVCONF_DISABLE_IPV6] = 0,
[DEVCONF_ACCEPT_DAD] = 1, [DEVCONF_FORCE_TLLAO] = 0,
[DEVCONF_NDISC_NOTIFY] = 0, [DEVCONF_MLDV1_UNSOLICITED_REPORT_INTERVAL] =
10000, [DEVCONF_MLDV2_UNSOLICITED_REPORT_INTERVAL] = 1000, ...]], [{nla_len=300,
nla_type=IFLA_INET6_STATS}, [[IPSTATS_MIB_NUM] = 37, [IPSTATS_MIB_INPKTS] = 0,
[IPSTATS_MIB_INOCTETS] = 0, [IPSTATS_MIB_INDELIVERS] = 0,
[IPSTATS_MIB_OUTFORWDATAGRAMS] = 0, [IPSTATS_MIB_OUTPKTS] = 0,
[IPSTATS_MIB_OUTOCTETS] = 0, [IPSTATS_MIB_INHDRERRORS] = 0,
[IPSTATS_MIB_INTOOBIGERRORS] = 0, [IPSTATS_MIB_INNOROUTES] = 0,
[IPSTATS_MIB_INADDRERRORS] = 0, [IPSTATS_MIB_INUNKNOWNPROTOS] = 0,
[IPSTATS_MIB_INTRUNCATEDPKTS] = 0, [IPSTATS_MIB_INDISCARDS] = 0,
[IPSTATS_MIB_OUTDISCARDS] = 0, [IPSTATS_MIB_OUTNOROUTES] = 0,
[IPSTATS_MIB_REASMTIMEOUT] = 0, [IPSTATS_MIB_REASMREQDS] = 0,
[IPSTATS_MIB_REASMOKS] = 0, [IPSTATS_MIB_REASMFAILS] = 0,
[IPSTATS_MIB_FRAGOKS] = 0, [IPSTATS_MIB_FRAGFAILS] = 0,
[IPSTATS_MIB_FRAGCREATES] = 0, [IPSTATS_MIB_INMCASTPKTS] = 0,
[IPSTATS_MIB_OUTMCASTPKTS] = 0, [IPSTATS_MIB_INBCASTPKTS] = 0,
[IPSTATS_MIB_OUTBCASTPKTS] = 0, [IPSTATS_MIB_INMCASTOCTETS] = 0,
[IPSTATS_MIB_OUTMCASTOCTETS] = 0, [IPSTATS_MIB_INBCASTOCTETS] = 0,
[IPSTATS_MIB_OUTBCASTOCTETS] = 0, [IPSTATS_MIB_CSUMERRORS] = 0, ...]], [{nla_len=52,
nla_type=IFLA_INET6_ICMP6STATS}, [[ICMP6_MIB_NUM] = 6, [ICMP6_MIB_INMSGS] = 0,
[ICMP6_MIB_INERRORS] = 0, [ICMP6_MIB_OUTMSGS] = 0, [ICMP6_MIB_OUTERRORS] = 0,
[ICMP6_MIB_CSUMERRORS] = 0]], [{nla_len=20, nla_type=IFLA_INET6_TOKEN},
inet_pton(AF_INET6, "::")], [{nla_len=5, nla_type=IFLA_INET6_ADDR_GEN_MODE},
IN6_ADDR_GEN_MODE_NONE]]]]], [{nla_len=17, nla_type=IFLA_PARENT_DEV_NAME}, "0000:03:00.0"], ...]]], iov_len=4096}], msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 2844

    20387 recvmsg(10, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000},

                                    …)

    20387 recvmsg(10, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, msg_namelen=12, msg_iov=[{iov_base=[{nlmsg_len=20, nlmsg_type=NLMSG_DONE, nlmsg_flags=NLM_F_MULTI, nlmsg_seq=1704658042, nlmsg_pid=20387}, 0], iov_len=4096}], msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 20

    20387 sendto(10, [{nlmsg_len=20, nlmsg_type=RTM_GETADDR, nlmsg_flags=NLM_F_REQUEST|NLM_F_DUMP, nlmsg_seq=1704658043, nlmsg_pid=0}, {ifa_family=AF_UNSPEC, ...}], 20, 0, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12) = 20

    20387 recvmsg(10, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000},

                                    …)

    20387 recvmsg(10, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000},

                                    …)

