МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Курсовой проект
по курсу «Операционные системы»

Выполнил: А. А. Устинов
Группа: М8О-207БВ-24
Преподаватель: Е. С. Миронов

Москва, 2025

Условие

Цель работы: 1. Приобретение практических навыков в использовании знаний, полученных в течении курса 2. Проведение исследования в выбранной предметной области

Задание: Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Вариант: 10

Метод Решения

Задача состояла в создании многопользовательской версии консольной игры "Быки и Коровы"с архитектурой Клиент-Сервер, реализованной на языке C++ с использованием Berkeley Sockets. Ключевая особенность решения — возможность динамического задания количества игроков при запуске сервера и обеспечение надежного, пошагового игрового процесса.

Архитектура представляет собой классическую модель Клиент-Сервер:

- **Сервер:** Отвечает за прием подключений, управление очередью ходов, хранение секретного кода, расчет результатов ("Быки"и "Коровы") и рассылку служебных сообщений.
- **Клиент:** Предоставляет пользовательский интерфейс для ввода ходов и отображает результаты, полученные от сервера.

Взаимодействие осуществляется через TCP-сокеты.

Сетевой Контракт (Протокол)

Для обеспечения совместимости между Клиентом и Сервером разработан простой текстовый протокол. Все сообщения состоят из команды и, при необходимости, аргументов, разделенных пробелом.

1. **Идентификация.** `YOUR_ID [ID]`
   Сервер сообщает клиенту его уникальный сокет-дескриптор, который будет использоваться в качестве ID игрока для управления очередью.
2. **Ожидание.** `WAIT [N]`
   Сообщение клиенту о том, сколько еще игроков необходимо для начала игры.
3. **Ход.** `TURN [ID]`
   Сервер информирует всех клиентов о том, чей сейчас ход. Клиент, чей ID совпадает с `[ID]`, получает разрешение на ввод.
4. **Результат.** `RESULT [Bulls] [Cows]`
   Сервер сообщает результат последней попытки.
5. **Победа.** `WIN [ID]`
   Объявление победителя.
6. **Завершение.** `SERVER_SHUTDOWN`
   Команда для корректного завершения работы клиента (высылается при остановке сервера или таймауте).

Методы Обеспечения Динамичности и Надежности

1. Количество необходимых для старта игроков (`players_required`) задается аргументом командной строки при запуске сервера (`./server N`).
2. Сервер использует функцию `select()` для неблокирующего ожидания подключений и ходов от текущего игрока, а также для контроля глобального флага остановки (`keep_running`) по сигналу `SIGINT`.
3. Введен таймаут на ход (60 секунд), по истечении которого сервер принудительно завершает игру, предотвращая зависание сессии из-за бездействия клиента.

Архитектура программы

```
lab4_network_game/
├── inc/
│   └── game.h
├── src/
│   ├── game.cpp
│   ├── server.cpp
│   └── client.cpp
└── CMakeListst
```

Описание Программных Модулей

Файл `inc/game.h` и `src/game.cpp`

Назначение: Инкапсуляция всей игровой логики.

- Установлена константа `CODE_LENGTH = 3`, что делает секретное число трехзначным.
- Реализована функция `generate_code()` для создания уникального 3-значного числа.
- Функция `process_guess()` выполняет сравнение хода игрока с секретным кодом, возвращая количество Быков и Коров. Условием победы является **3 Быка**.

Файл `src/server.cpp`

Назначение: Центральный узел управления игрой.
Логика:

- Принимает количество игроков в качестве аргумента командной строки.
- Использует прослушивающий сокет и цикл ‘while‘ для приема подключений.
- Управляет вектором клиентских сокет-дескрипторов (`clients`).
- Внутри игрового цикла перебирает клиентов по очереди (оператор `for` по вектору `clients`).

- Использует `select` для блокирующего ожидания ввода от текущего игрока с установленным таймаутом.

Файл `src/client.cpp`

Назначение: Интерфейс пользователя.
Логика:

- Устанавливает соединение с сервером.
- Сохраняет присвоенный сервером `server_client_id` для дальнейшей идентификации.
- Работает в бесконечном цикле `while` ожидания сообщений от сервера (`recv`).
- **Блокировка ввода:** Ввод хода разрешен только при получении команды `TURN` с ID, совпадающим с `server_client_id`.
- Реализована команда `QUIT` для корректного выхода.

