Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

# Лабораторная работа по курсу «Операционные системы» III Семестр

**Задание 6**
**Вариант 3**

| | |
|---|---|
| Студент: | Анисимов В.А. |
| Группа: | М80-208Б-18 |
| Преподаватель: | Миронов Е.С |
| | |
| Оценка: | |
| Дата: | |

Москва 2019

## 1. Описание задания

Реализовать распределенную систему по обработке запросов. В данной 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 сервера сообщений zmq. Также в данной системе необходимо предусмотреть проверку доступности узлов в соответствии с вариантом.
Вариант 41:
Топология 4 – бинарное дерево поиска. Набор команд 1 – подсчёт суммы n чисел. Команда проверки 1 – pingall.

## 2. Код программы

main.cpp

```
#include
<iostream>
            #include "zmq.hpp"
            #include <string>
            #include <zconf.h>
            #include <vector>
            #include <signal.h>
            #include <sstream>
            #include <set>
            #include <algorithm>
            #include "s_func.h"
            #include "zmq.h"


            class IdTree {
            public:
              IdTree() = default;
              ~IdTree() {
                delete_node(head_);
              }

              bool contains(int id) {
                TreeNode* temp = head_;
                while(temp != nullptr) {
                  if (temp->id_ == id) {
                        break;
                  }
                  if (id > temp->id_) {
                        temp = temp->right;
                  }
                  if (id < temp->id_) {
                        temp = temp->left;
```

```
    }
  }
  return temp != nullptr;
}

void insert(int id) {
  if (head_ == nullptr) {
    head_ = new TreeNode(id);
    return;
  }
  TreeNode* temp = head_;
  while(temp != nullptr) {
    if (id == temp->id_) {
      break;
    }
    if (id < temp->id_) {
      if (temp->left == nullptr) {
        temp->left = new TreeNode(id);
        break;
      }
      temp = temp->left;
    }
    if (id > temp->id_) {
      if (temp->right == nullptr) {
        temp->right = new TreeNode(id);
        break;
      }
      temp = temp->right;
    }
  }
}

void erase(int id) {
  TreeNode* prev_id = nullptr;
  TreeNode* temp = head_;
  while (temp != nullptr) {
    if (id == temp->id_) {
      if (prev_id == nullptr) {
        head_ = nullptr;
      } else {
        if (prev_id->left == temp) {
          prev_id->left = nullptr;
        } else {
          prev_id->right = nullptr;
        }
      }
      delete_node(temp);
      } else if (id < temp->id_) {
           prev_id = temp;
           temp = temp->left;
      } else if (id > temp->id_) {
           prev_id = temp;
           temp = temp->right;
      }
    }
  }

  std::vector<int> get_nodes() const {
    std::vector<int> result;
    get_nodes(head_, result);
    return result;
  }

private:

  struct TreeNode {
    TreeNode(int id) : id_(id) {}
    int id_;
    TreeNode* left = nullptr;
    TreeNode* right = nullptr;
  };

  void get_nodes(TreeNode* node, std::vector<int>& v) const {
    if (node == nullptr) {
      return;
    }
    get_nodes(node->left,v);
    v.push_back(node->id_);
    get_nodes(node->right, v);
  }

  void delete_node(TreeNode* node) {
    if (node == nullptr) {
      return;
    }
    delete_node(node->right);
    delete_node(node->left);
    delete node;
