

Курсовая работа

Общие сведения

- Цель работы – получить навыки разработки сетевого программного обеспечения.
- Сеть – на основе стека TCP/IP.
- Базовый проект – сервер и клиент игры «Крестики нолики 3x3».
- Сервер – программа, обслуживающая запросы от пользователей и управляющая несколькими игровыми ситуациями.
- Клиент – программа, обеспечивающая интерфейс пользователя (под GNU/Linux в текстовом режиме).

Часть 1. Разработка алгоритма игровой ситуации и протокола взаимодействия клиентской и серверной частей

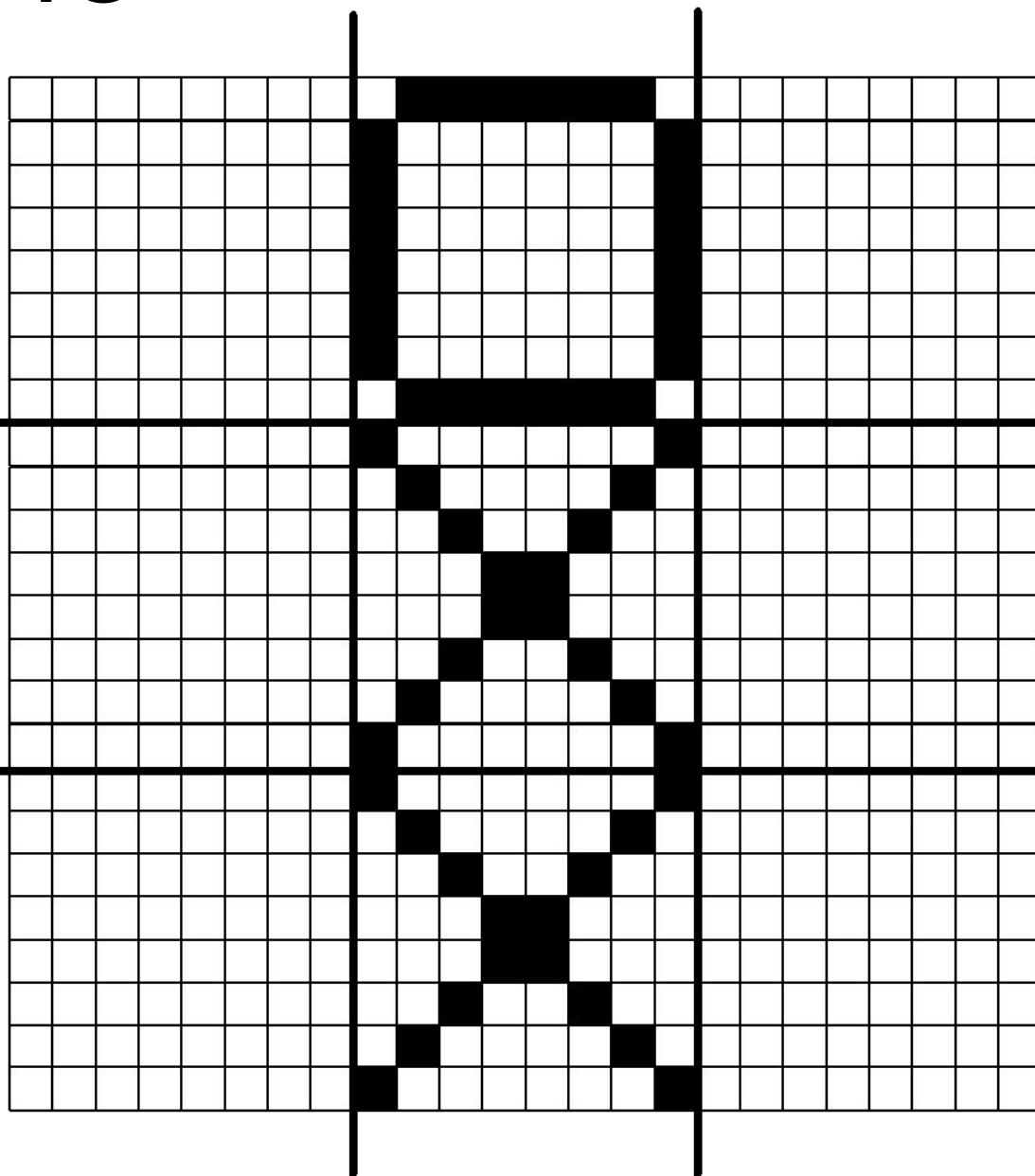
1. Опишите правила игры и алгоритм проведения игровой ситуации «крестики-нолики» (3x3).
2. Разработайте протокол взаимодействия клиентской и серверной частей приложения при: создании новой игры, подключении игроков к ранее созданной игре, проведении игровой ситуации и завершении её. Заложите возможность диалога игроков в режиме чата (через контролирующий сервер).
3. Определите тип протокола, который будет использоваться для взаимодействия клиентов и сервера (TCP или UDP).