Результаты

Разработанное сетевое решение было успешно протестировано, подтверждая корректность реализации архитектуры Клиент-Сервер, протокола взаимодействия и механизма динамической настройки игры.

Проверка Архитектуры Клиент-Сервер

Было подтверждено, что сетевое взаимодействие основано на протоколе TCP, и сервер корректно обрабатывает несколько одновременных подключений.

- **Идентификация:** Каждый клиент при подключении успешно получает от сервера уникальный `YOUR_ID` (сокет-дескриптор, например, 4, 5, 6), который используется для управления очередью ходов.
- **Старт игры:** Сервер корректно ждет заданное количество игроков (`players_required`) и только после этого рассылает команду `START`.
- **Зависимость:** Клиентский код не содержит логики игры, а полностью зависит от сервера, что подтверждает архитектуру тонкого клиента.

Проверка Механизма Управления Сессией и Очередностью

Протестирован игровой цикл с использованием двух и более клиентов.

- **Очередность ходов:** Сервер строго следует последовательности, рассылая команду `TURN [ID]` в порядке, соответствующем вектору подключенных клиентов.
- **Блокировка ввода (Клиент):** Клиентский модуль корректно блокирует пользовательский ввод, пока не получит команду `TURN` со своим `ID`, предотвращая преждевременную отправку данных (решена проблема из ранних этапов разработки).

- **Проверка логики игры:** Ввод трехзначных чисел корректно обрабатывается модулем `Game`, и результаты (`RESULT Bulls Cows`) точно соответствуют правилам игры.

Проверка Динамической Настройки и Надежности

Была проверена способность системы адаптироваться к изменяющимся условиям.

- **Динамическое количество игроков:** Запуск сервера с аргументом `./server 3` успешно заставил сервер ожидать трех клиентов, а цикл ходов корректно переключался между всеми тремя игроками.
- **Обработка таймаута** (`select`)**:** При бездействии текущего игрока в течение 60 секунд срабатывает таймаут, сервер корректно завершает сессию, рассылая команду `SERVER_SHUTDOWN` всем остальным клиентам.
- **Корректное завершение:** Команда `QUIT` от клиента или сигнал `SIGINT` (Ctrl+C) на сервере приводят к контролируемой остановке и закрытию всех сокетов.

Пример взаимодействия

Ниже приведен пример сессии взаимодействия с двумя клиентами, демонстрирующий строгую очередность ходов и корректный расчет результата для трехзначного кода.

Дано: Секретный код сервера: `359`. Требуется 2 игрока.

Сессия Клиента 1 (ID 4):

Листинг 1: Сессия работы Клиента 1

```
$ ./client
Connected! Waiting for server ID...
Server assigned you ID: 4
Server: Waiting for 1 more player(s)...

--- GAME STARTED ---
>>> YOUR TURN (4) <<<
Enter your guess (3 unique digits) or QUIT: 123
Result: Bulls: 0, Cows: 1 // 3 on 3rd pos
Waiting for Player 5's guess...
Result: Bulls: 0, Cows: 1

>>> YOUR TURN (4) <<<
Enter your guess (3 unique digits) or QUIT: 567
Result: Bulls: 0, Cows: 1 // 5 on 1st pos
Waiting for Player 5's guess...
Result: Bulls: 0, Cows: 1
```

Сессия Клиента 2 (ID 5):

Листинг 2: Сессия работы Клиента 2

```
$ ./client
Connected! Waiting for server ID...
Server assigned you ID: 5

```

```
--- GAME STARTED ---
Waiting for Player 4's guess...
Result: Bulls: 0, Cows: 1

>>> YOUR TURN (5) <<<
Enter your guess (3 unique digits) or QUIT: 890
Result: Bulls: 0, Cows: 1 // 9 on 9th pos
Waiting for Player 4's guess...
Result: Bulls: 0, Cows: 1

Waiting for Player 4's guess...
Result: Bulls: 0, Cows: 1

>>> YOUR TURN (5) <<<
Enter your guess (3 unique digits) or QUIT: 359
Result: Bulls: 3, Cows: 0 // Win!

!!! GAME OVER !!! Player 5 won the game!
```

Ключевые особенности разработанного решения

1. **Настраиваемая Сессия:** Возможность задать количество игроков (`N`) при запуске сервера, что обеспечивает гибкость и повторное использование кода для разного числа участников.

