Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования «Новгородский государственный университет имени Ярослава Мудрого »

Кафедра информационных технологий и систем

**Сетевая игра «Камень ножницы бумага»**

Курсовая работа

по дисциплине:

«Операционные системы»

Выполнил:

Студент группы 8091:

Кузин И. А.

«\_\_»\_\_\_\_\_2020г.

Проверил:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_»\_\_\_\_\_2020г.

**Великий Новгород**

**2020**

Оглавление

[Введение 3](#_Toc41827051)

[ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА 4](#_Toc41827052)

[1.1 Постановка задачи 4](#_Toc41827053)

[1.2 Выбор инструментальных средств 4](#_Toc41827054)

[1.3 Выбор модели 4](#_Toc41827055)

[1.4 Алгоритм решения задачи 4](#_Toc41827056)

[ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА 5](#_Toc41827057)

[2.1 Структура клиента и сервера 5](#_Toc41827058)

[2.2 Программный интерфейс сервера 5](#_Toc41827059)

[2.3 Описание программной реализации клиента 5](#_Toc41827060)

[2.4 Системные вызовы 5](#_Toc41827061)

[2.5 Пример работы приложения 7](#_Toc41827062)

[2.6 Результат работы 10](#_Toc41827063)

[ЗАКЛЮЧЕНИЕ 11](#_Toc41827064)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 12](#_Toc41827065)

[Листинг В1 — Код из файла client.с 13](#_Toc41827066)

[Листинг В2 — Код из файла server.c 16](#_Toc41827067)

Введение

При выполнении данной работы целью является разработка простейшей консольной версии игры «Камень ножницы бумага». **Камень, ножницы, бумага** — популярная игра на руках, известная во многих странах мира. Она часто используется как способ жеребьёвки для определения очерёдности хода в других играх.

Правила игры: Игроки произносят вместе, вслух «Камень… Ножницы… Бумага… Раз… Два… Три», одновременно качая кулаками. Существуют и другие варианты счёта, распространённость которых различается в разных городах и регионах, например, «Су-ли-фа», «Ю-зе-фа», «Бу-це-фа», «Аль… ман… джуз!» и др. На счёт «Три» они одновременно показывают при помощи руки один из трёх знаков: камень, ножницы или бумагу.

Победитель определяется по следующим правилам:

* Бумага побеждает камень («бумага обертывает камень»).
* Камень побеждает ножницы («камень затупляет или ломает ножницы»).
* Ножницы побеждают бумагу («ножницы разрезают бумагу»).

Если игроки показали одинаковый знак, то засчитывается ничья и игра переигрывается.

Для целей данной курсовой работы взята вариация игры с двумя игроками

Функции, реализованные в игре:

1. Подключение двух игроков
2. Возможность для игроков совершать ходы не зависимо друг от друга
3. Консольный интерфейс для ввода хода

ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА

1.1 Постановка задачи

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

1.2 Выбор инструментальных средств

Для реализации сетевого взаимодействия используются TCP – сокеты.

1.3 Выбор модели

Для взаимодействия между игроками выбрана клиент-серверная модель. Сервер обеспечивает параллельное получение информации о ходах, отправку результатов клиентам.

1.4 Алгоритм решения задачи

Подключение игрока/игроков происходит следующим образом:

К серверу подключаются оба игрока, введя адрес сервера и номер порта. Для синхронизации подключения используются семофоры. Ход игры:

Сервер ожидает ходы от обоих игроков, при этом для обработки ходов используются разные процессы. В связи с особенностью использования TCP сокетов, ожидание подключения происходит уже после открытия новых процессов. После того, как оба игрока совершили ходы, сервер отправляет результаты этих ходов. После этого, ждет от каждого игрока ответа о продолжении игры. Если один (или оба) игрока отказались от продолжения, то ожидается подключение новых игроков. Для синхронизации обработки хода, а также доступа к памяти используются семофоры.

ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА

2.1 Структура клиента и сервера

Соединение между клиентом и сервером происходит отдельно в процессе, открытом для данного клиента. Для игры также используются некоторые вспомогательные функции.

2.2 Программный интерфейс сервера

Для целей обмена информацией между клиентом и сервером используются специальные коды: 0 — код начала игры, 1- код победы, 2 — код поражения, 3 — код ничьей. Также для определения продолжения или прекращения игры от клиента ожидаются коды: 1 — продолжение, 2 — прекращение.

