Министерство цифрового развития, связи и массовых коммуникаций РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)
Кафедра вычислительных систем

Курсовая работа по дисциплине
Сетевое программирование
«Разработка сетевого приложения»

Выполнил: студент гр. ИП-115
Самарин Даниил Андреевич
Проверил: профессор кафедры ВС
Павский Кирилл Валерьевич

Новосибирск 2024г.

Оглавление

Постановка задачи

Для реализации курсового проекта необходимо разработать сетевое приложение «Сетевая игра». Мультипоточная реализация сервера, на базе протокола TCP; PTHREAD на языке программирования C/C+ в OS Linux.

Описание протокола

Протокол TCP (Transmission Control Protocol) — это один из основных протоколов транспортного уровня в модели OSI, используемый для надежной передачи данных между компьютерами в сети Интернет. TCP обеспечивает надежную, упорядоченную и безошибочную доставку данных между отправителем и получателем.

Основные компоненты и особенности протокола TCP:

TCP ориентирован на соединение, что означает необходимость установления соединения между двумя узлами перед началом передачи данных. Процесс установления соединения известен как трёхэтапное рукопожатие (three-way handshake):

1. SYN: Клиент отправляет серверу сегмент SYN для инициации соединения.
2. SYN-ACK: Сервер отвечает сегментом SYN-ACK, подтверждая получение SYN и предлагая синхронизацию.
3. ACK: Клиент отправляет серверу сегмент ACK, подтверждая получение SYN-ACK. На этом этапе соединение установлено.

Для идентификации отправляющих и принимающих приложений ***TCP использует номера портов***. Каждый порт представляет собой 16-битное число, что позволяет использовать до 65 536 портов на каждом узле. Порты позволяют различать различные приложения и процессы, работающие на одном и том же компьютере.

TCP реализует управление потоком, чтобы избежать перегрузки сети и обеспечить эффективное использование сетевых ресурсов. Это достигается с помощью механизма скользящего окна, который позволяет получателю контролировать количество данных, которые отправитель может передать до получения подтверждения.

TCP использует различные команды (флаги), чтобы управлять состоянием соединения и передачей данных:

SYN - Инициация соединения. ACK - Подтверждение получения данных. FIN - Завершение соединения. RST - Сброс соединения. PSH - Немедленная передача данных без буферизации. URG - Указание на наличие данных с высоким приоритетом.

После установления соединения ***данные передаются в виде последовательных сегментов***. Каждый сегмент содержит заголовок TCP, который включает порядковые номера, позволяющие получателю переупорядочивать сегменты, если они были доставлены не в том порядке.

TCP обеспечивает контроль ошибок с помощью контрольных сумм. Каждое сообщение TCP содержит контрольную сумму, которая проверяется получателем для обнаружения ошибок в переданных данных. В случае обнаружения ошибки сегмент данных будет повторно отправлен.

TCP сам по себе не предоставляет встроенные методы аутентификации. Однако он часто используется в сочетании с другими протоколами, такими как SSL/TLS, ***для обеспечения шифрования и аутентификации***, что позволяет защитить данные от несанкционированного доступа и подделки.

TCP постоянно развивается, и ***к нему добавляются новые функции и расширения*** для улучшения производительности и безопасности:
TCP Fast Open: ускоряет процесс установления соединения.
TCP Congestion Control: механизмы управления перегрузкой сети, такие как алгоритмы AIMD, CUBIC и другие, которые помогают оптимизировать передачу данных и предотвратить перегрузку сети.

Протокол TCP является фундаментальным элементом сетевых коммуникаций и обеспечивает надежную и эффективную передачу данных в сети Интернет. Благодаря своей структуре и расширяемости, TCP продолжает оставаться основным протоколом для многих приложений и сервисов, требующих гарантированной доставки данных.

