Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Кафедра информационных технологий и систем

**КУРСОВАЯ РАБОТА**

по дисциплине: «Операционные системы»

на тему: «Гонки»

Разработал:

Студент группы 8091

Шляханов Д. А. \_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2020г.

Проверил:

Ананьев В. В. \_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2020г.

**Великий Новгород**

**2020**

1. ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc42246550)

[ВВЕДЕНИЕ 3](#_Toc42246551)

[ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА. 4](#_Toc42246552)

[1.1 Постановка задачи. 4](#_Toc42246553)

[1.2 Выбор инструментальных средств. 4](#_Toc42246554)

[1.3 Выбор модели. 4](#_Toc42246555)

[1.4 Выбор способа решения проблемы блокирования ввода/вывода. 4](#_Toc42246556)

[1.5 Протокол взаимодействия клиента и сервера. 6](#_Toc42246557)

[1.6 Выбор протокола транспортного уровня OSI-модели. 6](#_Toc42246558)

[1.7 Алгоритм решения задачи. 7](#_Toc42246559)

[Со стороны клиента: 8](#_Toc42246560)

[Со стороны сервера: 9](#_Toc42246561)

[1.8 Вывод по первой главе. 10](#_Toc42246562)

[ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА. 11](#_Toc42246563)

[2.1 Структура клиента и сервера. 11](#_Toc42246564)

[2.2 Программный интерфейс сервера. 11](#_Toc42246565)

[Описание работы главного потока: 11](#_Toc42246566)

[Описание работы потока игрока: 12](#_Toc42246567)

[Описание работы потока бота: 12](#_Toc42246568)

[2.3 Описание программной реализации клиента 12](#_Toc42246569)

[2.4 Системные вызовы. 13](#_Toc42246570)

[2.5 Пример работы приложения. 14](#_Toc42246571)

[2.6 Результат работы. 14](#_Toc42246572)

[ЗАКЛЮЧЕНИЕ 16](#_Toc42246573)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ. 16](#_Toc42246574)

1. ВВЕДЕНИЕ

Целью данной работы является создание игры многопользовательской игры “Гонки”.

Особенность проектируемой игры заключается в том, что она должна работать под операционной системой Linux.

Основной функционал игры и правила игры следующие:

В одной игре может принимать участие от 2 до 10 человек. Игроки только компьютерные. Каждый ход каждая полоска получает случайное приращение. Вывод результатов гонки на экран производятся в основном потоке.

Перед началом игры игроки подключаются к серверу. После того как подключившиеся игроки будут соответствовать количеству указанном в аргументе сервера начинается Гонка. Каждый ход гонщики получают своё приращение и в консоль выводится его позиция. Об окончании Гонки для гонщика сообщат сервер. Также он передает таблицу результатов, а выводит ее клиент.

1. ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА.

1.1 Постановка задачи.

Задачей данной курсовой работы является разработка сетевой игры «Гонки», которая рассчитана от 2 до 10 игроков. Игра должна работать под управлением операционной системы семейства Unix/Linux. Сетевая часть иры должна быть выполнена на языке С/C++ с помощью сокетов, предоставляемых системной библиотекой <sys/socket.h>.

1.2 Выбор инструментальных средств.

Основным дистрибутивом, использовавшимся при разработке, был Ubuntu (18.04). Сетевая часть программы будет реализована с помощью библиотек сокетов на C++. Протокол передачи данных: TCP.

В представленной работе нет графического интерфейса, используется обычная консоль.

1.3 Выбор модели.

В сетевой игре, в которой может принять участие несколько игроков, необходима определенная модель, которая обеспечит их взаимодействие в игре. Для этого была выбрана модель взаимодействия компьютеров и программ в сети клиент-сервер. Основная часть игры, заключающаяся в изменении состояния треков и переводу игры в различные состояния (ожидания, игры, таблицы результатов) должна работать на одном компьютере, называемом сервер.

1.4 Выбор способа решения проблемы блокирования ввода/вывода.

Для проблемы блокирования ввода/вывода было решено использовать создания нового потока для каждого клиента.

