Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа
по курсу «Операционные системы»
III Семестр

Задание 6
Вариант 1

Студент:	Синявский А.В
Группа:	М80-208Б-18
Преподаватель:	Миронов Е.С
Оценка:	
Дата:	

Москва 2019

1. Описание задания

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.

Вариант 1:
Топология – дерево общего вида
Команда – поиск подстроки в строке
Проверка доступности узлов - heartbit

2. Код программы

2.1 functions.h/cpp

```
#ifndef SRC_FUNCTIONS_H
#define SRC_FUNCTIONS_H

#include <string>
#include <zconf.h>
#include "zmq.hpp"

bool send_message(zmq::socket_t& socket, const std::string& message_string);

std::string recieve_message(zmq::socket_t& socket);

std::string get_port_name(int port);

int bind_socket(zmq::socket_t& socket);

void create_node(int id, int parent_id, int port);

#endif //SRC_FUNCTIONS_H
```

functions.cpp

```
#include "functions.h"

bool send_message(zmq::socket_t& socket, const std::string& message_string) {
    zmq::message_t message(message_string.size());
    memcpy(message.data(), message_string.c_str(), message_string.size());
    return socket.send(message);
}

std::string recieve_message(zmq::socket_t& socket) {
    zmq::message_t message;
    bool ok;
    try {
        ok = socket.recv(&message);
    } catch(...) {
        ok = false;
    }
    std::string recieved_msg(static_cast<char*>(message.data()), message.size());
    if (recieved_msg.empty() || !ok) {
        return "ERROR! Node is unavailable";
    }
    return recieved_msg;
}

std::string get_port_name(int port) {
    return "tcp://127.0.0.1:" + std::to_string(port);
}
int bind_socket(zmq::socket_t& socket) {
    int port = 30000;
    while (true) {
        try {
            socket.bind(get_port_name(port));
            break;
        } catch(...) {
            port++;
        }
    }
    return port;
}

void create_node(int id, int parent_id, int port) {
    char* arg1 = strdup((std::to_string(id)).c_str());
    char* arg2 = strdup((std::to_string(parent_id)).c_str());
    char* arg3 = strdup((std::to_string(port)).c_str());
    char* args[] = {"./compute_node", arg1, arg2, arg3, nullptr};
    execv("./compute_node", args);
}
```

