Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу

«Операционные системы»

Управление серверами сообщений

Студент: Кудинов Сергей Владимирович
Группа: М80 – 206Б-18
Вариант: 32
Преподаватель: Соколов Андрей Алексеевич
Оценка: ___________
Дата: ___________
Подпись: ___________

Москва, 2019

Постановка задачи

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 задания: 41. Топология — AVL дерево. Тип вычислительной команды — сумма n чисел. Тип проверки узлов на доступность — пинг всех узлов.

Общие сведения о программе

Программа состоит из двух файлов, которые компилируются в исполнительные файлы(которые представляют управляющий и вычислительные узлы), а так же из статической библиотеки, которая подключается к вышеуказанным файлам. Общение между процессами происходит с помощью библиотеки zmq.

Общий метод и алгоритм решения

- Управляющий узел принимает команды, обрабатывает их и пересылает дочерним узлам(или выводит сообщение об ошибке).
- Дочерние узлы проверяют, может ли быть команда выполнена в данном узле, если нет, то команда пересылается в один из дочерних узлов, из которого возвращается некоторое сообщение(об успехе или об ошибке), которое потом пересылается обратно по дереву.
- Для корректной проверки на доступность узлов, используется дерево, эмулирующее поведение узлов в данной топологии(например, при удалении узла, удаляются все его потомки).
- Если узел недоступен, то по истечении таймаута будет сгенерировано сообщение о недоступности узла и оно будет передано вверх по дереву, к управляющему узлу.
- При удалении узла, все его потомки рекурсивно уничтожаются.

Код программы

main.cpp:

```cpp
#include <iostream>

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



struct TNode {
   unsigned long long Id;
   TNode* Left;
   long long Height;
   TNode* Right;
TNode(unsigned long long  id) {
   Id = id;
   Height = 1;
   Left = 0;
   Right = 0;
}

};

long long GetHeight(TNode* n) {
   if (n!=NULL) {
      return n->Height;
   }
   return 0;
}
long long GetBalance(TNode* n) {
   return GetHeight(n->Right)-GetHeight(n->Left);
}

void CountHeight(TNode* n)
{
   int hl = GetHeight(n->Left);
   int hr = GetHeight(n->Right);
   n->Height = (hl>hr?hl:hr)+1;
}
```

```cpp
TNode* RotateLeft(TNode* q) {
   TNode*p = q->Right;

   q->Right = p->Left;
   p->Left = q;
   CountHeight(q);
   CountHeight(p);
   return p;

}
TNode* RotateRight(TNode* q) {
   TNode* p = q->Left;
   q->Left = p->Right;
   p->Right = q;
   CountHeight(q);
   CountHeight(p);
   return p;
}
TNode* BalanceTree(TNode* p) {
   CountHeight(p);
   if (GetBalance(p)==2) {
      TNode* q = p->Right;
      if (GetBalance(q)<0) {
         p->Right = RotateRight(q);
      }
      return RotateLeft(p);
   }
   if (GetBalance(p)==-2) {
      TNode* q = p->Left;
      if (GetBalance(q)>0) {
         p->Left=RotateLeft(q);
      }
      return RotateRight(p);
   }
   return p;
}
  void get_nodes1(TNode* node, std::vector<int>& v)  {
      if (node == nullptr) {
         return;
      }
      get_nodes1(node->Left,v);
```

```cpp
    v.push_back(node->Id);
    get_nodes1(node->Right, v);
  }
std::vector<int> get_nodes(TNode* head)  {
    std::vector<int> result;
    get_nodes1(head, result);
    return result;
  }

TNode* AddElement(TNode* p, unsigned long long k) {
  if( !p ) {
    return new TNode(k);
  }
  if (k<p->Id) {
    p->Left = AddElement(p->Left,k);
  }
  else if (k>p->Id) {
    p->Right = AddElement(p->Right,k);
  }
  return BalanceTree(p);
}
TNode* FindMinimum(TNode* p)  {
  if (p->Left==0) {
    return p;
  }
  return FindMinimum(p->Left);
}
TNode* RemoveMinimum(TNode* p)
{
  if( p->Left==0 ) {
    return p->Right;
  }
  p->Left = RemoveMinimum(p->Left);
  return BalanceTree(p);
}
TNode* RemoveElement(TNode* p, unsigned long long k) {
  if (!p) {
    return 0;
  }
  if(k<p->Id) {
    p->Left = RemoveElement(p->Left,k);
  }
```

```
    else if(k>p->Id) {
        p->Right = RemoveElement(p->Right,k);
    }
    if (k==p->Id) {
        TNode* q = p->Left;
        TNode* r = p->Right;
        delete p;
        if (r==0) {
            return q;
        }
        TNode* min = FindMinimum(r);
        min->Right = RemoveMinimum(r);
        min->Left = q;
        return BalanceTree(min);
    }
    return BalanceTree(p);

}
void RemoveTree(TNode* p) {
    if (!p) {
        return;
    }
    RemoveTree(p->Left);
    RemoveTree(p->Right);
    delete p;
}
TNode* FindById(TNode* p, unsigned long long Id) {
    if (p==0) {
        return 0;
    }

    if (Id<p->Id) {
        return FindById(p->Left,Id);
    }
    if (Id>p->Id) {
        return FindById(p->Right,Id);
    }
    return p;
}


