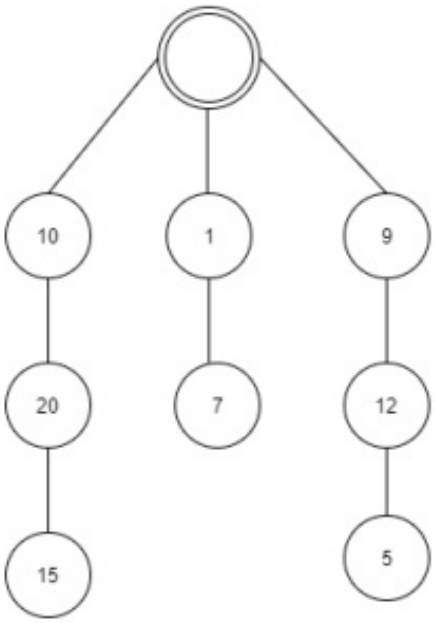
Лабораторная работа №6-8 по курсу "Операционные системы"

Студент группы: М80-207Б-21, Крючков Артемий Владимирович
Контакты: artemkr2003@mail.ru
Работа выполнена: 17.09.2022
Преподаватель: Миронов Евгений Сергеевич

Задание

Вариант 12 1 4 2

Топология 1



Все вычислительные узлы находятся в списке. Есть только один управляющий узел. Чтобы добавить новый вычислительный узел к управляющему, то необходимо выполнить команду: create id -1.

Набора команд 4 (поиск подстроки в строке)

Формат команды:

> exec id > text_string > pattern_string [result] – номера позиций, где найден образец, разделенный точкой с запятой

text_string — текст, в котором искать образец. Алфавит: [A-Za-z0-9]. Максимальная длина строки 10^8 символов

pattern_string — образец

Пример:

> exec 10

> abracadabra

> abra

Ok:10:0;7

> exec 10

> abracadabra

> mmm

Ok:10: -1

Примечания: Выбор алгоритма поиска не важен

Команда проверки 2

Формат команды: ping id

Команда проверяет доступность конкретного узла. Если узла нет, то необходимо выводить ошибку: «Error: Not found»

Пример:

> ping 10

Ok: 1 // узел 10 доступен

> ping 17

Ok: 0 // узел 17 недоступен

Тесты

```
[crewch@pc src]$ ./control
create 1 -1
OK: 2534
create 2 -1
OK: 2555
create 3 -1
OK: 2575
create 4 -1
OK: 2582
create 5 -1
OK: 2588
exec 1 ac acacsssssac
OK
exec 2 af afafafaffafa
OK
exec 3
123 123123123
OK
ping 1
OK: 1
ping 2
OK: 1
ping 3
OK: 1
ping 4
OK: 1
ping 5
OK: 1
back 1
OK: 2534 : 0, 2, 9
back 2
OK: 2555 : 0, 2, 4, 6, 9
back 3
OK: 2575 : 0, 3, 6
back 4
Error: No calculations to back
remove 5
OK
remove 4
OK
remove 3
OK
remove 2
OK
remove 1
OK
```