2. **Контроль Ввода/Вывода** (`select`)**:** Использование системного вызова `select()` обеспечивает эффективное управление сокетными дескрипторами и позволяет серверу одновременно отслеживать ожидание подключения и таймаут хода.

3. **Тонкий Клиент / Строгий Протокол:** Разделение логики: игровая логика полностью инкапсулирована в классе `Game` на сервере, а клиент выступает только в роли интерфейса, следующего четкому текстовому протоколу.

4. **Обработка исключений:** Реализованы механизмы корректного завершения работы (`QUIT`, `SIGINT`, Таймаут) с рассылкой широковещательных сообщений `SERVER_SHUTDOWN`.

Заключение

В результате выполнения курсовой работы была успешно разработана и протестирована многопользовательская сетевая игра "Быки и Коровы основанная на архитектуре Клиент-Сервер и реализованная на C++ с использованием Berkeley Sockets. Было достигнуто полное разделение логики: сервер управляет всей игровой сессией, очередностью ходов и содержит инкапсулированную логику расчета Быков/Коров, в то время как клиент выступает в роли тонкого интерфейса. Все поставленные задачи были решены, включая обеспечение стабильного TCP-взаимодействия и возможность динамического задания количества игроков при запуске сервера. Для повышения эффективности игры длина секретного кода была сокращена до трех цифр. Ключевым достижением стало применение системного вызова `select()`, что обеспечило надежное управление сокетными дескрипторами, эффективную обработку таймаутов на ход и корректную реакцию на сигналы завершения. Использование четко определенного текстового протокола гарантирует структурированный обмен данными и общую надежность системы. В целом, созданное решение является функциональным, демонстрирует грамотное проектирование сетевых приложений и готово к масштабированию.

Исходная программа

Файл game.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <random>
#include <chrono>
#include <sstream>
#include <set>

#include "game.h"

Game::Game(int players) : num_players(players) {
    generate_code();
    std::cout << "DEBUG: Secret code set to: " << secret_code << std::endl;
}

bool Game::is_valid(const std::string& guess) {
    if (guess.length() != CODE_LENGTH) {
        return false;
    }

    for (char c : guess) {
        if (!isdigit(c)) {
            return false;
        }
    }

    // Characters must be unique
    std::set<char> unique_digits(guess.begin(), guess.end());
    if (unique_digits.size() != CODE_LENGTH) {
        return false;
    }

    return true;
}

void Game::generate_code() {
    std::vector<int> digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    std::shuffle(digits.begin(), digits.end(), std::default_random_engine(seed));

    secret_code = "";


    int start_index = 0;
    if (CODE_LENGTH > 1 && digits[0] == 0) {
        auto it = std::find_if(digits.begin() + 1, digits.end(), [](int d){ return d !=
             0; });
        if (it != digits.end()) {
            std::swap(digits[0], *it);
        }
    }

    for (int i = 0; i < CODE_LENGTH; ++i) {
        secret_code += std::to_string(digits[i]);
```

```
    }
}

std::pair<int, int> Game::check_guess(const std::string& guess) {
    int bulls = 0;
    int cows = 0;

    for (int i = 0; i < CODE_LENGTH; ++i) {
        for (int j = 0; j < CODE_LENGTH; ++j) {
            if (guess[i] == secret_code[j]) {
                if (i == j) {
                    bulls++;
                } else {
                    cows++;
                }
            }
        }
    }
    return {bulls, cows};
}

std::string Game::process_guess(const std::string& guess, bool& win) {
    win = false;

    if (!is_valid(guess)) {
        return "RESULT 0 0 (Invalid)";
    }

    std::pair<int, int> result = check_guess(guess);
    int bulls = result.first;
    int cows = result.second;

    if (bulls == CODE_LENGTH) {
        win = true;
    }

    std::stringstream ss;
    ss << "RESULT " << bulls << " " << cows;
    return ss.str();
}
```

Листинг 3: Реализация логики игры

Файл server.cpp