2.2 control.node.cpp

```
#include <iostream>
#include "zmq.hpp"
#include <string>
#include <zconf.h>
#include <utility>
#include <vector>
#include <csignal>
#include <sstream>
#include <set>
#include <algorithm>
#include <memory>
#include <unordered_map>
#include "functions.h"

template <class T>
std::ostream& operator<<(std::ostream& os, std::vector<T> v) {
    for (const T& i : v) {
        os << i << " ";
    }
    return os;
}

struct TreeNode {
    TreeNode(int id, std::weak_ptr<TreeNode> parent) : id_(id), parent_(std::move(parent)) {}
    int id_;
    std::weak_ptr<TreeNode> parent_;
    std::unordered_map<int, std::shared_ptr<TreeNode>> nodes_;
};

class IdIndexingTree {
public:
    IdIndexingTree() = default;
    ~IdIndexingTree() = default;

    bool Insert(int elem, int parent_id) {
        if (root_ == nullptr) {
            root_ = std::make_shared<TreeNode>(elem, std::weak_ptr<TreeNode>());
            return true;
        }
        std::vector<int> path = GetPathTo(parent_id);
        if (path.empty()) {
            return false;
        }
        path.erase(path.begin());
        std::shared_ptr<TreeNode> node = root_;
        for (int i : path) {
            if (node->nodes_.count(i) == 0) {
                throw std::logic_error("Shit happened");
            }
```

```
            node = node->nodes_[i];
        }
        node->nodes_[elem] = std::make_shared<TreeNode>(elem,node);
        return true;
    }

    bool Erase(int elem) {
        std::vector<int> path = GetPathTo(elem);
        if (path.empty()) {
            return false;
        }
        path.erase(path.begin());
        std::shared_ptr<TreeNode> node = root_;
        for (int i : path) {
            if (node->nodes_.count(i) == 0) {
                throw std::logic_error("Shit happened");
            }
            node = node->nodes_[i];
        }
        if (node->parent_.lock() == nullptr) {
            root_ = nullptr;
            return true;
        }
        node = node->parent_.lock();
        node->nodes_.erase(elem);
        return true;
    }

    [[nodiscard]] std::vector<int> GetPathTo(int id) const {
        std::vector<int> v;
        if (!SearchFunc(root_, id, v)) {
            return {};
        }
        return v;
    }

    [[nodiscard]] std::vector<int> GetNodes() const {
        std::vector<int> v;
        GetNodes(root_, v);
        return v;
    }

private:
    bool SearchFunc(const std::shared_ptr<TreeNode>& node, int id, std::vector<int>& v) const {
        if (node == nullptr) {
            return false;
        }
        if (node->id_ == id) {
            v.push_back(node->id_);
            return true;
        }
        v.push_back(node->id_);
```

```
        for (auto [child_id, child_node] : node->nodes_) {
            if (SearchFunc(child_node, id, v)) {
                return true;
            }
        }
        v.pop_back();
        return false;
    }

    void GetNodes(const std::shared_ptr<TreeNode>& node, std::vector<int>& v) const {
        if (node == nullptr) {
            return;
        }
        v.push_back(node->id_);
        for (auto [child_id, child_ptr] : node->nodes_) {
            GetNodes(child_ptr, v);
        }
    }
    std::shared_ptr<TreeNode> root_ = nullptr;
};

int main() {
    std::string command;
    IdIndexingTree ids;
    pid_t child_pid = 0;
    int child_id = 0;
    zmq::context_t context(1);
    zmq::socket_t main_socket(context, ZMQ_REQ);
    int linger = 0;
    main_socket.setsockopt(ZMQ_SNDTIMEO, 2000);
    main_socket.setsockopt(ZMQ_LINGER, &linger, sizeof(linger));
    int port = bind_socket(main_socket);
    while (true) {
        std::cin >> command;
        if (command == "create") {
            size_t node_id, parent_id;
            std::string result;
            std::cin >> node_id >> parent_id;
            if (child_pid == 0) {
                child_pid = fork();
                if (child_pid == -1) {
                    std::cout << "Unable to create first worker node\n";
                    child_pid = 0;
                    exit(1);
                } else if (child_pid == 0) {
                    create_node(node_id, parent_id, port);
                } else {
                    parent_id = 0;
                    child_id = node_id;
                    send_message(main_socket,"pid");
                    result = recieve_message(main_socket);
                }
```

```
    } else {
        if (!ids.GetPathTo(node_id).empty()) {
            std::cout << "Error: Node already exists" << "\n";
            continue;
        }
        std::vector<int> path = ids.GetPathTo(parent_id);
        if (path.empty()) {
            std::cout << "Error: No parent node" << "\n";
            continue;
        }
        path.erase(path.begin());
        std::ostringstream msg_stream;
        msg_stream << "create " << path.size(); //сначала путь потом ид
        for (int i : path) {
            msg_stream << " " << i;
        }
        msg_stream << " " << node_id;
        send_message(main_socket, msg_stream.str());
        result = recieve_message(main_socket);
    }
    if (result.substr(0,2) == "Ok") {
        ids.Insert(node_id, parent_id);
    }
    std::cout << result << "\n";

} else if (command == "remove") {
    if (child_pid == 0) {
        std::cout << "Error: No such node\n";
        continue;
    }
    size_t node_id;
    std::cin >> node_id;
    if (node_id == child_id) {
        send_message(main_socket, "kill");
        recieve_message(main_socket);
        kill(child_pid, SIGTERM);
        kill(child_pid, SIGKILL);
        child_id = 0;
        child_pid = 0;
        std::cout << "Ok\n";
        ids.Erase(node_id);
        continue;
    }
    std::vector<int> path = ids.GetPathTo(node_id);
    if (path.empty()) {
        std::cout << "Error: No such node" << "\n";
        continue;
    }
    path.erase(path.begin());
    std::ostringstream msg_stream;
    msg_stream << "remove " << path.size() - 1;
    for (int i : path) {
```

```
            msg_stream << " " << i;
        }
        send_message(main_socket, msg_stream.str());
        std::string received_message = recieve_message(main_socket);
        if (received_message.substr(0, std::min<int>(received_message.size(), 2)) == "Ok") {
            ids.Erase(node_id);
        }
        std::cout << received_message << "\n";

    } else if (command == "exec") { //under development
        int id;
        std::string str, substr;
        std::cin >> id;
        std::vector<int> path = ids.GetPathTo(id);
        if (path.empty()) {
            std::cout << "Error: No such node\n";
            continue;
        }
        path.erase(path.begin());
        std::ostringstream msg_stream;
        msg_stream << "exec " << path.size();
        for (int i : path) {
            msg_stream << " " << i;
        }
        std::cin >> str;
        std::cin >> substr;
        msg_stream << " " << str << " " << substr;
        send_message(main_socket, msg_stream.str());
        std::string received_message = recieve_message(main_socket);
        std::cout << received_message << "\n";

    } else if (command == "heartbit") { //???
        if (child_pid == 0) {
            std::cout << "No nodes\n";
            continue;
        }
        send_message(main_socket,"heartbit");
        std::string recieved = recieve_message(main_socket);
        std::istringstream is(recieved);
        std::vector<int> recieved_nodes;
        int elem;
        while (is >> elem) {
            recieved_nodes.push_back(elem);
        }
        std::sort(recieved_nodes.begin(), recieved_nodes.end());
        std::vector<int> all_nodes = ids.GetNodes();
        std::sort(all_nodes.begin(), all_nodes.end());
        std::cout << "Recieved nodes " << recieved_nodes << "\n";
        std::cout << "All nodes " << all_nodes << "\n";
    } else if (command == "exit") {
        send_message(main_socket, "kill");
        recieve_message(main_socket);
```

```
            kill(child_pid, SIGTERM);
            kill(child_pid, SIGKILL);
            break;
        }
    }
    return 0;
}
```