  }

  TreeNode* head_ = nullptr;
};


int main() {
  std::string command;
  IdTree ids;
  size_t child_pid = 0;
  int child_id = 0;
  zmq::context_t context(1);
  zmq::socket_t main_socket(context, ZMQ_REQ); //з-о
  int linger = 0;
  main_socket.setsockopt(ZMQ_SNDTIMEO, 2000); //макс время до возврата операции отправки
  main_socket.setsockopt(ZMQ_LINGER, &linger, sizeof(linger)); //как долго неотправленные сообщения
тусят в памяти
  int port = bind_socket(main_socket);

  while (true) {
    std::cin >> command;
    if (command == "create") {
      size_t node_id;
      std::string result;
      std::cin >> node_id;
      if (child_pid == 0) {
	child_pid = fork();
	if (child_pid == -1) {
	  std::cout << "Unable to create first worker node\n";
	  child_pid = 0;
	  exit(1);
	} else if (child_pid == 0) {
	  create_node(node_id, port);
	} else {
	  child_id = node_id;
	  send_message(main_socket,"pid");
	  result = recieve_message(main_socket);
	}

      } else {
	std::ostringstream msg_stream;
	msg_stream << "create " << node_id;
	send_message(main_socket, msg_stream.str());
	result = recieve_message(main_socket);
      }

      if (result.substr(0,2) == "Ok") {
	ids.insert(node_id);
      }
      std::cout << result << "\n";

    } else if (command == "remove") {
      if (child_pid == 0) {
	std::cout << "Error:Not found\n";
	continue;
 }
 size_t node_id;
 std::cin >> node_id;
 if (node_id == child_id) {
	kill(child_pid, SIGTERM);
	kill(child_pid, SIGKILL);
	child_id = 0;
	child_pid = 0;
	std::cout << "Ok\n";
	ids.erase(node_id);
	continue;
 }
 std::string message_string = "remove " + std::to_string(node_id);
 send_message(main_socket, message_string);
 std::string recieved_message = recieve_message(main_socket);
 if (recieved_message.substr(0, std::min<int>(recieved_message.size(), 2)) == "Ok") {
	ids.erase(node_id);
 }
 std::cout << recieved_message << "\n";

} else if (command == "exec") {
 int id, n;
 std::cin >> id >> n;
 std::vector<int> numbers(n);
 for (int i = 0; i < n; ++i) {
	std::cin >> numbers[i];
 }

 std::string message_string = "exec " + std::to_string(id) + " " + std::to_string(n);
 for (int i = 0; i < n; ++i) {
	message_string += " " + std::to_string(numbers[i]);
 }

 send_message(main_socket, message_string);
 std::string recieved_message = recieve_message(main_socket);
 std::cout << recieved_message << "\n";

} else if (command == "pingall") {
 send_message(main_socket,"pingall");
 std::string recieved = recieve_message(main_socket);
 std::istringstream is;
 if (recieved.substr(0,std::min<int>(recieved.size(), 5)) == "Error") {
	is = std::istringstream("");
 } else {
	is = std::istringstream(recieved);
 }

 std::set<int> recieved_ids;
```

```
      int rec_id;
      while (is >> rec_id) {
           recieved_ids.insert(rec_id);
      }
      std::vector from_tree = ids.get_nodes();
      auto part_it = std::partition(from_tree.begin(), from_tree.end(), [&recieved_ids] (int a) {
            return recieved_ids.count(a) == 0;
           });
      if (part_it == from_tree.begin()) {
           std::cout << "Ok: -1\n";
      } else {
           std::cout << "Ok:";
           for (auto it = from_tree.begin(); it != part_it; ++it) {
            std::cout << " " << *it;
           }
           std::cout << "\n";
      }

    } else if (command == "exit") {
      break;
    }

  }

  return 0;
}
```