```
#include <arpa/inet.h>
#include <unistd.h>
#include <iostream>
#include <vector>
#include <cstring>
#include <signal.h>
#include <sys/select.h>
#include <sys/time.h>
#include <algorithm>
#include <stdexcept>

```

```cpp
#include "game.h"


volatile sig_atomic_t keep_running = 1;

void signal_handler(int sig) {
    if (sig == SIGINT) {
        keep_running = 0;
        std::cout << "\nSIGINT received. Shutting down gracefully..." << std::endl;
    }
}

void broadcast(const std::vector<int>& clients, const std::string& message) {
    std::string msg_with_newline = message + "\n";
    for (int c : clients) {
        send(c, msg_with_newline.c_str(), msg_with_newline.size(), 0);
    }
}

int main(int argc, char* argv[]) {

    signal(SIGINT, signal_handler);

    const int DEFAULT_PLAYERS = 2;
    int players_required = DEFAULT_PLAYERS;

    // Read cmd arguments
    if (argc == 2) {
        try {
            int requested_players = std::stoi(argv[1]);

            if (requested_players > 1) {
                players_required = requested_players;
            } else {
                std::cerr << "Error: Number of players must be greater than 1. Using
                    default (" << DEFAULT_PLAYERS << ")." << std::endl;
            }
        } catch (const std::exception& e) {
            std::cerr << "Error parsing player count argument. Using default (" <<
                DEFAULT_PLAYERS << ")." << std::endl;
        }
    }

    int server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // ports/adresses can be used again
    int opt = 1;
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt))) {
        perror("setsockopt failed");
        close(server_fd);
        return 1;
    }

    sockaddr_in addr{};
```

```
68      addr.sin_family = AF_INET;
69      addr.sin_port = htons(5555);
70      addr.sin_addr.s_addr = INADDR_ANY;
71
72      if (bind(server_fd, (sockaddr*)&addr, sizeof(addr)) < 0) {
73          perror("Bind failed (port 5555 likely in use)");
74          close(server_fd);
75          return 1;
76      }
77
78      if (listen(server_fd, 5) < 0) {
79          perror("Listen failed");
80          close(server_fd);
81          return 1;
82      }
83
84      std::cout << "Server started on port 5555. Required players: " << players_required
                << ". Waiting for clients..." << std::endl;
85
86      std::vector<int> clients;
87
88      // Find players
89      while ((int)clients.size() < players_required && keep_running) {
90          fd_set master_set;
91          FD_ZERO(&master_set);
92          FD_SET(server_fd, &master_set);
93
94          struct timeval timeout;
95          timeout.tv_sec = 1;
96          timeout.tv_usec = 0;
97
98          if (select(server_fd + 1, &master_set, nullptr, nullptr, &timeout) < 0) {
99              if (keep_running) perror("Select error during connect");
100             break;
101         }
102
103         if (FD_ISSET(server_fd, &master_set)) {
104             int c = accept(server_fd, nullptr, nullptr);
105             if (c < 0) continue;
106
107             clients.push_back(c);
108             std::cout << "Client connected: socket = " << c << std::endl;
109
110             std::string id_msg = "YOUR_ID " + std::to_string(c);
111             send(c, id_msg.c_str(), id_msg.size(), 0);
112
113             std::string wait_msg = "WAIT " + std::to_string(players_required - clients.
                    size());
114             send(c, wait_msg.c_str(), wait_msg.size(), 0);
115         }
116     }
117
118     // Game process
119     if ((int)clients.size() == players_required && keep_running) {
120
121         std::cout << "All players connected. Starting game" << std::endl;
122         Game game(players_required);
123         bool win = false;
```