```

```cpp
int main() {
  std::string command;
  TNode* head  = 0;
  size_t child_pid = 0;
  int child_id = 0;
  zmq::context_t context(1);
  zmq::socket_t main_socket(context, ZMQ_REQ);
  int linger = 0;
  main_socket.setsockopt(ZMQ_SNDTIMEO, 2000);
  //main_socket.setsockopt(ZMQ_RCVTIMEO, 2000);
  main_socket.setsockopt(ZMQ_LINGER, &linger, sizeof(linger));
  //main_socket.connect(get_connect_name(30000));
  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 {
//                  if (child_id == node_id) {
//                      std::cout << "Error: Already exists";
//                  }
        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") {
            head = AddElement(head,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";
            head = RemoveElement(head,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") {
            head = RemoveElement(head,node_id);
        }
        std::cout << recieved_message << "\n";

    } else if (command == "exec") {
        int id, n;
        std::cin >> id >> n;
        std::vector<int> numbers(n);
```

```cpp
      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 = get_nodes(head);
      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_node.cpp:

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


int main(int argc, char** argv) { //аргументы - айди и номер порта, к 
которому нужно подключиться
//    zmq::context_t context (1);

//    zmq::message_t msg(strlen(argv[1]));
//    zmq::message_t msg_2(strlen(argv[2]));
//    zmq::message_t rcv;
//    memcpy(msg.data(), argv[1],strlen(argv[1]));
//    memcpy(msg_2.data(), argv[2],strlen(argv[2]));
//    socket.send(msg);
//    socket.recv(&rcv);
//    socket.send(msg_2);
    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);

//    zmq::socket_t socket (context, ZMQ_REQ);
//    socket.connect ("tcp://127.0.0.1:5555");
    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_RCVTIMEO, 2000);
    left_socket.setsockopt(ZMQ_LINGER, &linger, sizeof(linger));
    right_socket.setsockopt(ZMQ_SNDTIMEO, 2000);
    //right_socket.setsockopt(ZMQ_RCVTIMEO, 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") {  // УБИТЬ ВСЕХ ДЕТЕЙ
            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;
        }
    }

}
```

server_functions.cpp:

```
#include "server.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 = 40000;
    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);
}
```

server_functions.h:

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

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);
```

Демонстрация работы программы

```
sergey@sergey-HP-Pavilion-dv7-Notebook-PC:~/Рабочий стол/work/os_lab_06$
strace ./terminal

create 1

Ok:10200

create 2

Ok:10205

create 3

Ok:10210

create 5

Ok:10215

exec 5 3 1 2 3

Ok:5:6

pingall

Ok: 1 2 3 5

create 5

Error: Already exists

exec 6 3 1 2 3

Error:6: Not found

remove 2

Ok

pingall

Ok: 1

exec 3 3 1 2 3

Error:3: Not found

exit

sergey@sergey-HP-Pavilion-dv7-Notebook-PC:~/Рабочий стол/work/os_lab_06$
strace ./terminal

