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»**

Студент: Путилин Д.Н.

Группа: М8О-207Б-21

Вариант: 36

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

[https://github.com/](https://github.com/Liguha/OS)putilin21dn/OC

**Постановка задачи**

**Цель работы**

Целью является приобретение практических навыков в:

* Управлении серверами сообщений (№6)
* Применение отложенных вычислений (№7)
* Интеграция программных систем друг с другом (№8)

**Задание**

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: create, exec, remove, heartbit.

**Общие сведения о программе**

Программа распределительного узла компилируется из файла annotations\_node.cpp, программа вычислительного узла компилируется из файла compiling\_node.cpp. В программе используется библиотека для работы с потоками и мьютексами, а также сторонняя библиотека для работы с сервером сообщений ZeroMQ.

**Общий метод и алгоритм решения**

Распределительный узел хранит в себе двоичное дерево, элементами которого являются async\_node – структуры, хранящие в себе id вычислительного узла, порт для связи с ним, поток отправляющий/принимающий запросы/ответы к узлу, очередь отправленных на узел запросов и мьютекс, обеспечивающий возможность работать с этой очередью в несколько потоков.

В функции main запускается бесконечный цикл обработки пользовательских запросов, при получении запроса он отправляется в очередь обработчика соответствующего узла, откуда позже будет извлечён потоком обработки и переслан требуемому узлу (обеспечение асинхронности). При удалении узла у записи, соответствующей удаляемому узлу устанавливается переменная активности в false, благодаря чему обработчик, отправив на узел запрос удаления и получив ответ, выходит из цикла и очищает память от уже ненужной записи. Подобные действия применяются и ко всем дочерним узлам удаляемого.

Heartbit. После запуска команды и введенного времени создается поток, который будет через время time проходится по всем узлам и проверяет их на установку соединения. В результате у нас в случае усспеха узел будет иметь 1, в противном случае – 0. Дальше уже в следующих командах мы можем удалять или добавлять узлы. Heartbit будет работать также и с ними. Для проверки работоспособности узла используем команду ping, которая в случае успеха возвращет Ок:1, иначе Ок:-1.