    20387 recvmsg(10, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, msg_namelen=12, msg_iov=[{iov_base=[{nlmsg_len=20, nlmsg_type=NLMSG_DONE, nlmsg_flags=NLM_F_MULTI, nlmsg_seq=1704658043, nlmsg_pid=20387}, 0], iov_len=4096}], msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 20

    20387 close(10)                    = 0

    20387 socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC, IPPROTO_TCP) = 10

    20387 setsockopt(10, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0

    20387 bind(10, {sa_family=AF_INET, sin_port=htons(5555), sin_addr=inet_addr("127.0.0.1")}, 16) = 0

    20387 listen(10, 100)                    = 0

    20387 getsockname(10, {sa_family=AF_INET, sin_port=htons(5555), sin_addr=inet_addr("127.0.0.1")}, [128 => 16]) = 0

    20387 getsockname(10, {sa_family=AF_INET, sin_port=htons(5555), sin_addr=inet_addr("127.0.0.1")}, [128 => 16]) = 0

    20387 getpid()                    = 20387

    20387 write(6, "\1\0\0\0\0\0\0\0", 8)   = 8

    20387 getpid( <unfinished ...>

    20390 <... epoll_wait resumed>[{events=EPOLLIN, data={u32=1509037216, u64=94507969418400}}], 256, -1) = 1
```

```
20387 <... getpid resumed>)            = 20387

20387 write(8, "\1\0\0\0\0\0\0\0", 8 <unfinished ...>

20390 getpid( <unfinished ...>

20387 <... write resumed>)             = 8

20390 <... getpid resumed>)            = 20387

20387 getpid( <unfinished ...>

20390 poll([{fd=6, events=POLLIN}], 1, 0 <unfinished ...>

20387 <... getpid resumed>)            = 20387

20390 <... poll resumed>)              = 1 ([{fd=6, revents=POLLIN}])

20387 poll([{fd=9, events=POLLIN}], 1, 0 <unfinished ...>

20390 getpid( <unfinished ...>

20387 <... poll resumed>)              = 0 (Timeout)

20390 <... getpid resumed>)            = 20387

20387 socket(AF_NETLINK, SOCK_RAW|SOCK_CLOEXEC, NETLINK_ROUTE <unfinished
...>

20390 read(6,  <unfinished ...>

20387 <... socket resumed>)            = 11

20390 <... read resumed>"\1\0\0\0\0\0\0\0", 8) = 8

20387 bind(11, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12) = 0

20387 getsockname(11, {sa_family=AF_NETLINK, nl_pid=20387, nl_groups=00000000}, [12]) =
0

20387 sendto(11, [{nlmsg_len=20, nlmsg_type=RTM_GETLINK,
nlmsg_flags=NLM_F_REQUEST|NLM_F_DUMP, nlmsg_seq=1704658042, nlmsg_pid=0},
{ifi_family=AF_UNSPEC, ...}], 20, 0, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12
<unfinished ...>

20390 epoll_ctl(7, EPOLL_CTL_ADD, 10, {events=0, data={u32=3154119536,
u64=140014793001840}} <unfinished ...>

20387 <... sendto resumed>)            = 20

20390 <... epoll_ctl resumed>)         = 0

20387 recvmsg(11,  <unfinished ...>

20390 epoll_ctl(7, EPOLL_CTL_MOD, 10, {events=EPOLLIN, data={u32=3154119536,
u64=140014793001840}}) = 0
```

```
20387 <... recvmsg resumed>{msg_name={sa_family=AF_NETLINK, nl_pid=0,
                                    …)

20390 getpid( <unfinished ...>

20387 recvmsg(11,  <unfinished ...>

20390 <... getpid resumed>)            = 20387

20387 <... recvmsg resumed>{msg_name={sa_family=AF_NETLINK, nl_pid=0,
                                    …)

20390 poll([{fd=6, events=POLLIN}], 1, 0 <unfinished ...>

20387 recvmsg(11,  <unfinished ...>

20390 <... poll resumed>)              = 0 (Timeout)

20387 <... recvmsg resumed>{msg_name={sa_family=AF_NETLINK, nl_pid=0,
nl_groups=00000000}, msg_namelen=12, msg_iov=[{iov_base=[{nlmsg_len=20,
nlmsg_type=NLMSG_DONE, nlmsg_flags=NLM_F_MULTI, nlmsg_seq=1704658042,
nlmsg_pid=20387}, 0], iov_len=4096}], msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 20