execve("./terminal", ["./terminal"], 0x7ffc446b5c20 /* 55 vars */) = 0
```

```
brk(NULL)                               = 0x561cd2db5000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=82780, ...}) = 0
mmap(NULL, 82780, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f5d5d1df000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/local/lib/libzmq.so.5", O_RDONLY|O_CLOEXEC) =
3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240\5\1\0\0\0\0\0"..., 832)
= 832
fstat(3, {st_mode=S_IFREG|0755, st_size=9150296, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7f5d5d1dd000
mmap(NULL, 2635120, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_DENYWRITE, 3, 0) = 0x7f5d5cd49000
mprotect(0x7f5d5cdc8000, 2093056, PROT_NONE) = 0
mmap(0x7f5d5cfc7000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x7e000) = 0x7f5d5cfc7000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/libstdc++.so.6", O_RDONLY|
O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\304\10\0\0\0\0\0"...,
832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=1594864, ...}) = 0
mmap(NULL, 3702848, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_DENYWRITE, 3, 0) = 0x7f5d5c9c0000
mprotect(0x7f5d5cb39000, 2097152, PROT_NONE) = 0
```

```
mmap(0x7f5d5cd39000, 49152, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x179000) = 0x7f5d5cd39000

mmap(0x7f5d5cd45000, 12352, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f5d5cd45000

close(3)                          = 0

access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libgcc_s.so.1", O_RDONLY|
O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\300*\0\0\0\0\0\0"..., 832)
= 832

fstat(3, {st_mode=S_IFREG|0644, st_size=96616, ...}) = 0

mmap(NULL, 2192432, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_DENYWRITE, 3, 0) = 0x7f5d5c7a8000

mprotect(0x7f5d5c7bf000, 2093056, PROT_NONE) = 0

mmap(0x7f5d5c9be000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x16000) = 0x7f5d5c9be000

close(3)                          = 0

access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|
O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0\0"...,
832) = 832

fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0

mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_DENYWRITE, 3, 0) = 0x7f5d5c3b7000

mprotect(0x7f5d5c59e000, 2097152, PROT_NONE) = 0

mmap(0x7f5d5c79e000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f5d5c79e000

mmap(0x7f5d5c7a4000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f5d5c7a4000

close(3)                          = 0

access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
```

```
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/librt.so.1", O_RDONLY|
O_CLOEXEC) = 3

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

fstat(3, {st_mode=S_IFREG|0644, st_size=31680, ...}) = 0

mmap(NULL, 2128864, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_DENYWRITE, 3, 0) = 0x7f5d5c1af000

mprotect(0x7f5d5c1b6000, 2093056, PROT_NONE) = 0

mmap(0x7f5d5c3b5000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x6000) = 0x7f5d5c3b5000

close(3)                        = 0

access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|
O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000b\0\0\0\0\0\0"..., 832) =
832

fstat(3, {st_mode=S_IFREG|0755, st_size=144976, ...}) = 0

mmap(NULL, 2221184, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_DENYWRITE, 3, 0) = 0x7f5d5bf90000

mprotect(0x7f5d5bfaa000, 2093056, PROT_NONE) = 0

mmap(0x7f5d5c1a9000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x19000) = 0x7f5d5c1a9000

mmap(0x7f5d5c1ab000, 13440, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f5d5c1ab000

close(3)                        = 0

access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.so.6", O_RDONLY|
O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\200\272\0\0\0\0\0\0"...,
832) = 832

fstat(3, {st_mode=S_IFREG|0644, st_size=1700792, ...}) = 0
```

```
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7f5d5d1db000

mmap(NULL, 3789144, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_DENYWRITE, 3, 0) = 0x7f5d5bbf2000

mprotect(0x7f5d5bd8f000, 2093056, PROT_NONE) = 0

mmap(0x7f5d5bf8e000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x19c000) = 0x7f5d5bf8e000

close(3)                            = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7f5d5d1d8000

arch_prctl(ARCH_SET_FS, 0x7f5d5d1d8740) = 0

mprotect(0x7f5d5c79e000, 16384, PROT_READ) = 0

mprotect(0x7f5d5bf8e000, 4096, PROT_READ) = 0

mprotect(0x7f5d5c1a9000, 4096, PROT_READ) = 0

mprotect(0x7f5d5c3b5000, 4096, PROT_READ) = 0

mprotect(0x7f5d5c9be000, 4096, PROT_READ) = 0

mprotect(0x7f5d5cd39000, 40960, PROT_READ) = 0

mprotect(0x7f5d5cfc7000, 20480, PROT_READ) = 0

mprotect(0x561cd1937000, 4096, PROT_READ) = 0

mprotect(0x7f5d5d1f4000, 4096, PROT_READ) = 0

munmap(0x7f5d5d1df000, 82780)           = 0

set_tid_address(0x7f5d5d1d8a10)         = 8452

set_robust_list(0x7f5d5d1d8a20, 24)     = 0

rt_sigaction(SIGRTMIN, {sa_handler=0x7f5d5bf95cb0, sa_mask=[],
sa_flags=SA_RESTORER|SA_SIGINFO, sa_restorer=0x7f5d5bfa2890}, NULL,
8) = 0

