Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №5-7 по курсу

«Операционные системы»

Группа: М80-206Б-22

Студент: Коломытцева Е.А.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 15.02.24

Москва, 2024

Цель работы:

Целью является приобретение практических навыков в:

Управлении серверами сообщений (№5)

Применение отложенных вычислений (№6)

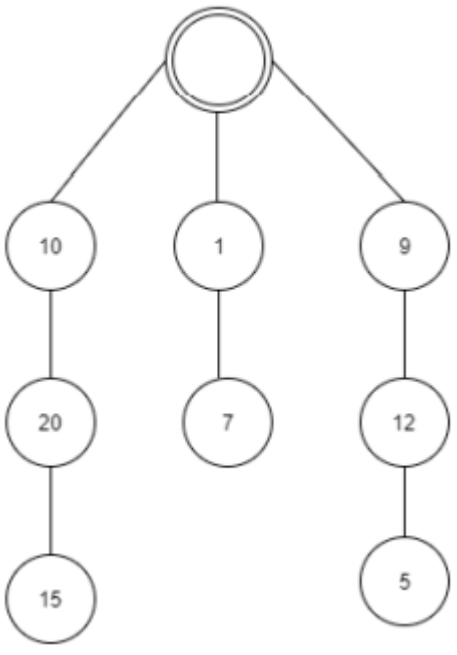
Интеграция программных систем друг с другом (№7)

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.

Вариант 38.



Топология 1. Все вычислительные узлы находятся в списках. Есть только один управляющий узел. Чтобы добавить новый вычислительный узел к управляющему, то необходимо выполнить команду: create id -1.

Тип команд 3. (локальный таймер) Формат команды сохранения значения: exec id subcommand

subcommand – одна из трех команд: start, stop, time.

start – запустить таймер

stop – остановить таймер

time – показать время локального таймера в миллисекундах

Тип проверки доступности узлов 3. Формат команды: heartbit time

Каждый узел начинает сообщать раз в time миллисекунд о том, что он работоспособен. Если от узла нет сигнала в течении 4*time миллисекунд, то должна выводится пользователю строка: «Heartbit: node id is unavailable now», где id – идентификатор недоступного вычислительного узла.

Общий метод и алгоритм решения

Для реализации связи между управляющим узлом и исполняющими я буду использовать очередь сообщений из библиотеки *ZeroMQ*.

Очередь сообщений предоставляет гарантии, что сообщение будет доставлено независимо от того, что происходит. Очередь сообщений позволяет асинхронно взаимодействовать между слабо связанными компонентами, а также обеспечивает строгую последовательность очереди.

ZeroMQ - это не системой очередей сообщений типа WebSphereMQ, или RabbitMQ, это библиотека, которая дает нам инструменты для создания собственной системы очередей сообщений. Её еще называют сокетами на стероидах.

Сокет — это виртуальная конструкция из IP-адреса и номера порта, предназначенная для связи приложений или компьютеров между собой. Еще это именованный канал - FIFO, именованный pipe.

Использованные системные вызовы из библиотеки *ZeroMQ:*

socket.bind("tcp://*IP*:*port*") - установление связи socket по IP адресу и указанному порту port

- socket.set(zmq::sockopt::, "filter prefix") - настройка подписки, принимает только те сообщения, которые начинаются с префикса: "filter prefix"
- socket.connect("tcp://*IP*:*port*") - соединение с указанным портом
- socket.disconnect("tcp://*IP*:*port*") - отсоединение от порта
- socket.recv(reply, zmq::recv_flags) - принятие сообщения
- socket.send(request1, zmq::send_flags) - отправка сообщения

Использование адреса 127.0.0.1 позволяет устанавливать соединение и передавать информацию для программ-серверов, работающих на том же компьютере, что и программа-клиент, независимо от конфигурации аппаратных сетевых средств компьютера.