Методы и алгоритмы решения

control.cpp

```cpp
#include <unistd.h>
#include <vector>

#include "topology.hpp"
#include "zmq_std.hpp"

using node_id_type = long long;

int main()
{
    int rc;
    topology_t<node_id_type> control_node;
    std::vector<std::pair<void*, void*> > childs;

    std::string s;
    node_id_type id;

    while (std::cin >> s >> id) {
        if (s == "create") {
            node_id_type parent_id;
            std::cin >> parent_id;

            if (parent_id == -1) {
                void* new_context = NULL;
                void* new_socket = NULL;
                zmq_std::init_pair_socket(new_context, new_socket);
                rc = zmq_bind(new_socket, ("tcp://*:" +
std::to_string(PORT_BASE + id)).c_str());
                assert(rc == 0);

                int fork_id = fork();
                if (fork_id == 0) {
                    rc = execl(NODE_EXECUTABLE_NAME, NODE_EXECUTABLE_NAME,
std::to_string(id).c_str(), NULL);
                    assert(rc != -1);
                    return 0;
                } else {
                    bool ok = true;
                    node_token_t reply_info({fail, id, id});
                    ok = zmq_std::recieve_msg_wait(reply_info, new_socket);

                    node_token_t* token = new node_token_t({ping, id, id});
                    node_token_t reply({fail, id, id});
                    ok = zmq_std::send_recieve_wait(token, reply, new_socket);

                    if (ok and reply.action == success) {
                        childs.push_back(std::make_pair(new_context,
new_socket));

                        control_node.insert(id);
                        std::cout << "OK: " << reply_info.id << std::endl;
                    } else {
```

```
                    rc = zmq_close(new_socket);
                    assert(rc == 0);
                    rc = zmq_ctx_term(new_context);
                    assert(rc == 0);
                }
            }
        } else if (control_node.find(parent_id) == -1) {
            std::cout << "Error: Not found" << std::endl;
        } else {
            if (control_node.find(id) != -1) {
                std::cout << "Error: Already exists" << std::endl;
            } else {
                int ind = control_node.find(parent_id);
                node_token_t* token = new node_token_t({create, parent_id,
id});
                node_token_t reply({fail, id, id});

                if (zmq_std::send_recieve_wait(token, reply,
childs[ind].second) and reply.action == success) {
                    std::cout << "OK: " << reply.id << std::endl;
                    control_node.insert(parent_id, id);
                } else {
                    std::cout << "Error: Parent is unavailable" <<
std::endl;
                }
            }
        }
    } else if (s == "remove") {
        int ind = control_node.find(id);

        if (ind != -1) {
            node_token_t* token = new node_token_t({destroy, id, id});
            node_token_t reply({fail, id, id});
            bool ok = zmq_std::send_recieve_wait(token, reply,
childs[ind].second);

    if (reply.action == destroy and reply.parent_id == id) {
                rc = zmq_close(childs[ind].second);
                assert(rc == 0);
                rc = zmq_ctx_term(childs[ind].first);
                assert(rc == 0);
                std::vector<std::pair<void*, void*> >::iterator it =
childs.begin();

     while (ind--) {
                    ++it;
                }

                childs.erase(it);
            } else if (reply.action == bind and reply.parent_id == id) {
                rc = zmq_close(childs[ind].second);
                assert(rc == 0);
```

```
                    rc = zmq_ctx_term(childs[ind].first);
                    assert(rc == 0);
                    zmq_std::init_pair_socket(childs[ind].first,
childs[ind].second);
                    rc = zmq_bind(childs[ind].second, ("tcp://*:" +
std::to_string(PORT_BASE + reply.id)).c_str());
                    assert(rc == 0);
                }

    if (ok) {
                    control_node.erase(id);
                    std::cout << "OK" << std::endl;
                } else {
                    std::cout << "Error: Node is unavailable" << std::endl;
                }
            } else {
                std::cout << "Error: Not found" << std::endl;
            }
        } else if (s == "ping") {
            int ind = control_node.find(id);

   if (ind != -1) {
                node_token_t* token = new node_token_t({ping, id, id});
                node_token_t reply({fail, id, id});

    if (zmq_std::send_recieve_wait(token, reply, childs[ind].second) and
reply.action == success) {
                    std::cout << "OK: 1" << std::endl;
                } else {
                    std::cout << "OK: 0" << std::endl;
                }
            } else {
                std::cout << "Error: Not found" << std::endl;
            }
        } else if (s == "back") {
            int ind = control_node.find(id);

            if (ind != -1) {
                node_token_t* token = new node_token_t({back, id, id});
                node_token_t reply({fail, id, id});

                if (zmq_std::send_recieve_wait(token, reply,
childs[ind].second)) {
                    if (reply.action == success) {
                        node_token_t* token_back = new node_token_t({back, id,
id});
                        node_token_t reply_back({fail, id, id});
                        std::vector<unsigned int> calculated;

                        while (zmq_std::send_recieve_wait(token_back,
reply_back, childs[ind].second) and reply_back.action == success) {
```

```
                            calculated.push_back(reply_back.id);
                        }

                        if (calculated.empty()) {
                            std::cout << "OK: " << reply.id << " : -1" <<
std::endl;
                        } else {
                            std::cout << "OK: " << reply.id << " : ";

                            for (size_t i = 0; i < calculated.size() - 1; ++i)
{
                                std::cout << calculated[i] << ", ";
                            }

                            std::cout << calculated.back() << std::endl;
                        }
                    } else {
                        std::cout << "Error: No calculations to back" <<
std::endl;
                    }
                } else {
                    std::cout << "Error: Node is unavailable" << std::endl;
                }
            } else {
                std::cout << "Error: Not found" << std::endl;
            }
        } else if (s == "exec") {
            std::string pattern, text;
            std::cin >> pattern >> text;
            int ind = control_node.find(id);

            if (ind != -1) {
                bool ok = true;
                std::string text_pattern = pattern + SENTINEL + text +
SENTINEL;

                for (size_t i = 0; i < text_pattern.size(); ++i) {
                    node_token_t* token = new node_token_t({exec,
text_pattern[i], id});
                    node_token_t reply({fail, id, id});

                    if (!zmq_std::send_recieve_wait(token, reply,
childs[ind].second) or reply.action != success) {
                        ok = false;
                        break;
                    }
                }

                if (ok) {
                    std::cout << "OK" << std::endl;
                } else {
```

```
                    std::cout << "Error: Node is unavailable" << std::endl;
                }
            } else {
                std::cout << "Error: Not found" << std::endl;
            }
        }
    }

    for (size_t i = 0; i < childs.size(); ++i) {
        rc = zmq_close(childs[i].second);
        assert(rc == 0);
        rc = zmq_ctx_term(childs[i].first);
        assert(rc == 0);
    }
}
```

