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: Яруллин А.Р.

Группа: М8О-207Б-21

Вариант: 25

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/nesrayr/OS-labs

**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, pingall.

**Общие сведения о программе**

Программа распределительного узла компилируется из файла distribution\_node.cpp, программа вычислительного узла компилируется из файла computing\_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.

Pingall выполняет обход дерева в ширину, проверяя доступность каждого узла. Проверка осуществляется попыткой подключиться к узлу: если монитор сокета получает сигнал ZMQ\_EVENT\_CONNECTED, то узел считается доступным, если же вместо этого приходит ZMQ\_EVENT\_CONNECT\_RETRIED, то узел считается недоступным.

**Исходный код**

|  |
| --- |
| **distribution\_node.cpp** |
| #include "zmq.h"  #include "string.h"  #include "unistd.h"  #include "stdlib.h"  #include "pthread.h"  #include <iostream>  #include <queue>  #include <vector>  #define CREATE 1  #define EXEC 0  #define REMOVE -1  #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)  const int MIN\_PORT = 1024;  using namespace std;  string protocol = "tcp://localhost:";  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 <vector <int>> 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(vector <int> 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;  }  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);  vector <int> 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[0])  {  case CREATE:  {  zmq\_msg\_t msg;  CHECK\_ZMQ(zmq\_msg\_init\_size(&msg, 2 \* sizeof(int)), "Error:" << node->id - MIN\_PORT << ": Message error\n", break);  memcpy(zmq\_msg\_data(&msg), &v[0], 2 \* sizeof(int));  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[1] < node->id)  node->L = new async\_node(v[1]);  else  node->R = new async\_node(v[1]);  cout << "Ok: " << pid << '\n';  zmq\_msg\_close(&msg);  break;  }  case EXEC:  {  zmq\_msg\_t msg;  int len = sizeof(int) \* v.size();  CHECK\_ZMQ(zmq\_msg\_init\_size(&msg, len), "Error:" << node->id - MIN\_PORT << ": Message error\n", break);  memcpy(zmq\_msg\_data(&msg), &v[0], len);  CHECK\_ZMQ(zmq\_msg\_send(&msg, req, 0), "Error:" << node->id - MIN\_PORT << ": Message error\n", break);  long long S;  CHECK\_ZMQ(zmq\_recv(req, &S, sizeof(long long), 0), "Error:" << node->id - MIN\_PORT << ": Message error\n", break);  cout << "Ok:" << node->id - MIN\_PORT << ':' << S << '\n';  zmq\_msg\_close(&msg);  break;  }  case REMOVE:  {  CHECK\_ZMQ(zmq\_send(req, &v[0], 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;  }  bool ping(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;  }  async\_node\* tree = nullptr;  int main()  {  while (true)  {  string command;  cin >> command;  if (command == "create")  {  int id;  cin >> id;  id += MIN\_PORT;  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 (!ping(node->id))  {  cerr << "Error:" << id - MIN\_PORT << ": Parent is unavailable\n";  continue;  }  if (node != nullptr)  node->make\_query({CREATE, id});  else  cerr << "Error: Already exists\n";  }  }  if (command == "exec")  {  int id, n;  cin >> id >> n;  id += MIN\_PORT;  vector <int> v(n + 2);  v[0] = EXEC;  v[1] = n;  for (int i = 0; i < n; i++)  cin >> v[i + 2];  if (!ping(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 (!ping(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 == "unsafe\_remove")  {  int id;  cin >> id;  id += MIN\_PORT;  bool state = destroy\_node(tree, id);  if (state)  cout << "Ok\n";  else  cerr << "Error: Not found\n";  }  if (command == "pingall")  {  queue <async\_node\*> q;  if (tree != nullptr)  q.push(tree);  vector <int> v;  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 = ping(ptr->id);  if (!check)  v.push\_back(ptr->id - MIN\_PORT);  }  if (v.empty())  v.push\_back(-1);  cout << "Ok: ";  for (int i = 0; i < v.size(); i++)  {  if (i != 0)  cout << ';';  cout << v[i];  }  cout << '\n';  }  }  } |

|  |
| --- |
| **computing\_node.cpp** |
| #include <zmq.h>  #include <unistd.h>  #include <string.h>  #include <iostream>  #include <vector>  #define CREATE 1  #define EXEC 0  #define REMOVE -1  #define CHECK\_ERROR(expr) \  do \  { \  int res = (expr); \  if (res == -1) \  return -1; \  } while (0)  using namespace std;  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);  CHECK\_ERROR(zmq\_bind(responder, port.c\_str()));  while (true)  {  zmq\_msg\_t msg;  CHECK\_ERROR(zmq\_msg\_init(&msg));  CHECK\_ERROR(zmq\_msg\_recv(&msg, responder, 0));  int\* data = (int\*)zmq\_msg\_data(&msg);  int t = \*data;  switch (t)  {  case CREATE:  {  int n = \*(++data);  string id\_str = to\_string(n);  int pid = fork();  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:  {  int n = \*(++data);  vector <int> v(n);  memcpy(&v[0], (++data), n \* sizeof(int));  long long S = 0;  for (int i = 0; i < n; i++)  S += v[i];  CHECK\_ERROR(zmq\_send(responder, &S, sizeof(long long), 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));  }  } |

**Демонстрация работы программы**

yarullin@yarullin-OMEN-by-HP-Laptop-15-dc1xxx:~/OS-Labs/lab678/build$ ./client

create 20

Ok: 2565

create 10

Ok: 2583

create 30

Ok: 2593

exec 20 1 2

Ok:20:2

exec 30 2 1 3

Ok:30:4

exec 10 3 0 0 7

Ok:10:7

pingall

Ok: -1

remove 10

Ok

exec 10 1 1

Error:10: Not found

exec 20 1 1

Ok:20:1

remove 20

Ok

exec 30 1 1

Error:30: Not found

exec 20 1 1

Error:20: Not found

create 20

Ok: 2759

create 10

Ok: 2767

create 30

Ok: 2775

pingall

Ok: 30 (процесс 2775 был предварительно убит)

pingall

Ok: 10;30 (дополнительно убит процесс 2767 для демонстрации pingall)

remove 20

Ok

pingall

Ok: -1

remove 20

Error: Not found

create 20

Ok: 2929

pingall

Ok: 20 (заранее был убит процесс 2929 для демонстрации ошибки при попытке exec)

exec 20 1 1

Error:20: Node is unavailable

**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.