Клиентское ПО

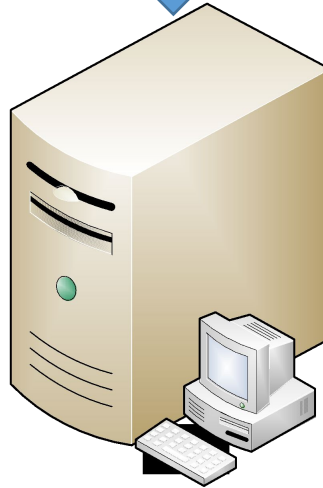
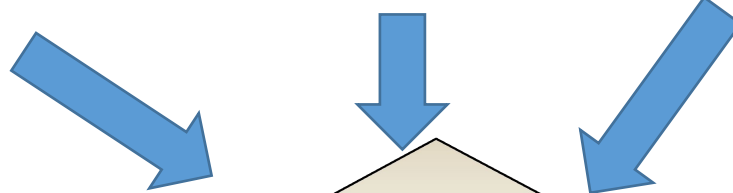
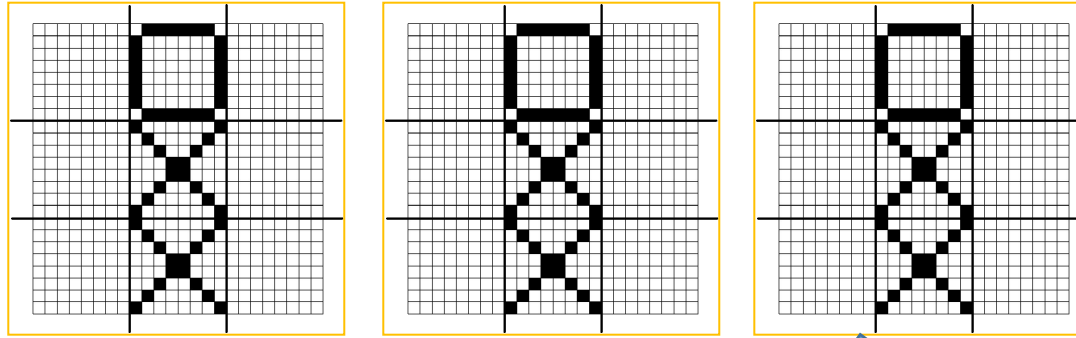
Имеется поле
3 на 3
«БОЛЬШИХ СИМВОЛОВ»
(библиотеки myTerm и
myBigChars)

Имеется место для
общения игроков
между собой

Контролем игры
занимается сервер



Серверное ПО



Сервер организует
несколько игровых
ситуаций

Для каждой пары
поддерживается
режим диалога

Взаимодействие
осуществляется по
TCP/IP

Часть 1. Разработка алгоритма игровой ситуации и протокола взаимодействия клиентской и серверной частей

4. С использованием библиотек MyTerm и BigChars, разработайте набор функций:
- `printBoard (struct board)`, выводящую на экран рабочее поле игры. Каждая ячейка поля представляется «большим» символом, само поле обведено в рамку. Рисунки символов «крестик» и «нолик» хранятся в одном файле (два больших символа, один - крестик, другой - нолик).
 - `setBoardPos (struct board, int, int, enum tPosSign)`, устанавливающую в указанную позицию поля крестик или нолик
 - `getBoardPos (struct board, int, int, enum tPosSign *)`, возвращает значение указанной позиции поля игры (крестик или нолик).
 - `editBoard (struct board *, int *, int *)`, реализующую редактирование поля (в свободное место устанавливается крестик (F5) или нолик (F6), на занятое место установить ничего нельзя) и возвращающую позицию установленного

«БОЛЬШОЙ СИМВОЛ»


- terminfo – база данных, описывающая характеристики терминалов

```
man terminfo
```