## child.cpp

```
#include
<iostream>
#include "zmq.hpp"
#include <string>
#include <sstream>
#include <zconf.h>
#include <exception>
#include <signal.h>
#include "s_func.h"
#include "zmq.h"




int main(int argc, char** argv) { //айди и номер порта
  int id = std::stoi(argv[1]);
  int parent_port = std::stoi(argv[2]);
  zmq::context_t context(3);
```

```
zmq::socket_t parent_socket(context, ZMQ_REP);
parent_socket.connect(get_port_name(parent_port));

int left_pid = 0;
int right_pid = 0;
int left_id = 0;
int right_id = 0;

zmq::socket_t left_socket(context, ZMQ_REQ);
zmq::socket_t right_socket(context, ZMQ_REQ);
int linger = 0;
left_socket.setsockopt(ZMQ_SNDTIMEO, 2000);
left_socket.setsockopt(ZMQ_LINGER, &linger, sizeof(linger));
right_socket.setsockopt(ZMQ_SNDTIMEO, 2000);
right_socket.setsockopt(ZMQ_LINGER, &linger, sizeof(linger));

int left_port = bind_socket(left_socket);
int right_port = bind_socket(right_socket);

while (true) {
  std::string request_string;

  request_string = recieve_message(parent_socket);
  std::istringstream command_stream(request_string);
  std::string command;
  command_stream >> command;

  if (command == "id") {
    std::string parent_string = "Ok:" + std::to_string(id);
    send_message(parent_socket, parent_string);
  } else if (command == "pid") {
    std::string parent_string = "Ok:" + std::to_string(getpid());
    send_message(parent_socket, parent_string);
  } else  if (command == "create") {
    int id_to_create;
    command_stream >> id_to_create;
    // управляюший узел сообщает id нового узла и порт, к которому его надо подключить
    if (id_to_create == id) {
         // если id равен данному, значит узел уже существует, посылаем ответ с ошибкой
         std::string message_string = "Error: Already exists";
         send_message(parent_socket, message_string);
    } else if (id_to_create < id) {
         if (left_pid == 0) {
           left_pid = fork();
           if (left_pid == -1) {
             send_message(parent_socket, "Error: Cannot fork");
             left_pid = 0;
           } else if (left_pid == 0) {
            create_node(id_to_create,left_port);
          } else {
            left_id = id_to_create;
            send_message(left_socket, "pid");
            send_message(parent_socket, recieve_message(left_socket));
          }
        } else {
          send_message(left_socket, request_string);
          send_message(parent_socket, recieve_message(left_socket));
        }
  } else {
        if (right_pid == 0) {
          right_pid = fork();
          if (right_pid == -1) {
            send_message(parent_socket, "Error: Cannot fork");
            right_pid = 0;
          } else if (right_pid == 0) {
            create_node(id_to_create,right_port);
          } else {
            right_id = id_to_create;
            send_message(right_socket, "pid");
            send_message(parent_socket, recieve_message(right_socket));
          }
        } else {
          send_message(right_socket, request_string);
          send_message(parent_socket, recieve_message(right_socket));
        }
  }

} else if (command == "remove") {
  int id_to_delete;
  command_stream >> id_to_delete;
  if (id_to_delete < id) {
        if (left_id == 0) {
          send_message(parent_socket, "Error: Not found");
        } else if (left_id == id_to_delete) {
          send_message(left_socket, "kill_children");
          recieve_message(left_socket);
          kill(left_pid,SIGTERM);
          kill(left_pid,SIGKILL);
          left_id = 0;
          left_pid = 0;
          send_message(parent_socket, "Ok");

        } else {
          send_message(left_socket, request_string);
          send_message(parent_socket, recieve_message(left_socket));
        }
```

```
 } else {
     if (right_id == 0) {
       send_message(parent_socket, "Error: Not found");
     } else if (right_id == id_to_delete) {
       send_message(right_socket, "kill_children");
       recieve_message(right_socket);
       kill(right_pid,SIGTERM);
       kill(right_pid,SIGKILL);
       right_id = 0;
       right_pid = 0;
       send_message(parent_socket, "Ok");
     } else {
       send_message(right_socket, request_string);
       send_message(parent_socket, recieve_message(right_socket));
     }
 }
} else if (command == "exec") {
 int exec_id;
 command_stream >> exec_id;
 if (exec_id == id) {
     int n;
     command_stream >> n;
     int sum = 0;
     for (int i = 0; i < n; ++i) {
       int cur_num;
       command_stream >> cur_num;
       sum += cur_num;
     }
     std::string recieve_message = "Ok:" + std::to_string(id) + ":" + std::to_string(sum);
     send_message(parent_socket, recieve_message);

 } else if (exec_id < id) {
     if (left_pid == 0) {
       std::string recieve_message = "Error:" + std::to_string(exec_id) + ": Not found";
       send_message(parent_socket, recieve_message);
     } else {
       send_message(left_socket, request_string);
       send_message(parent_socket, recieve_message(left_socket));
     }
 } else {
     if (right_pid == 0) {
       std::string recieve_message = "Error:" + std::to_string(exec_id) + ": Not found";
       send_message(parent_socket, recieve_message);
     } else {
       send_message(right_socket, request_string);
       send_message(parent_socket, recieve_message(right_socket));
     }
 }
```

```
    } else if (command == "pingall") {
      std::ostringstream res;
      std::string left_res;
      std::string right_res;
      if (left_pid != 0) {
            send_message(left_socket, "pingall");
            left_res = recieve_message(left_socket);
      }
      if (right_pid != 0) {
            send_message(right_socket, "pingall");
            right_res = recieve_message(right_socket);
      }
      if (!left_res.empty() && left_res.substr(std::min<int>(left_res.size(),5)) != "Error") {
            res << left_res;
      }


      if (!right_res.empty() && right_res.substr(std::min<int>(right_res.size(),5)) != "Error") {
            res << right_res;
      }
      send_message(parent_socket, res.str());
    } else if (command == "kill_children") {
      if (left_pid == 0 && right_pid == 0) {
            send_message(parent_socket, "Ok");
      } else {
            if (left_pid != 0) {
              send_message(left_socket, "kill_children");
              recieve_message(left_socket);
              kill(left_pid,SIGTERM);
              kill(left_pid,SIGKILL);
            }
            if (right_pid != 0) {
              send_message(right_socket, "kill_children");
              recieve_message(right_socket);
              kill(right_pid,SIGTERM);
              kill(right_pid,SIGKILL);
            }
            send_message(parent_socket, "Ok");

      }
    }
    if (parent_port == 0) {
      break;
    }
  }
}
```