2.3 Описание программной реализации клиента

При запуске клиента происходит подключение к серверу. Клиент ожидает от сервера код начала игры. После этого клиент отправляет серверу свой ход. Клиент по кодам, получаемым от сервера, узнает выиграл ли он. Узнав данную информацию, он выводит на экран соответствующее сообщение. После этого узнает у игрока, хочет ли он продолжить игру. В случае отказа отправляет код отказа — 2 на сервер и прекращает работу.

2.4 Системные вызовы

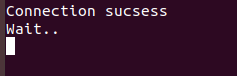
|  |  |
| --- | --- |
| socket() | Служит для создания сокета в ОС. Имеет три парметра. Первый указывает, к какому семейству протоколов относится создаваемый сокет, а второй и третий параметры определяют конкретный протокол внутри данного семейства. |
| Connect() | Служит для установки логического соединения со стороны клиента. Имеет три параметра: дескриптор активного сокета, через который будет устанавливаться соединение, полный адрес сокета сервера и его длина. |
| ioctl() | Выполняет различные управляющие действия над обычными устройствами и псевдоустройствами. Если в качестве параметров используются дескриптор сокета, флаг FIONREAD и целочисленная переменная, записывает в переменную, указанную третим параметром, количество доступных для чтения байт на сокете. |
| 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. |
| **shmget()** | возвращает идентификатор разделяемому сегменту памяти, соответствующий значению аргумента *key*. Создается новый разделяемый сегмент памяти с размером *size*( округленным до размера, кратного **PAGE\_SIZE**), если значение *key* равно **IPC\_PRIVATE** или если значение *key* не равно **IPC\_PRIVATE** и нет идентификатора, соответствующего *key*; причем, выражение *shmflg***&IPC\_CREAT** истинно |
| **fork** () | создает процесс-потомок, который отличается от родительского только значениями PID (идентификатор процесса) и PPID (идентификатор родительского процесса), а также тем фактом, что счетчики использования ресурсов установлены в 0. Блокировки файлов и сигналы, ожидающие обработки, не наследуются. |
| **int semget(key\_t** *key***,** **int** *nsems***,** **int** *semflg***)** | Системный вызов возвращает идентификатор набора семафоров, связанный с аргументом *key*. Создается новый набор из семафоров *nsems ,* если значение *key* равно **IPC\_PRIVATE** или c ключом *key* не связано ни одного существующего набора семафора, а выражение *semflg* & **IPC\_CREAT** истинно. |
| **int semop(int** *semid***,** **struct sembuf \****sops***,** **unsigned** *nsops***);** | Функция производит операции над выбранными элементами из набора семафоров *semid* |

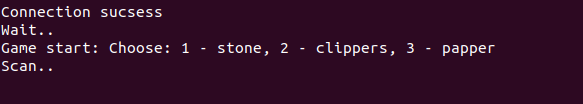
2.5 Пример работы приложения

Запуск сервера:

