Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №6-8 по курсу
«Операционные системы»

Тема работы
“Очереди сообщений”

Студент: Шатунова Юлия Викторовна
Группа: М8О-208Б-20
Вариант: 8
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

Репозиторий

https://github.com/s0bakkaa/OS/tree/main/lab6_8

Постановка задачи

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 8. Идеально сбалансированное бинарное дерево, поиск подстроки в строке, pingall.

Общие сведения о программе

Реализованы 5 файлов:

topology.h – реализация заданной структуры.

zmq_funcs.h – функции для работы с zmq.

calc_node.h – реализация команд вычислительного узла.

client.cpp – клиентская программа.

server.cpp – серверная программа

Общий метод и алгоритм решения

Запускается клиент, который будет в зависимости от получаемых сообщений посылать те или иные запросы серверу. Программа завершается по Ctrl+D. Поиск подстроки в строке реализован в виде наивного алгоритма.

Исходный код

calc_node.h

```
#include <bits/stdc++.h>
#include <vector>
#include "zmq_funcs.h"
#include "unistd.h"

std:: vector<unsigned int> find_substrings(std:: string pattern, std:: string text) {
                                                int i, j;
                                                std:: vector<unsigned int> vect;
                                                int tmp = pattern.length() - text.length() + 1;
                                                for (i = 0; i < tmp; ++i) {
                                                 j = 0;
                                                 while ((j < text.length()) && (text[j] == pattern[i + j])) {
                                                          j += 1;
                                                 }
                                                 if (j == text.length()) {
                                                          vect.push_back(i);
                                                 }
                                                }
                                                return vect;
}

class CalculationNode {
   private:
      zmq:: context_t context;
   public:
      zmq:: socket_t left, right, parent;
      int id, left_id = -2, right_id = -2, parent_id;
      int left_port, right_port, parent_port;
      CalculationNode(int id, int parent_port, int parent_id):
         id(id),
         parent_port(parent_port),
         parent_id(parent_id),
         left(context, ZMQ_REQ),
         right(context, ZMQ_REQ),
         parent(context, ZMQ_REP)
         {
         if (id != -1) {
            connect(parent, parent_port);
         }
      }
      std:: string create (int child_id) {
         int port;
         bool isleft = false;
         if (left_id == -2) {
            left_port = bind(left, child_id);
            left_id = child_id;
            port = left_port;
            isleft = true;
         }
         else if (right_id == -2) {
            right_port = bind(right, child_id);
            right_id = child_id;
            port = right_port;
         }
         else {
            std:: string fail = "Error: can not create the calculation node";
            return fail;
         }
         int fork_id = fork();
         if (fork_id == 0) {
```

```
            if (execl("./server", "server", std:: to_string(child_id).c_str(), std:: to_string(port).c_str(), std::
to_string(id).c_str(), (char*)NULL) == -1) {
                std:: cout << "Error: can not run the execl-command" << std:: endl;
                exit(EXIT_FAILURE);
            }
        }
        else {
            std:: string child_pid;
            try {
                if (isleft) {
                    left.setsockopt(ZMQ_SNDTIMEO, 3000);
                    send_message(left, "pid");
                    child_pid = receive_message(left);
                }
                else {
                    right.setsockopt(ZMQ_SNDTIMEO, 3000);
                    send_message(right, "pid");
                    child_pid = receive_message(right);
                }
                return "Ok: " + child_pid;
            }
            catch (int) {
                std:: string fail = "Error: can not connect to the child";
                return fail;
            }
        }
    }
    std:: string ping (int id) {
        std:: string answer = "Ok: 0";
        if (this->id == id) {
            answer = "Ok: 1";
            return answer;
        }
        else if (left_id == id) {
            std:: string message = "ping " + std:: to_string(id);
            send_message(left, message);
            try {
                message = receive_message(left);
                if (message == "Ok: 1") {
                    answer = message;
                }
            }
            catch(int){}
        }
        else if (right_id == id) {
            std:: string message = "ping " + std:: to_string(id);
            send_message(right, message);
            try {
                message = receive_message(right);
                if (message == "Ok: 1") {
                    answer = message;
                }
            }
            catch(int){}
        }
        return answer;
    }
    std:: string sendstring (std:: string string, int id) {
        std:: string answer = "Error: Parent not found";
        if (left_id == -2 && right_id == -2) {
            return answer;
        }
```

```
    else if (left_id == id) {
        if (ping(left_id) == "Ok: 1") {
            send_message(left, string);
            try{
                answer = receive_message(left);
            }
            catch(int){}
        }
    }
    else if (right_id == id) {
        if (ping(right_id) == "Ok: 1") {
            send_message(right, string);
            try {
                answer = receive_message(right);
            }
            catch(int){}
        }
    }
    else {
        if (ping(left_id) == "Ok: 1") {
            std:: string message = "send " + std:: to_string(id) + " " + string;
            send_message(left, message);
            try {
                message = receive_message(left);
            }
            catch(int) {
                message = "Error: Parent not found";
            }
            if (message != "Error: Parent not found") {
                answer = message;
            }
        }
        if (ping(right_id) == "Ok: 1") {
            std:: string message = "send " + std:: to_string(id) + " " + string;
            send_message(right, message);
            try {
