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»

Управлении серверами сообщений. Применение отложенных вычислений. Интеграция программных систем друг с другом.

Студент: Попов Илья Павлович

Группа: М80-206Б-20

Вариант: 35

Преподаватель: Соколов Андрей Алексеевич

Дата: 25.12.2021

Оценка: 5

Подпись: _____

Москва, 2021

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. Список основных поддерживаемых команд:

Создание нового вычислительного узла

Формат команды: create id [parent]

id – целочисленный идентификатор нового вычислительного узла

parent – целочисленный идентификатор родительского узла. Если топологией не предусмотрено введение данного параметра, то его необходимо игнорировать (если его ввели)

Формат вывода: «Ok: pid», где pid – идентификатор процесса для созданного вычислительного узла

«Error: Already exists» - вычислительный узел с таким идентификатором уже существует

«Error: Parent not found» - нет такого родительского узла с таким идентификатором

«Error: Parent is unavailable» - родительский узел существует, но по каким-то причинам с ним не удается связаться

«Error: [Custom error]» - любая другая обрабатываемая ошибка

Пример:

> create 10 5 Ok: 3128

Примечания: создание нового управляющего узла осуществляется пользователем программы при помощи запуска исполняемого файла. Id и pid — это разные идентификаторы.

Исполнение команды на вычислительном узле

Формат команды: exec id [params]

id – целочисленный идентификатор вычислительного узла, на который отправляется команда Формат вывода:

«Ok:id: [result]», где result – результат выполненной команды

«Error:id: Not found» - вычислительный узел с таким идентификатором не найден

«Error:id: Node is unavailable» - по каким-то причинам не удается связаться с вычислительным узлом

«Error:id: [Custom error]» - любая другая обрабатываемая ошибка

Пример: Можно найти в описании конкретной команды, определенной вариантом задания.

Примечание: выполнение команд должно быть асинхронным. Т.е. пока выполняется команда на одном из вычислительных узлов, то можно отправить следующую команду на другой вычислительный узел.

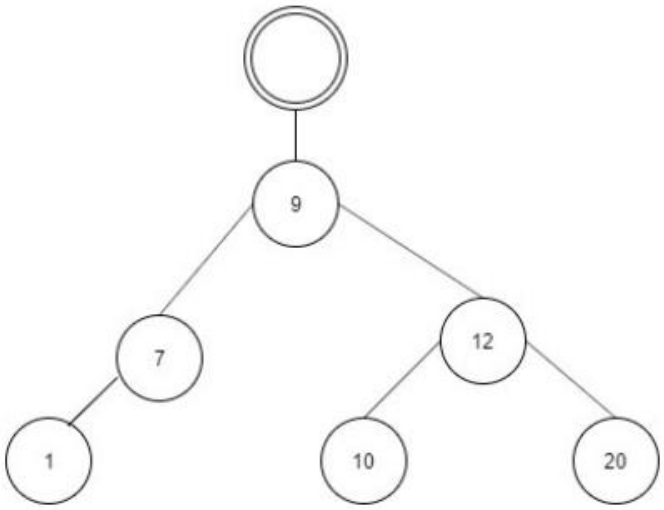
Вариант №35

Топология – 4

Тип команды – 1

Тип проверки доступности узлов - 2

Топология 3



Все вычислительные узлы хранятся в бинарном дереве поиска. [parent] — является необязательным параметром.

Топология 4

Аналогично топологии 3, но узлы находятся в идеально сбалансированном бинарном дереве. Каждый следующий узел должен добавляться в самое наименьшее поддерево.

Набор команд 1 (подсчет суммы n чисел)

Формат команды: exec id n k1 … kn id – целочисленный идентификатор вычислительного узла, на который отправляется команда n – количество складываемых чисел (от 1 до 108) k1 … kn – складываемые числа

Пример:

> exec 10 3 1 2 3 Ok:10: 6

Тип проверки доступности узлов

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

Листинг программы

BalancedTree.h

```
#ifndef BALANCED_TREE_H
#define BALANCED_TREE_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, 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 (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, 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 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:
    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
```

CalculationNode.h

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

class CalculationNode {
  private:
    zmq:: context_t context;
  public:
    zmq:: socket_t left, right, parent;
    int left_port, right_port, parent_port;
    int id, left_id = -2, right_id = -2, parent_id;
```

```
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);
    }
}
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 {
        string fail = "Error: can not create the calculation node";
        return fail;
    }
    int fork_id = fork();
    if (fork_id == 0) {
```

```
            if (execl("./client", "client",  to_string(child_id).c_str(),  to_string(port).c_str(),
to_string(id).c_str(), (char*)NULL) == -1) {
                cout << "Error: can not run the execl-command" <<  endl;
                exit(EXIT_FAILURE);
            }
        }
        else {
            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) {
                string fail = "Error: can not connect to the child";
                return fail;
            }
        }
    }
    string ping (int id) {
        string answer = "Ok: 0";
        if (this->id == id) {
            answer = "Ok: 1";
            return answer;
        }
```

```
    else if (left_id == id) {
        string message = "ping " +  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) {
        string message = "ping " +  to_string(id);
        send_message(right, message);
        try {
            message = receive_message(right);
            if (message == "Ok: 1") {
                answer = message;
            }
        }
        catch(int){}
    }
    return answer;
}

string sendstring (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 " +  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 " +  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;
}

string exec ( string string) {
   istringstream string_thread(string);
   int result = 0;
   int amount, number;
   string_thread >> amount;
   for (int i = 0; i < amount; ++i) {
      string_thread >> number;
      result += number;
   }
   std::string answer = "Ok: " +  to_string(id) + ": " +  to_string(result);
   return answer;
}

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  to_string(parent_id);
    }
    ~CalculationNode() {}
};
```

Client.cpp