## s_func.h

```
#pragma
once
        #include <string>
        #include <zconf.h>
        #include "zmq.hpp"
        #include "zmq.h"


        bool send_message(zmq::socket_t& socket, const std::string& message_string);


        std::string recieve_message(zmq::socket_t& socket);


        std::string get_port_name(int port);


        int bind_socket(zmq::socket_t& socket);


        void create_node(int id, int port);
```

## s_func.cpp

```
#include
"s_func.h"


        bool send_message(zmq::socket_t& socket, const std::string& message_string){
          zmq::message_t message(message_string.size());
          memcpy(message.data(), message_string.c_str(), message_string.size());
          return socket.send(message);
        }


        std::string recieve_message(zmq::socket_t& socket) {
          zmq::message_t message;
          bool ok;
          try {
            ok = socket.recv(&message);
          } catch (...) {
            ok = false;
          }
          std::string recieved_message(static_cast<char*>(message.data()), message.size());
          if (recieved_message.empty() || !ok) {
            return "Error: Node is not available";
```

```
  }
  return recieved_message;
}


std::string get_port_name(int port) {
  return "tcp://127.0.0.1:" + std::to_string(port);
}


int bind_socket(zmq::socket_t& socket) {
  int port = 30000;
  while (true) {
    try {
      socket.bind(get_port_name(port));
      break;
    } catch(...) {
      port++;
    }
  }
  return port;
}


void create_node(int id, int port) {
  char* arg1 = strdup((std::to_string(id)).c_str());
  char* arg2 = strdup((std::to_string(port)).c_str());
  char* args[] = {"./child", arg1, arg2, NULL};
  execv("./child", args);
}
```

## Протокол работы программы

```
walien@PC-name:~/2kurs/OS/lab6/tmp$ ./terminal
create 3
Ok:12986
create 1
Ok:12991
create 5
Ok:12996
exec 5 2 1 4
Ok:5:5
pingall
Ok: 1 3 5
remove 5
Ok
exec 5 2 1 3
Error:5: Not found
```

```
pingall
Ok: 1 3
exit
```

## . Общие сведения о программе

В файлах s_func.h и s_func.cpp осуществлена реализация функций сервера: отправка/принятие сообщений, получение номера порта, создание узла.

В файле main.cpp содержится класс бинарного дерева и реализованы управляющий сокет и команды взаимодействия с ним.

Файл child.cpp отвечает за реализацию вычислительных узлов.

Для общения между процессами используется библиотека zmq.

## Вывод

В результате данной работы были получены навыки владения технологией очереди сообщений и создания программ, процессы которых взаимодействуют при помощи данной очереди. Так же было полезным создать программу с процессами, связанными в определённой технологии. Особое внимание стоит уделить библиотеке zmq, которая позволяет осуществлять связь между процессами, исключая блокировки, т.к. процессы взаимодействуют между собой исключительно с помощью сообщений. Данная технология позволяет сильно ускорить работу многопроцессорной программы и защитить её от потери данных и зависания в случае самоблокировки.