rt_sigaction(SIGRT_1, {sa_handler=0x7f5d5bf95d50, sa_mask=[],
sa_flags=SA_RESTORER|SA_RESTART|SA_SIGINFO,
sa_restorer=0x7f5d5bfa2890}, 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)                              = 0x561cd2db5000

brk(0x561cd2dd6000)                    = 0x561cd2dd6000

futex(0x7f5d5cd4609c, FUTEX_WAKE_PRIVATE, 2147483647) = 0

futex(0x7f5d5cd460a8, 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

openat(AT_FDCWD, "/dev/urandom", O_RDONLY) = 4

read(4, "\306\220P\31", 4)             = 4

eventfd2(0, EFD_CLOEXEC)               = 5

fcntl(5, F_GETFL)                      = 0x2 (flags O_RDWR)

fcntl(5, F_SETFL, O_RDWR|O_NONBLOCK)   = 0

fcntl(5, F_GETFL)                      = 0x802 (flags O_RDWR|O_NONBLOCK)

fcntl(5, F_SETFL, O_RDWR|O_NONBLOCK)   = 0

epoll_create1(EPOLL_CLOEXEC)           = 6

epoll_ctl(6, EPOLL_CTL_ADD, 5, {0, {u32=3537673040,
u64=94681796746064}}) = 0

epoll_ctl(6, EPOLL_CTL_MOD, 5, {EPOLLIN, {u32=3537673040,
u64=94681796746064}}) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|
MAP_STACK, -1, 0) = 0x7f5d5b3f1000

mprotect(0x7f5d5b3f2000, 8388608, PROT_READ|PROT_WRITE) = 0

clone(child_stack=0x7f5d5bbf0fb0, flags=CLONE_VM|CLONE_FS|
CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|
CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
parent_tidptr=0x7f5d5bbf19d0, tls=0x7f5d5bbf1700,
child_tidptr=0x7f5d5bbf19d0) = 8453
```

```
sched_getparam(8453, [0])            = 0
sched_getscheduler(8453)             = 0 (SCHED_OTHER)
sched_setscheduler(8453, SCHED_OTHER, [0]) = 0
eventfd2(0, EFD_CLOEXEC)             = 7
fcntl(7, F_GETFL)                    = 0x2 (flags O_RDWR)
fcntl(7, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
fcntl(7, F_GETFL)                    = 0x802 (flags O_RDWR|O_NONBLOCK)
fcntl(7, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
epoll_create1(EPOLL_CLOEXEC)         = 8
epoll_ctl(8, EPOLL_CTL_ADD, 7, {0, {u32=3537674912,
u64=94681796747936}}) = 0
epoll_ctl(8, EPOLL_CTL_MOD, 7, {EPOLLIN, {u32=3537674912,
u64=94681796747936}}) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|
MAP_STACK, -1, 0) = 0x7f5d5abf0000
mprotect(0x7f5d5abf1000, 8388608, PROT_READ|PROT_WRITE) = 0
clone(child_stack=0x7f5d5b3effb0, flags=CLONE_VM|CLONE_FS|
CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|
CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
parent_tidptr=0x7f5d5b3f09d0, tls=0x7f5d5b3f0700,
child_tidptr=0x7f5d5b3f09d0) = 8454
sched_getparam(8454, [0])            = 0
sched_getscheduler(8454)             = 0 (SCHED_OTHER)
sched_setscheduler(8454, SCHED_OTHER, [0]) = 0
eventfd2(0, EFD_CLOEXEC)             = 9
fcntl(9, F_GETFL)                    = 0x2 (flags O_RDWR)
fcntl(9, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
fcntl(9, F_GETFL)                    = 0x802 (flags O_RDWR|O_NONBLOCK)
fcntl(9, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
poll([{fd=9, events=POLLIN}], 1, 0)     = 0 (Timeout)
```