- infocmp – вывод информации о характеристиках терминалов

```
infocmp -1L xterm
```


«БОЛЬШОЙ СИМВОЛ»

```
enter_alt_charset_mode=\E(0  
ACS_CKBOARD a   
exit_alt_charset_mode=\E(B
```

```
#include <unistd.h>  
#include <string.h>  
#define ENTER_ALT_MODE "\E(0"  
#define EXIT_ALT_MODE "\E(B"  
#define ACS_CKBOARD "a"
```

```
gcc alt_mode.c  
./a.out  
  
a
```

```
int main()  
{  
    write(1, ENTER_ALT_MODE, strlen(ENTER_ALT_MODE));  
    write(1, ACS_CKBOARD, strlen(ACS_CKBOARD));  
    write(1, EXIT_ALT_MODE, strlen(EXIT_ALT_MODE));  
    write(1, "\n", 1);  
    write(1, ACS_CKBOARD, strlen(ACS_CKBOARD));  
    return 0;  
}
```

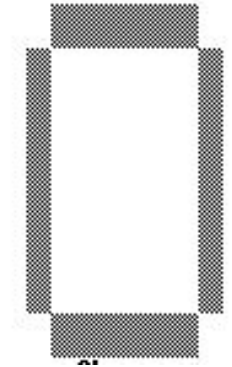
«БОЛЬШОЙ СИМВОЛ»

```
clear_screen=\E[H\E[2J  
cursor_address=\E[%i%p1%d;%p2%dH
```

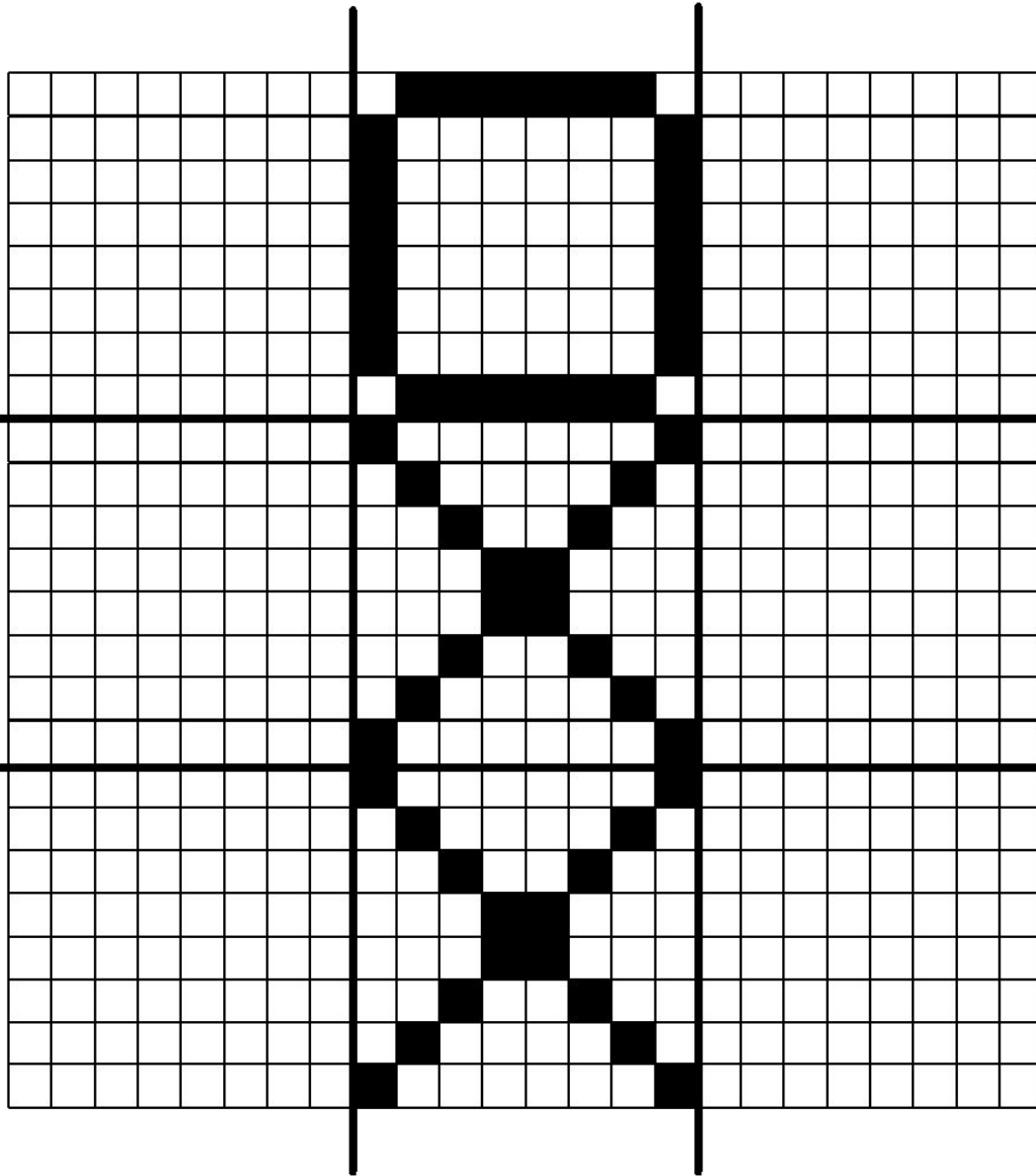
```
#define CLRSCR "\E[H\E[2J"  
#define GOTOXY "\E[%d;%dH"  
#define STDOUT 1  
int clrscr(){  
    if(write(STDOUT, CLRSCR, strlen(CLRSCR)) == -1)  
        { return -1; }  
    return 0;  
}  
int gotoXY(int x, int y){  
    char buf[15];  
    if(sprintf(buf, GOTOXY, x, y) > 0){  
        if(write(STDOUT, buf, strlen(buf)) == -1){  
            return -1;  
        }  
    }else{return -1;}  
    return 0;  
}
```