## Strace

```
walien@PC-name:~/2kurs/OS/lab6/tmp$ strace ./terminal
execve("./terminal", ["./terminal"], 0x7ffd969fda30 /* 66 vars */) = 0
brk(NULL)                               = 0x55f81be61000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f939a4b5000
mmap(NULL, 3789144, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f9398eba000
mprotect(0x7f9399057000, 2093056, PROT_NONE) = 0
mmap(0x7f9399256000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x19c000) = 0x7f9399256000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f939a4b2000
arch_prctl(ARCH_SET_FS, 0x7f939a4b2740) = 0
mprotect(0x7f9399a66000, 16384, PROT_READ) = 0
mprotect(0x7f9399256000, 4096, PROT_READ) = 0
mprotect(0x7f9399471000, 4096, PROT_READ) = 0
mprotect(0x7f939967d000, 4096, PROT_READ) = 0
mprotect(0x7f9399c86000, 4096, PROT_READ) = 0
mprotect(0x7f939a001000, 40960, PROT_READ) = 0
mprotect(0x7f939a29f000, 28672, PROT_READ) = 0
mprotect(0x55f81bbdb000, 4096, PROT_READ) = 0
mprotect(0x7f939a4ce000, 4096, PROT_READ) = 0
munmap(0x7f939a4b9000, 83936)           = 0
set_tid_address(0x7f939a4b2a10)         = 13317
set_robust_list(0x7f939a4b2a20, 24)     = 0
rt_sigaction(SIGRTMIN, {sa_handler=0x7f939925dcb0, sa_mask=[], sa_flags=SA_RESTORER|SA_SIGINFO,
sa_restorer=0x7f939926a8a0}, NULL, 8) = 0
rt_sigaction(SIGRT_1, {sa_handler=0x7f939925dd50, sa_mask=[],
sa_flags=SA_RESTORER|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f939926a8a0}, NULL, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
brk(NULL)                               = 0x55f81be61000
brk(0x55f81be82000)                     = 0x55f81be82000
futex(0x7f939a00e09c, FUTEX_WAKE_PRIVATE, 2147483647) = 0
futex(0x7f939a00e0a8, FUTEX_WAKE_PRIVATE, 2147483647) = 0
eventfd2(0, EFD_CLOEXEC)                = 3
fcntl(3, F_GETFL)                       = 0x2 (flags O_RDWR)
fcntl(3, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
fcntl(3, F_GETFL)                       = 0x802 (flags O_RDWR|O_NONBLOCK)
fcntl(3, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
eventfd2(0, EFD_CLOEXEC)                = 4
fcntl(4, F_GETFL)                       = 0x2 (flags O_RDWR)
fcntl(4, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
fcntl(4, F_GETFL)                       = 0x802 (flags O_RDWR|O_NONBLOCK)
fcntl(4, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
epoll_create1(EPOLL_CLOEXEC)            = 5
epoll_ctl(5, EPOLL_CTL_ADD, 4, {0, {u32=468151808, u64=94524108402176}}) = 0
epoll_ctl(5, EPOLL_CTL_MOD, 4, {EPOLLIN, {u32=468151808, u64=94524108402176}}) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f93986b9000
mprotect(0x7f93986ba000, 8388608, PROT_READ|PROT_WRITE) = 0
clone(child_stack=0x7f9398eb8fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLON
E_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7f9398eb99d0,
tls=0x7f9398eb9700, child_tidptr=0x7f9398eb99d0) = 13318
eventfd2(0, EFD_CLOEXEC)                = 6
fcntl(6, F_GETFL)                       = 0x2 (flags O_RDWR)
fcntl(6, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
fcntl(6, F_GETFL)                       = 0x802 (flags O_RDWR|O_NONBLOCK)
fcntl(6, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
epoll_create1(EPOLL_CLOEXEC)            = 7
epoll_ctl(7, EPOLL_CTL_ADD, 6, {0, {u32=468153824, u64=94524108404192}}) = 0
epoll_ctl(7, EPOLL_CTL_MOD, 6, {EPOLLIN, {u32=468153824, u64=94524108404192}}) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f9397eb8000
mprotect(0x7f9397eb9000, 8388608, PROT_READ|PROT_WRITE) = 0