Код программы

topology.h:

```
#include <iostream>
#include <list>
#include <stdexcept>

class topology {
private:
 std::list<std::list<int>> container;

public:
 void insert(int id, int parent_id) {
   if (parent_id == -1) {
```

```
    std::list<int> new_list;
    new_list.push_back(id);
    container.push_back(new_list);
  } else {
    int list_id = find(parent_id);
    if (list_id == -1) {
      throw std::runtime_error("Wrong parent id");
    }
    auto it1 = container.begin();
    std::advance(it1, list_id);
    for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {
      if (*it2 == parent_id) {
        it1->insert(++it2, id);
        return;
      }
    }
  }
}
int find(int id) {  // проходимся по спискам в контейнере и ищем элемент
  int cur_list_id = 0;
  for (auto it1 = container.begin(); it1 != container.end(); ++it1) {
    for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {
      if (*it2 == id) {
        return cur_list_id;
      }
    }
    ++cur_list_id;
  }
  return -1;
}

void erase(int id) {
  int list_id = find(id);
  if (list_id == -1) {
    throw std::runtime_error("Wrong id");
  }
  auto it1 = container.begin();
  std::advance(it1, list_id);
  for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {
    if (*it2 == id) {
      it1->erase(it2, it1->end());
      if (it1->empty()) {
        container.erase(it1);
      }
      return;
    }
  }
}
```

```
 int get_first_id(int list_id) {
   auto it1 = container.begin();
   std::advance(it1, list_id);
   if (it1->begin() == it1->end()) {
     return -1;
   }
   return *(it1->begin());
 }

 int get_children_count(topology& network, int id) {
   int list_id = network.find(id);
   if (list_id == -1) {
     throw std::runtime_error("Wrong id");
   }
   auto it1 = container.begin();
   std::advance(it1, list_id);
   return it1->size();
 }

 bool hasChildren(int parent_id) const {
   for (const auto& list : container) {
     if (!list.empty() && list.front() == parent_id && list.size() > 1) {
       return true;
     }
   }
   return false;
 }
};
```

zmq_functions.h:

```
#include <iostream>
#include <zmq.hpp>

#define MAIN_PORT 6060

void sendMessage(zmq::socket_t& socket, const std::string& msg) {
 zmq::message_t message(msg.size());  // создается объект сообщение, которое
                                      // будет отправлено через сокет
 memcpy(message.data(), msg.c_str(), msg.size());
 socket.send(message);
}

std::string receiveMessage(zmq::socket_t& socket) {
 zmq::message_t message;
 bool charsRead = false;  // статус успешности приема сообщения
 try {
   charsRead =
       socket.recv(&message);  // извлекаем сообщение из сокета и сохраняем
 } catch (const zmq::error_t& e) {
```

```
    std::cerr << "ZeroMQ error: " << e.what() << std::endl;
  }
  if (!charsRead) {
    return "Error: failed to receive message from node";
  }
  std::string receivedMsg(
      static_cast<char*>(message.data()),
      message
          .size());  // копируем сообщение в строку с помощьюпреобразования
                     // указателя message.data() в указатель на массив символов
  return receivedMsg;
}

void connectToNode(zmq::socket_t& socket, int id) {
  std::string address = "tcp://127.0.0.1:" + std::to_string(MAIN_PORT + id);
  try {
    socket.connect(address);
  } catch (const zmq::error_t& e) {
    std::cerr << "ZeroMQ error: " << e.what() << std::endl;
  }
}

void disconnectFromNode(zmq::socket_t& socket, int id) {
  std::string address = "tcp://127.0.0.1:" + std::to_string(MAIN_PORT + id);
  try {
    socket.disconnect(address);
  } catch (const zmq::error_t& e) {
    std::cerr << "ZeroMQ error: " << e.what() << std::endl;
  }
}

void bindToAddress(zmq::socket_t& socket,
                   int id) {  // привязка сокета к указанному адресу, чтобы
                              // другие узлы могли подключиться к этому сокету
  std::string address = "tcp://127.0.0.1:" + std::to_string(MAIN_PORT + id);
  try {
    socket.bind(address);
  } catch (const zmq::error_t& e) {
    std::cerr << "ZeroMQ error: " << e.what() << std::endl;
  }
}

void unbindFromAddress(zmq::socket_t& socket, int id) {
  std::string address = "tcp://127.0.0.1:" + std::to_string(MAIN_PORT + id);
  try {
    socket.unbind(address);
  } catch (const zmq::error_t& e) {
    std::cerr << "ZeroMQ error: " << e.what() << std::endl;
  }
```

```
}
```