```
socket(AF_NETLINK, SOCK_RAW|SOCK_CLOEXEC, NETLINK_ROUTE) =
10

bind(10, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12) = 0

getsockname(10, {sa_family=AF_NETLINK, nl_pid=8452,
nl_groups=00000000}, [12]) = 0

sendto(10, {{len=20, type=RTM_GETLINK, flags=NLM_F_REQUEST|
NLM_F_DUMP, seq=1579764091, pid=0}, {ifi_family=AF_UNSPEC, ...}}, 20,
0, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12) = 20

recvmsg(10, {msg_name={sa_family=AF_NETLINK, nl_pid=0,
nl_groups=00000000}, msg_namelen=12, msg_iov=[{iov_base=[{{len=1316,
type=RTM_NEWLINK, flags=NLM_F_MULTI, seq=1579764091, pid=8452},
{ifi_family=AF_UNSPEC, ifi_type=ARPHRD_LOOPBACK,
ifi_index=if_nametoindex("lo"), ifi_flags=IFF_UP|IFF_LOOPBACK|
IFF_RUNNING|0x10000, 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=0x32 /* IFLA_??? */}, "\x00\x00\x00\x00"}, {{nla_len=8,
nla_type=0x33 /* IFLA_??? */}, "\x00\x00\x00\x00"}, {{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=12, nla_type=IFLA_QDISC},
"noqueue"}, {{nla_len=8, nla_type=IFLA_CARRIER_CHANGES}, 0},
{{nla_len=5, nla_type=IFLA_PROTO_DOWN}, 0}, {{nla_len=8, nla_type=0x2f /
* IFLA_??? */}, "\x00\x00\x00\x00"}, {{nla_len=8, nla_type=0x30 /* IFLA_???
*/}, "\x00\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}, "\x00\x00\x00\x00\x00\x00"}, {{nla_len=10,
nla_type=IFLA_BROADCAST}, "\x00\x00\x00\x00\x00\x00"}, {{nla_len=196,
nla_type=IFLA_STATS64}, {rx_packets=3483, tx_packets=3483,
rx_bytes=354119, tx_bytes=354119, 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=3483, tx_packets=3483,
rx_bytes=354119, tx_bytes=354119, 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=760,
nla_type=IFLA_AF_SPEC}, "\x88\x00\x02\x00\x84\x00\x01\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x01\x00\x00\x00\x01\
x00\x00\x00"...}]}, {{len=1324, type=RTM_NEWLINK, flags=NLM_F_MULTI,
seq=1579764091, pid=8452}, {ifi_family=AF_UNSPEC,
ifi_type=ARPHRD_ETHER, ifi_index=if_nametoindex("eno1"),
ifi_flags=IFF_UP|IFF_BROADCAST|IFF_MULTICAST, ifi_change=0},
[{{nla_len=9, nla_type=IFLA_IFNAME}, "eno1"}, {{nla_len=8,
nla_type=IFLA_TXQLEN}, 1000}, {{nla_len=5, nla_type=IFLA_OPERSTATE},
2}, {{nla_len=5, nla_type=IFLA_LINKMODE}, 0}, {{nla_len=8,
nla_type=IFLA_MTU}, 1500}, {{nla_len=8, nla_type=0x32 /* IFLA_??? */}, "\
x3c\x00\x00\x00"}, {{nla_len=8, nla_type=0x33 /* IFLA_??? */}, "\xf0\x23\x00\
x00"}, {{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}, 0}, {{nla_len=13, nla_type=IFLA_QDISC},
"fq_codel"}, {{nla_len=8, nla_type=IFLA_CARRIER_CHANGES}, 1},
{{nla_len=5, nla_type=IFLA_PROTO_DOWN}, 0}, {{nla_len=8, nla_type=0x2f /
* IFLA_??? */}, "\x00\x00\x00\x00"}, {{nla_len=8, nla_type=0x30 /* IFLA_???
*/}, "\x01\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}, "\xa0\xb3\xcc\x44\x4a\xad"}, {{nla_len=10,
nla_type=IFLA_BROADCAST}, "\xff\xff\xff\xff\xff\xff"}, {{nla_len=196,
nla_type=IFLA_STATS64}, {rx_packets=0, tx_packets=0, rx_bytes=0,
tx_bytes=0, 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=0, tx_packets=0, rx_bytes=0, tx_bytes=0,
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=760,
nla_type=IFLA_AF_SPEC}, "\x88\x00\x02\x00\x84\x00\x01\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x01\x00\x00\x00\x01\
x00\x00\x00"...}]}], iov_len=4096}], msg_iovlen=1, msg_controllen=0,
msg_flags=0}, 0) = 2640

