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

Управлении серверами сообщений

Применение отложенных вычислений

Интеграция программных систем друг с другом

Студент: Шаларь Игорь Павлович
Группа: М8О-208Б-20
Вариант: 5
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

Репозиторий

https://github.com/IgShalar/OS/tree/main/os_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.

Топология: идеально сбалансированное бинарное дерево

Команда: сложение последовательности чисел

exec id n k_1 ... k_n

Тип проверки доступности узлов: pingall

Общие сведения о программе

Запуск:

_$ cmake .

_$ make

_$ ./main

Написано для Unix.

mylib.h:

Вспомогательные функции. Например, работа с сокетами zeromq организована в виде классов.

Системные вызовы:

zmq_ctx_new() - создание контекста

zmq_socket() - создание сокета

zmq_connect() - подключение сокета к порту

zmq_setsockopt() - настройка сокета

zmq_close() - отключение сокета

zmq_ctx_destroy() - уничтожение контекста

zmq_send() - отправка сообщения

zmq_recv() - получение сообщения

execl() - заменяет текущий образ процесса новым образом процесса

main.cpp:

Управляющий узел.

Системные вызовыы

fork() - создание дочернего процесса

kill() - остановка процесса

calc.cpp:

вычислительный узел

tree.h:

Определяет куда добавить вершину

Общий метод и алгоритм решения

Каждый узел состоит из двух процессов, один передает выше команды снизу, другой слушает и передает ниже (если надо) команды сверху. При остановке узла, верхние узлы остаются работоспособными.

Исходный код

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.16)

set(CMAKE_CXX_STANDARD 14)

add_executable(main main.cpp)
add_executable(calc calc.cpp)
target_link_libraries(main zmq)
target_link_libraries(calc zmq)
```

mylib.h:

```
#include <zmq.h>
#include <unistd.h>
#include<iostream>
#include <sys/types.h>
#include <signal.h>
#include <chrono>
#include <thread>

using namespace std;

const int WAIT = 10;
const int LIM = 50;
int port = 48654;

class Request{

public:
    Request(string s){
        s = "tcp://localhost:" + s;
```

```
//        cout << "Connecting request to " << s << endl;
        ctx = zmq_ctx_new();
        if (ctx == NULL){
            perror("ctx");
        }
        req = zmq_socket(ctx, ZMQ_REQ);
        if (req == NULL){
            perror ("Can't create socket");
        }
        if (zmq_connect (req, s.c_str()) == -1) perror("Can't connect");
        if (zmq_setsockopt (req, ZMQ_RCVTIMEO, &WAIT, sizeof(int)) == -1){
            perror("Can't set wait");
        }
        if (zmq_setsockopt (req, ZMQ_LINGER, &WAIT, sizeof(int)) == -1){
            perror("Can't set linger");
        }
//        cout << "Request to " << s << " connected." << endl;
    }

    ~Request(){
        if (zmq_close(req) == -1) perror ("Can't close socket");
        if (zmq_ctx_destroy(ctx) == -1) perror("Can't destroy context.");
    }

    string send(const string &s){
        string res;
        char ch[size];
        zmq_send (req, s.c_str(), s.size() + 1, 0);
        if (zmq_recv (req, ch, size, 0) == -1) res = "-1";
        else{
            string temp(ch);
            res = temp;
        }
        return res;
    }

private:
```

```
    const int size = LIM;
    void * req, * ctx;
};


class Reply{

public:
    Reply(string s){
        s = "tcp://*:" + s;
//        cout << "Connecting reply to " << s << endl;
        ctx = zmq_ctx_new();
        if (ctx == NULL){
            perror("ctx");
        }
        rep = zmq_socket(ctx, ZMQ_REP);
        if (rep == NULL){
            perror ("Can't create socket");
        }
        int rc = 0;
        rc = zmq_bind (rep, s.c_str());
        if (rc == -1) perror ("Connection failed");
//        cout << "Reply to " << s << " connected." << endl;
    }

    ~Reply(){
        if (zmq_close(rep) == -1) perror ("Can't close socket");
        if (zmq_ctx_destroy(ctx) == -1) perror("Can't destroy context.");
    }

    string recieve(){
        char ch[size];
        zmq_recv (rep, ch, size, 0);
        string res(ch);
        return res;
    }
```

```
    void reply(const string &s){
        zmq_send (rep, s.c_str(), s.size() + 1, 0);
    }

private:
    const int size = LIM;
    void * rep, * ctx;
};

void remove_spaces (string &s){
    int now = 0;
    while (s[now] == ' ' && now < s.size()) now++;
    s.erase(0, now);
}

string get_word (string &s){
    int now = 0;
    while (s[now] == ' ' && now < s.size()) now++;
    s.erase(0, now);
    now = 0;
    while (s[now] != ' ' && now < s.size()) now++;
    string res = s.substr(0, now);
    s.erase(0, now);
    remove_spaces(s);
    return res;
}

string first_word (const string &s){
    int now = 0;
    string res = "";
    while (s[now] == ' ' && now < s.size()) now++;
    while (s[now] != ' ' && now < s.size()){
        res += s[now];
        now++;
    }
    return res;
}
```