clone(child_stack=0x7f93986b7fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLON
E_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7f93986b89d0,
tls=0x7f93986b8700, child_tidptr=0x7f93986b89d0) = 13319
eventfd2(0, EFD_CLOEXEC)                = 8
fcntl(8, F_GETFL)                       = 0x2 (flags O_RDWR)
fcntl(8, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
fcntl(8, F_GETFL)                       = 0x802 (flags O_RDWR|O_NONBLOCK)
fcntl(8, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
socket(AF_NETLINK, SOCK_RAW|SOCK_CLOEXEC, NETLINK_ROUTE) = 9
bind(9, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12) = 0
getsockname(9, {sa_family=AF_NETLINK, nl_pid=13317, nl_groups=00000000}, [12]) = 0
"\x84\x00\x02\x00\x80\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x01\x00\x0
0\x00\x01\x00\x00\x00"...}]}, {{len=1304, type=RTM_NEWLINK, flags=NLM_F_MULTI, seq=1616614489,
pid=13317}, {ifi_family=AF_UNSPEC, ifi_type=ARPHRD_ETHER, ifi_index=if_nametoindex("enp0s3"),
ifi_flags=IFF_UP|IFF_BROADCAST|IFF_RUNNING|IFF_MULTICAST|0x10000, ifi_change=0}, [{{nla_len=11,
nla_type=IFLA_IFNAME}, "enp0s3"}, {{nla_len=8, nla_type=IFLA_TXQLEN}, 1000}, {{nla_len=5,
nla_type=IFLA_OPERSTATE}, 6}, {{nla_len=5, nla_type=IFLA_LINKMODE}, 0}, {{nla_len=8,
nla_type=IFLA_MTU}, 1500}, {{nla_len=8, nla_type=IFLA_GROUP}, 0}, {{nla_len=8,
nla_type=IFLA_PROMISCUITY}, 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_NUM_RX_QUEUES}, 1}, {{nla_len=5, nla_type=IFLA_CARRIER}, 1}, {{nla_len=13,
nla_type=IFLA_QDISC}, "fq_codel"}, {{nla_len=8, nla_type=IFLA_CARRIER_CHANGES}, 26}, {{nla_len=5,
nla_type=IFLA_PROTO_DOWN}, 0}, {{nla_len=8, nla_type=0x2f /* IFLA_??? */}, "\x0d\x00\x00\x00"},
{{nla_len=8, nla_type=0x30 /* IFLA_??? */}, "\x0d\x00\x00\x00"}, {{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},
"\x08\x00\x27\xcd\x19\xf6"}, {{nla_len=10, nla_type=IFLA_BROADCAST}, "\xff\xff\xff\xff\xff\xff"},
{{nla_len=196, nla_type=IFLA_STATS64}, {rx_packets=2912052, tx_packets=559448, rx_bytes=3386863270,
tx_bytes=35567091, rx_errors=0, tx_errors=0, rx_dropped=0, tx_dropped=0, multicast=0, collisions=0,
rx_length_errors=0, rx_over_errors=0, rx_crc_errors=0, rx_frame_errors=0, rx_fifo_errors=0, rx_missed_errors=0,
tx_aborted_errors=0, tx_carrier_errors=0, tx_fifo_errors=0, tx_heartbeat_errors=0, tx_window_errors=0,
rx_compressed=0, tx_compressed=0, rx_nohandler=0}}, {{nla_len=100, nla_type=IFLA_STATS},
{rx_packets=2912052, tx_packets=559448, rx_bytes=3386863270, tx_bytes=35567091, rx_errors=0, tx_errors=0,
rx_dropped=0, tx_dropped=0, multicast=0, collisions=0, rx_length_errors=0, rx_over_errors=0, rx_crc_errors=0,
rx_frame_errors=0, rx_fifo_errors=0, rx_missed_errors=0, tx_aborted_errors=0, tx_carrier_errors=0, tx_fifo_errors=0,
tx_heartbeat_errors=0, tx_window_errors=0, rx_compressed=0, tx_compressed=0, rx_nohandler=0}}, {{nla_len=12,
nla_type=IFLA_XDP}, {{nla_len=5, nla_type=IFLA_XDP_ATTACHED}, 0}}, {{nla_len=756,
nla_type=IFLA_AF_SPEC},
"\x84\x00\x02\x00\x80\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x01\x00\x0
0\x00\x01\x00\x00\x00"...}]}], iov_len=4096}], msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 2600
recvmsg(9, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, msg_namelen=12,
msg_iov=[{iov_base={{len=20, type=NLMSG_DONE, flags=NLM_F_MULTI, seq=1616614489, pid=13317}, 0},
iov_len=4096}], msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 20
sendto(9, {{len=20, type=RTM_GETADDR, flags=NLM_F_REQUEST|NLM_F_DUMP, seq=1616614490, pid=0},