Потоки Pthread в POSIX: Основные Принципы

Pthread (POSIX threads) — это набор интерфейсов программирования (API) для создания и управления потоками, предоставляемый стандартом POSIX (Portable Operating System Interface). Потоки позволяют выполнять несколько потоков (нитей) исполнения в пределах одного процесса, что значительно улучшает производительность и эффективность программ.

Основные компоненты и особенности потоков Pthread:

1. **Создание потоков:**

   Для создания нового потока используется функция pthread_create. Она принимает аргументы, указывающие на идентификатор потока, атрибуты потока (если есть), указатель на функцию потока и аргумент для этой функции.

   Пример:

   pthread_t thread;

   pthread_create(&thread, NULL, thread_function, (void *) arg);

2. **Атрибуты потоков:**

   Атрибуты потоков могут быть использованы для управления различными аспектами создания и исполнения потоков, такими как размер стековой области и политики планирования. Для инициализации и установки атрибутов используются функции *pthread_attr_init* и *pthread_attr_set*....

3. **Управление потоками:**

   У Pthread имеется обширный набор API для управления жизненным циклом потоков. Это включает функции указанного создания потоков (*pthread_create*), прекращения работы потока (*pthread_exit*), ожидания завершения потока (*pthread_join*) и отмены потока (*pthread_cancel*).

   Пример использования pthread_join:

   pthread_join(thread, NULL);

4. **Синхронизация потоков**:

Неотъемлемой частью многопоточного программирования является синхронизация. POSIX предоставляет механизмы для предотвращения состояния гонок и других проблем синхронизации:

Мьютексы (pthread_mutex_t):

Применяются для взаимного исключения (защиты критических секций кода). Операции типа блокирования и разблокирования мьютексов реализуются функциями *pthread_mutex_lock* и *pthread_mutex_unlock.*

Условные переменные (pthread_cond_t):

Используются для блокировки потоков до тех пор, пока не произойдет определенное событие. Для ожидания на условной переменной используется функция *pthread_cond_wait*, а для оповещения — *pthread_cond_signal* или *pthread_cond_broadcast.*

Семафоры:

POSIX также предоставляет семафоры для контроля доступа к ресурсам в многопоточной среде.

5. **Планирование и политики:**

   POSIX потоки поддерживают различные политики планирования, которые определяют порядок исполнения потоков. Существует несколько уровней приоритетов, и потоки могут быть настроены на использование реального времени или политики свопинга времени. Эти параметры могут быть установлены и изменены через функции *pthread_setschedparam* и *pthread_getschedparam*.

6. **Сигналы и активация:**

   Потоки могут получать и обрабатывать сигналы (события), что используется для управления потоками, особенно в асинхронных системах. POSIX предоставляет механизмы для настройки и обработки сигналов потоками, позволяя гибко управлять их поведением.

Описание реализации

server.c

Программа представляет собой многопоточный сервер на основе TCP, который обрабатывает игровой процесс по поиску сокровищ несколькими игроками. Взаимодействие игроков с сервером осуществляется через сокеты. Каждый игрок подключается к серверу, который распределяет их по играм. При достижении максимального количества игроков в игре игра начинается, и сервер управляет игровым процессом.

Краткое описание программы:

Главная функция (main):

Создает серверный TCP-сокет.

Привязывает сокет к адресу и порту сервера.

Переходит в режим прослушивания входящих соединений.

Принимает входящие соединения и создает отдельный поток для каждого клиента, вызывая функцию ***handle_client***.

Функция start_game:

Создает новый поток для игры, вызывающий функцию ***game_thread***.

Функция game_thread:

Основной игровой цикл, в котором сервер взаимодействует с игроками.

Отправляет уведомление всем игрокам о начале игры.

Цикл, который обрабатывает ходы игроков:

Получает от игрока координаты попытки найти сокровище.

Проверяет, угаданы ли координаты.

Если сокровище найдено, уведомляет всех игроков и завершает игру.

Если нет, отправляет игрокам информацию о неудачной попытке.