«БОЛЬШОЙ СИМВОЛ»

```
int main() {
    long int n = 0x7E81818181817E;
    int i = 0, x = 10, y = 10;
    clrscr();
    for(i = sizeof(n)*8; i > 0; i--){
long int j = (n >> (i - 1)) & 0x1;
if((i % 8) == 0){
    x = x + 1;
    y = 10;
}else{ y = y + 1; }
gotoXY(x, y);
if(j == 0) { write(1, " ", 1); }
else {
    write(1, ENTER_ALT_MODE,
                                strlen(ENTER_ALT_MODE));
    write(1, ACS_CKBOARD, strlen(ACS_CKBOARD));
    write(1, EXIT_ALT_MODE, strlen(EXIT_ALT_MODE));
}
}
return 0;
}
```



Игровое поле



<code>ACS_HLINE</code>	<code>q</code>
<code>ACS_VLINE</code>	<code>x</code>
<code>ACS_LLCORNER</code>	<code>m</code>
<code>ACS_LRCORNER</code>	<code>j</code>
<code>ACS_ULCORNER</code>	<code>l</code>
<code>ACS_URCORNER</code>	<code>k</code>

Обработка нажатия клавиш

```
key_f5=\E[15~
```

```
key_f6=\E[17~
```

Режимы работы терминала:

- *Канонический* - информация передаётся в ЭВМ только в виде законченных строк (после нажатия «ENTER»);
- *Неканонический* - вводимая информация сразу поступает в ЭВМ

Режимы работы терминала

```
#include <termios.h>
#include <unistd.h>

int tcgetattr (int fd, struct termios *tsaved);
int tcsetattr (int fd, int actions, const struct
               termios *tnew);

struct termios{
tcflag_t c_iflag;
tcflag_t c_oflag;
tcflag_t c_lflag; // поведение терминала при обработке
                  // и передаче информации в ЭВМ
tcflag_t c_cflag;
tcflag_t c_cc[NCCS]; // перечень символов, которые
                    // будут интерпретироваться как
                    // управляющие
};
```

Режимы работы терминала

```
struct termios tty, tty_new;
tcgetattr(1, &tty);

tty_new = tty;
tty_new.c_lflag &= ~(ICANON | ECHO);
tty_new.c_lflag |= ISIG; // разрешена обработка клавиш
                        // прерывания и аварийного
                        // завершения
tty_new.c_cc[VTIME] = 0; // read завершится только после
tty_new.c_cc[VMIN] = 1; // того, как будут считаны VMIN
                        // символов
tcsetattr(1, TCSADRAIN, &tty_new);
```