```
124         char buf[128];
125 
126         broadcast(clients, "START");
127 
128         while (!win && keep_running) {
129             for (int i = 0; i < players_required; ++i) {
130                 if (!keep_running) break;
131 
132                 int sock = clients[i];
133 
134                 std::string turn_msg = "TURN " + std::to_string(sock);
135                 broadcast(clients, turn_msg);
136 
137                 fd_set read_set;
138                 FD_ZERO(&read_set);
139                 FD_SET(sock, &read_set);
140 
141                 struct timeval timeout_turn;
142                 timeout_turn.tv_sec = 60;
143                 timeout_turn.tv_usec = 0;
144 
145                 int ready_fds = select(sock + 1, &read_set, nullptr, nullptr, &
                        timeout_turn);
146 
147                 if (!keep_running) break;
148 
149                 if (ready_fds <= 0) {
150                     std::cout << "Client " << sock << " timed out or select error.
                            Aborting." << std::endl;
151                     broadcast(clients, "SERVER_SHUTDOWN");
152                     keep_running = 0;
153                     break;
154                 }
155 
156                 memset(buf, 0, sizeof(buf));
157                 int r = recv(sock, buf, sizeof(buf)-1, 0);
158 
159                 if (r <= 0) {
160                     std::cout << "Client " << sock << " disconnected unexpectedly (r="
                            << r << ")." << std::endl;
161                     broadcast(clients, "SERVER_SHUTDOWN");
162                     keep_running = 0;
163                     break;
164                 }
165 
166                 std::string guess(buf);
167                 guess.erase(std::remove_if(guess.begin(), guess.end(), [](char c){
                        return c == '\n' || c == '\r'; }), guess.end());
168 
169                 if (guess == "QUIT") {
170                     std::cout << "Client " << sock << " requested QUIT. Aborting game."
                            << std::endl;
171                     broadcast(clients, "SERVER_SHUTDOWN");
172                     keep_running = 0;
173                     break;
174                 }
175 
176                 std::cout << "Received guess from client " << sock << ": " << guess <<
```

```
                    std::endl;

                std::string resp = game.process_guess(guess, win);

                broadcast(clients, resp);

                if (win) {
                    std::string win_msg = "WIN " + std::to_string(sock);
                    broadcast(clients, win_msg);
                    break;
                }
            }
        }
    } else {
        broadcast(clients, "SERVER_SHUTDOWN");
    }

    std::cout << "Game over." << std::endl;

    for(int c : clients) {
        close(c);
    }

    close(server_fd);
    std::cout << "Server successfully shut down." << std::endl;

    return 0;
}
```

Листинг 4: Реализация работы сервера.

Файл client.cpp

```
#include <arpa/inet.h>
#include <unistd.h>
#include <iostream>
#include <string>
#include <cstring>
#include <algorithm>
#include <vector>
#include <sstream>

// str -> characters
std::vector<std::string> split(const std::string &s, char delimiter) {
    std::vector<std::string> tokens;
    std::string token;
    std::istringstream tokenStream(s);
    while (std::getline(tokenStream, token, delimiter)) {
        tokens.push_back(token);
    }
    return tokens;
}

void trim_message(std::string& str) {
    str.erase(std::remove_if(str.begin(), str.end(), [](char c){ return c == '\n' || c
        == '\r'; }), str.end());
```

```
}

int main() {
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("Socket creation failed");
        return 1;
    }

    sockaddr_in addr{};
    addr.sin_family = AF_INET;
    addr.sin_port = htons(5555);

    if (inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr) <= 0) {
        perror("Invalid address / Address not supported");
        close(sock);
        return 1;
    }

    std::cout << "Connecting to server..." << std::endl;

    if (connect(sock, (sockaddr*)&addr, sizeof(addr)) < 0) {
        perror("Connection Failed.");
        close(sock);
        return 1;
    }

    int server_client_id = -1;

    std::cout << "Connected! Waiting for server ID..." << std::endl;

    char buf[128];
    while (true) {
        memset(buf, 0, sizeof(buf));

        int r = recv(sock, buf, sizeof(buf)-1, 0);
        if (r <= 0) {
            std::cout << "\nServer disconnected or error occurred." << std::endl;
            break;
        }

        std::string msg(buf);
        trim_message(msg);

        std::vector<std::string> parts = split(msg, ' ');
        std::string command = parts[0];


        if (command == "YOUR_ID") {
            server_client_id = std::stoi(parts[1]);
            std::cout << "Server assigned you ID: " << server_client_id << std::endl;
        }
        else if (command == "WAIT") {
            std::cout << "Server: Waiting for " << parts[1] << " more player(s)..." <<
                std::endl;
        }
        else if (command == "START") {
            std::cout << "\nGAME STARTED \n";
```