```
#include <bits/stdc++.h>
#include "CalculationNode.h"
#include "ZMQFunctions.h"
#include "BalancedTree.h"



int main(int argc, char *argv[]) {

   CalculationNode node(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]));
```

```
while(true) {
    string message;
    string command;
    message = receive_message(node.parent);
    istringstream request(message);
    request >> command;
    if (command == "pid") {
        string answer =  to_string(getpid());
        send_message(node.parent, answer);
    }
    else if (command == "ping") {
        int child;
        request >> child;
        string answer = node.ping(child);
        send_message(node.parent, answer);
    }
    else if (command == "create") {
        int child;
        request >> child;
        string answer = node.create(child);
        send_message(node.parent, answer);
    }
    else if (command == "exec") {
        string str;
        getline(request, str);
        string answer = node.exec(str);
        send_message(node.parent, answer);
    }
    else if (command == "kill") {
        string answer = node.kill();
        send_message(node.parent, answer);
        disconnect(node.parent, node.parent_port);
        node.parent.close();
```

```
            break;
        }
    }
    return 0;
}
```

Makefile

```
files: server client

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

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

clean:
	rm  -rf server client
```

Server.cpp

```
#include <bits/stdc++.h>
#include "CalculationNode.h"
#include "ZMQFunctions.h"
#include "BalancedTree.h"

void menu(){
    cout << "Avaliable commands:\n";
    cout << "1. create <id>\n";
    cout << "2. exec <id> <n> <n1 n2... n>\n";
    cout << "3. ping <id>\n";
    cout << "4. kill <id>\n";
}



int main() {
```

```
string command;
CalculationNode node(-1, -1, -1);
string answer;
BalancedTree tree;

menu();

while (( cout << "> ") && ( cin >> command)) {

   if (command == "create") {
      int child; cin >> child;

      if (tree.Exist(child)) {
          cout << "Error: Already exists\n";
      }
      else {
         while (true) {
            int idParent = tree.FindID();
            if (idParent == node.id) {
               answer = node.create(child);
               tree.AddInTree(child, idParent);
               break;
            }
            else {
               string message = "create " +  to_string(child);
               answer = node.sendstring(message, idParent);
               if (answer == "Error: Parent not found") {
                  tree.AvailabilityCheck(idParent);
               }
               else {
                  tree.AddInTree(child, idParent);
                  break;
               }
```

```
            }
        }
        cout << answer << endl;
    }
}

else if (command == "exec") {

    if (!tree.Exist(child)) {
        cout << "Error: Node does not exist!\n";
    }

    string str;
    int child; cin >> child;
    getline(cin, str);

    if (!tree.Exist(child)) {
        cout << "Error: Parent is not existed\n";
    }
    else {
        string message = "exec " + str;
        answer = node.sendstring(message, child);
        cout << answer <<  endl;
    }
}

else if (command == "ping") {
    int child; cin >> child;

    if (!tree.Exist(child)) {
        cout << "Ok: 0\n";
    }
    else if (node.left_id == child || node.right_id == child) {
```

```
        answer = node.ping(child);
        cout << answer <<  endl;
    }
    else {
        string message = "ping " +  to_string(child);
        answer = node.sendstring(message, child);
        cout << answer <<  endl;
    }
}

else if (command == "kill") {
    int child; cin >> child;

    string message = "kill";
    if (!tree.Exist(child)) {
        cout << "Error: Parent is not existed\n";
    }
    else {
        answer = node.sendstring(message, child);
        if (answer != "Error: Parent not found") {
            tree.RemoveFromRoot(child);
            if (child == node.left_id){
                node.left_id = -2;
                unbind(node.left, node.left_port);
                answer = "Ok";
            }
            else if (child == node.right_id) {
                node.right_id = -2;
                unbind(node.right, node.right_port);
                answer = "Ok";
            }
            else {
                message = "clear " +  to_string(child);
```

```
                answer = node.sendstring(message,  stoi(answer));
            }
            cout << answer <<  endl;
          }
        }
    }
    else {
        cout << "Please enter correct command!\n\n";
        menu();
    }
  }
  node.kill();
  return 0;
}
```

ZMQFunctions.h

```
#ifndef ZMQ_H
#define ZMQ_H
#include <bits/stdc++.h>
#include <zmq.hpp>
const int MAIN_PORT = 4040;

using namespace std;

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

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;
    }
    string received_msg(static_cast<char*>(message.data()), message.size());
    return received_msg;
}

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

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

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

```
        }
    }
    return port;
}

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

#endif
```

Примеры работы

```
lunidep@lunidep-VirtualBox:~/Desktop/OS/lab6$ ./server
Avaliable commands:
1. create <id>
2. exec <id> <n> <n1 n2... n>
3. ping <id>
4. kill <id>
> create 5
Ok: 7746
> create 2
Ok: 7771
> create 7
Ok: 7774
> ping 2
Ok: 1
> kill 2
Ok
> ping 2
Ok: 0
> exec 7 3 1 2 3
Ok: 7: 6
```

Вывод

В процессе выполнения данной лабораторной работы мною была реализована распределенная система по асинхронной обработке запросов. В моей программе использовался протокол передачи данных tcp, в котором, в отличие от ipc общение между процессами происходит через определенные порты, а не через временные файлы.

Обмен сообщений происходит посредством функций библиотеки ZMQ, а в частности, ее паттерном «Request – Reply». Это один из самых простых и прямолинейных паттернов, который своей реализацией очень напоминает pipe. Материала для реализации данной лабораторной работы потребовалось довольно много и я получил полезный опыт изучения англоязычной документации.

Также хорошей тренировкой стала реализация идеально сбалансированного бинарного дерева на C++.