calculation_node.cpp

```
#include <pthread.h>
#include <unistd.h>
#include <list>
#include <queue>
#include <tuple>

#include "search.hpp"
#include "zmq_std.hpp"

const std::string SENTINEL_STR = "$";

long long node_id;
pthread_mutex_t mutex;
pthread_cond_t cond;
std::queue<std::pair<std::string, std::string> > calc_queue;
std::queue<std::list<unsigned int> > done_queue;

void* thread_func(void*)
{
    while (1) {
        pthread_mutex_lock(&mutex);

        while (calc_queue.empty()) {
            pthread_cond_wait(&cond, &mutex);
        }

        std::pair<std::string, std::string> cur = calc_queue.front();
        calc_queue.pop();
        pthread_mutex_unlock(&mutex);

        if (cur.first == SENTINEL_STR and cur.second == SENTINEL_STR) {
            break;
        } else {
            std::vector<unsigned int> res = KMPStrong(cur.first, cur.second);
```

```
            std::list<unsigned int> res_list;

            for (const unsigned int& elem : res) {
                res_list.push_back(elem);
            }

            pthread_mutex_lock(&mutex);
            done_queue.push(res_list);
            pthread_mutex_unlock(&mutex);
        }
    }
    return NULL;
}

int main(int argc, char** argv)
{
    int rc;
    assert(argc == 2);
    node_id = std::stoll(std::string(argv[1]));

    void* node_parent_context = zmq_ctx_new();
    void* node_parent_socket = zmq_socket(node_parent_context, ZMQ_PAIR);
    rc = zmq_connect(node_parent_socket, ("tcp://localhost:" +
std::to_string(PORT_BASE + node_id)).c_str());
    assert(rc == 0);

    long long child_id = -1;
    void* node_context = NULL;
    void* node_socket = NULL;

    pthread_t calculation_thread;
    rc = pthread_mutex_init(&mutex, NULL);
    assert(rc == 0);
    rc = pthread_cond_init(&cond, NULL);
    assert(rc == 0);
    rc = pthread_create(&calculation_thread, NULL, thread_func, NULL);
    assert(rc == 0);

    std::string pattern, text;
    bool flag_sentinel = true;

    node_token_t* info_token = new node_token_t({info, getpid(), getpid()});
    zmq_std::send_msg_dontwait(info_token, node_parent_socket);

    std::list<unsigned int> cur_calculated;

    bool has_child = false;
    bool awake = true;
    bool calc = true;

    while (awake) {
        node_token_t token;
```

```
        zmq_std::recieve_msg(token, node_parent_socket);

        node_token_t* reply = new node_token_t({fail, node_id, node_id});

        if (token.action == back) {
            if (token.id == node_id) {
                if (calc) {
                    pthread_mutex_lock(&mutex);

                    if (done_queue.empty()) {
                        reply->action = exec;
                    } else {
                        cur_calculated = done_queue.front();
                        done_queue.pop();
                        reply->action = success;
                        reply->id = getpid();
                    }

                    pthread_mutex_unlock(&mutex);
                    calc = false;
                } else {
                    if (cur_calculated.size() > 0) {
                        reply->action = success;
                        reply->id = cur_calculated.front();
                        cur_calculated.pop_front();
                    } else {
                        reply->action = exec;
                        calc = true;
                    }
                }
            } else {
                node_token_t* token_down = new node_token_t(token);
                node_token_t reply_down(token);
                reply_down.action = fail;

                if (zmq_std::send_recieve_wait(token_down, reply_down,
node_socket) and reply_down.action == success) {
                    *reply = reply_down;
                }
            }
        } else if (token.action == bind and token.parent_id == node_id) {
            // Привязка может быть получена при создании родительского узла и
этот узел должен привязываться к дочернему узлу родителя
            zmq_std::init_pair_socket(node_context, node_socket);
            rc = zmq_bind(node_socket, ("tcp://*:" + std::to_string(PORT_BASE +
token.id)).c_str());
            assert(rc == 0);
            has_child = true;
            child_id = token.id;
            node_token_t* token_ping = new node_token_t({ping, child_id,
child_id});
            node_token_t reply_ping({fail, child_id, child_id});
```

```
   if (zmq_std::send_recieve_wait(token_ping, reply_ping, node_socket) and
reply_ping.action == success) {
                reply->action = success;
            }
        } else if (token.action == create) {
            if (token.parent_id == node_id) {
                if (has_child) {
                    rc = zmq_close(node_socket);
                    assert(rc == 0);
                    rc = zmq_ctx_term(node_context);
                    assert(rc == 0);
                }

                zmq_std::init_pair_socket(node_context, node_socket);
                rc = zmq_bind(node_socket, ("tcp://*:" +
std::to_string(PORT_BASE + token.id)).c_str());
                assert(rc == 0);

                int fork_id = fork();
                if (fork_id == 0) {
                    rc = execl(NODE_EXECUTABLE_NAME, NODE_EXECUTABLE_NAME,
std::to_string(token.id).c_str(), NULL);
                    assert(rc != -1);
                    return 0;
                } else {
                    bool ok = true;
                    node_token_t reply_info({fail, token.id, token.id});
                    ok = zmq_std::recieve_msg_wait(reply_info, node_socket);

                    if (reply_info.action != fail) {
                        reply->id = reply_info.id;
                        reply->parent_id = reply_info.parent_id;
                    }

                    if (has_child) {
                        node_token_t* token_bind = new node_token_t({bind,
token.id, child_id});
                        node_token_t reply_bind({fail, token.id, token.id});
                        ok = zmq_std::send_recieve_wait(token_bind, reply_bind,
node_socket);
                        ok = ok and (reply_bind.action == success);
                    }

                    if (ok) {  // Специальное сообщение для
родителейподключился ли дочерний элем к узлу
                        node_token_t* token_ping = new node_token_t({ping,
token.id, token.id});
                        node_token_t reply_ping({fail, token.id, token.id});
                        ok = zmq_std::send_recieve_wait(token_ping, reply_ping,
node_socket);
                        ok = ok and (reply_ping.action == success);
```

```
                    if (ok) {
                        reply->action = success;
                        child_id = token.id;
                        has_child = true;
                    } else {
                        rc = zmq_close(node_socket);
                        assert(rc == 0);
                        rc = zmq_ctx_term(node_context);
                        assert(rc == 0);
                    }
                }
            }
        } else if (has_child) {
            node_token_t* token_down = new node_token_t(token);
            node_token_t reply_down(token);
            reply_down.action = fail;

            if (zmq_std::send_recieve_wait(token_down, reply_down,
node_socket) and reply_down.action == success) {
                *reply = reply_down;
            }
        }
    } else if (token.action == ping) {
        if (token.id == node_id) {
            reply->action = success;
        } else if (has_child) {
            node_token_t* token_down = new node_token_t(token);
            node_token_t reply_down(token);
            reply_down.action = fail;

            if (zmq_std::send_recieve_wait(token_down, reply_down,
node_socket) and reply_down.action == success) {
                *reply = reply_down;
            }
        }
    } else if (token.action == destroy) {
        if (has_child) {
            if (token.id == child_id) {
                bool ok = true;
                node_token_t* token_down = new node_token_t({destroy,
node_id, child_id});
                node_token_t reply_down({fail, child_id, child_id});
                ok = zmq_std::send_recieve_wait(token_down, reply_down,
node_socket);
                // Мы должны получить специальный ответ от дочернего

                if (reply_down.action == destroy and reply_down.parent_id
== child_id) {
                    rc = zmq_close(node_socket);
                    assert(rc == 0);
                    rc = zmq_ctx_term(node_context);
```

```
                        assert(rc == 0);
                        has_child = false;
                        child_id = -1;
                    } else if (reply_down.action == bind and
reply_down.parent_id == node_id) {
                        rc = zmq_close(node_socket);
                        assert(rc == 0);
                        rc = zmq_ctx_term(node_context);
                        assert(rc == 0);
                        zmq_std::init_pair_socket(node_context, node_socket);
                        rc = zmq_bind(node_socket, ("tcp://*:" +
std::to_string(PORT_BASE + reply_down.id)).c_str());
                        assert(rc == 0);
                        child_id = reply_down.id;
                        node_token_t* token_ping = new node_token_t({ping,
child_id, child_id});
                        node_token_t reply_ping({fail, child_id, child_id});

                        if (zmq_std::send_recieve_wait(token_ping, reply_ping,
node_socket) and reply_ping.action == success) {
                            ok = true;
                        }
                    }

                    if (ok) {
                        reply->action = success;
                    }
                } else if (token.id == node_id) {
                    rc = zmq_close(node_socket);
                    assert(rc == 0);
                    rc = zmq_ctx_term(node_context);
                    assert(rc == 0);
                    has_child = false;
                    reply->action = bind;
                    reply->id = child_id;
                    reply->parent_id = token.parent_id;
                    awake = false;
                } else {
                    node_token_t* token_down = new node_token_t(token);
                    node_token_t reply_down(token);
                    reply_down.action = fail;

                    if (zmq_std::send_recieve_wait(token_down, reply_down,
node_socket) and reply_down.action == success) {
                        *reply = reply_down;
                    }
                }
            } else if (token.id == node_id) {
                //  Специальное сообщение для родителя
                reply->action = destroy;
                reply->parent_id = node_id;
                reply->id = node_id;
```

```
                awake = false;
            }
        } else if (token.action == exec) {
            if (token.id == node_id) {
                char c = token.parent_id;

                if (c == SENTINEL) {
                    if (flag_sentinel) {
                        std::swap(text, pattern);
                    } else {
                        pthread_mutex_lock(&mutex);

                        if (calc_queue.empty()) {
                            pthread_cond_signal(&cond);
                        }

                        calc_queue.push({pattern, text});
                        pthread_mutex_unlock(&mutex);
                        text.clear();
                        pattern.clear();
                    }
                    flag_sentinel = flag_sentinel ^ 1;
                } else {
                    text = text + c;
                }
                reply->action = success;
            } else if (has_child) {
                node_token_t* token_down = new node_token_t(token);
                node_token_t reply_down(token);
                reply_down.action = fail;

    if (zmq_std::send_recieve_wait(token_down, reply_down, node_socket) and
reply_down.action == success) {
                    *reply = reply_down;
                }
            }
        }
        zmq_std::send_msg_dontwait(reply, node_parent_socket);
    }

    if (has_child) {
        rc = zmq_close(node_socket);
        assert(rc == 0);
        rc = zmq_ctx_term(node_context);
        assert(rc == 0);
    }

    rc = zmq_close(node_parent_socket);
    assert(rc == 0);
    rc = zmq_ctx_term(node_parent_context);
    assert(rc == 0);
```

```
    pthread_mutex_lock(&mutex);

    if (calc_queue.empty()) {
        pthread_cond_signal(&cond);
    }

    calc_queue.push({SENTINEL_STR, SENTINEL_STR});
    pthread_mutex_unlock(&mutex);

    rc = pthread_join(calculation_thread, NULL);
    assert(rc == 0);

    rc = pthread_cond_destroy(&cond);
    assert(rc == 0);
    rc = pthread_mutex_destroy(&mutex);
    assert(rc == 0);
}
```