...

```
tcsetattr(1, TCSADRAIN, &tty_new);
```

```
int cmd = -1;
while(cmd != 0){
    read(1, &c, 1);
    switch(c){
        case '\033': cmd = 1;
            break;
        case '[':
            if(cmd == 1)
            {
                cmd = 2;
            }else{
                // Unknown key!
            }
            break;
        case '1':
            if(cmd == 2)
            {
                cmd = 3;
            }else{
                // Unknown key!
            }
            break;
```



```
case '5':
    if(cmd == 3)
    {
        cmd = 4;
    }else{
        // Unknown key!
    }
    break;
case '7':
    if(cmd == 3)
    {
        cmd = 4;
    }else{
        // Unknown key!
    }
    break;
```

```
case '~':  
    if(cmd == 4)  
    {  
        // F5!  
        cmd = 0;  
    }else if(cmd == 5)  
    {  
        // F6!  
        cmd = 0;  
    }else{  
        // Unknown key!  
    }  
    break;
```

```
}
```

Часть 1. Разработка алгоритма игровой ситуации и протокола взаимодействия клиентской и серверной частей

5. Реализуйте простейшую программу-сервер.
 - При запуске анализируются параметры командной строки, в которых указывается
 - количество одновременно поддерживаемых игр (опция `--count-games | -g`). По умолчанию возможно проведение только одной игры.
 - номер порта, который будет использоваться сервером для ожидания начальных запросов от клиентов. По умолчанию используется порт 7777.
 - Сервер открывает сокет, «привязывает» его к соответствующему интерфейсу и порту и «слушает» его.
 - При подключении клиента передаем ему строку «HELLO <IP>», в которой вместо IP указывается адрес только что подключившегося клиента.
 - В течение 60 секунд ожидается подключение ещё одного клиента. Если клиент подключился, то ему отправляется строка «HELLO. YOU ARE NUMBER TWO. THE FIRST CLIENT IS <IP>», в которой вместо IP отправляется адрес первого подключившегося клиента. Первому клиенту в этом случае отправляется строка вида «CLIENT NUMBER TWO IS <IP>», в которой вместо IP указывается адрес второго подключившегося клиента.
 - Порт закрывается и сервер завершает свою работу.

Часть 1. Разработка алгоритма игровой ситуации и протокола взаимодействия клиентской и серверной частей

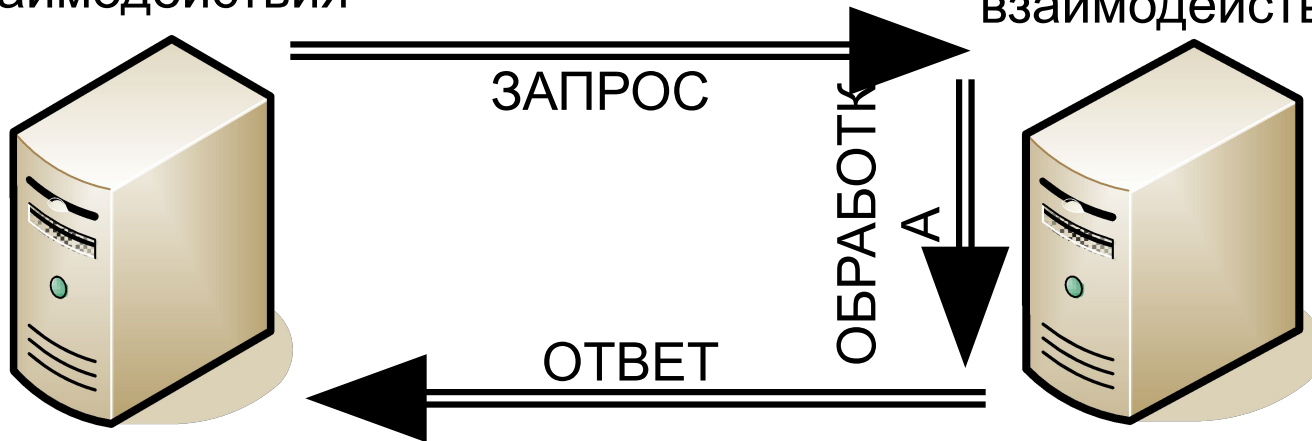
6. Клиент

- При запуске анализирует параметры командной строки, в которых указывается:
 - адрес игрового сервера (опция `--game-server | -s`). По умолчанию используется адрес `127.0.0.1`;
 - порт, на котором сервер ожидает сообщений от клиентов (опция `--game-server-port | -p`). По умолчанию используется порт `7777`.
- Клиент подключается к указанному серверу и порту.
- Ожидает от сервера строку (в течение 60 секунд).
- Если сервер присылает строку вида «HELLO <IP>», то ожидается (в течение 60 секунд) ещё одна строка вида «CLIENT NUMBER TWO IS <IP>».
- На экран выводится адрес второго игрока.
- Сокет закрывается и работа клиента завершается.