**Исходный код**

|  |
| --- |
| **annotations\_node.cpp** |
| #include "zmq.h"  #include "string.h"  #include "unistd.h"  #include "stdlib.h"  #include "pthread.h"  #include <iostream>  #include <queue>  #include <vector>  #include "header.hpp"  #include <ctime>  #include <cstdlib>  #define CHECK\_ERROR(expr, stream, act) \  do \  { \  int res = (expr); \  if (res != 0) \  { \  std::cerr << stream; \  act; \  } \  } while (0)  #define CHECK\_ZMQ(expr, stream, act) \  do \  { \  int res = (expr); \  if (res == -1) \  { \  std::cerr << stream; \  act; \  } \  } while (0)  using namespace std;  string protocol = "tcp://localhost:";  int MIN\_PORT;  vector<pair<int,bool>> act;  int times;  bool flag = true;  void\* async\_node\_thd(void\*);  struct async\_node  {  int id;  string port;  bool active;  async\_node\* L;  async\_node\* R;  pthread\_mutex\_t mutex;  pthread\_t thd;  queue <vec> q;    async\_node(int i)  {  id = i;  port = protocol + to\_string(i);  active = true;  L = nullptr;  R = nullptr;  CHECK\_ERROR(pthread\_mutex\_init(&mutex, NULL), "Error:" << i - MIN\_PORT << ": Gateway mutex error\n", return);  CHECK\_ERROR(pthread\_create(&thd, NULL, async\_node\_thd, this), "Error:" << i - MIN\_PORT << ": Gateway thread error\n", return);  CHECK\_ERROR(pthread\_detach(thd), "Error:" << i << ": Gateway thread error\n", return);  }  void make\_query(vec V)  {  CHECK\_ERROR(pthread\_mutex\_lock(&mutex), "Error:" << id - MIN\_PORT << ": Gateway mutex lock error\n", active = false; return);  q.push(V);  CHECK\_ERROR(pthread\_mutex\_unlock(&mutex), "Error:" << id - MIN\_PORT << ": Gateway mutex unlock error\n", active = false);  }  ~async\_node()  {  pthread\_mutex\_destroy(&mutex);  }  };  async\_node\* find\_node\_exec(async\_node\* ptr, int id)  {  if (ptr == nullptr)  return nullptr;  if (ptr->id > id)  return find\_node\_exec(ptr->L, id);  if (ptr->id < id)  return find\_node\_exec(ptr->R, id);  return ptr;  }  async\_node\* find\_node\_create(async\_node\* ptr, int id)  {  if (ptr == nullptr)  return nullptr;  if (ptr->L == nullptr && ptr->id > id)  return ptr;  if (ptr->R == nullptr && ptr->id < id)  return ptr;  if (ptr->id > id)  return find\_node\_create(ptr->L, id);  if (ptr->id < id)  return find\_node\_create(ptr->R, id);  return nullptr;  }  bool destroy\_node(async\_node\*& ptr, int id)  {  if (ptr == nullptr)  return false;  if (ptr->id > id)  return destroy\_node(ptr->L, id);  if (ptr->id < id)  return destroy\_node(ptr->R, id);  ptr->active = false;  ptr->make\_query({REMOVE});  if (ptr->L != nullptr)  destroy\_node(ptr->L, ptr->L->id);  if (ptr->R != nullptr)  destroy\_node(ptr->R, ptr->R->id);  ptr = nullptr;  return true;  }  bool pings(int);  void\* async\_node\_thd(void\* ptr)  {  async\_node\* node = (async\_node\*)ptr;  void\* context = zmq\_ctx\_new();  void\* req = zmq\_socket(context, ZMQ\_REQ);  CHECK\_ZMQ(zmq\_connect(req, node->port.c\_str()), "Error: Connection with" << node->id - MIN\_PORT << "\n",);  while (node->active)  {  if (node->q.empty())  continue;  CHECK\_ERROR(pthread\_mutex\_lock(&node->mutex), "Error:" << node->id - MIN\_PORT << ": Gateway mutex lock error\n", node->active = false; break);  vec V = node->q.front();  node->q.pop();  CHECK\_ERROR(pthread\_mutex\_unlock(&node->mutex), "Error:" << node->id - MIN\_PORT << ": Gateway mutex unlock error\n", node->active = false; break);  switch (V.ex)  {  case CREATE:  {  zmq\_msg\_t msg;  msg = vec2msg(V);  CHECK\_ZMQ(zmq\_msg\_send(&msg, req, 0), "Error:" << node->id - MIN\_PORT << ": Message error\n", break);  int pid;  CHECK\_ZMQ(zmq\_recv(req, &pid, sizeof(int), 0), "Error:" << node->id - MIN\_PORT << ": Message error\n", break);  if (V.id < node->id)  node->L = new async\_node(V.id);  else  node->R = new async\_node(V.id);  cout << "Ok: " << pid << '\n';  zmq\_msg\_close(&msg);  break;  }  case EXEC:  {  zmq\_msg\_t msg;  msg = vec2msg(V);  CHECK\_ZMQ(zmq\_msg\_send(&msg, req, 0), "Error:" << node->id - MIN\_PORT << ": Message error\n", break);  string ans;  CHECK\_ZMQ(zmq\_msg\_recv(&msg, req, 0), "Error:" << node->id - MIN\_PORT << ": Message error\n", break);  cout << "Ok:" << node->id - MIN\_PORT << ":";  ans = msg2str(msg);  for(int i=0; i<ans.length();++i){  if(ans[i]!='#')  cout << ans[i];  else{  if(i==(ans.length()-1))  cout << '\n';  else{  cout << ";";  }  }  }  zmq\_msg\_close(&msg);  break;  }  case REMOVE:  {  CHECK\_ZMQ(zmq\_send(req, &V.ex, sizeof(int), 0), "Error:" << node->id - MIN\_PORT << ": Message error\n", break);  int ans;  CHECK\_ZMQ(zmq\_recv(req, &ans, sizeof(int), 0), "Error:" << node->id - MIN\_PORT << ": Message error\n", break);  break;  }  }  }  zmq\_close(req);  zmq\_ctx\_destroy(context);  delete node;  return NULL;  }  async\_node\* tree = nullptr;  struct th{  async\_node \*tree;  int time;  };  bool pings(int id)  {    string port = protocol + to\_string(id);  string ping = "inproc://ping" + to\_string(id);  void\* context = zmq\_ctx\_new();  void \*req = zmq\_socket(context, ZMQ\_REQ);  zmq\_socket\_monitor(req, ping.c\_str(), ZMQ\_EVENT\_CONNECTED | ZMQ\_EVENT\_CONNECT\_RETRIED);  void \*soc = zmq\_socket(context, ZMQ\_PAIR);  zmq\_connect(soc, ping.c\_str());  zmq\_connect(req, port.c\_str());  zmq\_msg\_t msg;  zmq\_msg\_init(&msg);  zmq\_msg\_recv(&msg, soc, 0);  uint8\_t\* data = (uint8\_t\*)zmq\_msg\_data(&msg);  uint16\_t event = \*(uint16\_t\*)(data);  zmq\_close(req);  zmq\_close(soc);  zmq\_msg\_close(&msg);  zmq\_ctx\_destroy(context);  return event % 2;  }    void ping(int id)  {  for (auto x : act){  if(x.first == id){  cout << "Ok : 1" << '\n';  return;  }  }  cout << "Ok : -1" << '\n';  return;  }  void \* heartbit(void \*args){  // int times = \*((int\*)(&args));  int time = ((th \*)args)->time;  async\_node\* node = ((th \*)args)->tree;  while(times>0){  queue <async\_node\*> q;  if (node != nullptr)  q.push(node);  act.resize(0);    while (!q.empty())  {  async\_node\* ptr = q.front();  q.pop();  if (ptr->L != nullptr)  q.push(ptr->L);  if (ptr->R != nullptr)  q.push(ptr->R);  bool check = pings(ptr->id);    act.push\_back({ptr->id,check});  }  sleep(4\*times/1000);  }  return NULL;  }  int main()  {  srand( time(0) );  MIN\_PORT = 1024 + rand()%1000;  while (true)  {  string command;  cin >> command;    if (command == "create")  {  int id;  cin >> id;  id += MIN\_PORT;  vec V;  V.ex = CREATE;  V.id = id;  if (tree == nullptr)  {  string id\_str = to\_string(id);  int pid = fork();  if (pid == 0)  CHECK\_ERROR(execl("server", id\_str.c\_str(), NULL), "Error:" << id - MIN\_PORT << ": Creating error\n", break);  cout << "Ok: " << pid << '\n';  tree = new async\_node(id);  }  else  {  async\_node\* node = find\_node\_create(tree, id);  if (node != nullptr){  if (!pings(node->id))  {  cerr << "Error:" << id - MIN\_PORT << ": Parent is unavailable\n";  continue;  }  node->make\_query({CREATE, id});  }  else  cerr << "Error: Already exists\n";  }  }  if (command == "exec")  {  int id;  cin >> id;  id += MIN\_PORT;  string str,substr;  cin >> str >> substr;  vec V;  V.ex = EXEC;  V.id = id;  V.lenstr = str.length();  V.str = str;  V.lensubstr = substr.length();  V.substr = substr;  if (!pings(id))  {  cerr << "Error:" << id - MIN\_PORT << ": Node is unavailable\n";  continue;  }  async\_node\* node = find\_node\_exec(tree, id);  if (node != nullptr)  node->make\_query(V);  else  cerr << "Error:" << id - MIN\_PORT << ": Not found\n";  }  if (command == "remove")  {  int id;  cin >> id;  id += MIN\_PORT;  if (!pings(id))  {  cerr << "Error:" << id - MIN\_PORT << ": Node is unavailable\n";  continue;  }  bool state = destroy\_node(tree, id);  if (state)  cout << "Ok\n";  else  cerr << "Error: Not found\n";  }  if (command == "heartbit"){    cin >> times;  vector<pthread\_t> threads = vector<pthread\_t>(1);  th V;  V.time = times;  V.tree = tree;  if(flag){  if (times>0){    flag = false;  if(int err = pthread\_create(&threads[0],NULL, heartbit, (void \*)&V))    if (pthread\_join(threads[0],NULL) != 0) {  cout << "Can't wait for thread\n";  }  }  }  }  if(command == "ping"){  int id;  cin >> id;  id += MIN\_PORT;  ping(id);  }  }  } |