                message = receive_message(right);
            }
            catch(int) {
                message = "Error: Parent not found";
            }
            if (message != "Error: Parent not found") {
                answer = message;
            }
        }
    }
    return answer;
}

std:: string exec (std:: string string) {
    std:: istringstream string_thread(string);
    std:: string pattern, text, result;
    string_thread >> pattern;
    string_thread >> text;
    std:: string answer = "Ok: " + std:: to_string(id) + ": ";
                                                    std:: vector<unsigned int> res = find_substrings(pattern, text);
                                                    if (res.empty()) {
                                                            return answer;
                                                    }
                                                    else {
                                                            for (int i = 0; i < res.size() - 1; ++i) {
                                                                    answer += std:: to_string(res[i]);
```

```
                answer += ";";
            }
            answer += std:: to_string(res.back());
        }

        return answer;
    }
    std:: string treeclear (int child) {
        if (left_id == child) {
            left_id = -2;
            unbind(left, left_port);
        }
        else {
            right_id = -2;
            unbind(right, right_port);
        }
        return "Ok";
    }
    std:: string kill () {
        if (left_id != -2){
            if (ping(left_id) == "Ok: 1") {
                std:: string message = "kill";
                send_message(left, message);
                try {
                    message = receive_message(left);
                }
                catch(int){}
                unbind(left, left_port);
                left.close();
            }
        }
        if (right_id != -2) {
            if (ping(right_id) == "Ok: 1") {
                std:: string message = "kill";
                send_message(right, message);
                try {
                    message = receive_message(right);
                }
                catch (int){}
                unbind(right, right_port);
                right.close();
            }
        }
        return std:: to_string(parent_id);
    }
    ~CalculationNode() {}
};

client.cpp
#include <bits/stdc++.h>
#include "calc_node.h"
#include "zmq_funcs.h"
#include "topology.h"

int main() {
    std:: string command;
    CalculationNode node(-1, -1, -1);
    std:: string answer;
    std:: cout << "create id" << std:: endl;
    std:: cout << "exec id" << std:: endl;
    std:: cout << "pingall" << std:: endl;
    std:: cout << "kill id" << std:: endl;
```

```
BalancedTree tree;
while ((std:: cout << "Command: ") && (std:: cin >> command)) {
   if (command == "create") {
      int child;
      std:: cin >> child;
      if (tree.Exist(child)) {
         std:: cout << "Error: Already exists" << std:: endl;
      }
      else {
         while (true) {
            int idParent = tree.FindID();
            if (idParent == node.id) {
               answer = node.create(child);
               tree.AddInTree(child, idParent);
               break;
            }
            else {
               std:: string message = "create " + std:: to_string(child);
               answer = node.sendstring(message, idParent);
               if (answer == "Error: Parent not found") {
                  tree.AvailabilityCheck(idParent);
               }
               else {
                  tree.AddInTree(child, idParent);
                  break;
               }
            }
         }
         std:: cout << answer << std::endl;
      }
   }
   else if (command == "exec") {
      std:: string pattern, text;
      int child;
      std:: cin >> child;
      if (!tree.Exist(child)) {
         std:: cout << "Error: Parent is not existed" << std:: endl;
      }
      else {
                                                std:: cout << "Enter pattern text: ";
                                                std:: cin >> pattern;
                                                std:: cout << "Enter text: ";
                                                std:: cin >> text;
         std:: string message = "exec " + pattern + " " + text;
         answer = node.sendstring(message, child);
         std:: cout << answer << std:: endl;
      }
   }
   else if (command == "pingall") {
      std:: string str;
      std:: vector<int> not_available;

         for (int j : tree.ids) {
            std:: string answer = node.ping(j);
            if (answer != "Ok: 1") {
               not_available.push_back(j);
            }
         }
         if (not_available.empty()) {
            std:: cout << "Ok: -1" << std:: endl;
         }
         else {
```

```
                std:: cout << "Ok: ";
                for (int z = 0; z < not_available.size(); ++z) {
                    std:: cout << not_available[z] << " ";
                }
                std:: cout << std:: endl;
                not_available.clear();
            }

        }
        else if (command == "kill") {
            int child;
            std:: cin >> child;
            std:: string message = "kill";
            if (!tree.Exist(child)) {
                std:: cout << "Error: Parent is not existed" << std:: endl;
            }
            else {
                answer = node.sendstring(message, child);
                if (answer != "Error: Parent not found") {
                    tree.RemoveFromRoot(child);
                    if (child == node.left_id){
                        unbind(node.left, node.left_port);
                        node.left_id = -2;
                        answer = "Ok";
                    }
                    else if (child == node.right_id) {
                        node.right_id = -2;
                        unbind(node.right, node.right_port);
                        answer = "Ok";
                    }
                    else {
                        message = "clear " + std:: to_string(child);
                        answer = node.sendstring(message, std:: stoi(answer));
                    }
                    std:: cout << answer << std:: endl;
                }
            }
        }
        else {
            std:: cout << "Please enter correct command!" << std:: endl;
        }
    }
    node.kill();
    return 0;
}
```