search.cpp

```
#include "search.hpp"

std::vector<unsigned int> PrefixFunction(const std::string& s)
{
    unsigned int n = s.size();
    std::vector<unsigned int> p(n);

    for (unsigned int i = 1; i < n; ++i) {
        p[i] = p[i - 1];

        while (p[i] > 0 and s[i] != s[p[i]]) {
            p[i] = p[p[i] - 1];
        }

        if (s[i] == s[p[i]]) {
            ++p[i];
        }
    }
    return p;
}

std::vector<unsigned int> KMPWeak(const std::string& pattern, const
std::string& text)
{
    std::vector<unsigned int> p = PrefixFunction(pattern);
    unsigned int m = pattern.size();
    unsigned int n = text.size();
    unsigned int i = 0;
    std::vector<unsigned int> ans;

    if (m > n) {
        return ans;
```

```
    }

    while (i < n - m + 1) {
        unsigned int j = 0;

        while (j < m and pattern[j] == text[i + j]) {
            ++j;
        }

        if (j == m) {
            ans.push_back(i);
        } else {
            if (j > 0 and j > p[j - 1]) {
                i = i + j - p[j - 1] - 1;
            }
        }
        ++i;
    }
    return ans;
}

std::vector<unsigned int> ZFunction(const std::string& s)
{
    unsigned int n = s.size();
    std::vector<unsigned int> z(n);
    unsigned int l = 0, r = 0;

    for (unsigned int i = 1; i < n; ++i) {
        if (i <= r) {
            z[i] = std::min(z[i - l], r - i);
        }

        while (i + z[i] < n and s[i + z[i]] == s[z[i]]) {
            ++z[i];
        }

        if (i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }
    return z;
}

std::vector<unsigned int> StrongPrefixFunction(const std::string& s)
{
    std::vector<unsigned int> z = ZFunction(s);
    unsigned int n = s.size();
    std::vector<unsigned int> sp(n);

    for (unsigned int i = n - 1; i > 0; --i) {
        sp[i + z[i] - 1] = z[i];
```

```
    }

    return sp;
}

std::vector<unsigned int> KMPStrong(const std::string& pattern, const
std::string& text)
{
    std::vector<unsigned int> p = StrongPrefixFunction(pattern);
    unsigned int m = pattern.size();
    unsigned int n = text.size();
    unsigned int i = 0;
    std::vector<unsigned int> ans;

    if (m > n) {
        return ans;
    }

    while (i < n - m + 1) {
        unsigned int j = 0;

        while (j < m and pattern[j] == text[i + j]) {
            ++j;
        }

        if (j == m) {
            ans.push_back(i);
        } else {
            if (j > 0 and j > p[j - 1]) {
                i = i + j - p[j - 1] - 1;
            }
        }
        ++i;
    }
    return ans;
}
```