|  |
| --- |
| **compiling\_node.cpp** |
| #include <zmq.h>  #include <unistd.h>  #include <string.h>  #include "header.hpp"  #include <iostream>  #include <vector>  #define CREATE 1  #define EXEC 0  #define REMOVE -1  #define CHECK\_ERROR(expr) \  do \  { \  int res = (expr); \  if (res == -1){ \  std::cout << errno << '\n'; \  return -1;} \  } while (0)  using namespace std;  string z\_func(int lensubstr, string &s){  int n = s.size();  vector<int> z(n,0);  string ans;  int l=-1,r=-1;  z[0]=n;  for(int i=1;i<n;++i){  if(i<=r){  z[i]=min(z[i-l], r-i);  }  while((i+z[i]<n) && (s[z[i]]==s[i+z[i]])){  ++z[i];  }  if(i+z[i]>r){  r=i+z[i];  l=i;  }  }  for(int i=lensubstr; i<n;++i){  if(z[i]==lensubstr){  ans += to\_string(i-lensubstr-1) + "#";  }  }  return ans;  }  int main(int argc, char\* argv[])  {  int id = atoi(argv[0]);  string port = "tcp://\*:" + to\_string(id);  void \*context = zmq\_ctx\_new();  void \*responder = zmq\_socket(context, ZMQ\_REP);  zmq\_bind(responder, port.c\_str());    while (true)  {  zmq\_msg\_t msg;  CHECK\_ERROR(zmq\_msg\_init(&msg));  // cout << "heresa" <<'\n';  CHECK\_ERROR(zmq\_msg\_recv(&msg, responder, 0));  // cout << "heresdfs" <<'\n';  vec V = msg2vec(msg);        switch (V.ex)  {  case CREATE:  {  string id\_str = to\_string(V.id);  // cout << "here2" << '\n';  int pid = fork();  // cout << "here 2" << pid <<'\n';  if (pid == -1)  return -1;  if (pid == 0)  CHECK\_ERROR(execl("server", id\_str.c\_str(), NULL));  else  CHECK\_ERROR(zmq\_send(responder, &pid, sizeof(int), 0));  break;  }  case EXEC:  {  string s = V.substr + "#" + V.str;  string ans;  ans = z\_func(V.lensubstr,s);    zmq\_msg\_t msg = str2msg(ans);  zmq\_msg\_send(&msg,responder,0);  break;  }  case REMOVE:  {  zmq\_send(responder, &id, sizeof(int), 0);  zmq\_close(responder);  zmq\_ctx\_destroy(context);  return 0;  }  }  CHECK\_ERROR(zmq\_msg\_close(&msg));  }  } |