1.5 Выбор протокола транспортного уровня OSI-модели.

Для разработки игры необходимо выбрать какой протокол из TCP и UDP выбрать.

TCP-протокол обеспечивает надежность передаваемых данных. Он посылает подтверждения о приеме данных и в случае, если данные не получены или искажены запрашивает повторную передачу. Также этот протокол обеспечивает правильный порядок принятых данных. Это сложный, требующий больших затрат времени протокол, но он берет на себя заботу о гарантированной доставке пакетов, избавляя программиста от необходимости включать эту функциональную возможность в прикладной протокол.

UDP-протокол в отличии от TCP — очень быстрый протокол, поскольку в нем определен самый минимальный механизм, необходимый для передачи данных. В нем есть свои недостатки. Сообщения поступают в любом порядке, и то, которое отправлено первым, может быть получено последним. Доставка сообщений в UDP не гарантируется. Сообщения могут потеряться, или могут быть получены две копии одного и того же сообщения.

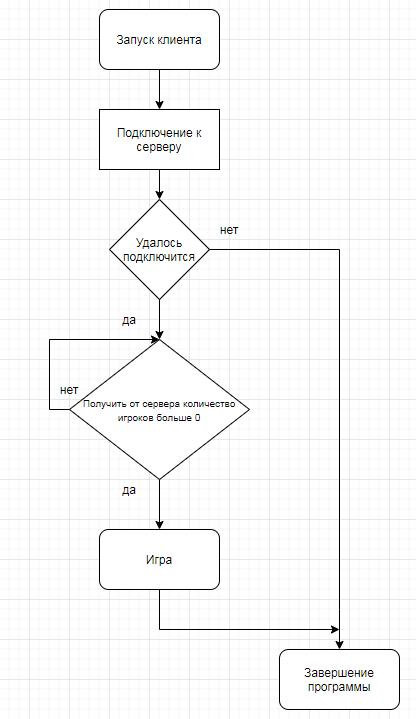
Для своей курсовой работы я решил использовать TCP-протокол, поскольку

придуманный мной протокол взаимодействия клиента и сервера основывается на том, что сообщения приходят с строго определённом порядке, а основные недостатки TCP относительно UDP нивелируются, тем для каждого игрока в любом случае должен создаваться отдельный поток.

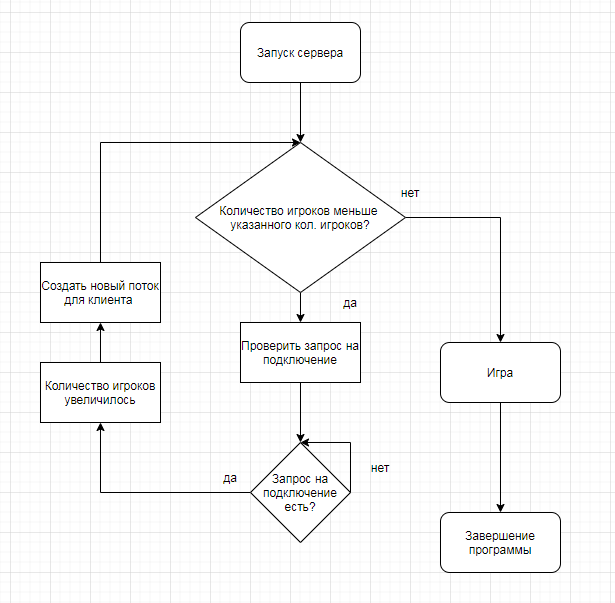
1.7 Алгоритм решения задачи.

Алгоритм начала игры выглядит следующим образом:

Со стороны клиента:



Со стороны сервера:



1.8 Вывод по первой главе.

Таким образом, для реализации игры «Гонки» основной ОС выбрана Ubuntu, языками программирования процедурной части C и С.

Для проблемы блокирования ввода/вывода было решено использовать создания нового потока для каждого клиента.

Был составлен протокол взаимодействия клиентов и сервера.

Используемым протоколом транспортного уровня OSI-модели был выбран TCP.

1. ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА.

2.1 Структура клиента и сервера.