```
void create_node(int my_id, const string &p){ // add request port
    string s = p;
    s += " " + to_string (my_id);
//    cout << "SEND_STRING: " << s << endl;
    int id = fork();
    if (id == -1){
        perror("Fork error");
    }
    else if (id == 0 && execl("calc", s.c_str(), NULL) < 0){
        perror("execl error");
    }
}

string get_port(){
    if (port == 49152){
        cout << "ERROR: Port limit exceeded." << endl;
        return "ERROR";
    }
    string res = to_string(port);
    port++;
    return res;
}
```

tree.h:

```
#include<set>

#include<iostream>


using namespace std;


struct node{

    node * l, * r;

    int v;

    int s;
```

```
    node(int a){
        l = NULL;
        r = NULL;
        v = a;
        s = 1;
    }
};

struct Tree{
    node * root;

    Tree(){
        root = new node(-2);
    }




    void print(node * now){
        if (now == NULL) return;
        cout << now->v << ": " << now->s << endl;
        print(now->l);
        print(now->r);
    }

    int cnt_dfs(node * now){
        if (now == NULL) return 0;
```

```
    int l = cnt_dfs(now->l);
    int r = cnt_dfs(now->r);
    now->s = l + r + 1;
    return now->s;
}

void cnt(){
    cnt_dfs(root);
}



void del_dfs(node * now){
    if (now == NULL) return;
    del_dfs(now->l);
    del_dfs(now->r);
    delete now;
}

void add_dfs(node * now, int p, int val){
    if (now == NULL) return;
    if (now->v == p){
        if (now->l == NULL) now->l = new node(val);
        else if (now->r == NULL) now->r = new node (val);
        return;
    }
    add_dfs(now->l, p, val);
    add_dfs(now->r, p, val);
```

```
}

void add(int p, int val){
    add_dfs(root, p, val);
    cnt();
}

void find_dfs(node * now, int &res, set <int> &s){
    if (now == NULL) return;
    if (now->l == NULL){
        if (s.find(now->v) == s.end()) res = now->v;
        return;
    }
    if (now->r == NULL){
         if (s.find(now->v) == s.end()) res = now->v;
        return;
    }
    if (now->l->s <= now->r->s) find_dfs(now->l, res, s);
    if (now->l->s >= now->r->s) find_dfs(now->r, res, s);
}

int find(set <int> &s){
    int res = -1;
    find_dfs(root, res, s);
    return res;
}
```

```
    ~Tree(){
        del_dfs(root);
    }
};
```

main.cpp:

```
#include"mylib.h"
#include<queue>
#include<set>
#include"tree.h"

using namespace std;

Tree tree;
set <int> nodes;
Request * l = NULL, * r = NULL;

set <int> ping (){
//    this_thread::sleep_for(std::chrono::milliseconds(500 * WAIT));
    string m = "ping";
    string s = "";
    if (l != NULL) s += l->send(m);
    s += " ";
    if (r != NULL) s += r->send(m);
    set <int> res = nodes;
    remove_spaces(s);
    while (s != ""){
```

```
        string t = get_word(s);
        int a = stoi(t);
        if (a != -1) res.erase(a);
    }
    return res;
}


int main (){
    string up_port = get_port();
    string s = "";
    int pr_id = fork();
    if (pr_id == -1){
        perror ("Fork error.");
        exit(-1);
    }
    else if (pr_id == 0){
        Reply * rep = new Reply(up_port);
        while(1){
            string m = rep->recieve();
            cout << m << endl;
            rep->reply("GOT");
        }
    }
    while (1){
        cin >> s;
//        cout << "INPUT: " << s << endl;
```

```
if (s == "exit"){
    if (l != NULL) l->send("stop");
    if (r != NULL) r->send("stop");
    break;
}
else if (s == "create"){
    int id;
    string t;
    cin >> id;
    if (nodes.find(id) != nodes.end()){
        cout << "Error: Already exists" << endl;
        continue;
    }
    set <int> busy = ping();
    int wh = tree.find(busy);
    if (wh == -1){
        cout << "Error: can't find avaliable parent" << endl;
        continue;
    }
    nodes.insert(id);
    tree.add(wh, id);
    cout << "WHERE: " << wh << endl; //root has id = -2
    if (wh == -2){
        string temp = get_port();
        create_node (id, get_port() + " " + up_port + " " + temp);
        if (l == NULL) l = new Request(temp);
        else r = new Request(temp);
```

```
            }
            else{
                string m = to_string(wh) + " c " + to_string(id) + " " + get_port() + " " +
get_port();
//                cout << "TO SEND: " << m << endl;
                if (l != NULL) l->send(m);
                if (r != NULL) r->send(m);
            }
        }
        else if (s == "exec"){
            string t, res;
            int n;
            cin >> res;
            res += " s";
            cin >> n;
            res += " " + to_string(n);
            for (int i = 0; i < n; i++){
                cin >> t;
                res += " " + t;
            }
//            cout << "TO SEND: " << res << endl;
            string wh = first_word(res);
            int tr = stoi(wh);
            if (nodes.find(tr) == nodes.end()){
                cout << "Error:" << tr << ": Not found" << endl;
                continue;
            }
            set <int> qw = ping();
```