server.cpp

```
#include <bits/stdc++.h>
#include "calc_node.h"
#include "zmq_funcs.h"
#include "topology.h"

int main(int argc, char *argv[]) {
    if (argc != 4) {
        std:: cout << "Usage: 1)./main, 2) child_id, 3) parent_port, 4) parent_id" << std:: endl;
        exit(EXIT_FAILURE);
    }
    CalculationNode node(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]));
    while(true) {
        std:: string message;
        std:: string command;
        message = receive_message(node.parent);
```

```
        std:: istringstream request(message);
        request >> command;
        if (command == "pid") {
            std:: string answer = std:: to_string(getpid());
            send_message(node.parent, answer);
        }
        else if (command == "ping") {
            int child;
            request >> child;
            std:: string answer = node.ping(child);
            send_message(node.parent, answer);
        }
        else if (command == "create") {
            int child;
            request >> child;
            std:: string answer = node.create(child);
            send_message(node.parent, answer);
        }
        else if (command == "send"){
            int child;
            std:: string str;
            request >> child;
            getline(request, str);
            str.erase(0, 1);
            std:: string answer = node.sendstring(str, child);
            send_message(node.parent, answer);
        }
        else if (command == "exec") {
            std:: string str;
            getline(request, str);
            std:: string answer = node.exec(str);
            send_message(node.parent, answer);
        }
        else if (command == "kill") {
            std:: string answer = node.kill();
            send_message(node.parent, answer);
            disconnect(node.parent, node.parent_port);
            node.parent.close();
            break;
        }
        else if (command == "clear") {
            int child;
            request >> child;
            std:: string answer = node.treeclear(child);
            send_message(node.parent, answer);
        }
    }
    return 0;
}

topology.h
#ifndef TOPOLOGY_H
#define TOPOLOGY_H
#include <bits/stdc++.h>
class BalancedTree {
        class BalancedTreeNode {
            public:
                int id;
                BalancedTreeNode* left;
                BalancedTreeNode* right;
                int height;
                bool available;
```

```
BalancedTreeNode (int id) {
   this->id = id;
   available = true;
   left = NULL;
   right = NULL;
}
void CheckAvailability (int id) {
   if (this->id == id){
      available = false;
   }
   else {
      if (left != NULL) {
         left->CheckAvailability(id);
      }
      if (right != NULL) {
         right->CheckAvailability(id);
      }
   }
}
void Remove (int id, std::set<int> &ids) {
   if (left != NULL && left->id == id) {
      left->RecursionRemove(ids);
      ids.erase(left->id);
      delete left;
      left = NULL;
   }
   else if (right != NULL && right->id == id) {
      right->RecursionRemove(ids);
      ids.erase(right->id);
      delete right;
      right = NULL;
   }
   else {
      if (left != NULL) {
         left->Remove(id, ids);
      }
      if (right != NULL) {
         right->Remove(id, ids);
      }
   }
}
void RecursionRemove (std::set<int> &ids) {
   if (left != NULL) {
      left->RecursionRemove(ids);
      ids.erase(left->id);
      delete left;
      left = NULL;
   }
   if (right != NULL) {
      right->RecursionRemove(ids);
      ids.erase(right->id);
      delete right;
      right = NULL;
   }
}
void AddInNode (int id, int parent_id, std::set<int> &ids) {
   if (this->id == parent_id) {
      if (left == NULL){
         left = new BalancedTreeNode(id);
      }
      else {
         right = new BalancedTreeNode(id);
```

```
            }
            ids.insert(id);
        }
        else {
            if (left != NULL) {
                left->AddInNode(id, parent_id, ids);
            }
            if (right != nullptr) {
                right->AddInNode(id, parent_id, ids);
            }
        }
    }
    int MinimalHeight() {
        if (left == NULL || right == NULL) {
            return 0;
        }
        int left_height = -1;
        int right_height = -1;
        if (left != NULL && left->available == true) {
            left_height = left->MinimalHeight();
        }
        if (right != NULL && right->available == true) {
            right_height = right->MinimalHeight();
        }
        if (right_height == -1 && left_height == -1) {
            available = false;
            return -1;
        }
        else if (right_height  == -1) {
            return left_height + 1;
        }
        else if (left_height == -1) {
            return right_height + 1;
        }
        else {
            return std::min(left_height, right_height) + 1;
        }
    }
    int IDMinimalHeight(int height, int current_height) {
        if (height < current_height) {
            return -2;
        }
        else if (height > current_height) {
            int current_id = -2;
            if (left != NULL && left->available == true) {
                current_id = left->IDMinimalHeight(height, (current_height + 1));
            }
            if (right != NULL && right->available == true && current_id == -2){
                current_id = right->IDMinimalHeight(height, (current_height + 1));
            }
            return current_id;
        }
        else {
            if (left == NULL || right == NULL){
                return id;
            }
            return -2;
        }
    }
    ~BalancedTreeNode() {}
};
private:
```

```
        BalancedTreeNode* root;
    public:
        std::set<int> ids;
        BalancedTree() {
            root = new BalancedTreeNode(-1);
        }
        bool Exist(int id) {
            if (ids.find(id) != ids.end()) {
                return true;
            }
            return false;
        }
        void AvailabilityCheck(int id) {
            root->CheckAvailability(id);
        }
        int FindID() {
            int h = root->MinimalHeight();
            return root->IDMinimalHeight(h, 0);
        }
        void AddInTree(int id, int parent) {
            root->AddInNode(id, parent, ids);
        }
        void RemoveFromRoot(int idElem) {
            root->Remove(idElem, ids);
        }
        ~BalancedTree() {
            root->RecursionRemove(ids);
            delete root;
        }
};
#endif // TOPOLOGY_H
```