search.hpp

```
#ifndef SEARCH_HPP
#define SEARCH_HPP

#include <string>
#include <vector>

std::vector<unsigned int> PrefixFunction(const std::string& s);
std::vector<unsigned int> KMPWeak(const std::string& pattern, const
std::string& text);

std::vector<unsigned int> ZFunction(const std::string& s);
std::vector<unsigned int> StrongPrefixFunction(const std::string& s);
std::vector<unsigned int> KMPStrong(const std::string& pattern, const
std::string& text);

#endif
```

zmq_std.hpp

```
#ifndef ZMQ_STD_HPP
#define ZMQ_STD_HPP

#include <assert.h>
#include <errno.h>
#include <string.h>
#include <zmq.h>
#include <string>

const char* NODE_EXECUTABLE_NAME = "calculation";
const char SENTINEL = '









;
const int PORT_BASE = 8000;
const int WAIT_TIME = 1000;

enum actions_t {
    fail = 0,
    success = 1,
    create = 2,
    destroy = 3,
    bind = 4,
    ping = 5,
    exec = 6,
    info = 7,
    back = 8
};
```

```
struct node_token_t {
    actions_t action;
    long long parent_id, id;
};

namespace zmq_std {
    void init_pair_socket(void*& context, void*& socket)
    {
        int rc;
        context = zmq_ctx_new();
        socket = zmq_socket(context, ZMQ_PAIR);
        rc = zmq_setsockopt(socket, ZMQ_RCVTIMEO, &WAIT_TIME, sizeof(int));
        assert(rc == 0);
        rc = zmq_setsockopt(socket, ZMQ_SNDTIMEO, &WAIT_TIME, sizeof(int));
        assert(rc == 0);
    }

    template <class T>
    void recieve_msg(T& reply_data, void* socket)
    {
        int rc = 0;
        zmq_msg_t reply;
        zmq_msg_init(&reply);
        rc = zmq_msg_recv(&reply, socket, 0);
        assert(rc == sizeof(T));
        reply_data = *(T*)zmq_msg_data(&reply);
        rc = zmq_msg_close(&reply);
        assert(rc == 0);
    }

    template <class T>
    void send_msg(T* token, void* socket)
    {
        int rc = 0;
        zmq_msg_t message;
        zmq_msg_init(&message);
        rc = zmq_msg_init_size(&message, sizeof(T));
        assert(rc == 0);
        rc = zmq_msg_init_data(&message, token, sizeof(T), NULL, NULL);
        assert(rc == 0);
        rc = zmq_msg_send(&message, socket, 0);
        assert(rc == sizeof(T));
    }

    template <class T>
    bool send_msg_dontwait(T* token, void* socket)
    {
        int rc;
        zmq_msg_t message;
        zmq_msg_init(&message);
        rc = zmq_msg_init_size(&message, sizeof(T));
```

```
    assert(rc == 0);
    rc = zmq_msg_init_data(&message, token, sizeof(T), NULL, NULL);
    assert(rc == 0);
    rc = zmq_msg_send(&message, socket, ZMQ_DONTWAIT);

    if (rc == -1) {
        zmq_msg_close(&message);
        return false;
    }

    assert(rc == sizeof(T));
    return true;
}

template <class T>
bool recieve_msg_wait(T& reply_data, void* socket)
{
    int rc = 0;
    zmq_msg_t reply;
    zmq_msg_init(&reply);
    rc = zmq_msg_recv(&reply, socket, 0);

    if (rc == -1) {
        zmq_msg_close(&reply);
        return false;
    }

    assert(rc == sizeof(T));
    reply_data = *(T*)zmq_msg_data(&reply);
    rc = zmq_msg_close(&reply);
    assert(rc == 0);
    return true;
}

// Возвращает значение true, если T был успешно помещен в очередь сокета
template <class T>
bool send_msg_wait(T* token, void* socket)
{
    int rc;
    zmq_msg_t message;
    zmq_msg_init(&message);
    rc = zmq_msg_init_size(&message, sizeof(T));
    assert(rc == 0);
    rc = zmq_msg_init_data(&message, token, sizeof(T), NULL, NULL);
    assert(rc == 0);
    rc = zmq_msg_send(&message, socket, 0);

    if (rc == -1) {
        zmq_msg_close(&message);
        return false;
    }
```

```
        assert(rc == sizeof(T));
        return true;
    }

    // Возвращает значение true, если сокет успешно поставил сообщение в
очередь и получил ответ
    template <class T>
    bool send_recieve_wait(T* token_send, T& token_reply, void* socket)
    {
        if (send_msg_wait(token_send, socket)) {
            if (recieve_msg_wait(token_reply, socket)) {
                return true;
            } else {
                return false;
            }
        } else {
            return false;
        }
    }
}

#endif
```