20390 epoll_wait(7,  <unfinished ...>

20387 sendto(11, [{nlmsg_len=20, nlmsg_type=RTM_GETADDR,
nlmsg_flags=NLM_F_REQUEST|NLM_F_DUMP, nlmsg_seq=1704658043, nlmsg_pid=0},
{ifa_family=AF_UNSPEC, ...}], 20, 0, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000},
12) = 20

20387 recvmsg(11, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000},
                                    …)

20387 recvmsg(11, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000},
                                    …)

20387 recvmsg(11, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000},
msg_namelen=12, msg_iov=[{iov_base=[{nlmsg_len=20, nlmsg_type=NLMSG_DONE,
nlmsg_flags=NLM_F_MULTI, nlmsg_seq=1704658043, nlmsg_pid=20387}, 0], iov_len=4096}],
msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 20

20387 close(11)                        = 0

20387 socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC, IPPROTO_TCP) = 11

20387 setsockopt(11, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0

20387 bind(11, {sa_family=AF_INET, sin_port=htons(5556), sin_addr=inet_addr("127.0.0.1")},
16) = 0

20387 listen(11, 100)                  = 0
```

```
        20387 getsockname(11, {sa_family=AF_INET, sin_port=htons(5556),
sin_addr=inet_addr("127.0.0.1")}, [128 => 16]) = 0

        20387 getsockname(11, {sa_family=AF_INET, sin_port=htons(5556),
sin_addr=inet_addr("127.0.0.1")}, [128 => 16]) = 0

        20387 getpid()                    = 20387

        20387 write(6, "\1\0\0\0\0\0\0\0", 8)   = 8

        20390 <... epoll_wait resumed>[{events=EPOLLIN, data={u32=1509037216,
u64=94507969418400}}], 256, -1) = 1

        20387 getpid()                    = 20387

        20390 getpid( <unfinished ...>

        20387 write(9, "\1\0\0\0\0\0\0\0", 8 <unfinished ...>

        20390 <... getpid resumed>)          = 20387

        20387 <... write resumed>)           = 8

        20390 poll([{fd=6, events=POLLIN}], 1, 0) = 1 ([{fd=6, revents=POLLIN}])

        20387 newfstatat(1, "",  <unfinished ...>

        20390 getpid( <unfinished ...>

        20387 <... newfstatat resumed>{st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...},
AT_EMPTY_PATH) = 0

        20390 <... getpid resumed>)          = 20387

        20387 write(1, "Welcome in our programm! This is"..., 56 <unfinished ...>

        20390 read(6,  <unfinished ...>

        20387 <... write resumed>)           = 56

        20390 <... read resumed>"\1\0\0\0\0\0\0\0", 8) = 8

        20387 write(1, " - create 'id' \n", 16 <unfinished ...>

        20390 epoll_ctl(7, EPOLL_CTL_ADD, 11, {events=0, data={u32=3154119568,
u64=140014793001872}} <unfinished ...>

        20387 <... write resumed>)           = 16

        20390 <... epoll_ctl resumed>)        = 0

        20387 write(1, " - exec 'id' 'command(start/stop"..., 41 <unfinished ...>

        20390 epoll_ctl(7, EPOLL_CTL_MOD, 11, {events=EPOLLIN, data={u32=3154119568,
u64=140014793001872}} <unfinished ...>

        20387 <... write resumed>)           = 41
```

```
20390 <... epoll_ctl resumed>)          = 0
20387 write(1, " - heartbit 'time' (in ms) \n", 28 <unfinished ...>
20390 getpid( <unfinished ...>
20387 <... write resumed>)              = 28
20390 <... getpid resumed>)             = 20387
20387 write(1, " - draw\n", 8 <unfinished ...>
20390 poll([{fd=6, events=POLLIN}], 1, 0 <unfinished ...>
20387 <... write resumed>)              = 8
20390 <... poll resumed>)               = 0 (Timeout)
20387 write(1, "Or enter q or ^D to exit\n", 25 <unfinished ...>
20390 epoll_wait(7,  <unfinished ...>
20387 <... write resumed>)              = 25
20387 write(1, "Enter you command: \n", 20) = 20
20387 write(1, " ->", 3)                = 3
20387 newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...},
 AT_EMPTY_PATH) = 0
20387 read(0, "create 2\n", 1024)       = 9
20387 clone(child_stack=NULL,
 flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
 child_tidptr=0x7f57c3833c90) = 20397
20397 set_robust_list(0x7f57c3833ca0, 24 <unfinished ...>
20387 write(1, "Ok: 20397\n", 10 <unfinished ...>
20397 <... set_robust_list resumed>)    = 0
20387 <... write resumed>)              = 10
20387 write(1, " ->", 3)                = 3
20387 read(0,  <unfinished ...>
20397 execve("./node", ["./node", "2"], 0x7ffe4ad488a8 /* 56 vars */) = 0
20397 arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd7f3e2270) = -1 EINVAL (Invalid argument)
20397 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
20397 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
```

