

**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
Дата: _____

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 (высыпается при остановке сервера или таймауте).

Методы Обеспечения Динаминости 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
CMakeLists.txt
```

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 рассыпает 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, рассыпая 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

```

1 $ ./client
2 Connected! Waiting for server ID...
3 Server assigned you ID: 4
4 Server: Waiting for 1 more player(s)...
5
6 --- GAME STARTED ---
7 >>> YOUR TURN (4) <<<
8 Enter your guess (3 unique digits) or QUIT: 123
9 Result: Bulls: 0, Cows: 1 // 3 on 3rd pos
10 Waiting for Player 5's guess...
11 Result: Bulls: 0, Cows: 1
12
13 >>> YOUR TURN (4) <<<
14 Enter your guess (3 unique digits) or QUIT: 567
15 Result: Bulls: 0, Cows: 1 // 5 on 1st pos
16 Waiting for Player 5's guess...
17 Result: Bulls: 0, Cows: 1

```

Сессия Клиента 2 (ID 5):

Листинг 2: Сессия работы Клиента 2

```

1 $ ./client
2 Connected! Waiting for server ID...
3 Server assigned you ID: 5
4

```

```
5 | --- GAME STARTED ---
6 | Waiting for Player 4's guess...
7 | Result: Bulls: 0, Cows: 1
8 |
9 | >>> YOUR TURN (5) <<<
10 | Enter your guess (3 unique digits) or QUIT: 890
11 | Result: Bulls: 0, Cows: 1 // 9 on 9th pos
12 | Waiting for Player 4's guess...
13 | Result: Bulls: 0, Cows: 1
14 |
15 | Waiting for Player 4's guess...
16 | Result: Bulls: 0, Cows: 1
17 |
18 | >>> YOUR TURN (5) <<<
19 | Enter your guess (3 unique digits) or QUIT: 359
20 | Result: Bulls: 3, Cows: 0 // Win!
21 |
22 | !!! GAME OVER !!! Player 5 won the game!
```

Ключевые особенности разработанного решения

- Настраиваемая Сессия:** Возможность задать количество игроков (N) при запуске сервера, что обеспечивает гибкость и повторное использование кода для разного числа участников.
- Контроль Ввода/Вывода (select):** Использование системного вызова `select()` обеспечивает эффективное управление сокетными дескрипторами и позволяет серверу одновременно отслеживать ожидание подключения и таймаут хода.
- 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.
- 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

```
1 #include <iostream>
2 #include <algorithm>
3 #include <random>
4 #include <chrono>
5 #include <sstream>
6 #include <set>
7
8 #include "game.h"
9
10 Game::Game(int players) : num_players(players) {
11     generate_code();
12     std::cout << "DEBUG: Secret code set to: " << secret_code << std::endl;
13 }
14
15 bool Game::is_valid(const std::string& guess) {
16     if (guess.length() != CODE_LENGTH) {
17         return false;
18     }
19
20     for (char c : guess) {
21         if (!isdigit(c)) {
22             return false;
23         }
24     }
25
26     // Characters must be unique
27     std::set<char> unique_digits(guess.begin(), guess.end());
28     if (unique_digits.size() != CODE_LENGTH) {
29         return false;
30     }
31
32     return true;
33 }
34
35 void Game::generate_code() {
36     std::vector<int> digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
37
38     unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
39     std::shuffle(digits.begin(), digits.end(), std::default_random_engine(seed));
40
41     secret_code = "";
42
43
44     int start_index = 0;
45     if (CODE_LENGTH > 1 && digits[0] == 0) {
46         auto it = std::find_if(digits.begin() + 1, digits.end(), [] (int d){ return d != 0; });
47         if (it != digits.end()) {
48             std::swap(digits[0], *it);
49         }
50     }
51
52     for (int i = 0; i < CODE_LENGTH; ++i) {
53         secret_code += std::to_string(digits[i]);
```

```

54     }
55 }
56
57 std::pair<int, int> Game::check_guess(const std::string& guess) {
58     int bulls = 0;
59     int cows = 0;
60
61     for (int i = 0; i < CODE_LENGTH; ++i) {
62         for (int j = 0; j < CODE_LENGTH; ++j) {
63             if (guess[i] == secret_code[j]) {
64                 if (i == j) {
65                     bulls++;
66                 } else {
67                     cows++;
68                 }
69             }
70         }
71     }
72     return {bulls, cows};
73 }
74
75 std::string Game::process_guess(const std::string& guess, bool& win) {
76     win = false;
77
78     if (!is_valid(guess)) {
79         return "RESULT 0 0 (Invalid)";
80     }
81
82     std::pair<int, int> result = check_guess(guess);
83     int bulls = result.first;
84     int cows = result.second;
85
86     if (bulls == CODE_LENGTH) {
87         win = true;
88     }
89
90     std::stringstream ss;
91     ss << "RESULT " << bulls << " " << cows;
92     return ss.str();
93 }

```

Листинг 3: Реализация логики игры

Файл server.cpp