timer.h:

```
#include <iostream>
#include <chrono>
#include <ctime>
#include <cmath>

class Timer {
public:
   void start() {
       m_StartTime = std::chrono::system_clock::now();
       m_bRunning = true;
   }

   void stop() {
       m_EndTime = std::chrono::system_clock::now();
       m_bRunning = false;
   }

   double elapsedMilliseconds() {
       std::chrono::time_point<std::chrono::system_clock> endTime;

       if(m_bRunning)
       {
           endTime = std::chrono::system_clock::now();
       }
       else
       {
           endTime = m_EndTime;
       }

       return std::chrono::duration_cast<std::chrono::milliseconds>(endTime -
m_StartTime).count();
   }

   double elapsedSeconds() {
       return elapsedMilliseconds() / 1000.0;
   }

private:
   std::chrono::time_point<std::chrono::system_clock> m_StartTime;
   std::chrono::time_point<std::chrono::system_clock> m_EndTime;
   bool m_bRunning = false;
};
```

control.cpp:

```
#include <unistd.h>

#include <set>
#include <sstream>

#include "topology.h"
#include "zmq_functions.h"

int main() {
 topology network;
 std::vector<zmq::socket_t> branches;  // сокеты для подключения к узлам
 zmq::context_t context;

 std::string cmd;
 while (std::cin >> cmd) {
       if (cmd == "create") {
     int node_id, parent_id;
     std::cin >> node_id >> parent_id;
     if (parent_id != -1 && network.hasChildren(parent_id)) {
       std::cout << "Error: parent node has children" << std::endl;
     } else {
       if (network.find(node_id) != -1) {
         std::cout << "Error: already exists" << std::endl;
       } else if (parent_id == -1) {
         pid_t pid =
             fork();  // создаем дочерний процесс для вычислительных узлов
         if (pid < 0) {
           perror("Can't create new process");
           return -1;
         }
         if (pid == 0) {
           if (execl("./counting", "./counting",
                     std::to_string(node_id).c_str(), "-1", NULL) < 0) {
             perror("Can't execute new process");
             return -2;
           }
         }
         branches.emplace_back(
             context,
             ZMQ_REQ);  // создаем сокет для отправки запросов
                        // в сеть и помещаем в вектор сокетов
         branches.back().setsockopt(
             ZMQ_SNDTIMEO,
             5000);  // устанавливаем таймаут ожидания отправки сообщения
                     // (5сек) последнему сокету
         bindToAddress(branches.back(),
                       node_id);  // привязываем сокет к адресу
         sendMessage(
             branches.back(),
```

```
            std::to_string(node_id) +
                "pid");  // отправляем сообщение с идентификатором узла

        std::string reply =
            receiveMessage(branches.back());  // получаем ответ
        std::cout << reply << std::endl;
        network.insert(node_id, parent_id);  // добавляем узел в сеть
      } else if (network.find(parent_id) == -1) {
        std::cout << "Error: parent not found" << std::endl;
      } else {
        int branch = network.find(parent_id);
        sendMessage(branches[branch], std::to_string(parent_id) + " create " +
                                          std::to_string(node_id));

        std::string reply = receiveMessage(branches[branch]);
        std::cout << reply << std::endl;
        network.insert(node_id, parent_id);
      }
    }
  } else if (cmd == "exec") {
    int dest_id;
    std::string numbers;
    std::cin >> dest_id;
    std::getline(std::cin >> std::ws,
                 numbers);  // пропускаем пробелы и считываем числа
    int branch = network.find(dest_id);
    if (branch == -1) {
      std::cout << "ERROR: incorrect node id" << std::endl;
    } else {
      sendMessage(branches[branch],
                  std::to_string(dest_id) + " exec " + numbers);
      std::string reply = receiveMessage(branches[branch]);
      std::cout << reply << std::endl;
    }
  } else if (cmd == "kill") {
    int id;
    std::cin >> id;
    int branch = network.find(id);
    if (branch == -1) {
      std::cout << "ERROR: incorrect node id" << std::endl;
    } else {
      bool is_first =
          (network.get_first_id(branch) ==
           id);  // проверяем, является ли идентификатор первым узлом в списке
      sendMessage(branches[branch], std::to_string(id) + " kill");

      std::string reply = receiveMessage(branches[branch]);
      std::cout << reply << std::endl;
      network.erase(id);
```

```
          if (is_first) {
            unbindFromAddress(branches[branch], id);
            branches.erase(branches.begin() + branch);
          }
        }
      } else if (cmd == "heartbeat") {
        std::set<int> available_nodes;
        for (size_t i = 0; i < branches.size(); ++i) {
          int first_node_id = network.get_first_id(i);
          sendMessage(branches[i], std::to_string(first_node_id) + " heartbeat");

          std::string received_message = receiveMessage(branches[i]);
          std::istringstream reply(received_message);
          int node;
          while (reply >> node) {
            available_nodes.insert(
                node);  // сохраняем доступные (уникальные) узлы
          }
        }
        std::cout << "OK: ";
        if (available_nodes.empty()) {
          std::cout << "No available nodes" << std::endl;
        } else {
          for (auto v : available_nodes) {
            std::cout << v << " ";
          }
          std::cout << std::endl;
        }
      } else if (cmd == "exit") {
        for (size_t i = 0; i < branches.size(); ++i) {
          int first_node_id = network.get_first_id(i);
          sendMessage(branches[i], std::to_string(first_node_id) + " kill");
          std::string reply = receiveMessage(branches[i]);
          if (reply != "OK") {
            std::cout << reply << std::endl;
          } else {
            unbindFromAddress(branches[i], first_node_id);
          }
        }
        exit(0);
      } else {
        std::cout << "Incorrect cmd" << std::endl;
      }
    }
  }
```