В представленном проекте используется 3 основных файлов.

server.c - реализация сервера.

client.c - реализация клиента.

colors.h – содержит массив цветов.

2.2 Программный интерфейс сервера.

Описание работы главного потока:

При запуске сервера создаётся сокет на порте, указанном в параметрах командной строки, происходит ожидание подключения игроков в количестве также указанном в параметрах командной строки. Для каждого игрока создается создаётся отдельный поток. После того как все игроки подключились главный поток дает разрешение начать Гонку потокам игроков. В главном потоке также начинается Гонка которая закончиться если кто-нибудь из гонщиков доедет до финиша. В течении Гонки основной поток каждый ход получает текущую позицию и выводит гоночную трассу каждого игрока. Когда игрок доезжает до финиша то он получает свое место в таблице результатов и ждет пока остальные доберутся до финиша. После того как все игроки доедут до финиша основной поток отправляет команду потокам клиентов, что Гонка окончена. После этого выводится таблица результатов.

Описание работы потока игрока:

После создания потока игрока происходит установка информации о собственном подключении в общем состоянии игры.

Затем поток находится в состоянии ожидания остальных игроков(то есть команды старта Гонки). После старта отправляем клиенту количество игроков, длину трека и номер игрока. В течении Гонки происходит отправка клиенту статуса Гонки, считается приращение и оно прибавляется к текущей позиции и происходит отправка позиций всех игроков. Приращение считается рандомно с ограничением по скорости, если посчитанное рандомное приращение больше чем разница длинны трека и текущей позиции то приращение будет считаться по длинны и позиции. После получения команды об окончании гонки это статус посылается клиенту. Также посылается таблица результатов.

2.3 Описание программной реализации клиента

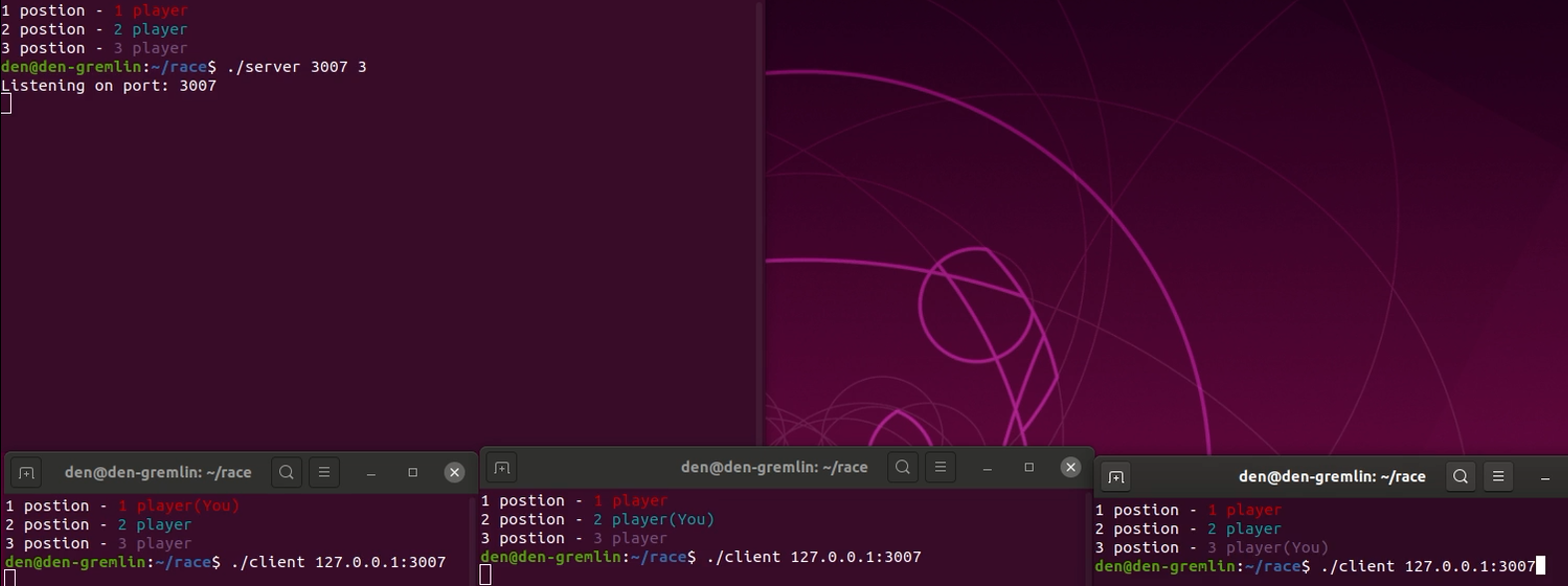
Клиент подключается к серверу по ip и порту указанным как параметр запуска и ожидает начала Гонки. Если клиент получает от сервера размер игроков отличных от нуля то Гонка началась которая закончиться если кто-нибудь из гонщиков доедет до финиша. Также клиент получает длину трека и номер гонщика. В течении Гонки клиент каждый ход получает от сервера текущую позицию и выводит гоночную трассу каждого игрока. Свидетельство о том, что Гонка для гонщика закончилась является получение от сервера статуса об окончании Гонки либо то что он доехал до финиша. После окончания Гонки клиент получает от сервера таблицу результатов и выводит ее.