Завершает игру и закрывает соединения с игроками.

client.c

Эта программа представляет собой клиентское приложение для игры "Искатели сокровищ". Она использует TCP-сокет для соединения с сервером и взаимодействует с ним для участия в многопользовательской

игре. Ниже приведено краткое описание функциональности программы.

Краткое описание программы:

Главная функция (main):

Создает сокет и устанавливает его параметры.

Инициирует соединение с сервером, используя IP-адрес и порт из config.h.

Получает и обрабатывает приветственное сообщение от сервера.

Определяет индекс игрока, принимая его от сервера.

Основной игровой цикл:

- Выполняется до тех пор, пока не будет найдено сокровище.
- Поочередно запрашивает у текущего игрока координаты для раскопок и отправляет их на сервер.
- Получает и обрабатывает сообщения от сервера.
- Обновляет и отображает текущую игровую доску после каждого хода.
- Завершает игру, когда один из игроков находит сокровище.

Функция initializeBoard:

Заполняет игровое поле размером BOARD_SIZE x BOARD_SIZE символами '.', обозначающими пустые клетки.

Функция changeSymbol:

Модифицирует символ на заданной координате доски. Символ заменяется на '*', если координаты корректны.

Функция printBoard:

Отображает текущую доску с координатной сеткой.

Результаты работы программы

```
master@Ubuntu:~/Рабочий стол/SibSUTIS/6/NetworkProgramming/cursework$ ./build.sh
Компиляция успешно завершена. Исполняемый файл 'server' создан.
Компиляция успешно завершена. Исполняемый файл 'client' создан.
master@Ubuntu:~/Рабочий стол/SibSUTIS/6/NetworkProgramming/cursework$ ./server
СЕРВЕР: Включен!
```

Рисунок 1 - Запуск сервера.

```
master@Ubuntu:~/Рабочий стол/SibSUTIS/6/NetworkProgramming/cursework$ ./client
Игры: Искатели сокровищ(на трех игроков)
 У каждого игрорка своя карта 8x8, где он ищет свое сокровище, кто первый нашел тот и победил
 Карта начинается с позиции (0,0)
Подождите больше игроков для начала игры 1. Вы игрок № 1
Индекс игрока: 1
```

Рисунок 2 - Запуск сервера.

```
Ход №1.
Индекс игрока: 1
_|0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . . . . . . .
3 . . . . . . . .
4 . . . . . . . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . . . .
Введите координаты раскопки клада(x y):
```

Рисунок 3 - Результат работы клиента

```
Ход №2.
Игрок 1 копает (1, 5) - сокровищ нет
Игрок 2 копает (2, 3) - сокровищ нет
Индекс игрока: 3
_|0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . * . . .
2 . . . . . . . .
3 . . . . . . . .
4 . . . . . . . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . . . .
Введите координаты раскопки клада(x y):
```

Рисунок 4 - Результат работы клиента

Текст программы

Ссылка на git: github.com/SamarinDaniil/SibSUTIS/6/NetworkProgramming/cursework

Server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <arpa/inet.h>
#include "config.h"

typedef struct {
    int x;
    int y;
} Position;

typedef struct {
    int sockfd;
    Position hidden_treasure;
    int game_id;
    int player_num;
} Player;

typedef struct {
    Player players[MAX_PLAYERS];
    int current_player_count;
    int game_id;
    int is_active;
} Game;

Game games[MAX_GAMES];
int current_game_count = 0;
pthread_mutex_t mutex =
        PTHREAD_MUTEX_INITIALIZER;
```