```

1 #include <arpa/inet.h>
2 #include <unistd.h>
3 #include <iostream>
4 #include <vector>
5 #include <cstring>
6 #include <signal.h>
7 #include <sys/select.h>
8 #include <sys/time.h>
9 #include <algorithm>
10 #include <stdexcept>
11

```

```

12 #include "game.h"
13
14
15 volatile sig_atomic_t keep_running = 1;
16
17 void signal_handler(int sig) {
18     if (sig == SIGINT) {
19         keep_running = 0;
20         std::cout << "\nSIGINT received. Shutting down gracefully..." << std::endl;
21     }
22 }
23
24 void broadcast(const std::vector<int>& clients, const std::string& message) {
25     std::string msg_with_newline = message + "\n";
26     for (int c : clients) {
27         send(c, msg_with_newline.c_str(), msg_with_newline.size(), 0);
28     }
29 }
30
31 int main(int argc, char* argv[]) {
32     signal(SIGINT, signal_handler);
33
34     const int DEFAULT_PLAYERS = 2;
35     int players_required = DEFAULT_PLAYERS;
36
37     // Read cmd arguments
38     if (argc == 2) {
39         try {
40             int requested_players = std::stoi(argv[1]);
41
42             if (requested_players > 1) {
43                 players_required = requested_players;
44             } else {
45                 std::cerr << "Error: Number of players must be greater than 1. Using
46                         default (" << DEFAULT_PLAYERS << ")." << std::endl;
47             }
48         } catch (const std::exception& e) {
49             std::cerr << "Error parsing player count argument. Using default (" <<
50                         DEFAULT_PLAYERS << ")." << std::endl;
51         }
52     }
53
54     int server_fd = socket(AF_INET, SOCK_STREAM, 0);
55     if (server_fd < 0) {
56         perror("Socket creation failed");
57         return 1;
58     }
59
60     // ports/adresses can be used again
61     int opt = 1;
62     if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt))) {
63         perror("setsockopt failed");
64         close(server_fd);
65         return 1;
66     }
67     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
85             << ". Waiting for clients..." << std::endl;
86
87     std::vector<int> clients;
88
89     // Find players
90     while ((int)clients.size() < players_required && keep_running) {
91         fd_set master_set;
92         FD_ZERO(&master_set);
93         FD_SET(server_fd, &master_set);
94
95         struct timeval timeout;
96         timeout.tv_sec = 1;
97         timeout.tv_usec = 0;
98
99         if (select(server_fd + 1, &master_set, nullptr, nullptr, &timeout) < 0) {
100             if (keep_running) perror("Select error during connect");
101             break;
102         }
103
104         if (FD_ISSET(server_fd, &master_set)) {
105             int c = accept(server_fd, nullptr, nullptr);
106             if (c < 0) continue;
107
108             clients.push_back(c);
109             std::cout << "Client connected: socket = " << c << std::endl;
110
111             std::string id_msg = "YOUR_ID " + std::to_string(c);
112             send(c, id_msg.c_str(), id_msg.size(), 0);
113
114             std::string wait_msg = "WAIT " + std::to_string(players_required - clients.
115                     size());
116             send(c, wait_msg.c_str(), wait_msg.size(), 0);
117         }
118
119     // Game process
120     if ((int)clients.size() == players_required && keep_running) {
121
122         std::cout << "All players connected. Starting game" << std::endl;
123         Game game(players_required);
124         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, &
146             timeout_turn);
147
148             if (!keep_running) break;
149
150             if (ready_fds <= 0) {
151                 std::cout << "Client " << sock << " timed out or select error.
152                 Aborting." << std::endl;
153                 broadcast(clients, "SERVER_SHUTDOWN");
154                 keep_running = 0;
155                 break;
156             }
157
158             memset(buf, 0, sizeof(buf));
159             int r = recv(sock, buf, sizeof(buf)-1, 0);
160
161             if (r <= 0) {
162                 std::cout << "Client " << sock << " disconnected unexpectedly (r="
163                 << r << ")." << std::endl;
164                 broadcast(clients, "SERVER_SHUTDOWN");
165                 keep_running = 0;
166                 break;
167             }
168
169             std::string guess(buf);
170             guess.erase(std::remove_if(guess.begin(), guess.end(), [] (char c){
171                 return c == '\n' || c == '\r'; }), guess.end());
172
173             if (guess == "QUIT") {
174                 std::cout << "Client " << sock << " requested QUIT. Aborting game."
175                 << std::endl;
176                 broadcast(clients, "SERVER_SHUTDOWN");
177                 keep_running = 0;
178                 break;
179             }
180
181             std::cout << "Received guess from client " << sock << ":" << guess <<

```