recvmsg(10, {msg_name={sa_family=AF_NETLINK, nl_pid=0,
nl_groups=00000000}, msg_namelen=12, msg_iov=[{iov_base={{len=1316,
type=RTM_NEWLINK, flags=NLM_F_MULTI, seq=1579764091, pid=8452},
{ifi_family=AF_UNSPEC, ifi_type=ARPHRD_ETHER,
ifi_index=if_nametoindex("wlo1"), ifi_flags=IFF_UP|IFF_BROADCAST|
IFF_RUNNING|IFF_MULTICAST|0x10000, ifi_change=0}, [{{nla_len=9,
nla_type=IFLA_IFNAME}, "wlo1"}, {{nla_len=8, nla_type=IFLA_TXQLEN},
1000}, {{nla_len=5, nla_type=IFLA_OPERSTATE}, 6}, {{nla_len=5,
nla_type=IFLA_LINKMODE}, 1}, {{nla_len=8, nla_type=IFLA_MTU}, 1500},
{{nla_len=8, nla_type=0x32 /* IFLA_??? */}, "\x00\x01\x00\x00"}, {{nla_len=8,
nla_type=0x33 /* IFLA_??? */}, "\x00\x09\x00\x00"}, {{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}, 4}, {{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=7, nla_type=IFLA_QDISC}, "mq"},
{{nla_len=8, nla_type=IFLA_CARRIER_CHANGES}, 4}, {{nla_len=5,
nla_type=IFLA_PROTO_DOWN}, 0}, {{nla_len=8, nla_type=0x2f /* IFLA_???
*/}, "\x02\x00\x00\x00"}, {{nla_len=8, nla_type=0x30 /* IFLA_??? */}, "\x02\
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}, "\x68\x5d\x43\x91\x76\xca"}, {{nla_len=10,
nla_type=IFLA_BROADCAST}, "\xff\xff\xff\xff\xff\xff"}, {{nla_len=196,
nla_type=IFLA_STATS64}, {rx_packets=363477, tx_packets=154844,
rx_bytes=445906664, tx_bytes=22551467, 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=363477, tx_packets=154844, rx_bytes=445906664,
tx_bytes=22551467, 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=760, nla_type=IFLA_AF_SPEC}, "\x88\x00\x02\x00\x84\x00\x01\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x01\x00\
x00\x00\x01\x00\x00\x00"...}]}, iov_len=4096}], msg_iovlen=1,
msg_controllen=0, msg_flags=0}, 0) = 1316

recvmsg(10, {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=1579764091, pid=8452}, 0},
iov_len=4096}], msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 20

sendto(10, {{len=20, type=RTM_GETADDR, flags=NLM_F_REQUEST|
NLM_F_DUMP, seq=1579764092, pid=0}, {ifa_family=AF_UNSPEC, ...}}, 20,
0, {sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, 12) = 20

recvmsg(10, {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=1579764092, pid=8452},
{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=2151, tstamp=2151}}]}, {{len=88,
type=RTM_NEWADDR, flags=NLM_F_MULTI, seq=1579764092, pid=8452},
{ifa_family=AF_INET, ifa_prefixlen=16, ifa_flags=0,
ifa_scope=RT_SCOPE_UNIVERSE, ifa_index=if_nametoindex("wlo1")},
[{{nla_len=8, nla_type=IFA_ADDRESS}, 172.17.142.166}, {{nla_len=8,
nla_type=IFA_LOCAL}, 172.17.142.166}, {{nla_len=8,
nla_type=IFA_BROADCAST}, 172.17.255.255}, {{nla_len=9,
nla_type=IFA_LABEL}, "wlo1"}, {{nla_len=8, nla_type=IFA_FLAGS},
IFA_F_NOPREFIXROUTE}, {{nla_len=20, nla_type=IFA_CACHEINFO},
{ifa_prefered=1093, ifa_valid=1093, cstamp=1071314, tstamp=1146136}}]}],
iov_len=4096}], msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 164