zmq_funcs.h

```
#pragma once
#include <bits/stdc++.h>
#include <zmq.hpp>
const int MAIN_PORT = 4040;

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

std::string receive_message(zmq::socket_t &socket) {
    zmq::message_t message;
    int chars_read;
    try {
        chars_read = (int)socket.recv(&message);
    }
    catch (...) {
        chars_read = 0;
    }
    if (chars_read == 0) {
        throw -1;
    }
    std::string received_msg(static_cast<char*>(message.data()), message.size());
    return received_msg;
}

void connect(zmq::socket_t &socket, int port) {
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
```

```
    socket.connect(address);
}

void disconnect(zmq::socket_t &socket, int port) {
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
    socket.disconnect(address);
}

int bind(zmq::socket_t &socket, int id) {
    int port = MAIN_PORT + id;
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
    while(1){
        try{
            socket.bind(address);
            break;
        }
        catch(...){
            port++;
        }
    }
    return port;
}

void unbind(zmq::socket_t &socket, int port) {
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
    socket.unbind(address);
}
```

Makefile

```
files: client server

client: client.cpp
                                        g++ -fsanitize=address client.cpp -lzmq -o client -w

server: server.cpp
                                        g++ -fsanitize=address server.cpp -lzmq -o server -w

clean:
                                        rm  -rf client  server
```

Демонстрация работы программы

```
[yulia@andromeda lab6_8]$ ./client
create id
exec id
pingall
kill id
Command: create 4
Ok: 7300
Command: create 5
Ok: 7309
Command: create 6
Ok: 7314
Command: create 7
Ok: 7317
Command: exec 5
Enter pattern text: abracadabra
Enter text: abra
Ok: 5: 0;7
Command: create 11
```

```
Ok: 7329
Command: kill 5
Ok
Command: exec 7
Enter pattern text: pipipupu
Enter text: ip
Ok: 7: 1;3
Command: pingall
Ok: 6 7
Command: kill 6
Ok
Command: pingall
Ok: 7
Command: create 11
Ok: 7479
```

Выводы

Лабораторная работа была очень сложной, нужно было понять, как между собой взаимодействуют клиент и сервер. Методом проб и ошибок, а также долгих часов серфинга в Интернете с запросами по работе той или иной части zmq мне наконец-то удалось реализовать программу, которая бы отвечала заявленным требованиям.