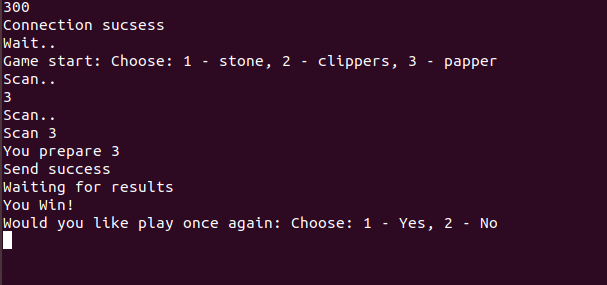


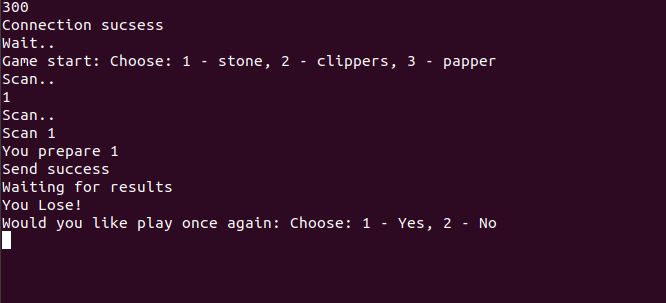
Подключение клиентов

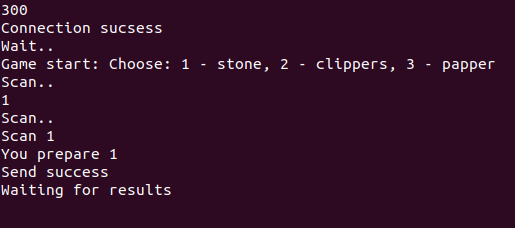


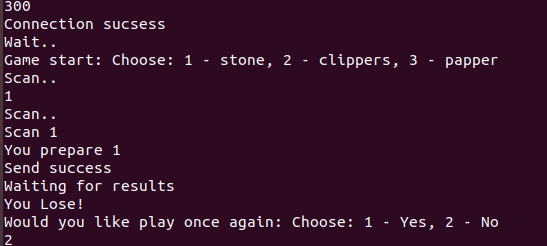


Выполнение ходов:









2.6 Результат работы

В результате проделанной работы, выполнена разработка работающей сетевой игры «Камень ножницы бумага», в которой для сетевого взаимодействия игроков используются клиент-серверная модель, а обмен данными осуществляется через TCP – сокеты.

ЗАКЛЮЧЕНИЕ

В результате, было создано серверное приложение, соответствующее цели работы. Приложение работает на операционных системах семейства Linux. Для построения сетевой части приложения использован язык C++ и компилятор g++. Сетевое взаимодействие было построено с помощью протокола TCP. Для обмена информацией между процессами использована разделяемая память, а для синхронизации процессов и доступа к разделяемой памяти использованы семафоры. В целом созданную сетевую игру можно считать логически завершенной.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Скотт Граннеман «Linux. Карманный справочник»
2. Брюс Моли «Unix/Linux. Теория и практика программирования»
3. https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BC%D0%B5%D0%BD%D1%8C,\_[%D0%BD%D0%BE%D0%B6%D0%BD%D0%B8%D1%86%D1%8B,\_%D0%B1%D1%83%D0%BC%D0%B0%D0%B3%D0%B0](https://ru.wikipedia.org/wiki/Камень,_ножницы,_бумага)
4. https://linux.die.net

Листинг В1 — Код из файла client.с

#include <stdio.h>

#include <sys/shm.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/sem.h>

#include <sys/msg.h>

#include <sys/ipc.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <signal.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <errno.h>

#include <iostream>

void game(int sockfd)

{

int check;

while (1)

{

read (sockfd, &check, sizeof(int)); // получаем инфомацию, что оба игрока готовы

check = ntohs(check);

char in;

do

{

printf("Game start: Choose: 1 - stone, 2 - clippers, 3 - papper\n");

printf("Scan..\n");

scanf("%c", &in);

printf("Scan..\n");

//printf("Scan %d\n", check);

} while ((in!='1')&&(in!='2')&&(in!='3'));

switch (in)

{

case '1':

check = 1;

break;

case '2':

check = 2;

break;

case '3':

check = 3;

break;

}

printf("You prepare %d\n", check);

check = htons(check);

write (sockfd, &check, sizeof(int)); // отправляем ход

printf("Send success\n");

printf("Waiting for results\n");

read (sockfd, &check, sizeof(int)); // получаем результаты игры

check = ntohs(check);

if (check == 1)

{

printf("You Win!\n");

}

else if (check == 2)

{

printf("You Lose!\n");

}

else if (check == 3)

{

printf("Stand off\n");

}

printf("Would you like play once again: Choose: 1 - Yes, 2 - No\n");

scanf("%d", &check);

check = htons(check);

write (sockfd, &check, sizeof(int)); // отправляем наше решение

if (ntohs(check)==2) // если не хочет играть, завершаем игру

break;

printf ("Wait..."); // Ждет второго игрока в случае продолжения

}

}

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

int sockfd = 0, n = 0;

char recvBuff[1024];

struct sockaddr\_in serv\_addr;

if(argc != 3) {

printf("\n Usage: %s <ip of server> \n",argv[0]);

return 1;

}

memset(recvBuff, '0',sizeof(recvBuff));

if((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

printf("\n Error : Could not create socket \n");

return 1;

}

memset(&serv\_addr, '0', sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(atoi(argv[2]));

if(inet\_pton(AF\_INET, argv[1], &serv\_addr.sin\_addr)<=0)

{

printf("\n inet\_pton error occured\n");

return 1;

}

if( connect(sockfd, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0)

{

printf("\n Error : Connect Failed \n");

return 1;