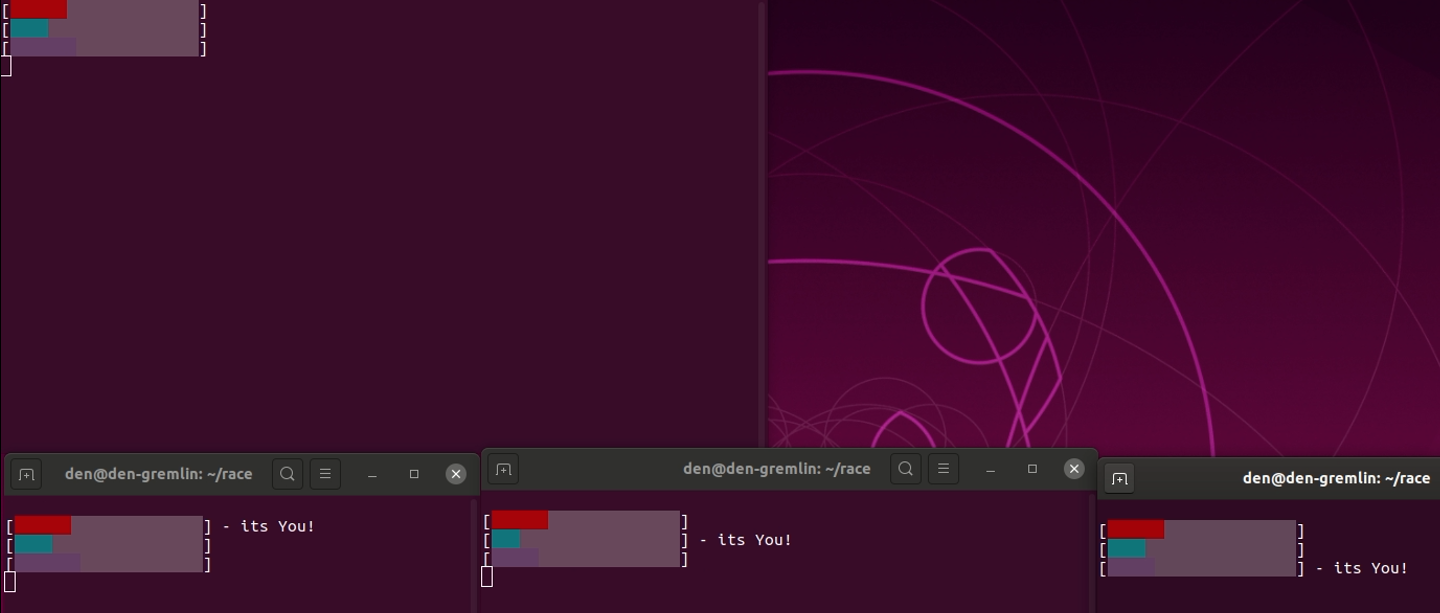
2.4 Системные вызовы.