```
80          }
81          else if (command == "TURN") {
82              int turn_id = std::stoi(parts[1]);
83
84              if (turn_id == server_client_id) {
85                  std::cout << "\n>>> YOUR TURN (" << server_client_id << ") <<<" << std
                        ::endl;
86                  std::string guess;
87
88                  while (true) {
89                      std::cout << "Enter your guess (3 unique digits) or QUIT: " << std::
                            flush;
90                      std::cin >> guess;
91
92                      if (guess == "QUIT") {
93                          send(sock, "QUIT\n", 5, 0);
94                          goto exit_cleanup;
95                      }
96
97                      std::string guess_with_newline = guess + "\n";
98                      send(sock, guess_with_newline.c_str(), guess_with_newline.size(), 0)
                            ;
99                      break;
100                 }
101             } else {
102                 std::cout << "Waiting for Player " << turn_id << "'s guess...\n";
103             }
104         }
105         else if (command == "RESULT") {
106             std::cout << "Result: Bulls: " << parts[1] << ", Cows: " << parts[2] << "\n
                    ";
107         }
108         else if (command == "WIN") {
109             std::cout << "\nGAME OVER Player " << parts[1] << " won the game!\n";
110             break;
111         }
112         else if (command == "SERVER_SHUTDOWN") {
113              std::cout << "\nServer has been shut down. Exiting.\n";
114              break;
115         }
116         else {
117             std::cout << "Server: " << msg << std::endl;
118         }
119     }
120
121 exit_cleanup:
122     std::cout << "Closing socket..." << std::endl;
123     close(sock);
124     return 0;
125 }
```

Листинг 5: Реализация работы клиента.

Вывод strace

Для server

```
execve("./server", ["./server"], 0x7ffd0cca2c30 /* 26 vars */) = 0
brk(NULL)                                       = 0x5bd75e7cd000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7d9db9d8
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=32015, ...}) = 0
mmap(NULL, 32015, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7d9db9d87000
close(3)                                        = 0
openat(AT_FDCWD, "/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\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 8
fstat(3, {st_mode=S_IFREG|0644, st_size=2592224, ...}) = 0
mmap(NULL, 2609472, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7d9db9a00000
mmap(0x7d9db9a9d000, 1343488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRIT
mmap(0x7d9db9be5000, 552960, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e5
mmap(0x7d9db9c6c000, 57344, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
mmap(0x7d9db9c7a000, 12608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS
close(3)                                        = 0
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\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 8
fstat(3, {st_mode=S_IFREG|0644, st_size=183024, ...}) = 0
mmap(NULL, 185256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7d9db9d59000
mmap(0x7d9db9d5d000, 147456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
mmap(0x7d9db9d81000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2800
mmap(0x7d9db9d85000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
close(3)                                        = 0
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\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832)
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 6
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 6
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7d9db9600000
mmap(0x7d9db9628000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRIT
mmap(0x7d9db97b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0
mmap(0x7d9db97ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
mmap(0x7d9db9805000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS
close(3)                                        = 0
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\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 8
fstat(3, {st_mode=S_IFREG|0644, st_size=952616, ...}) = 0
mmap(NULL, 950296, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7d9db9917000
mmap(0x7d9db9927000, 520192, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
mmap(0x7d9db99a6000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x8f0
mmap(0x7d9db99fe000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
close(3)                                        = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7d9db9d5
```