topology.hpp

```
#ifndef TOPOLOGY_HPP
#define TOPOLOGY_HPP

#include <iostream>
#include <list>

template <class T>
class topology_t {
private:
    using list_type = std::list<std::list<T> >;
    using iterator = typename std::list<T>::iterator;
    using list_iterator = typename list_type::iterator;

    list_type container;
    size_t container_size;

public:
    explicit topology_t() noexcept
        : container(), container_size(0) {}
    ~topology_t() {}

    bool erase(const T& elem)
    {
        for (list_iterator it1 = container.begin(); it1 != container.end();
++it1) {
            for (iterator it2 = it1->begin(); it2 != it1->end(); ++it2) {
                if (*it2 == elem) {
```

```
                    if (it1->size() > 1) {
                        it1->erase(it2);
                    } else {
                        container.erase(it1);
                    }

                    --container_size;
                    return true;
                }
            }
        }
        return false;
    }

    long long find(const T& elem)
    {
        long long ind = 0;
        for (list_iterator it1 = container.begin(); it1 != container.end();
++it1) {
            for (iterator it2 = it1->begin(); it2 != it1->end(); ++it2) {
                if (*it2 == elem) {
                    return ind;
                }
            }
            ++ind;
        }
        return -1;
    }

    bool insert(const T& parent, const T& elem)
    {
        for (list_iterator it1 = container.begin(); it1 != container.end();
++it1) {
            for (iterator it2 = it1->begin(); it2 != it1->end(); ++it2) {
                if (*it2 == parent) {
                    it1->insert(++it2, elem);
                    ++container_size;
                    return true;
                }
            }
        }
        return false;
    }

    void insert(const T& elem)
    {
        std::list<T> new_list;
        new_list.push_back(elem);
        ++container_size;
        container.push_back(new_list);
    }
```

```
    size_t size()
    {
        return container_size;
    }

    template <class U>
    friend std::ostream& operator<<(std::ostream& of, const topology_t<U>& top)
    {
        for (auto it1 = top.container.begin(); it1 != top.container.end();
++it1) {
            of << "{";
            for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {
                of << *it2 << " ";
            }
            of << "}" << std::endl;
        }
        return of;
    }
};

#endif
```