```
void *game_thread(void *arg) {
    sleep(1);
    Game *game = (Game *)arg;
    char buffer[1024];
    pthread_mutex_lock(&mutex);
    if (!game->is_active) {
        for (int i = 0; i < MAX_PLAYERS; i++) {
            close(game->players[i].sockfd);
        }
        pthread_mutex_unlock(&mutex);
        pthread_exit(NULL);
        //return;
    }
    pthread_mutex_unlock(&mutex);
    sprintf(buffer, "Игра началась!\n");
    for (int j = 0; j < MAX_PLAYERS; j++) {
        send(game->players[j].sockfd, buffer,
         strlen(buffer), 0);
    }

    while (1) {

        for (int i = 0; i < MAX_PLAYERS; i++) {

            int sockfd = game->players[i].sockfd;
            memset(buffer, 0, sizeof(buffer));

            // Получить ход игрока
            ssize_t bytesRead = recv(sockfd, buffer,
          sizeof(buffer) - 1, 0);
            buffer[bytesRead] = '\0';
            printf("%s", buffer);
            if (bytesRead == 0)
```

```
  {
      printf("СЕРВЕР: Клиент отключился.\n");
      game->is_active = 0;
      for (int i = 0; i < MAX_PLAYERS; i++) {
          close(game->players[i].sockfd);
      }
      pthread_exit(NULL);
  }
  else if (bytesRead < 0)
  {
      perror("СЕРВЕР: Плохой socket клиента.");
      game->is_active = 0;
      for (int i = 0; i < MAX_PLAYERS; i++) {
          close(game->players[i].sockfd);
      }
      pthread_exit(NULL);
  }


  // Обработка хода (поиск сокровища)
  int x, y;
  sscanf(buffer, "%d %d", &x, &y);


  if (x == game->players[i].hidden_treasure.x && y == game->players[i].hidden_treasure.y) {
      sprintf(buffer, "Игрок %d нашел сокровище!\n", i + 1);
      for (int j = 0; j < MAX_PLAYERS; j++) {
          if (send(game->players[j].sockfd, buffer, strlen(buffer), 0) < 0) {
              perror("СЕРВЕР: Ошибка при отправке сообщения");
              exit(1);
          }
      }
      game->is_active = 0;
      break;
    } else {
      sprintf(buffer, "Игрок %d копает (%d, %d) - сокровищ нет\n", i + 1, x, y);
      for (int j = 0; j < MAX_PLAYERS; j++) {
          if (send(game->players[j].sockfd, buffer, strlen(buffer), 0) < 0) {
              perror("СЕРВЕР: Ошибка при отправке сообщения");
              exit(1);
          }
      }
    }
  }


  // Закрытие соединений с игроками
  for (int i = 0; i < MAX_PLAYERS; i++) {
      close(game->players[i].sockfd);
  }
  game->is_active = 0;


  pthread_exit(NULL);
}


void start_game(Game *game) {
  pthread_t tid;
  pthread_create(&tid, NULL, game_thread, (void *)game);
}


void *handle_client(void *arg) {
  int sockfd = *(int *)arg;
  free(arg);
```

```
pthread_mutex_lock(&mutex);


// Найти или создать игру для игрока
Game *game = NULL;
for (int i = 0; i < current_game_count; i++) {
   if (games[i].current_player_count <
    MAX_PLAYERS) {
      game = &games[i];
      break;
   } else {
      if (i == current_game_count-1)
      {
         i = 0;
      }
   }
}


if (game == NULL && current_game_count <
    MAX_GAMES) {
   game = &games[current_game_count++];
   game->game_id = current_game_count;
   game->current_player_count = 0;
   game->is_active = 0;
}


int player_num = game-
    >current_player_count++;
game->players[player_num].sockfd = sockfd;
game-
    >players[player_num].hidden_treasure.x =
    1; //rand() % BOARD_SIZE;
game-
    >players[player_num].hidden_treasure.y =
    1; //rand() % BOARD_SIZE;
game->players[player_num].game_id = game-
    >game_id;
game->players[player_num].player_num =
    player_num;

// Проверка на заполнение игры
if (game->current_player_count ==
    MAX_PLAYERS) {
   game->is_active = 1;
   start_game(game);
}
if (current_game_count == MAX_GAMES-1)
{
   current_game_count = 0;
}


pthread_mutex_unlock(&mutex);