counting.cpp:

```
#include <unistd.h>
```

```
#include <chrono>
#include <sstream>
#include <unordered_map>

#include "timer.h"
#include "zmq_functions.h"

int main(int argc, char* argv[]) {
 if (argc != 2 && argc != 3) {
   throw std::runtime_error("Wrong args for counting node");
 }
 int cur_id = std::atoi(argv[1]);
 int child_id = -1;
 if (argc == 3) {
   child_id = std::atoi(argv[2]);
 }

 std::unordered_map<std::string, int> dictionary;

 zmq::context_t context;
 zmq::socket_t parent_socket(context, ZMQ_REP);  // создаем сокет-ответчик
 connectToNode(parent_socket, cur_id);

 zmq::socket_t child_socket(context, ZMQ_REQ);  // создаем сокет-запросчик
 child_socket.setsockopt(ZMQ_SNDTIMEO, 5000);
 if (child_id != -1) {
   bindToAddress(child_socket, child_id);
 }

 Timer timer;
 std::string message;
 while (true) {
   message = receiveMessage(parent_socket);
   std::istringstream request(message);
   int dest_id;
   request >> dest_id;  // получаем идентификатор получателя

   std::string cmd;
   request >> cmd;

   if (dest_id == cur_id) {
     if (cmd == "pid") {
       sendMessage(parent_socket, "OK: " + std::to_string(getpid()));
     }

     else if (cmd == "create") {
       int new_child_id;
       request >> new_child_id;
       if (child_id != -1) {
```

```
    unbindFromAddress(child_socket, child_id);
  }
  bindToAddress(child_socket, new_child_id);
  pid_t pid = fork();
  if (pid < 0) {
    perror("Can't create new process");
    return -1;
  }
  if (pid == 0) {
    execl("./counting", "./counting",
          std::to_string(new_child_id).c_str(),
          std::to_string(child_id).c_str(), NULL);
    perror("Can't execute new process");
    return -2;
  }
  sendMessage(child_socket, std::to_string(new_child_id) + "pid");
  child_id = new_child_id;
  sendMessage(parent_socket, receiveMessage(child_socket));
} else if (cmd == "exec") {
  int sum = 0;
  std::string subcommand, answer;
  request >> subcommand;
  if (subcommand == "start") {
    timer.start();
    answer = "Ok: " + std::to_string(cur_id) + ", start timer";
  } else if (subcommand == "stop") {
    timer.stop();
    answer = "Ok: " + std::to_string(cur_id) + ", stop timer";
  } else if (subcommand == "time") {
    answer = "Ok: " + std::to_string(cur_id) + ", time is " +
             std::to_string(timer.elapsedSeconds());
  } else {
    answer = "Ok: " + std::to_string(cur_id) + ", wrong command";
  }
  sendMessage(parent_socket, answer);
}

else if (cmd == "heartbeat") {
  std::string reply;
  if (child_id != -1) {
    sendMessage(child_socket, std::to_string(child_id) + " heartbeat");
    std::string msg = receiveMessage(child_socket);
    reply += " " + msg;
  }
  sendMessage(parent_socket, std::to_string(cur_id) + reply);
} else if (cmd == "kill") {
  if (child_id != -1) {
    sendMessage(child_socket, std::to_string(child_id) + " kill");
    std::string msg = receiveMessage(child_socket);
```

```
          if (msg == "OK") {
            sendMessage(parent_socket, "OK");
          }
          unbindFromAddress(child_socket, child_id);
          disconnectFromNode(parent_socket, cur_id);
          break;
        }
        sendMessage(parent_socket, "OK");
        disconnectFromNode(parent_socket, cur_id);
        break;
      }
    } else if (child_id != -1) {
      sendMessage(child_socket, message);
      sendMessage(parent_socket, receiveMessage(child_socket));
      if (child_id == dest_id && cmd == "kill") {
        child_id = -1;
      }
    } else {
      sendMessage(parent_socket, "Error: node is unavailable");
    }
  }
}
```