Распределенная система

Активная сторона
взаимодействия

Пассивная сторона
взаимодействия



Клиент

Сервер

Передаёт серверу запрос, включающий требование выполнить какое-либо действие или выдать какую-либо информацию.

Ожидает ответа от сервера.

Клиент получает ответ от сервера.

Ожидает от клиента запрос, включающий требование выполнить какое-либо действие или выдать какую-либо информацию.

Получив запрос, сервер обрабатывает его соответствующим образом и выдает клиенту ответ, содержащий результат его действий.

Распределенная система (РС)

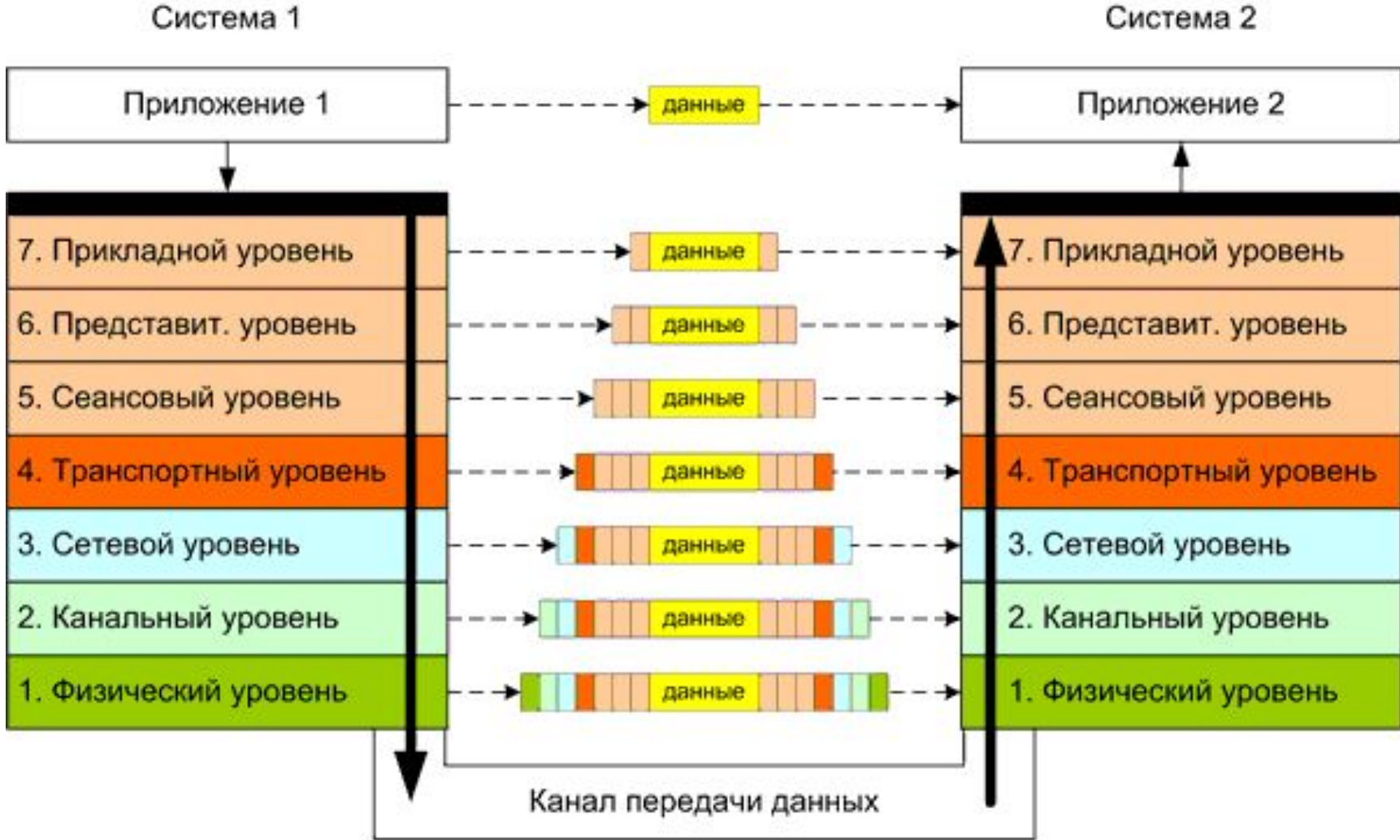
Вопрос. Как узлы РС могут взаимодействовать друг с другом?