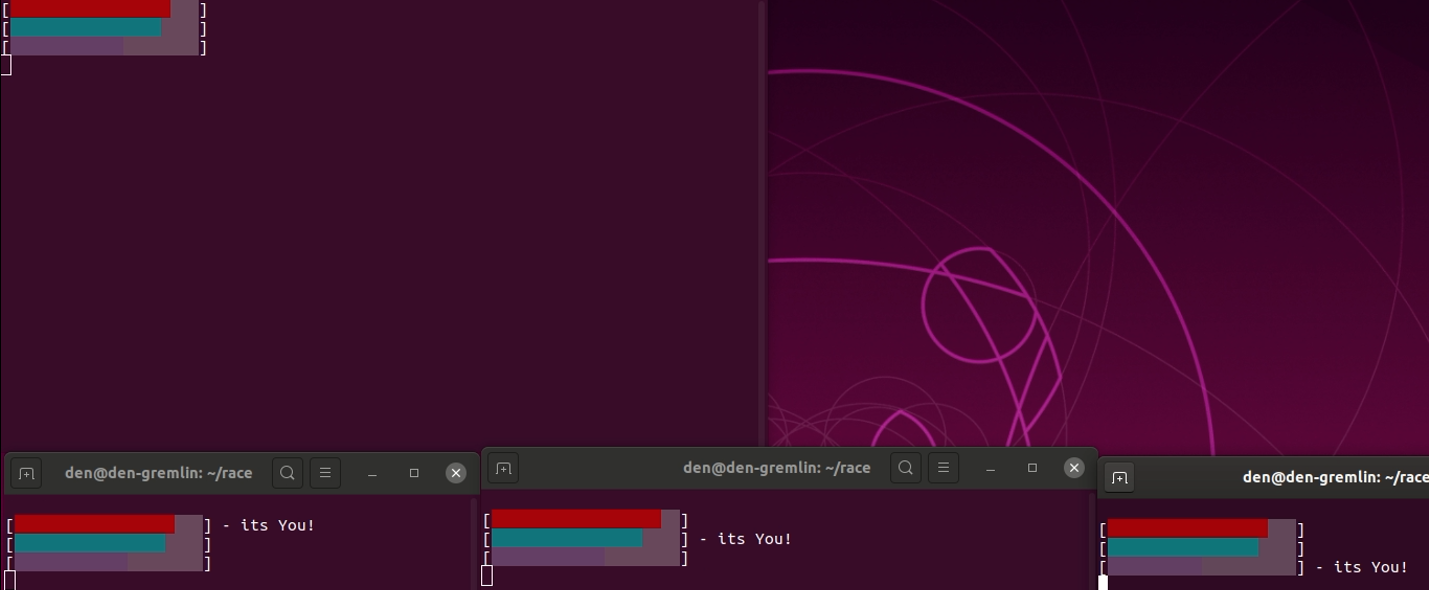
Для данного проекта использовались следующие системные вызовы:

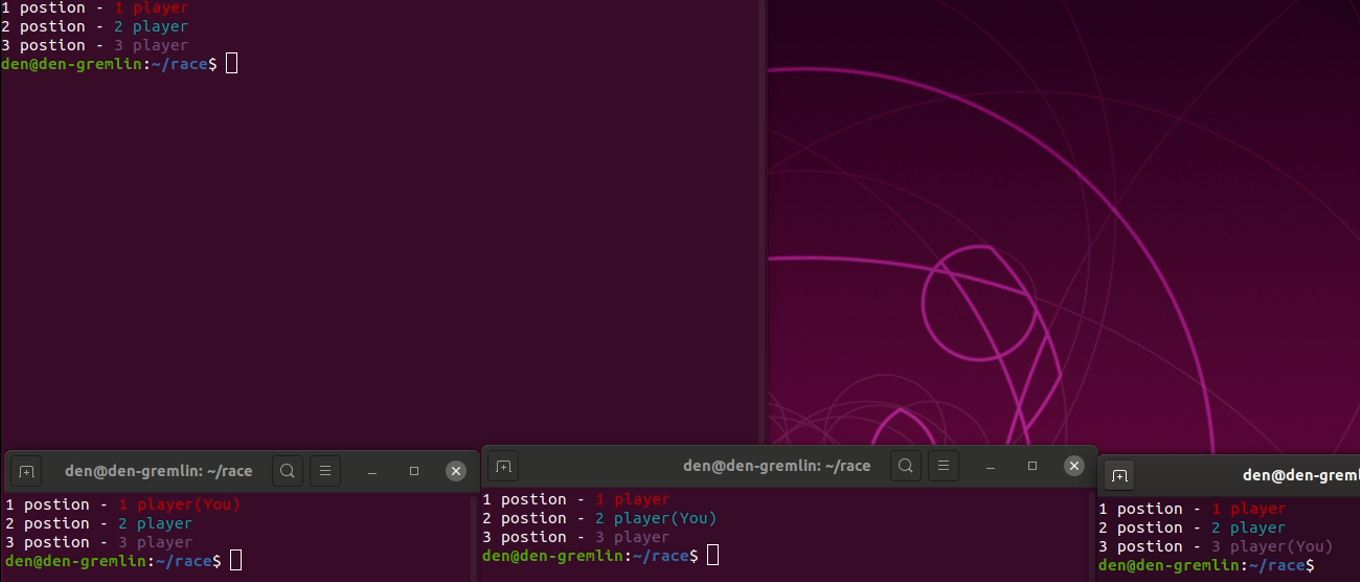
|  |  |
| --- | --- |
| Системный вызов | Описание |
| socket() | Служит для создания сокета в ОС. Имеет три парметра. Первый указывает, к какому семейству протоколов относится создаваемый сокет, а второй и третий параметры определяют конкретный протокол внутри данного семейства. |
| connect() | Служит для установки логического соединения со стороны клиента. Имеет три параметра: дескриптор активного сокета, через который будет устанавливаться соединение, полный адрес сокета сервера и его длина. |
| read() и write() | Используются для чтения и записи файлов. Могут быть использованы для обмена данных, если в качестве параметров вместо дескрипторов файлов в них задаются дескрипторы сокетов. |
| bind() | Настраивает адрес для созданного сокета. Первый параметр вызова должен содержать дескриптор сокета, для которого производится настройка адреса. Второй и третий параметры задают этот адрес. |
| listen() | Прослушивает соединения на сокете. Позволяет показать готовность принимать соединения и задать лимит входящих соединений. В качестве первого параметра используется дескриптор сокета. Второй параметр определяет максимальную длину очереди входящих соединений. |
| accept() | Извлекает первый запрос на соединение из очереди ожидающих соединений, создаёт новый подключенный сокет и выделяет для сокета новый файловый дескриптор, который и возвращается. В качестве параметров принимает дескриптор слушающего сокета, указатель на структуру адреса и размер структуры адреса. |

2.5 Пример работы приложения.





**



2.6 Результат работы.

В ходе выполнения данной курсовой работы удалось разработать сетевую игру “Гонки”, соответствующую поставленным к ней требованиям.

1. ЗАКЛЮЧЕНИЕ

Таким образом, результат курсовой работы соответствует требованиям к реализации. Приложение работает на операционных системах семейства Linux. Для построения серверной части приложения использован язык C и компилятор gсс, а для клиентской язык C++ и компилятор g++. Сетевое взаимодействие было построено с помощью протокола TCP. Сервер может одновременно обслуживать несколько клиентов. В целом созданная сетевая игра логически завершена.

1. СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.
2. Карпов В.Е., Коньков К.А. - Основы операционных систем. Практикум.
3. Карпов В.Е., Коньков К.А. - Основы операционных систем. Лекции+семинары.
4. ПРИЛОЖЕНИЕ С КОДОМ ПРОГРАММ
5. Листинг A1 — Код из файла server.c

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <string.h>

#include <stdio.h>

#include <errno.h>

#include <unistd.h>

#include <limits.h>

#include <stdlib.h>

#include <pthread.h>

#include <sys/sem.h>

#include "colors.h"

#define TRACK\_LEN 20

#define MAX\_SPEED 5

struct thread\_args

{

int sockfd;

int semId;

char ind;

char \*players;

char \*gameStarted;

char \*connected;

char \*positions;

char \*leaderboard;

char \*raceFinished;

};

char min(char a, char b)

{

if (a > b)

{

return b;

}

else

{

return a;

}

}

void sem(int semId, int n, int d)

{

struct sembuf op;

op.sem\_op = d;

op.sem\_flg = 0;

op.sem\_num = n;

semop(semId, &op, 1);

}

void semUnlock(int semId, int n)

{

sem(semId, n, 1);

}

void semLock(int semId, int n)

{

sem(semId, n, -1);

}

void \*player\_thread(void \*t\_args)

{

struct thread\_args \*args = (struct thread\_args \*)t\_args;

int newsockfd = args->sockfd;

int semId = args->semId;

char ind = args->ind;

char \*players = args->players;

char \*gameStarted = args->gameStarted;

char \*connected = args->connected;

char \*positions = args->positions;

char \*raceFinished = args->raceFinished;

char \*leaderboard = args->leaderboard;

free(t\_args);

while (!(\*gameStarted))

{

char sync = 0;

if (!write(newsockfd, &sync, sizeof(sync)))

{

(\*connected) = 0;

(\*players)--;

return NULL;

}

sleep(1);

}

semUnlock(semId, 1 + ind);

semLock(semId, 0);

char track\_len = TRACK\_LEN;

write(newsockfd, players, sizeof(\*players));

write(newsockfd, &track\_len, sizeof(track\_len));

write(newsockfd, &ind, sizeof(ind));

char status = 0;

while (!(\*raceFinished))

{

write(newsockfd, &status, sizeof(status));

(positions[ind]) += min(rand() % MAX\_SPEED + 1, TRACK\_LEN - positions[ind]);

sleep(1);

write(newsockfd, positions, sizeof(\*positions) \* (\*players));

}

status = 1;

write(newsockfd, &status, sizeof(status));

write(newsockfd, leaderboard, sizeof(\*leaderboard) \* (\*players));

semUnlock(semId, 0);

}

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

{

int listener, sock;

listener = socket(PF\_INET, SOCK\_STREAM, 0);

struct sockaddr\_in addr, cliaddr;

bzero(&addr, sizeof(addr));

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(atoi(argv[1]));

addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)) < 0)

{

addr.sin\_port = 0;

if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)) < 0)

{

perror("Bind");

close(listener);

exit(2);

}

}

char gameStarted = 0;

char players = 0;

char playerSlots = atoi(argv[2]);

char playersConnected[playerSlots];

char playersPositions[playerSlots];

char playersFinished[playerSlots];

char leaderrboard[playerSlots];

char leaderrboardInd = 0;

bzero(playersConnected, sizeof(\*playersConnected) \* playerSlots);

bzero(playersPositions, sizeof(\*playersPositions) \* playerSlots);

bzero(playersFinished, sizeof(\*playersFinished) \* playerSlots);

socklen\_t servlen = sizeof(addr);

getsockname(listener, (struct sockaddr \*)&addr, &servlen);

printf("Listening on port: %d\n", ntohs(addr.sin\_port));

listen(listener, 10);

int semId = semget(IPC\_PRIVATE, 1 + playerSlots, 0600 | IPC\_CREAT);

char globalRaceFinished = 0;

while (players < playerSlots)

{

socklen\_t clilen = sizeof(cliaddr);

sock = accept(listener, (struct sockaddr \*)&cliaddr, &clilen);

struct thread\_args \*new\_thread\_args = malloc(sizeof(struct thread\_args));

new\_thread\_args->sockfd = sock;

new\_thread\_args->semId = semId;

new\_thread\_args->gameStarted = &gameStarted;

new\_thread\_args->players = &players;

new\_thread\_args->raceFinished = &globalRaceFinished;

int i;

for (i = 0; i < playerSlots && playersConnected[i]; i++)