{ifa_family=AF_UNSPEC, ...}}, 20, 0, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12) = 20
recvmsg(9, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, msg_namelen=12,
msg_iov=[{iov_base=[{{len=76, type=RTM_NEWADDR, flags=NLM_F_MULTI, seq=1616614490, pid=13317},
{ifa_family=AF_INET, ifa_prefixlen=8, ifa_flags=IFA_F_PERMANENT, ifa_scope=RT_SCOPE_HOST,
ifa_index=if_nametoindex("lo")}, [{{nla_len=8, nla_type=IFA_ADDRESS}, 127.0.0.1}, {{nla_len=8,
nla_type=IFA_LOCAL}, 127.0.0.1}, {{nla_len=7, nla_type=IFA_LABEL}, "lo"}, {{nla_len=8,
nla_type=IFA_FLAGS}, IFA_F_PERMANENT}, {{nla_len=20, nla_type=IFA_CACHEINFO},
{ifa_prefered=4294967295, ifa_valid=4294967295, cstamp=855, tstamp=855}}]}, {{len=88, type=RTM_NEWADDR,
flags=NLM_F_MULTI, seq=1616614490, pid=13317}, {ifa_family=AF_INET, ifa_prefixlen=24, ifa_flags=0,
ifa_scope=RT_SCOPE_UNIVERSE, ifa_index=if_nametoindex("enp0s3")}, [{{nla_len=8, nla_type=IFA_ADDRESS},
10.0.2.15}, {{nla_len=8, nla_type=IFA_LOCAL}, 10.0.2.15}, {{nla_len=8, nla_type=IFA_BROADCAST},
10.0.2.255}, {{nla_len=11, nla_type=IFA_LABEL}, "enp0s3"}, {{nla_len=8, nla_type=IFA_FLAGS},
IFA_F_NOPREFIXROUTE}, {{nla_len=20, nla_type=IFA_CACHEINFO}, {ifa_prefered=58282, ifa_valid=58282,
cstamp=7657, tstamp=7927239}}]}], iov_len=4096}], msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 164
recvmsg(9, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, msg_namelen=12,
msg_iov=[{iov_base=[{{len=72, type=RTM_NEWADDR, flags=NLM_F_MULTI, seq=1616614490, pid=13317},
{ifa_family=AF_INET6, ifa_prefixlen=128, ifa_flags=IFA_F_PERMANENT, ifa_scope=RT_SCOPE_HOST,
ifa_index=if_nametoindex("lo")}, [{{nla_len=20, nla_type=IFA_ADDRESS}, ::1}, {{nla_len=20,
nla_type=IFA_CACHEINFO}, {ifa_prefered=4294967295, ifa_valid=4294967295, cstamp=855, tstamp=855}},
{{nla_len=8, nla_type=IFA_FLAGS}, IFA_F_PERMANENT}]}, {{len=72, type=RTM_NEWADDR,
flags=NLM_F_MULTI, seq=1616614490, pid=13317}, {ifa_family=AF_INET6, ifa_prefixlen=64,
ifa_flags=IFA_F_PERMANENT, ifa_scope=RT_SCOPE_LINK, ifa_index=if_nametoindex("enp0s3")}, [{{nla_len=20,
nla_type=IFA_ADDRESS}, fe80::e014:18bb:37cd:786b}, {{nla_len=20, nla_type=IFA_CACHEINFO},
{ifa_prefered=4294967295, ifa_valid=4294967295, cstamp=7542, tstamp=7665}}, {{nla_len=8,
nla_type=IFA_FLAGS}, IFA_F_PERMANENT|IFA_F_NOPREFIXROUTE}]}], iov_len=4096}], msg_iovlen=1,
msg_controllen=0, msg_flags=0}, 0) = 144
recvmsg(9, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, msg_namelen=12,
msg_iov=[{iov_base={{len=20, type=NLMSG_DONE, flags=NLM_F_MULTI, seq=1616614490, pid=13317}, 0},
iov_len=4096}], msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 20
close(9)                    = 0
socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC, IPPROTO_TCP) = 9
setsockopt(9, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
bind(9, {sa_family=AF_INET, sin_port=htons(30020), sin_addr=inet_addr("127.0.0.1")}, 16) = 0
listen(9, 100)              = 0
getsockname(9, {sa_family=AF_INET, sin_port=htons(30020), sin_addr=inet_addr("127.0.0.1")}, [128->16]) = 0
getsockname(9, {sa_family=AF_INET, sin_port=htons(30020), sin_addr=inet_addr("127.0.0.1")}, [128->16]) = 0
```

```
write(6, "\1\0\0\0\0\0\0\0", 8)        = 8
write(8, "\1\0\0\0\0\0\0\0", 8)        = 8
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
read(0, create 3
"create 3\n", 1024)            = 9
clone(child_stack=NULL,                          flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f939a4b2a10) = 13320