}

printf("Connection sucsess\n");

printf("Wait..\n");

game(sockfd);

}

Листинг В2 — Код из файла server.c

#include <stdio.h>

#include <sys/shm.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/sem.h>

#include <sys/msg.h>

#include <sys/ipc.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <signal.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <errno.h>

#include <time.h>

void initSemValue(int semId, int n)

{

struct sembuf op;

op.sem\_op = 1;

op.sem\_flg = 0;

op.sem\_num = n;

semop(semId, &op, 1);

}

void tryToGetSemOrWait(int semId, int n)

{

struct sembuf op;

op.sem\_op = -1;

op.sem\_flg = 0;

op.sem\_num = n;

semop(semId, &op, 1);

}

void releaseSem(int semId, int n)

{

initSemValue(semId, n);

}

int checkWin (int semId, int \*mem)

{

int res1, res2;

if (mem[1]==mem[2])

{

res1 = 3;

res1 = htons(res1);

write (mem[3], &res1, sizeof(int));

}

else if ((mem[1]==1)&&(mem[2]==2))

{

res1 = 1;

res2 = 2;

res1 = htons(res1);

res2 = htons(res2);

write (mem[3], &res1, sizeof(int));

}

else if ((mem[1]==2)&&(mem[2]==1))

{

res1 = 2;

res2 = 1;

res1 = htons(res1);

res2 = htons(res2);

write (mem[3], &res1, sizeof(int));

}

else if ((mem[1]==1)&&(mem[2]==3))

{

res1 = 2;

res2 = 1;

res1 = htons(res1);

res2 = htons(res2);

write (mem[3], &res1, sizeof(int));

}

else if ((mem[1]==3)&&(mem[2]==1))

{

res1 = 1;

res2 = 2;

res1 = htons(res1);

res2 = htons(res2);

write (mem[3], &res1, sizeof(int));

}

else if ((mem[1]==2)&&(mem[2]==3))

{

res1 = 1;

res2 = 2;

res1 = htons(res1);

res2 = htons(res2);

write (mem[3], &res1, sizeof(int));

}

else if ((mem[1]==3)&&(mem[2]==2))

{

res1 = 2;

res2 = 1;

res1 = htons(res1);

res2 = htons(res2);

write (mem[3], &res1, sizeof(int));

}

}

int checkWin1 (int semId, int \*mem)

{

int res1, res2;

if (mem[1]==mem[2])

{

res1 = 3;

res1 = htons(res1);

write (mem[4], &res1, sizeof(int));

}

else if ((mem[1]==1)&&(mem[2]==2))

{

res1 = 1;

res2 = 2;

res1 = htons(res1);

res2 = htons(res2);

write (mem[4], &res2, sizeof(int));

}

else if ((mem[1]==2)&&(mem[2]==1))

{

res1 = 2;

res2 = 1;

res1 = htons(res1);

res2 = htons(res2);

write (mem[4], &res2, sizeof(int));

}

else if ((mem[1]==1)&&(mem[2]==3))

{

res1 = 2;

res2 = 1;

res1 = htons(res1);

res2 = htons(res2);

write (mem[4], &res2, sizeof(int));

}

else if ((mem[1]==3)&&(mem[2]==1))

{

res1 = 1;

res2 = 2;

res1 = htons(res1);

res2 = htons(res2);

write (mem[4], &res2, sizeof(int));

}

else if ((mem[1]==2)&&(mem[2]==3))

{

res1 = 1;

res2 = 2;

res1 = htons(res1);

res2 = htons(res2);

write (mem[4], &res2, sizeof(int));

}

else if ((mem[1]==3)&&(mem[2]==2))

{

res1 = 2;

res2 = 1;

res1 = htons(res1);

res2 = htons(res2);

write (mem[4], &res2, sizeof(int));

}

}

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

int listenfd = 0;

struct sockaddr\_in serv\_addr;

char sendBuff[1025];

time\_t ticks;

listenfd = socket(AF\_INET, SOCK\_STREAM, 0);

memset(&serv\_addr, '0', sizeof(serv\_addr));

memset(sendBuff, '0', sizeof(sendBuff));

serv\_addr.sin\_family = AF\_INET;

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

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

bind(listenfd, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr));

listen(listenfd, 10);

int memId = shmget(IPC\_PRIVATE, 5\*sizeof(int), 0600|IPC\_CREAT|IPC\_EXCL);

if (memId <= 0)