```

    std::endl;
177
178     std::string resp = game.process_guess(guess, win);
179
180     broadcast(clients, resp);
181
182     if (win) {
183         std::string win_msg = "WIN " + std::to_string(sock);
184         broadcast(clients, win_msg);
185         break;
186     }
187 }
188 }
189 } else {
190     broadcast(clients, "SERVER_SHUTDOWN");
191 }
192
193 std::cout << "Game over." << std::endl;
194
195 for(int c : clients) {
196     close(c);
197 }
198
199 close(server_fd);
200 std::cout << "Server successfully shut down." << std::endl;
201
202 return 0;
203 }
```

Листинг 4: Реализация работы сервера.

Файл client.cpp

```

1 #include <arpa/inet.h>
2 #include <unistd.h>
3 #include <iostream>
4 #include <string>
5 #include <cstring>
6 #include <algorithm>
7 #include <vector>
8 #include <sstream>
9
10 // str -> characters
11 std::vector<std::string> split(const std::string &s, char delimiter) {
12     std::vector<std::string> tokens;
13     std::string token;
14     std::istringstream tokenStream(s);
15     while (std::getline(tokenStream, token, delimiter)) {
16         tokens.push_back(token);
17     }
18     return tokens;
19 }
20
21 void trim_message(std::string& str) {
22     str.erase(std::remove_if(str.begin(), str.end(), [] (char c){ return c == '\n' || c
23             == '\r'; }), str.end());
```

```

23 || }
24
25 int main() {
26     int sock = socket(AF_INET, SOCK_STREAM, 0);
27     if (sock < 0) {
28         perror("Socket creation failed");
29         return 1;
30     }
31
32     sockaddr_in addr{};
33     addr.sin_family = AF_INET;
34     addr.sin_port = htons(5555);
35
36     if (inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr) <= 0) {
37         perror("Invalid address / Address not supported");
38         close(sock);
39         return 1;
40     }
41
42     std::cout << "Connecting to server..." << std::endl;
43
44     if (connect(sock, (sockaddr*)&addr, sizeof(addr)) < 0) {
45         perror("Connection Failed.");
46         close(sock);
47         return 1;
48     }
49
50     int server_client_id = -1;
51
52     std::cout << "Connected! Waiting for server ID..." << std::endl;
53
54     char buf[128];
55     while (true) {
56         memset(buf, 0, sizeof(buf));
57
58         int r = recv(sock, buf, sizeof(buf)-1, 0);
59         if (r <= 0) {
60             std::cout << "\nServer disconnected or error occurred." << std::endl;
61             break;
62         }
63
64         std::string msg(buf);
65         trim_message(msg);
66
67         std::vector<std::string> parts = split(msg, ' ');
68         std::string command = parts[0];
69
70
71         if (command == "YOUR_ID") {
72             server_client_id = std::stoi(parts[1]);
73             std::cout << "Server assigned you ID: " << server_client_id << std::endl;
74         }
75         else if (command == "WAIT") {
76             std::cout << "Server: Waiting for " << parts[1] << " more player(s)..." <<
77                         std::endl;
78         }
79         else if (command == "START") {
79             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
86            ::endl;
87            std::string guess;
88
89            while (true) {
90                std::cout << "Enter your guess (3 unique digits) or QUIT: " << std::
91                flush;
92                std::cin >> guess;
93
94                if (guess == "QUIT") {
95                    send(sock, "QUIT\n", 5, 0);
96                    goto exit_cleanup;
97                }
98
99                std::string guess_with_newline = guess + "\n";
100               send(sock, guess_with_newline.c_str(), guess_with_newline.size(), 0
101                   );
102               break;
103           }
104       } else {
105           std::cout << "Waiting for Player " << turn_id << "'s guess...\n";
106       }
107   }
108   else if (command == "RESULT") {
109       std::cout << "Result: Bulls: " << parts[1] << ", Cows: " << parts[2] << "\n
110       ";
111   }
112   else if (command == "WIN") {
113       std::cout << "\nGAME OVER Player " << parts[1] << " won the game!\n";
114       break;
115   }
116   else if (command == "SERVER_SHUTDOWN") {
117       std::cout << "\nServer has been shut down. Exiting.\n";
118       break;
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

```
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\x a9", 8, GRND_NONBLOCK) = 8
brk(NULL) = 0x5bd75e7cd000
brk(0x5bd75e7ee000) = 0x5bd75e7ee000
rt_sigaction(SIGINT, {sa_handler=0x5bd743604a0a, sa_mask=[INT], sa_flags=SA_RESTORER|SA_RESTART}, {sa_handler=0x5bd743604a0a, sa_mask=[INT], sa_flags=SA_RESTORER|SA_RESTART}) = 0
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) = 0
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

) = 36