2.3 compute_node.cpp

```
#include <iostream>
#include <string>
#include <sstream>
#include <zconf.h>
#include <csignal>
#include <unordered_map>
#include "functions.h"

int main (int argc, char* argv[]) {
    int id = std::atoi(argv[1]);
    int parent_id = std::atoi(argv[2]);
    int parent_port = std::atoi(argv[3]);

    zmq::context_t context(3);
    zmq::socket_t parent_socket(context, ZMQ_REP);

    parent_socket.connect(get_port_name(parent_port));

    std::unordered_map<int, zmq::socket_t> sockets;
    std::unordered_map<int, int> pids;
    std::unordered_map<int, int> ports;

    while(true) {
        std::string request_string;
        request_string = recieve_message(parent_socket);
        std::istringstream command_stream(request_string);
        std::string command;
        command_stream >> command;
        if (command == "id") {
            std::string parent_string = "Ok:" + std::to_string(id);
            send_message(parent_socket, parent_string);
        } else if (command == "pid") {
            std::string parent_string = "Ok:" + std::to_string(getpid());
            send_message(parent_socket, parent_string);
        } else if (command == "create") {
            int size, node_id;
            command_stream >> size;
            std::vector<int> path(size);
            for (int i = 0; i < size; ++i) {
```

```
            command_stream >> path[i];
        }
        command_stream >> node_id;
        if (size == 0) {
            sockets.emplace(std::piecewise_construct,
                            std::forward_as_tuple(node_id),
                            std::forward_as_tuple(context, ZMQ_REQ));
            int port = bind_socket(sockets.at(node_id));
            int pid = fork();
            if (pid == -1) {
                send_message(parent_socket, "Cannot fork");
                continue;
            } else if (pid == 0) {
                create_node(node_id, id, port);
            } else {
                ports[node_id] = port;
                pids[node_id] = pid;
                send_message(sockets.at(node_id), "pid");
                send_message(parent_socket, recieve_message(sockets.at(node_id)));
            }
        } else {
            int next_id = path.front();
            path.erase(path.begin());
            std::ostringstream msg_stream;
            msg_stream << "create " << path.size();
            for (int i : path) {
                msg_stream << " " << i;
            }
            msg_stream << " " << node_id;
            send_message(sockets.at(next_id), msg_stream.str());
            send_message(parent_socket, recieve_message(sockets.at(next_id)));
        }
    } else if (command == "remove") {
        int size, node_id;
        command_stream >> size;
        std::vector<int> path(size);
        for (int i = 0; i < size; ++i) {
            command_stream >> path[i];
        }
        command_stream >> node_id;
        if (path.empty()) {
            send_message(sockets.at(node_id), "kill");
            recieve_message(sockets.at(node_id));
            kill(pids[node_id], SIGTERM);
            kill(pids[node_id], SIGKILL);
            pids.erase(node_id);
            sockets.at(node_id).disconnect(get_port_name(ports[node_id]));
            ports.erase(node_id);
            sockets.erase(node_id);
            send_message(parent_socket, "Ok");
        } else {
            int next_id = path.front();
```

```
            path.erase(path.begin());
            std::ostringstream msg_stream;
            msg_stream << "remove " << path.size();
            for (int i : path) {
                msg_stream << " " << i;
            }
            msg_stream << " " << node_id;
            send_message(sockets.at(next_id), msg_stream.str());
            send_message(parent_socket, recieve_message(sockets.at(next_id)));
        }
    } else if (command == "exec") {
        int path_size;
        std::string str, substr;
        command_stream >> path_size;
        std::vector<int> path(path_size);
        for (int i = 0; i < path_size; ++i) {
            command_stream >> path[i];
        }
        command_stream >> str;
        command_stream >> substr;
        if (path.empty()) {
            std::string msg_string("Ok ");
            msg_string += std::to_string(id);
            int pos = str.find(substr);
            if (pos != std::string::npos) {
                msg_string += ": " + std::to_string(pos);
            } else {
                msg_string += ": no matches";
            }
            send_message(parent_socket, msg_string);
        } else {
            int next_id = path.front();
            path.erase(path.begin());
            std::ostringstream message_stream;
            message_stream << "exec " << path.size();
            for (int i : path) {
                message_stream << " " << i;
            }
            message_stream << " " << str << " " << substr;
            send_message(sockets.at(next_id),message_stream.str());
            send_message(parent_socket, recieve_message(sockets.at(next_id)));
        }
    } else if (command == "heartbit") {
        std::ostringstream res;
        for (auto& [child_id, child_socket] : sockets) {
            send_message(child_socket, "heartbit");
            std::string local_result = recieve_message(child_socket);
            if (!local_result.empty() && local_result.substr(std::min<int>(local_result.size(),5)) !=
"Error") {
                res << local_result << " ";
            }
        }
```

```
            res << id << " ";
            send_message(parent_socket, res.str());
        } else if (command == "kill") {
            for (auto& [child_id, child_socket] : sockets) {
                send_message(child_socket, "kill");
                recieve_message(child_socket);
                kill(pids[child_id], SIGTERM);
                kill(pids[child_id], SIGKILL);
            }
            send_message(parent_socket, "Ok");
        }
        if (parent_port == 0) {
            break;
        }
    }
}
```