poll([{fd=8, events=POLLIN}], 1, 0)     = 1 ([{fd=8, revents=POLLIN}])
read(8, "\1\0\0\0\0\0\0\0", 8)         = 8
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, 2000)  = 1 ([{fd=8, revents=POLLIN}])
read(8, "\1\0\0\0\0\0\0\0", 8)         = 8
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, -1)    = 1 ([{fd=8, revents=POLLIN}])
read(8, "\1\0\0\0\0\0\0\0", 8)         = 8
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, -1)    = 1 ([{fd=8, revents=POLLIN}])
read(8, "\1\0\0\0\0\0\0\0", 8)         = 8
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(6, "\1\0\0\0\0\0\0\0", 8)        = 8
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
write(1, "Ok:13320\n", 9Ok:13320
)              = 9
read(0, create 1
"create 1\n", 1024)            = 9
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(6, "\1\0\0\0\0\0\0\0", 8)        = 8
poll([{fd=8, events=POLLIN}], 1, -1)    = 1 ([{fd=8, revents=POLLIN}])
read(8, "\1\0\0\0\0\0\0\0", 8)         = 8
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(1, "Ok:13325\n", 9Ok:13325
)              = 9
read(0, create 5
"create 5\n", 1024)            = 9
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(6, "\1\0\0\0\0\0\0\0", 8)        = 8
poll([{fd=8, events=POLLIN}], 1, -1)    = 1 ([{fd=8, revents=POLLIN}])
read(8, "\1\0\0\0\0\0\0\0", 8)         = 8
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(1, "Ok:13330\n", 9Ok:13330
)              = 9
read(0, pingall
"pingall\n", 1024)             = 8
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(6, "\1\0\0\0\0\0\0\0", 8)        = 8
poll([{fd=8, events=POLLIN}], 1, -1)    = 1 ([{fd=8, revents=POLLIN}])
read(8, "\1\0\0\0\0\0\0\0", 8)         = 8
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(1, "Ok: 1 3 5\n", 10Ok: 1 3 5
)            = 10
read(0, exec 1 2 1 5
"exec 1 2 1 5\n", 1024)        = 13
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(6, "\1\0\0\0\0\0\0\0", 8)        = 8
poll([{fd=8, events=POLLIN}], 1, -1)    = 1 ([{fd=8, revents=POLLIN}])
read(8, "\1\0\0\0\0\0\0\0", 8)         = 8
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(1, "Ok:1:6\n", 7Ok:1:6
)              = 7
read(0, remove 3
"remove 3\n", 1024)            = 9
kill(13320, SIGTERM)                    = 0
kill(13320, SIGKILL)                    = 0
```

```
write(1, "Ok\n", 3Ok
)                   = 3
read(0, 0x55f81be78a10, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_KILLED, si_pid=13320, si_uid=1000, si_status=SIGTERM,
si_utime=0, si_stime=0} ---
read(0, pingall
"pingall\n", 1024)              = 8
poll([{fd=8, events=POLLIN}], 1, 0)     = 1 ([{fd=8, revents=POLLIN}])
read(8, "\1\0\0\0\0\0\0\0", 8)          = 8
poll([{fd=8, events=POLLIN}], 1, 0)     = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, 2000)  = 0 (Timeout)
write(1, "Ok: -1\n", 7Ok: -1
)                = 7
read(0, exit
"exit\n", 1024)                 = 5
write(4, "\1\0\0\0\0\0\0\0", 8)         = 8
write(8, "\1\0\0\0\0\0\0\0", 8)         = 8
poll([{fd=3, events=POLLIN}], 1, -1)    = 1 ([{fd=3, revents=POLLIN}])
read(3, "\1\0\0\0\0\0\0\0", 8)          = 8
write(6, "\1\0\0\0\0\0\0\0", 8)         = 8
close(7)                                = 0
close(6)                                = 0
close(5)                                = 0
close(4)                                = 0
close(3)                                = 0
lseek(0, -1, SEEK_CUR)                  = -1 ESPIPE (Illegal seek)
exit_group(0)                           = ?
+++ exited with 0 +++
```