// Сообщение игроку о его статусе
    ожидания/начала игры
char message[256];
if (game->is_active) {
   printf("СЕРВЕР: Игра %d началась. Вы
    игрок № %d\n", game->game_id,
    player_num + 1);
   sprintf(message, "Игра %d началась. Вы
    игрок № %d\n", game->game_id,
    player_num + 1);
} else {
   printf("СЕРВЕР: Подождите больше
    игроков для начала игры %d. Вы игрок
    № %d\n", game->game_id, player_num +
    1);
   sprintf(message, "Подождите больше
    игроков для начала игры %d. Вы игрок
    № %d\n", game->game_id, player_num +
    1);
}
if (send(sockfd, message, strlen(message),0) <
    0) {
   perror("СЕРВЕР: Ошибка отправки
    сообщения обратно клиенту.");
   //exit(1);
}
```

```
    pthread_exit(NULL);
}

int main() {
    int sockfd, newsockfd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t cli_len;
    int *new_sockfd;

    sockfd = socket(AF_INET, SOCK_STREAM,
        0);
    if (sockfd < 0) {
        perror("СЕРВЕР: Сервер не может
         открыть TCP-сокет.");
        exit(1);
    }

    memset(&server_addr, 0,
        sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr =
        INADDR_ANY;
    server_addr.sin_port =
        htons(SERVER_PORT);

    if (bind(sockfd, (struct sockaddr
        *)&server_addr, sizeof(server_addr)) < 0)
        {
        perror("СЕРВЕР: Связывание сервера
         неудачно.");
        close(sockfd);
        exit(1);
    }

    printf("СЕРВЕР: Включен!\n");
    listen(sockfd, 5);
    cli_len = sizeof(client_addr);

    while (1) {
        newsockfd = accept(sockfd, (struct
         sockaddr *)&client_addr, &cli_len);
        if (newsockfd < 0) {
            perror("Ошибка при принятии
         входящего соединения.");
            continue;
        }
        printf("СЕРВЕР: Принято входящее
         соединение.\n");
        char clientIp[INET_ADDRSTRLEN];
        inet_ntop(AF_INET,
         &(client_addr.sin_addr), clientIp,
         INET_ADDRSTRLEN);
        printf("СЕРВЕР: IP адрес клиента: %s\n",
         clientIp);
        printf("СЕРВЕР: PORT клиента: %d\n",
         ntohs(client_addr.sin_port));

        pthread_t tid;
        new_sockfd = malloc(sizeof(int));
        *new_sockfd = newsockfd;
        pthread_create(&tid, NULL, handle_client,
         (void *)new_sockfd);
    }