Strace

```
[crewch@pc src]$ strace ./control
execve("./control", ["./control"], 0x7ffd7503ca30 /* 70 vars */) = 0
brk(NULL)                               = 0x5573b1d55000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdabc97d00) = -1 EINVAL (Недопустимый
аргумент)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или
каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=158627, ...}, AT_EMPTY_PATH) =
0
mmap(NULL, 158627, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f462196d000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libzmq.so.5", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 \200\1\0\0\0\0\0"...,
832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=743320, ...}, AT_EMPTY_PATH) =
0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f462196b000
mmap(NULL, 745560, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f46218b4000
mmap(0x7f46218cc000, 471040, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x18000) = 0x7f46218cc000
mmap(0x7f462193f000, 143360, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x8b000) = 0x7f462193f000
mmap(0x7f4621962000, 36864, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xad000) = 0x7f4621962000
close(3)                                = 0
```

```
openat(AT_FDCWD, "/usr/lib/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"...,
832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=19198496, ...}, AT_EMPTY_PATH)
= 0
mmap(NULL, 2320384, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f462167d000
mmap(0x7f4621716000, 1138688, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x99000) = 0x7f4621716000
mmap(0x7f462182c000, 487424, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1af000) = 0x7f462182c000
mmap(0x7f46218a3000, 57344, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x225000) = 0x7f46218a3000
mmap(0x7f46218b1000, 10240, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f46218b1000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"...,
832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=944600, ...}, AT_EMPTY_PATH) =
0
mmap(NULL, 946368, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f4621595000
mmap(0x7f46215a3000, 499712, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe000) = 0x7f46215a3000
mmap(0x7f462161d000, 385024, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x88000) = 0x7f462161d000
mmap(0x7f462167b000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe5000) = 0x7f462167b000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"...,
832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=571848, ...}, AT_EMPTY_PATH) =
0
mmap(NULL, 127304, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f4621575000
mmap(0x7f4621578000, 94208, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x3000) = 0x7f4621578000
mmap(0x7f462158f000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1a000) = 0x7f462158f000
mmap(0x7f4621593000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1d000) = 0x7f4621593000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P4\2\0\0\0\0\0"..., 832)
= 832
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
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1953472, ...}, AT_EMPTY_PATH)
= 0
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
mmap(NULL, 1994384, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
```

```
0x7f462138e000
mmap(0x7f46213b0000, 1421312, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7f46213b0000
mmap(0x7f462150b000, 356352, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x17d000) = 0x7f462150b000
mmap(0x7f4621562000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1d4000) = 0x7f4621562000
mmap(0x7f4621568000, 52880, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f4621568000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libsodium.so.23", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 \320\0\0\0\0\0\0"...,
832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=362968, ...}, AT_EMPTY_PATH) =
0
mmap(NULL, 365576, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f4621334000
mmap(0x7f4621341000, 233472, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xd000) = 0x7f4621341000
mmap(0x7f462137a000, 73728, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x46000) = 0x7f462137a000
mmap(0x7f462138c000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x57000) = 0x7f462138c000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libpgm-5.3.so.0", O_RDONLY|O_CLOEXEC) = 3
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
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=309872, ...}, AT_EMPTY_PATH) =
0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f4621332000
mmap(NULL, 304440, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f46212e7000
mprotect(0x7f46212eb000, 266240, PROT_NONE) = 0
mmap(0x7f46212eb000, 155648, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x4000) = 0x7f46212eb000
mmap(0x7f4621311000, 106496, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x2a000) = 0x7f4621311000
mmap(0x7f462132c000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x44000) = 0x7f462132c000
mmap(0x7f462132e000, 13624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f462132e000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/librt.so.1", O_RDONLY|O_CLOEXEC) = 3
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
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=14480, ...}, AT_EMPTY_PATH) =
0
mmap(NULL, 16400, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f46212e2000
mmap(0x7f46212e3000, 4096, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7f46212e3000
mmap(0x7f46212e4000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x2000) = 0x7f46212e4000
```

```
mmap(0x7f46212e5000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f46212e5000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\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
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=14416, ...}, AT_EMPTY_PATH) =
0
mmap(NULL, 16400, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f46212dd000
mmap(0x7f46212de000, 4096, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7f46212de000
mmap(0x7f46212df000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x2000) = 0x7f46212df000
mmap(0x7f46212e0000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f46212e0000
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f46212db000
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f46212d8000
arch_prctl(ARCH_SET_FS, 0x7f46212d8980) = 0
set_tid_address(0x7f46212d8c50)         = 11419
set_robust_list(0x7f46212d8c60, 24)     = 0
rseq(0x7f46212d92a0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f4621562000, 16384, PROT_READ) = 0
mprotect(0x7f46212e0000, 4096, PROT_READ) = 0
mprotect(0x7f46212e5000, 4096, PROT_READ) = 0
mprotect(0x7f462167b000, 4096, PROT_READ) = 0
mprotect(0x7f462132c000, 4096, PROT_READ) = 0
mprotect(0x7f462138c000, 4096, PROT_READ) = 0
mprotect(0x7f4621593000, 4096, PROT_READ) = 0
mprotect(0x7f46218a3000, 53248, PROT_READ) = 0
mprotect(0x7f4621962000, 32768, PROT_READ) = 0
mprotect(0x5573b1c8f000, 4096, PROT_READ) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f46212d6000
mprotect(0x7f46219c5000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f462196d000, 158627)          = 0
getrandom("\x3e\xa8\x7d\xf2\x7b\x2e\xd5\xcb", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x5573b1d55000
brk(0x5573b1d76000)                     = 0x5573b1d76000
futex(0x7f46218b16bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0
newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x4), ...},
AT_EMPTY_PATH) = 0
read(0, create 1 -1
"create 1 -1\n", 1024)          = 12
openat(AT_FDCWD, "/sys/devices/system/cpu/online", O_RDONLY|O_CLOEXEC) = 3
read(3, "0-11\n", 1024)                 = 5
close(3)                                = 0
```

```
openat(AT_FDCWD, "/sys/devices/system/cpu/possible", O_RDONLY|O_CLOEXEC) = 3
read(3, "0-31\n", 1024)                 = 5
close(3)                                = 0
getpid()                                = 11419
sched_getaffinity(11419, 128, [0 1 2 3 4 5 6 7 8 9 10 11]) = 40
newfstatat(AT_FDCWD, "/etc/nsswitch.conf", {st_mode=S_IFREG|0644, st_size=391,
...}, 0) = 0