3. Протокол работы программы

```
anri@andrew-HP-250-G6:~/Documents/Github_repositories/OS_labs/lab6/src/build$
./control_node
create 1 -1
Ok:16896
create 2 1
Ok:16901
exec 2 qwerty we
Ok 2: 1
create 3 1
Ok:16909
exec 3 asdf as
Ok 3: 0
heartbit
Recieved nodes 1 2 3
All nodes 1 2 3
exit
anri@andrew-HP-250-G6:~/Documents/Github_repositories/OS_labs/lab6/src/build$
```

4. Объяснение работы программы

Для запуска программы запускается код control_node. В нём задаются параметры управляющего узла (его сокет, дерево id вычислительных узлов), и в цикле считываются команды, которые он при помощи очереди сообщений отправляет по дереву, а затем ждёт сообщения об успешном выполнении команды от указанного в команде узла. Вычислительный узел же после создания в цикле ждёт команду, и либо передаёт её дальше, либо выполняет её, после чего передаёт сообщение обратно в управляющий узел. Для передачи сообщений по дереву используется вектор, представляющий собой путь от вершины дерева до нужного узла, вычисляемый отдельной функцией.

5. Strace

```
anri@andrew-HP-250-G6:~/Documents/Github_repositories/OS_labs/lab6/src/build$ strace -f -e trace=clone ./control_node
clone(strace: Process 24331 attached
child_stack=0x7faca69e4b70, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7faca69e59d0, tls=0x7faca69e5700, child_tidptr=0x7faca69e59d0) = 24331
[pid 24330] clone(strace: Process 24332 attached
child_stack=0x7faca61e3b70, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7faca61e49d0, tls=0x7faca61e4700, child_tidptr=0x7faca61e49d0) = 24332
create 1 -1
[pid 24330] clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7faca89b5e50) = 24339
strace: Process 24339 attached
[pid 24339] clone(strace: Process 24340 attached
child_stack=0x7f3f57fdab70, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7f3f57fdb9d0, tls=0x7f3f57fdb700, child_tidptr=0x7f3f57fdb9d0) = 24340
[pid 24339] clone(strace: Process 24341 attached
child_stack=0x7f3f577d9b70, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7f3f577da9d0, tls=0x7f3f577da700, child_tidptr=0x7f3f577da9d0) = 24341
[pid 24339] clone(strace: Process 24342 attached
child_stack=0x7f3f56fd8b70, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7f3f56fd99d0, tls=0x7f3f56fd9700, child_tidptr=0x7f3f56fd99d0) = 24342
[pid 24339] clone(strace: Process 24343 attached
child_stack=0x7f3f567d7b70, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7f3f567d89d0, tls=0x7f3f567d8700, child_tidptr=0x7f3f567d89d0) = 24343
Ok:24339
create 2 1
[pid 24339] clone(strace: Process 24351 attached
child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f3f59fabe50) = 24351
[pid 24351] clone(strace: Process 24352 attached
child_stack=0x7fc893574b70, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fc8935759d0, tls=0x7fc893575700, child_tidptr=0x7fc8935759d0) = 24352
[pid 24351] clone(child_stack=0x7fc892d73b70, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|
```

CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fc892d749d0, tls=0x7fc892d74700, child_tidptr=0x7fc892d749d0) = 24353
[pid 24351] clone(strace: Process 24354 attached
strace: Process 24353 attached
child_stack=0x7fc892572b70, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fc8925739d0, tls=0x7fc892573700, child_tidptr=0x7fc8925739d0) = 24354
[pid 24351] clone(strace: Process 24355 attached
child_stack=0x7fc891d71b70, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fc891d729d0, tls=0x7fc891d72700, child_tidptr=0x7fc891d729d0) = 24355
Ok:24351
exec 1 qwerty we
Ok 1: 1
remove 2
[pid 24351] --- SIGTERM {si_signo=SIGTERM, si_code=SI_USER, si_pid=24339, si_uid=1000} ---
[pid 24355] +++ killed by SIGTERM +++
[pid 24354] +++ killed by SIGTERM +++
[pid 24353] +++ killed by SIGTERM +++
[pid 24352] +++ killed by SIGTERM +++
[pid 24351] +++ killed by SIGTERM +++
[pid 24339] --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_KILLED, si_pid=24351, si_uid=1000, si_status=SIGTERM, si_utime=0, si_stime=1} ---
Ok
heartbit
Recieved nodes 1
All nodes 1
create 2 1
[pid 24339] clone(strace: Process 24390 attached
child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f3f59fabe50) = 24390
[pid 24390] clone(strace: Process 24391 attached
child_stack=0x7ff72a908b70, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7ff72a9099d0, tls=0x7ff72a909700, child_tidptr=0x7ff72a9099d0) = 24391
[pid 24390] clone(strace: Process 24392 attached
child_stack=0x7ff72a107b70, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7ff72a1089d0, tls=0x7ff72a108700, child_tidptr=0x7ff72a1089d0) = 24392
[pid 24390] clone(strace: Process 24393 attached
child_stack=0x7ff729906b70, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7ff7299079d0, tls=0x7ff729907700, child_tidptr=0x7ff7299079d0) = 24393
[pid 24390] clone(strace: Process 24394 attached
child_stack=0x7ff729105b70, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|

```
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7ff7291069d0, tls=0x7ff729106700, child_tidptr=0x7ff7291069d0) = 24394
Ok:24390
heartbit
Recieved nodes 1 2
All nodes 1 2
exit
[pid 24390] --- SIGTERM {si_signo=SIGTERM, si_code=SI_USER, si_pid=24339, si_uid=1000} ---
[pid 24393] +++ killed by SIGTERM +++
[pid 24392] +++ killed by SIGTERM +++
[pid 24391] +++ killed by SIGTERM +++
[pid 24394] +++ killed by SIGTERM +++
[pid 24390] +++ killed by SIGTERM +++
[pid 24343] ????( <unfinished ...>
[pid 24343] +++ killed by SIGKILL +++
[pid 24342] +++ killed by SIGKILL +++
[pid 24341] +++ killed by SIGKILL +++
[pid 24340] +++ killed by SIGKILL +++
[pid 24339] +++ killed by SIGKILL +++
[pid 24330] --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_KILLED, si_pid=24339, si_uid=1000, si_status=SIGKILL, si_utime=0, si_stime=1} ---
[pid 24331] +++ exited with 0 +++
[pid 24332] +++ exited with 0 +++
+++ exited with 0 +++
```

Вывод

В связи с СЕРЬЁЗНЫМИ ошибками в планировании своего времени, я не проделал работу в полной мере, из-за надвигающегося срока дедлайна мне пришлось несколько схалтурить и упростить себе задачу: топология дерева общего вида у меня должным образом не работает, и представляет собой скорее топологию вектора или списка с произвольным доступом. Вычислительные узлы по команде exec находят не все вхождения подстроки в строку, а только первое вхождение, а команда heartbit вместо вывода узлов, которые не ответили втечение n милисекунд, просто выводит список узлов, до которых дошла команда, и узлов, которые формально числятся в дереве в управляющем узле. Но, несмотря на эти серьёзные упрощения, работа всё равно оказалась далеко не самой простой, хотя и интересной.