```
20397 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=81715, ...}, AT_EMPTY_PATH) = 0
20397 close(3)                    = 0
20397 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libzmq.so.5", O_RDONLY|O_CLOEXEC) = 3
20397 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240\233\1\0\0\0\0\0"..., 832) = 832
20397 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=634936, ...}, AT_EMPTY_PATH) = 0
20397 close(3)                    = 0
```

*...Где фулл? А фулл лежит в логе...*

```
20389 poll([{fd=4, events=POLLIN}], 1, 0 <unfinished ...>
20387 write(6, "\1\0\0\0\0\0\0\0", 8 <unfinished ...>
20389 <... poll resumed>)         = 0 (Timeout)
20387 <... write resumed>)        = 8
20390 <... epoll_wait resumed>[{events=EPOLLIN, data={u32=1509037216, u64=94507969418400}}], 256, -1) = 1
20387 futex(0x7f57c27b5910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 20390, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>
20389 rt_sigprocmask(SIG_BLOCK, ~[RT_1],  <unfinished ...>
20390 getpid( <unfinished ...>
20389 <... rt_sigprocmask resumed>NULL, 8) = 0
20390 <... getpid resumed>)       = 20387
20389 madvise(0x7f57c27b6000, 8368128, MADV_DONTNEED <unfinished ...>
20390 poll([{fd=6, events=POLLIN}], 1, 0 <unfinished ...>
20389 <... madvise resumed>)      = 0
20390 <... poll resumed>)         = 1 ([{fd=6, revents=POLLIN}])
20389 exit(0 <unfinished ...>
20390 getpid( <unfinished ...>
20389 <... exit resumed>)         = ?
20390 <... getpid resumed>)       = 20387
20390 read(6,  <unfinished ...>
20389 +++ exited with 0 +++

20390 <... read resumed>"\1\0\0\0\0\0\0\0", 8) = 8

20390 epoll_ctl(7, EPOLL_CTL_DEL, 6, 0x55f459f214a4) = 0

20390 getpid()                    = 20387

20390 poll([{fd=6, events=POLLIN}], 1, 0) = 0 (Timeout)

20390 rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0

20390 madvise(0x7f57c1fb5000, 8368128, MADV_DONTNEED) = 0

20390 exit(0)                     = ?

20387 <... futex resumed>)            = 0

20390 +++ exited with 0 +++

20387 close(7)                    = 0

20387 close(6)                    = 0

20387 close(5)                    = 0

20387 close(4)                    = 0

20387 close(3)                    = 0

20387 lseek(0, -1, SEEK_CUR)          = -1 ESPIPE (Illegal seek)

20387 exit_group(0)                 = ?

20387 +++ exited with 0 +++

20399 <... epoll_wait resumed> <unfinished ...>) = ?

20398 <... epoll_wait resumed> <unfinished ...>) = ?

20397 <... poll resumed> <unfinished ...>) = ?

20399 +++ killed by SIGKILL +++

20398 +++ killed by SIGKILL +++

20404 <... epoll_wait resumed> <unfinished ...>) = ?

20397 +++ killed by SIGKILL +++

20403 <... epoll_wait resumed> <unfinished ...>) = ?

20404 +++ killed by SIGKILL +++

20402 <... poll resumed> <unfinished ...>) = ?

20403 +++ killed by SIGKILL +++

20402 +++ killed by SIGKILL +++
```

## Вывод

Эта лабораторная работа фактически завершает наше знакомство с курсом ОС и в принципе с темой IPC. И завершает его, знакомя нас с последним и самым популярным способом передачи данных между разными процессами - сокетами, в нашем случае в обертке очередей сообщений.

Честно скажу, что эта лабораторная мне очень понравилась, т.к. было поставлено интересное ТЗ и предоставлен достаточно простой в освоении, но при этом очень мощный инструмент в видео очередей сообщений.

В итоге я научился работать с очередями сообщений, проникся этой темой и смог написать исправно работающий код, поэтому считаю, что с поставленной задачей справился успешно.