{

printf("error with shmget()\n");

return -1;

}

int \*mem = (int \*)shmat(memId, 0, 0);

if (NULL == mem)

{

printf("error with shmat()\n");

return -2;

}

const size\_t semCount = 6;

int semId = semget(IPC\_PRIVATE, semCount, 0600|IPC\_CREAT);

if (semId < 0)

{

perror("error with semget()");

return -1;

}

else

{

printf("semaphore set id = %i\n", semId);

}

printf("1+\n");

printf("2+\n");

printf("3+\n");

printf("4+\n");

/\* mem[3] = accept(listenfd, (struct sockaddr\*)NULL, NULL); //ждем подключеня 1 клиента

printf("ID1 %d\n", mem[3]);

mem[4] = accept(listenfd, (struct sockaddr\*)NULL, NULL); //ждем подключеня 2 клиента \*/

printf("ID2 %d\n", mem[4]);

pid\_t childId = fork();

if (childId < 0)

{

perror("error with fork()\n");

}

else if (childId > 0)

{

releaseSem (semId, 1);

printf("1-\n");

mem[3] = accept(listenfd, (struct sockaddr\*)NULL, NULL); //ждем подключеня 1 клиента

tryToGetSemOrWait(semId, 1);

printf("1+\n");

releaseSem (semId, 5);

printf("5-\n");

releaseSem (semId, 2);

printf("2-\n");

releaseSem (semId, 4);

printf("4-\n");

while (1)

{

int check = 0;

check = htons (check);

write (mem[3], &check, sizeof(int)); // отпраляем статус начала

//write (mem[4], &check, sizeof(int));

printf("Game start\n");

printf("Wait 1\n");

read (mem[3], &check, sizeof(int)); // ждем ход

printf("Get 1\n");

check = ntohs (check);

printf("1 %d\n", check);

mem[1]=check; // конец хода

releaseSem (semId, 0);

printf("0-\n");

tryToGetSemOrWait(semId, 5);

printf("5+\n");

tryToGetSemOrWait(semId, 2);

printf("2+\n");

checkWin (semId, mem); // проверяем результат

printf("Check result success\n"); //end

releaseSem (semId, 1);

printf("1-\n");

releaseSem (semId, 2);

printf("2-\n");

int temp = mem[3];

read (temp, &check, sizeof(int)); // читаем ответ о продолжени

check = ntohs (check);

if (check==2){

close (mem[3]);

mem[3] = accept(listenfd, (struct sockaddr\*)NULL, NULL);

}

releaseSem (semId, 5);

printf("5-\n");

tryToGetSemOrWait(semId, 0);

printf("0+\n");

tryToGetSemOrWait(semId, 1);

printf("1+\n");

}

}

else

{

tryToGetSemOrWait(semId, 1);

printf("c1+\n");

mem[4] = accept(listenfd, (struct sockaddr\*)NULL, NULL); //ждем подключеня 2 клиента

releaseSem (semId, 1);

printf("c1-\n");

tryToGetSemOrWait(semId, 5);

printf("c5+\n");

tryToGetSemOrWait(semId, 2);

printf("c2+\n");

tryToGetSemOrWait(semId, 4);

printf("c4+\n");

while (1)

{

int check = 0;

write (mem[4], &check, sizeof(int));

printf("C: Game start\n");

printf("Wait 2\n");

printf("Check ID2 %d\n", mem[4]);

int spec;

read (mem[4], &spec, sizeof(int)); // ждем ход

printf("Get 2\n");

check = ntohs (spec);

printf("2 %d\n", check);

mem[2]=check;

releaseSem(semId, 5);

printf("c5-\n");

tryToGetSemOrWait(semId, 0);

printf("c0+\n");

releaseSem(semId, 2);

printf("c2-\n");

tryToGetSemOrWait(semId, 1);

printf("c1+\n");

tryToGetSemOrWait(semId, 2);

printf("c2+\n");

checkWin1 (semId, mem); // проверяем результат

printf("C: Check result success\n");

read (mem[4], &check, sizeof(int)); //otwet

check = ntohs (check);

if (check==2){

close (mem[4]);

mem[4] = accept(listenfd, (struct sockaddr\*)NULL, NULL);

}

releaseSem(semId, 0);

printf("c0-\n");

tryToGetSemOrWait(semId, 5);

printf("c5+\n");

releaseSem(semId, 1);

printf("c1-\n");

}

}

}