recvmsg(10, {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=1579764092, pid=8452},
{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=2151, tstamp=2151}},
{{nla_len=8, nla_type=IFA_FLAGS}, IFA_F_PERMANENT}]}, {{len=72,
type=RTM_NEWADDR, flags=NLM_F_MULTI, seq=1579764092, pid=8452},
{ifa_family=AF_INET6, ifa_prefixlen=64, ifa_flags=IFA_F_PERMANENT,
ifa_scope=RT_SCOPE_LINK, ifa_index=if_nametoindex("wlo1")},
[{{nla_len=20, nla_type=IFA_ADDRESS}, fe80::9159:e8ba:926d:aa5a},
{{nla_len=20, nla_type=IFA_CACHEINFO}, {ifa_prefered=4294967295,
ifa_valid=4294967295, cstamp=1071308, tstamp=1071436}}, {{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(10, {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=1579764092, pid=8452}, 0},
iov_len=4096}], msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 20

close(10)                        = 0

socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC, IPPROTO_TCP) = 10

setsockopt(10, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0

bind(10, {sa_family=AF_INET, sin_port=htons(40000),
sin_addr=inet_addr("127.0.0.1")}, 16) = 0

listen(10, 100)                  = 0

getsockname(10, {sa_family=AF_INET, sin_port=htons(40000),
sin_addr=inet_addr("127.0.0.1")}, [128->16]) = 0

write(7, "\1\0\0\0\0\0\0\0", 8)        = 8

write(9, "\1\0\0\0\0\0\0\0", 8)        = 8

fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0

read(0, create 3

"create 3\n", 1024)              = 9

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|
CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f5d5d1d8a10) = 8657

poll([{fd=9, events=POLLIN}], 1, 0)    = 1 ([{fd=9, revents=POLLIN}])

read(9, "\1\0\0\0\0\0\0\0", 8)         = 8

poll([{fd=9, events=POLLIN}], 1, 0)    = 0 (Timeout)

poll([{fd=9, events=POLLIN}], 1, 2000)  = 1 ([{fd=9, revents=POLLIN}])
```

```
read(9, "\1\0\0\0\0\0\0\0", 8)          = 8
poll([{fd=9, events=POLLIN}], 1, 0)     = 0 (Timeout)
poll([{fd=9, events=POLLIN}], 1, -1)    = 1 ([{fd=9, revents=POLLIN}])
read(9, "\1\0\0\0\0\0\0\0", 8)          = 8
poll([{fd=9, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(7, "\1\0\0\0\0\0\0\0", 8)         = 8
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0
write(1, "Ok:8657\n", 8Ok:8657
)                = 8
read(0, create 5
"create 5\n", 1024)             = 9
poll([{fd=9, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(7, "\1\0\0\0\0\0\0\0", 8)         = 8
poll([{fd=9, events=POLLIN}], 1, -1)    = 1 ([{fd=9, revents=POLLIN}])
read(9, "\1\0\0\0\0\0\0\0", 8)          = 8
poll([{fd=9, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(1, "Ok:8664\n", 8Ok:8664
)                = 8
read(0, exec 3 3 1 2 3
"exec 3 3 1 2 3\n", 1024)       = 15
poll([{fd=9, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(7, "\1\0\0\0\0\0\0\0", 8)         = 8
poll([{fd=9, events=POLLIN}], 1, -1)    = 1 ([{fd=9, revents=POLLIN}])
read(9, "\1\0\0\0\0\0\0\0", 8)          = 8
poll([{fd=9, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(1, "Ok:3:6\n", 7Ok:3:6
)                 = 7
read(0, pingall
"pingall\n", 1024)              = 8
```

```
poll([{fd=9, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(7, "\1\0\0\0\0\0\0\0", 8)         = 8
poll([{fd=9, events=POLLIN}], 1, -1)    = 1 ([{fd=9, revents=POLLIN}])
read(9, "\1\0\0\0\0\0\0\0", 8)          = 8
poll([{fd=9, events=POLLIN}], 1, 0)     = 0 (Timeout)
write(1, "Ok: 3 5\n", 8Ok: 3 5
)                = 8
read(0, exit
"exit\n", 1024)                 = 5
write(5, "\1\0\0\0\0\0\0\0", 8)         = 8
write(9, "\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(7, "\1\0\0\0\0\0\0\0", 8)         = 8
close(8)                                = 0
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 +++
```

Вывод

В результате данной лабораторной работы я научился работать с технологией очереди сообщений, создавать программы, создающие и связывающие процессы в определенные топологии. Так же я приобрел полезные навыки в отладке многопроцессорных приложений.