    close(sockfd);
    return 0;
}
```

Client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
```

```
#include <netdb.h>
#include <errno.h>
#include "config.h"



char board[BOARD_SIZE][BOARD_SIZE];


// Инициализация доски
void initializeBoard(char board[BOARD_SIZE][BOARD_SIZE]) {
    for (int i = 0; i < BOARD_SIZE; i++) {
        for (int j = 0; j < BOARD_SIZE; j++) {
            board[i][j] = '.';
        }
    }
}


// Изменение символа на доске
void changeSymbol(char board[BOARD_SIZE][BOARD_SIZE], int x, int y) {
    if (x >= 0 && x < BOARD_SIZE && y >= 0 && y < BOARD_SIZE) {
        board[x][y] = '*';
    } else {
        printf("Некорректные координаты\n");
    }
}


// Вывод доски
void printBoard(char board[BOARD_SIZE][BOARD_SIZE]) {
    printf("_|");
    for (int i = 0; i < BOARD_SIZE; i++)
        printf("%d ", i);
    printf("\n");
    for (int i = 0; i < BOARD_SIZE; i++) {
        printf("%d ", i);
        for (int j = 0; j < BOARD_SIZE; j++) {
            printf("%c ", board[i][j]);
        }
        printf("\n");
    }
}


int main() {
    int sockfd;
    struct sockaddr_in server_addr;
    char buffer[1024];
    struct hostent* hp;



    // Создание сокета
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("Клиент не может открыть TCP-сокет.");
        exit(1);
    }


    // Задание параметров сервера
```

```
    memset(&server_addr, 0,
sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr =
inet_addr(SERVER_IP);
    server_addr.sin_port =
htons(SERVER_PORT);


    hp = gethostbyname(SERVER_IP);
    if (hp == NULL)
    {
        printf("Не удалось получить IP-
адрес для указанного хоста\n");
        exit(1);
    }


    memcpy((char*)&server_addr.sin_addr,
hp->h_addr_list[0], hp->h_length);


    // Подключение к серверу
    if (connect(sockfd, (struct sockaddr
*)&server_addr, sizeof(server_addr)) < 0)
{
        perror("Ошибка при подключении к
серверу");
        exit(1);
    }
    // Чтение приветственного сообщения
от сервера
    initializeBoard(board);
    printf("Игры: Искатели сокровищ(на
трех игроков) \n У каждого игрорка своя
карта 8x8, где он ищет свое сокровище,
кто первый нашел тот и победил \n
Карта начинается с позиции (0,0)\n");
    memset(buffer, 0, sizeof(buffer));
    recv(sockfd, buffer, sizeof(buffer) - 1,
0);
    printf("%s", buffer);
    int length = strlen(buffer);
    char lastChar = buffer[length - 2];
    int numPlayer = lastChar - '0';
    printf("Индекс игрока: %c\n",
lastChar);


    //Game start
    memset(buffer, 0, sizeof(buffer));
    recv(sockfd, buffer, sizeof(buffer) - 1,
0);
    printf("%s", buffer);
    //
    sleep(2);
    system("clear");
    // Основной игровой цикл
    int con = 0;
    while (1) {
        // Чтение хода от игрока
        con++;
        printf("Ход №%d.\n", con);
        int endgame = 0;
        for (int count = 0; count <
MAX_PLAYERS; count++)
        {
            if (count == numPlayer-1)
            {
                int x = -1, y = -1;
                printf("Индекс игрока: %c\n",
lastChar);
                printBoard(board);
```

```
            while (1 == 1) {
                printf("Введите координаты раскопки клада(x y): ");
                fgets(buffer, sizeof(buffer), stdin);
                sscanf(buffer, "%d %d", &x, &y);
                if (x >= 0 && x < BOARD_SIZE && y >= 0 && y < BOARD_SIZE) {
                    break;
                } else {
                    printf("Некорректные координаты\n");
                }
            }

            changeSymbol(board, x, y);
            buffer[strcspn(buffer, "\n")] = 0;

            // Отправка хода серверу
            send(sockfd, buffer, strlen(buffer), 0);

        }

        // Получение ответа от сервера
        memset(buffer, 0, sizeof(buffer));

        ssize_t bytesRead = recv(sockfd, buffer, sizeof(buffer) - 1, 0);
        if (bytesRead < 0) {
            perror("Ошибка при приеме сообщения");
            exit(1);
        }
        buffer[bytesRead] = '\0';
        printf("%s", buffer);
        // Проверка на конец игры
        if (strstr(buffer, "нашел сокровище!") != NULL) {
            endgame = 1;
            break;
        }
    }
    sleep(2);
    system("clear");
    if (endgame == 1)
    {

        break;
    }
  }

  // Закрытие соединения
  close(sockfd);

  return 0;
}
```

Список источников

1. Beej's Guide to Network Programming /// URL - https://www.beej.us/guide/bgnet/
2. https://pro-prof.com/forums/topic/libpcap
3. C++ Concurrency in Action /// URL - https://www.bogotobogo.com/cplusplus/files/CplusplusConcurrencyInAction_PracticalMultithreading.pdf