newfstatat(AT_FDCWD, "/", {st_mode=S_IFDIR|0755, st_size=4096, ...}, 0) = 0
openat(AT_FDCWD, "/etc/nsswitch.conf", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=391, ...}, AT_EMPTY_PATH) = 0
read(3, "# Name Service Switch configurat"..., 4096) = 391
read(3, "", 4096)                       = 0
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=391, ...}, AT_EMPTY_PATH) = 0
close(3)                                = 0
openat(AT_FDCWD, "/etc/protocols", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=3196, ...}, AT_EMPTY_PATH) = 0
lseek(3, 0, SEEK_SET)                   = 0
read(3, "# Full data: /usr/share/iana-etc"..., 4096) = 3196
close(3)                                = 0
eventfd2(0, EFD_CLOEXEC)                = 3
fcntl(3, F_GETFL)                       = 0x2 (flags O_RDWR)
fcntl(3, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
fcntl(3, F_GETFL)                       = 0x802 (flags O_RDWR|O_NONBLOCK)
fcntl(3, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
getpid()                                = 11419
getpid()                                = 11419
getrandom("\x0b\xc1\x41\xcb\x16\x4c\x1a\xc6\xba\xb2\x63\x85\x28\x96\x97\xbd",
16, 0) = 16
getrandom("\x2e\x49\xdb\xb2\x74\x67\x8b\x5b\xfe\x26\xe3\x02\xb7\xd7\x2d\x16",
16, 0) = 16
eventfd2(0, EFD_CLOEXEC)                = 4
fcntl(4, F_GETFL)                       = 0x2 (flags O_RDWR)
fcntl(4, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
fcntl(4, F_GETFL)                       = 0x802 (flags O_RDWR|O_NONBLOCK)
fcntl(4, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
getpid()                                = 11419
epoll_create1(EPOLL_CLOEXEC)            = 5
epoll_ctl(5, EPOLL_CTL_ADD, 4, {events=0, data={u32=2983622144,
u64=93955393222144}}) = 0
epoll_ctl(5, EPOLL_CTL_MOD, 4, {events=EPOLLIN, data={u32=2983622144,
u64=93955393222144}}) = 0
getpid()                                = 11419
rt_sigaction(SIGRT_1, {sa_handler=0x7f4621411d00, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
sa_restorer=0x7f46213c6a00}, NULL, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f4620ad5000
mprotect(0x7f4620ad6000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SY
```

```
SVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f46212d5990, parent_tid=0x7f46212d5990, exit_signal=0,
stack=0x7f4620ad5000, stack_size=0x7ffd40, tls=0x7f46212d56c0} => {parent_tid=
[11444]}, 88) = 11444
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
eventfd2(0, EFD_CLOEXEC)                = 6
fcntl(6, F_GETFL)                       = 0x2 (flags O_RDWR)
fcntl(6, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
fcntl(6, F_GETFL)                       = 0x802 (flags O_RDWR|O_NONBLOCK)
fcntl(6, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
getpid()                                = 11419
epoll_create1(EPOLL_CLOEXEC)            = 7
epoll_ctl(7, EPOLL_CTL_ADD, 6, {events=0, data={u32=2983643744,
u64=93955393243744}}) = 0
epoll_ctl(7, EPOLL_CTL_MOD, 6, {events=EPOLLIN, data={u32=2983643744,
u64=93955393243744}}) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f46202d4000
mprotect(0x7f46202d5000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SY
SVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f4620ad4990, parent_tid=0x7f4620ad4990, exit_signal=0,
stack=0x7f46202d4000, stack_size=0x7ffd40, tls=0x7f4620ad46c0} => {parent_tid=
[11445]}, 88) = 11445
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
eventfd2(0, EFD_CLOEXEC)                = 8
fcntl(8, F_GETFL)                       = 0x2 (flags O_RDWR)
fcntl(8, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
fcntl(8, F_GETFL)                       = 0x802 (flags O_RDWR|O_NONBLOCK)
fcntl(8, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
getpid()                                = 11419
getpid()                                = 11419
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC, IPPROTO_TCP) = 9
setsockopt(9, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
bind(9, {sa_family=AF_INET, sin_port=htons(8001),
sin_addr=inet_addr("0.0.0.0")}, 16) = 0
listen(9, 100)                          = 0
getsockname(9, {sa_family=AF_INET, sin_port=htons(8001),
sin_addr=inet_addr("0.0.0.0")}, [128 => 16]) = 0
getsockname(9, {sa_family=AF_INET, sin_port=htons(8001),
sin_addr=inet_addr("0.0.0.0")}, [128 => 16]) = 0
getpid()                                = 11419
write(6, "\1\0\0\0\0\0\0\0", 8)         = 8
getpid()                                = 11419
write(8, "\1\0\0\0\0\0\0\0", 8)         = 8
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f46212d8c50) = 11446
getpid()                                = 11419
poll([{fd=8, events=POLLIN}], 1, 1000)  = 1 ([{fd=8, revents=POLLIN}])
getpid()                                = 11419
```

```
read(8, "\1\0\0\0\0\0\0\0", 8)          = 8
getpid()                                = 11419
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
getpid()                                = 11419
poll([{fd=8, events=POLLIN}], 1, 1000)  = 1 ([{fd=8, revents=POLLIN}])
getpid()                                = 11419
read(8, "\1\0\0\0\0\0\0\0", 8)          = 8
getpid()                                = 11419
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
getpid()                                = 11419
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
getpid()                                = 11419
write(6, "\1\0\0\0\0\0\0\0", 8)         = 8
getpid()                                = 11419
poll([{fd=8, events=POLLIN}], 1, 1000)  = 1 ([{fd=8, revents=POLLIN}])
getpid()                                = 11419
read(8, "\1\0\0\0\0\0\0\0", 8)          = 8
getpid()                                = 11419
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x4), ...},
AT_EMPTY_PATH) = 0
write(1, "OK: 11446\n", 10OK: 11446
)             = 10
read(0, ping 1
"ping 1\n", 1024)               = 7
getpid()                                = 11419
poll([{fd=8, events=POLLIN}], 1, 0)     = 1 ([{fd=8, revents=POLLIN}])
getpid()                                = 11419
read(8, "\1\0\0\0\0\0\0\0", 8)          = 8
getpid()                                = 11419
write(6, "\1\0\0\0\0\0\0\0", 8)         = 8
getpid()                                = 11419
poll([{fd=8, events=POLLIN}], 1, 0)     = 1 ([{fd=8, revents=POLLIN}])
getpid()                                = 11419
read(8, "\1\0\0\0\0\0\0\0", 8)          = 8
getpid()                                = 11419
write(6, "\1\0\0\0\0\0\0\0", 8)         = 8
getpid()                                = 11419
write(6, "\1\0\0\0\0\0\0\0", 8)         = 8
getpid()                                = 11419
write(6, "\1\0\0\0\0\0\0\0", 8)         = 8
getpid()                                = 11419
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
getpid()                                = 11419
write(6, "\1\0\0\0\0\0\0\0", 8)         = 8
getpid()                                = 11419
poll([{fd=8, events=POLLIN}], 1, 1000)  = 1 ([{fd=8, revents=POLLIN}])
getpid()                                = 11419
read(8, "\1\0\0\0\0\0\0\0", 8)          = 8
getpid()                                = 11419
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
```

```
write(1, "OK: 1\n", 6OK: 1
)                  = 6
read(0, exit
"exit\n", 1024)                 = 5
read(0, "", 1024)                       = 0
getpid()                                = 11419
write(4, "\1\0\0\0\0\0\0\0", 8)         = 8
getpid()                                = 11419
getpid()                                = 11419
write(8, "\1\0\0\0\0\0\0\0", 8)         = 8
futex(0x5573b1d676e8, FUTEX_WAKE_PRIVATE, 1) = 1
getpid()                                = 11419
poll([{fd=3, events=POLLIN}], 1, -1)    = 1 ([{fd=3, revents=POLLIN}])
getpid()                                = 11419
read(3, "\1\0\0\0\0\0\0\0", 8)          = 8
getpid()                                = 11419
write(6, "\1\0\0\0\0\0\0\0", 8)         = 8
futex(0x7f4620ad4990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 11445, NULL,
FUTEX_BITSET_MATCH_ANY) = -1 EAGAIN (Ресурс временно недоступен)
close(7)                                = 0
close(6)                                = 0
close(5)                                = 0
close(4)                                = 0
close(3)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```