;

if (i >= playerSlots)

continue;

new\_thread\_args->ind = i;

new\_thread\_args->connected = playersConnected + i;

new\_thread\_args->positions = playersPositions;

new\_thread\_args->leaderboard = leaderrboard;

players++;

playersConnected[i] = 1;

pthread\_t thread;

pthread\_create(&thread, NULL, player\_thread, new\_thread\_args);

}

gameStarted = 1;

for (int i = 1; i <= playerSlots; i++)

{

semLock(semId, i);

}

sem(semId, 0, playerSlots);

char raceFinished;

do

{

raceFinished = 1;

for (int i = 0; i < playerSlots; i++)

{

if (playersPositions[i] < TRACK\_LEN)

{

raceFinished = 0;

}

else

{

if (!playersFinished[i])

{

playersFinished[i] = 1;

leaderrboard[leaderrboardInd] = i;

leaderrboardInd++;

}

}

printf("[");

for (int j = 0; j < TRACK\_LEN; j++)

{

if (playersPositions[i] > j)

{

printf("%s▓", colors[i]);

printf(RESET\_COLOR);

}

else

{

printf("░");

}

}

printf("]\n");

}

sleep(1);

system("clear");

} while (!raceFinished);

globalRaceFinished = 1;

for (int i = 0; i < playerSlots; i++)

{

printf("%d postion - %s%d player%s\n", i + 1, colors[leaderrboard[i]], leaderrboard[i] + 1, RESET\_COLOR);

}

sem(semId, 0, -players);

semctl(semId, 0, IPC\_RMID);

}

1. Листинг A2 — Код из файла client.c

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/mman.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <fcntl.h>

#include <string.h>

#include <stdio.h>

#include <errno.h>

#include <unistd.h>

#include <limits.h>

#include <stdlib.h>

#include <pthread.h>

#include "colors.h"

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

{

int port = atoi(strchr(argv[1], ':') + 1);

int pointer = strchr(argv[1], ':') - argv[1];

char ip[pointer + 1];

strncpy(ip, argv[1], pointer);

ip[pointer] = '\0';

int sock;

sock = socket(PF\_INET, SOCK\_STREAM, 0);

struct sockaddr\_in addr;

bzero(&addr, sizeof(addr));

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(port);

inet\_aton(ip, &addr.sin\_addr);

if (connect(sock, (struct sockaddr \*)&addr, sizeof(addr)) < 0)

{

perror("Connect");

exit(2);

}

char players;

do

{

read(sock, &players, sizeof(players));

} while (players == 0);

char track\_len;

read(sock, &track\_len, sizeof(track\_len));

char ind;

read(sock, &ind, sizeof(ind));

char positions[players];

while (1)

{

system("clear");

char status;

if (!read(sock, &status, sizeof(status)))

{

break;

}

if (status == 1)

{

break;

}

if (!read(sock, positions, sizeof(\*positions) \* players))

{

break;

}

printf("\n");

for (int i = 0; i < players; i++)

{

printf("[");

for (int j = 0; j < track\_len; j++)

{

if (positions[i] > j)

{

printf("%s▓", colors[i]);

printf(RESET\_COLOR);

}

else

{

printf("░");

}

}

printf("]");

if (i == ind)

{

printf(" - its You!");

}

printf("\n");

}

sleep(1);

}

char leaderrboard[players];

read(sock, leaderrboard, sizeof(\*leaderrboard) \* players);

for (int i = 0; i < players; i++)

{

printf("%d postion - %s%d player", i + 1, colors[leaderrboard[i]], leaderrboard[i] + 1);

if (leaderrboard[i] == ind)

{

printf("(You)");

}

printf("%s\n", RESET\_COLOR);

}

close(sock);

}

1. Листинг A2 — Код из файла colors.h

#define RESET\_COLOR "\033[0m"

char \*colors[] = {

"\033[0;31m",

"\033[0;36m",

"\033[0;35m",

"\033[1;32m",

"\033[0;33m",

"\033[1;36m",

"\033[0;34m",

"\033[0;32m",

"\033[1;34m",

"\033[1;35m",

"\033[1;31m",

"\033[1;33m",

};