```
            if (qw.find(tr) != qw.end()){
                cout << "Error:" << tr << ": Node is unavailable" << endl;
                continue;
            }
            if (l != NULL) l->send(res);
            if (r != NULL) r->send(res);
        }
        else if (s == "pingall"){
            set<int> temp = ping();
            cout << "OK: ";
            if (temp.empty()) cout << "-1";
            else for (auto i : temp) cout << i << "; ";
            cout << endl;
        }
        else if (s == "kill"){
            int w;
            cin >> w;
            string m = to_string(w) + " k";
            if (l != NULL) l->send(m);
            if (r != NULL) r->send(m);
        }
    }
    if (kill(pr_id, SIGKILL) == -1) perror ("Can't kill");
    if (l != NULL) delete l;
    if (r != NULL) delete r;
    return 0;
}
```

calc.cpp:

```
#include"mylib.h"

using namespace std;



int main (int a, char * tr[]){
    string qw (* tr);
//    cout << "GOT_STRING: " << qw << endl;
    string up_port = get_word(qw);
    string req_port = get_word(qw);
    string rep_port = get_word(qw);
    string my_id = get_word(qw);
    Request * p = new Request(req_port), * l = NULL, * r = NULL;
    Reply * prep = new Reply(rep_port), * rep = NULL;
    int pr_id = fork();
    if (pr_id == -1){
        perror ("Fork error.");
        exit(-1);
    }
    else if (pr_id == 0){
        Request * p_sec = new Request(req_port);
        rep = new Reply(up_port);
        while(1){
            string m = rep->recieve();
            rep->reply("OK");
```

```
            p_sec->send(m);
//            cout << my_id << ": up " << m << endl;
        }
    }
//    cout << my_id << ": started" << endl;
    p->send("OK: " + to_string(getpid()) + " " + to_string(pr_id));
    while (1){
        string s = prep->recieve();
        if (first_word(s) == "ping"){
            string t1 = "-1";
            string t2 = "-1";
            if (l != NULL) t1 = l->send(s);
            if (r != NULL) t2 = r->send(s);
            if (t1 != "-1") t1 += " " + my_id;
            else t1 = my_id;
            if (t2 != "-1") t1 += " " + t2;
            prep->reply(t1);
        }
        prep->reply("OK");
        if (first_word(s) == "stop"){
            if (l != NULL) l->send(s);
            if (r != NULL) r->send(s);
            break;
        }
        else if (first_word(s) != my_id){
            if (l != NULL) l->send(s);
            if (r != NULL) r->send(s);
```

```
        }
        else{
            get_word(s);
            string t = get_word(s);
            if (t == "c"){
                int new_id = stoi(get_word(s));
                string up = get_word(s);
                string rep = get_word(s);
                create_node(new_id, up + " " + up_port + " " + rep);
                if (l == NULL) l = new Request(rep);
                else r = new Request(rep);
//                cout << my_id << ": new node id = " << new_id << endl;
            }
            else if (t == "s"){
                long long n, a, res = 0;
                n = stoi(get_word(s));
                for (int i = 0; i < n; i++){
                    a = stoi(get_word(s));
                    res += a;
                }
//                cout << my_id << ": SOLVED" << endl;
                string result = "OK:" + my_id + ": " + to_string(res);
                p->send(result);

                //cout << my_id << ": " << res << endl;
            }
            else if (t == "k"){
```

```
            if (l != NULL) l->send("stop");
            if (r != NULL) r->send("stop");
            break;
        }
    }
  }
  if (kill(pr_id, SIGKILL) == -1) perror ("Can't kill");
  if (p != NULL) delete p;
  delete prep;
  if (l != NULL) delete l;
  if (rep != NULL) delete rep;
  if (r != NULL) delete r;
  cout << my_id << ": STOPPED" << endl;
  return 0;
}
```

Демонстрация работы программы

```
igor@igor-VirtualBox:/media/sf_VM_Shared$ ./main
create 1
WHERE: -2
OK: 2294 2301
create 2
WHERE: -2
OK: 2306 2313
create 3
WHERE: 2
OK: 2318 2325
create 4
WHERE: 1
OK: 2330 2337
create 5
WHERE: 2
OK: 2342 2349
create 6
```

WHERE: 1
OK: 2354 2361
exec 6 2 3 4
OK:6: 7
kill 2
5: STOPPED
3: STOPPED
2: STOPPED
pingall
OK: 2; 3; 5;
exec 1 2 3 4
OK:1: 7
kill 6
6: STOPPED
pingall
OK: 2; 3; 5; 6;
exit
4: STOPPED
1: STOPPED

Выводы

Познакомился с серверами сообщений, отложенными вычислениями. Получил ценный опыт использования этих технологий для реализации распределенной системы по асинхронной обработке запросов.