|  |
| --- |
| **Header.hpp** |
| #ifndef CONSTANTS\_HPP  #define CONSTANTS\_HPP  #include <string>  #include <zmq.h>  #include <string.h>  #include <iostream>  using namespace std;  #define CREATE 1  #define EXEC 0  #define REMOVE -1  struct vec{  int ex;  int id;  int lenstr;  string str;  int lensubstr;  string substr;  };  zmq\_msg\_t vec2msg(vec V){    zmq\_msg\_t msg;  zmq\_msg\_init\_size(&msg, 4 \* sizeof(int) + V.lenstr+V.lensubstr);  void \* z = zmq\_msg\_data(&msg);  memcpy(z, &V.ex, sizeof(int));  memcpy(z = (void \*)((unsigned long)z + sizeof(int)), &V.id, sizeof(int));  memcpy(z = (void \*)((unsigned long)z + sizeof(int)), &V.lenstr, sizeof(int));  memcpy(z = (void \*)((unsigned long)z + sizeof(int)), V.str.c\_str(), sizeof(char)\*V.lenstr);  memcpy(z = (void \*)((unsigned long)z + sizeof(char)\*V.lenstr), &V.lensubstr, sizeof(int));  memcpy(z = (void \*)((unsigned long)z + sizeof(int)), V.substr.c\_str(), sizeof(char)\*V.lensubstr);  return msg;  }  vec msg2vec(zmq\_msg\_t msg){  void \* z = zmq\_msg\_data(&msg);  vec V;    memcpy(&V.ex, z, sizeof(int));  memcpy(&V.id, z=(void \*)((unsigned long)z + sizeof(int)), sizeof(int));  memcpy(&V.lenstr, z=(void \*)((unsigned long)z + sizeof(int)), sizeof(int));  char \* s = (char \*)calloc(V.lenstr,sizeof(char));  memcpy(s, z = (void \*)((unsigned long)z + sizeof(int)) , V.lenstr);  V.str = s;  free(s);  memcpy(&V.lensubstr, z=(void \*)((unsigned long)z + sizeof(char)\*V.lenstr), sizeof(int));  s = (char \*)calloc(V.lensubstr,sizeof(char));  s = s + '\0';  memcpy(s, z=(void \*)((unsigned long)z + sizeof(int)) , V.lensubstr);  V.substr = s;  free(s);  return V;  }  zmq\_msg\_t str2msg(string str){  zmq\_msg\_t msg;  zmq\_msg\_init\_size(&msg, str.length());  void \* z = zmq\_msg\_data(&msg);  memcpy(z, str.c\_str(), sizeof(char)\*str.length());  return msg;  }  string msg2str(zmq\_msg\_t msg){  void \* z = zmq\_msg\_data(&msg);  int len = zmq\_msg\_size(&msg);  char \* s = (char \*)calloc(len,sizeof(char));  memcpy(s,z,len\*sizeof(char));  return s;  }  #endif |

**Демонстрация работы программы**

dmitry@dmitry-VirtualBox:~/Рабочий стол/OC/lab6-8/build$ ./client

create 10

Ok: 9956

create 20

Ok: 9964

create 30

Ok: 9976

exec 30

abasbdbabab

ab

Ok:30:0;7;9

remove 30

Ok

exec 3

asndas

sa

Error:3: Node is unavailable

exec 20

hello

a

Ok:20:-1

heartbit 1000

ping 20

Ok : 1

ping 20 // after kill 9964

Ok : -1

**Выводы**

Составлена и отлажена программа на языке C++, осуществляющая отложенные вычисления на нескольких вычислительных узлах. Пользователь управляет программой через распределительный узел, который перенаправляет запросы в асинхронном режиме.