```
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7d9db9d
arch_prctl(ARCH_SET_FS, 0x7d9db9d54740) = 0
set_tid_address(0x7d9db9d54a10)         = 22169
set_robust_list(0x7d9db9d54a20, 24)     = 0
rseq(0x7d9db9d55060, 0x20, 0, 0x53053053) = 0
mprotect(0x7d9db97ff000, 16384, PROT_READ) = 0
mprotect(0x7d9db99fe000, 4096, PROT_READ) = 0
mprotect(0x7d9db9d85000, 4096, PROT_READ) = 0
mprotect(0x7d9db9c6c000, 45056, PROT_READ) = 0
mprotect(0x5bd74360f000, 4096, PROT_READ) = 0
mprotect(0x7d9db9dc7000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7d9db9d87000, 32015)           = 0
futex(0x7d9db9c7a7bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0
getrandom("\x84\x1f\x8e\xb9\xcb\x5b\xf0\xa9", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x5bd75e7cd000
brk(0x5bd75e7ee000)                     = 0x5bd75e7ee000
rt_sigaction(SIGINT, {sa_handler=0x5bd743604a0a, sa_mask=[INT], sa_flags=SA_RESTORER|
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 3
setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
bind(3, {sa_family=AF_INET, sin_port=htons(5555), sin_addr=inet_addr("0.0.0.0")}, 16)
listen(3, 5)                            = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
write(1, "Server started on port 5555. Req"..., 73Server started on port 5555. Requir
) = 73
```

Для client

```
execve("./client", ["./client"], 0x7ffe6e174840 /* 26 vars */) = 0
brk(NULL)                               = 0x572422526000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7627ec66
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=32015, ...}) = 0
mmap(NULL, 32015, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7627ec65a000
close(3)                                = 0
openat(AT_FDCWD, "/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\0\0\0\0\0\0\0\0"..., 832) = 8
fstat(3, {st_mode=S_IFREG|0644, st_size=2592224, ...}) = 0
mmap(NULL, 2609472, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7627ec200000
mmap(0x7627ec29d000, 1343488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRIT
mmap(0x7627ec3e5000, 552960, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e5
mmap(0x7627ec46c000, 57344, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
mmap(0x7627ec47a000, 12608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS
close(3)                                = 0
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\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 8
fstat(3, {st_mode=S_IFREG|0644, st_size=183024, ...}) = 0
```

```
mmap(NULL, 185256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7627ec62c000
mmap(0x7627ec630000, 147456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
mmap(0x7627ec654000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2800
mmap(0x7627ec658000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
close(3)                                = 0
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\220\243\2\0\0\0\0\0"..., 832)
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7627ebe00000
mmap(0x7627ebe28000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRIT
mmap(0x7627ebfb0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0
mmap(0x7627ebfff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
mmap(0x7627ec005000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS
close(3)                                = 0
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\0\0\0\0\0\0\0\0\0"..., 832) = 83
fstat(3, {st_mode=S_IFREG|0644, st_size=952616, ...}) = 0
mmap(NULL, 950296, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7627ec543000
mmap(0x7627ec553000, 520192, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
mmap(0x7627ec5d2000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x8f0
mmap(0x7627ec62a000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7627ec54
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7627ec53
arch_prctl(ARCH_SET_FS, 0x7627ec53e740) = 0
set_tid_address(0x7627ec53ea10)         = 22568
set_robust_list(0x7627ec53ea20, 24)     = 0
rseq(0x7627ec53f060, 0x20, 0, 0x53053053) = 0
mprotect(0x7627ebfff000, 16384, PROT_READ) = 0
mprotect(0x7627ec62a000, 4096, PROT_READ) = 0
mprotect(0x7627ec658000, 4096, PROT_READ) = 0
mprotect(0x7627ec46c000, 45056, PROT_READ) = 0
mprotect(0x5723fc0a3000, 4096, PROT_READ) = 0
mprotect(0x7627ec69a000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7627ec65a000, 32015)           = 0
futex(0x7627ec47a7bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0
getrandom("\x00\x6a\xcb\xe1\xdf\xdf\x81\xed", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x572422526000
brk(0x572422547000)                     = 0x572422547000
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 3
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
write(1, "Connecting to server...\n", 24Connecting to server...
) = 24
connect(3, {sa_family=AF_INET, sin_port=htons(5555), sin_addr=inet_addr("127.0.0.1")}
write(1, "Connected! Waiting for server ID"..., 36Connected! Waiting for server ID...
```

) = 36