Makefile:

```
all: control counting
control:
 g++ control.cpp -lzmq -o control  --std=c++17 -L /usr/local/include
counting:
 g++ counting.cpp -lzmq -o counting --std=c++17 -L /usr/local/include
clean:
 rm -rf control counting
```

Протокол работы программы

Тестирование:

katya@katya:~/MAI_2/OS/github/OS_MAI/lab5-7$ ./control
create 1 -1
OK: 417131
create 2 -1
OK: 417206
create 3 1
OK: 417239
create 4 1
Error: parent node has children
create 4 2
OK: 417334

```
exec 4 start
Ok: 4, start timer
exec 2 start
Ok: 2, start timer
exec 1 start
Ok: 1, start timer
exec 2 stop
Ok: 2, stop timer
exec 2 time
Ok: 2, time is 17.657000
exec 4 stop
Ok: 4, stop timer
exec 4 time
Ok: 4, time is 38.301000
exec 1 stop
Ok: 1, stop timer
exec 1 time
Ok: 1, time is 31.971000
heartbeat
OK: 1 2 3 4
kill 4
OK
heartbeat
OK: 1 2 3
kill 1
OK
keartbeat
Incorrect cmd
heartbeat
OK: 2
kill 2
OK
exit
```

==========

Strace:

```
katya@katya:~/MAI_2/OS/github/OS_MAI/lab5-7$ strace -f -e
trace=\!brk,clock_nanosleep,mmap,mprotect,munmap -owrite-simple3.log ./control
create 1 -1
OK: 423891
create 2 -1
OK: 423918
exec 1 start
Ok: 1, start timer
```

exec 2 start
Ok: 2, start timer
heartbeat
OK: 1 2
create 3 2
OK: 424240
exec 1 stop
Ok: 1, stop timer
exec 1 time
Ok: 1, time is 24.721000
exec 2 stop
Ok: 2, stop timer
exec 2 time
Ok: 2, time is 27.183000
heartbeat
OK: 1 2 3
kill 2
OK
heartbeat
OK: 1
kill 1
OK
exit

write-simple3.log:

Пояснения:

sendto - отправление сообщения на сокет

recvmsg - получение сообщения с сокета

socket - создать конечную точку для связи

setsockopt() - set the socket options

bind() - bind a name to a socket

listen() - network listener daemon

getsockname() - get socket name

epoll_ctl - интерфейс управления описателями epoll (очень полезная штука, которая позволяет отложить реакцию на событие и продолжить ждать остальные события)

poll - input/output multiplexing (мультиплексирование — уплотнение канала, то есть передача нескольких потоков данных с меньшей скоростью по одному каналу)

423819 execve("./control", ["./control"], 0x7fff129cd840 /* 46 vars */) = 0

423819 arch_prctl(0x3001 /* ARCH_??? */, 0x7ffffb36a440) = -1 EINVAL (Недопустимый аргумент)

423819 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)

423819 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

423819 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=121023, ...}, AT_EMPTY_PATH) = 0

423819 close(3) = 0

423819 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libzmq.so.5", O_RDONLY|O_CLOEXEC) = 3

423819 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

423819 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=639000, ...}, AT_EMPTY_PATH) = 0

423819 close(3) = 0

…

423819 <... epoll_ctl resumed>) = 0

423891 <... read resumed>"\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

423892 prctl(PR_SET_NAME, "ZMQbg/Reaper" <unfinished ...>

423891 newfstatat(3, "", <unfinished ...>

423892 <... prctl resumed>) = 0

423891 <... newfstatat resumed>{st_mode=S_IFREG|0644, st_size=639000, ...}, AT_EMPTY_PATH) = 0

423892 epoll_wait(5, <unfinished ...>

423819 rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

423819 clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f856f2b7990, parent_tid=0x7f856f2b7990, exit_signal=0, stack=0x7f856eab7000, stack_size=0x7ffd40, tls=0x7f856f2b76c0} => {parent_tid=[423893]}, 88) = 423893

423893 rseq(0x7f856f2b7fe0, 0x20, 0, 0x53053053 <unfinished ...>

423819 rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

423893 <... rseq resumed>) = 0

423819 <... rt_sigprocmask resumed>NULL, 8) = 0

423819 getpid(<unfinished ...>

423893 <... sched_getscheduler resumed>) = 0 (SCHED_OTHER)

423819 <... getpid resumed>) = 423819

423891 newfstatat(3, "", <unfinished ...>

423819 poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

423893 sched_setscheduler(423893, SCHED_OTHER, [0] <unfinished ...>

423819 <... poll resumed>) = 0 (Timeout)

423891 <... newfstatat resumed>{st_mode=S_IFREG|0644, st_size=2522552, ...}, AT_EMPTY_PATH) = 0

423893 <... sched_setscheduler resumed>) = 0

423819 socket(AF_NETLINK, SOCK_RAW|SOCK_CLOEXEC, NETLINK_ROUTE <unfinished ...>

423893 prctl(PR_SET_NAME, "ZMQbg/IO/0" <unfinished ...>

423819 <... socket resumed>) = 9

423893 <... prctl resumed>) = 0

423819 bind(9, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12 <unfinished ...>

423893 epoll_wait(7, <unfinished ...>

423819 <... bind resumed>) = 0

423819 getsockname(9, {sa_family=AF_NETLINK, nl_pid=423819, nl_groups=00000000}, [12]) = 0

423819 sendto(9, [{nlmsg_len=20, nlmsg_type=RTM_GETLINK, nlmsg_flags=NLM_F_REQUEST|NLM_F_DUMP, nlmsg_seq=1707990386, nlmsg_pid=0}, {ifi_family=AF_UNSPEC, ...}], 20, 0, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12) = 20

423819 recvmsg(9, <unfinished ...>

423891 close(3 <unfinished ...>

423819 <... recvmsg resumed>{msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, msg_namelen=12, msg_iov=[{iov_base=[[{nlmsg_len=1404, nlmsg_type=RTM_NEWLINK, nlmsg_flags=NLM_F_MULTI, nlmsg_seq=1707990386, nlmsg_pid=423819}, {ifi_family=AF_UNSPEC, ifi_type=ARPHRD_LOOPBACK, ifi_index=if_nametoindex("lo"), ifi_flags=IFF_UP|IFF_LOOPBACK|IFF_RUNNING|IFF_LOWER_UP, ifi_change=0}, [[{nla_len=7, nla_type=IFLA_IFNAME}, "lo"], [{nla_len=8, nla_type=IFLA_TXQLEN}, 1000], [{nla_len=5, nla_type=IFLA_OPERSTATE}, 0], [{nla_len=5, nla_type=IFLA_LINKMODE}, 0], [{nla_len=8, nla_type=IFLA_MTU}, 65536], [{nla_len=8, nla_type=IFLA_MIN_MTU}, 0], [{nla_len=8, nla_type=IFLA_MAX_MTU}, 0], [{nla_len=8, nla_type=IFLA_GROUP}, 0], [{nla_len=8, nla_type=IFLA_PROMISCUITY}, 0], [{nla_len=8, nla_type=IFLA_ALLMULTI}, 0], [{nla_len=8, nla_type=IFLA_NUM_TX_QUEUES}, 1], [{nla_len=8, nla_type=IFLA_GSO_MAX_SEGS}, 65535],

[{nla_len=8, nla_type=IFLA_GSO_MAX_SIZE}, 65536], [{nla_len=8, nla_type=IFLA_GRO_MAX_SIZE}, 65536], [{nla_len=8, nla_type=IFLA_TSO_MAX_SIZE}, 524280], [{nla_len=8, nla_type=IFLA_TSO_MAX_SEGS}, 65535], [{nla_len=8, nla_type=IFLA_NUM_RX_QUEUES}, 1], [{nla_len=5, nla_type=IFLA_CARRIER}, 1], [{nla_len=12, nla_type=IFLA_QDISC}, "noqueue"], [{nla_len=8, nla_type=IFLA_CARRIER_CHANGES}, 0], [{nla_len=8, nla_type=IFLA_CARRIER_UP_COUNT}, 0], [{nla_len=8, nla_type=IFLA_CARRIER_DOWN_COUNT}, 0], [{nla_len=5, nla_type=IFLA_PROTO_DOWN}, 0], [{nla_len=36, nla_type=IFLA_MAP}, {mem_start=0, mem_end=0, base_addr=0, irq=0, dma=0, port=0}], [{nla_len=10, nla_type=IFLA_ADDRESS}, 00:00:00:00:00:00],

…

423819 socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC, IPPROTO_TCP) = 9

423819 setsockopt(9, SOL_SOCKET, SO_REUSEADDR, [1], 4 <unfinished ...>

423891 close(3 <unfinished ...>

423819 <... setsockopt resumed>) = 0

423891 <... close resumed>) = 0

423819 bind(9, {sa_family=AF_INET, sin_port=htons(6061), sin_addr=inet_addr("127.0.0.1")}, 16 <unfinished ...>

423891 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC <unfinished ...>

423819 <... bind resumed>) = 0

423819 listen(9, 100 <unfinished ...>

423891 <... openat resumed>) = 3

423819 <... listen resumed>) = 0

423891 read(3, <unfinished ...>

…

423893 +++ exited with 0 +++

423892 +++ exited with 0 +++

423819 +++ exited with 0 +++

Вывод

Данный проект представляет собой систему управления и координации вычислительными узлами с использованием ZeroMQ. Файл control.cpp отвечает за управление системой и коммуникацию между узлами. Файл counting.cpp представляет собой вычислительный узел, который выполняет задачи. Использование ZeroMQ обеспечивает эффективную коммуникацию между узлами. Каждый узел имеет свой уникальный идентификатор для обмена сообщениями. Работа системы зависит от эффективной коммуникации между управляющим и вычислительными

узлами. Данная работа мне понравилась, так как я познакомилась с совершенно новой для меня технологией.