Ответ. Только по средством передачи сообщений.

Вопрос. Что нужно реализовать, чтобы узлы РС «понимали» друг друга при передачи сообщений?

Ответ. Принцип открытой системы.

Модель OSI



- Логическое соединение между уровнями
- Реализация передачи данных

Модель OSI — Стек TCP/IP

Модель OSI

TCP/IP

Прикладной уровень

Уровень представления

Сеансовый уровень

Транспортный уровень

Сетевой уровень

Канальный уровень

Физический уровень

Уровень приложений /
процессов

Транспортный уровень

Сетевой уровень

Уровень сетевого
интерфейса

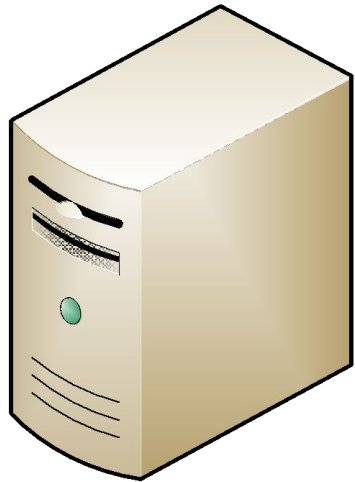
HTTP, RTP, FTP, DNS

TCP, UDP, SCTP, DCCP

IP, ICMP/ICMPv6, IGMP

Ethernet, IEEE 802.11
Wireless Ethernet, SLIP,
Token Ring, ATM и MPLS

Сокеты



Адрес:Порт

⬡ *:80
⬡ 127.0.0.1:22
⬡ 192.168.1.1:80
⬡ 1.0.0.1:80

```
#include <sys/socket.h>
#include <sys/types.h>
```

```
int socket(int domain,
           int type,
           int protocol);
```

	PF_INET	Протоколы семейства IPv4
domain	PF_INET6	Протоколы семейства IPv6
	PF_LOCAL	Локальные именованные каналы
	SOCK_STREAM	Протокол последовательной передачи данных с подтверждением доставки (TCP)
type	SOCK_DGRAM	Протокол пакетной передачи данных без подтверждения доставки (UDP)
protocol		32-разрядное целое число с сетевым порядком следования байтов (0)

СОКЕТЫ

```
#include <sys/socket.h>
```

```
#include <sys/types.h>
```

```
int connect(int sd, struct sockaddr *server,  
            int addr_len);
```

```
struct sockaddr  
{  
    short int sa_family;  
    char sa_data[14];  
}
```

```
struct sockaddr_in  
{  
    sa_family_t sin_family;  
    in_port_t sin_port;  
    struct in_addr sin_addr;  
    unsigned char sin_zero[8]  
}
```

СОКЕТЫ

```
#define PORT 13
struct sockaddr_in dest;
char host[] = "127.0.0.1";
int sd;

/**** Создание сокета ****/
bzero(&dest, sizeof(dest)); /* обнуляем структуру */
dest.sin_family = AF_INET; /* выбираем протокол */
dest.sin_port = htons(PORT); /* выбираем порт */
inet_aton(host, &dest.sin_addr); /* задаем адрес */

/* подключаемся! */
if(connect(sd, &dest, sizeof(dest)) != 0)
{
    perror(" socket connection ");
    abort();
}
```

СОКЕТЫ

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);

int sd, bytes_read;
sd = socket (PF_INET, SOCK_STREAM, 0);
bytes_read = read(sd, buffer, MAXBUF);
if(bytes_read < 0)
{
    /* сообщить об ошибках; завершить работу */
}

#include <unistd.h>
int close(int fd);
```

Сокеты

Клиент

```
socket()  
connect()  
... // Обмен данными  
close()
```

Сервер

```
socket()  
bind()  
listen()  
while()  
{  
    accept()  
    ... // Обмен данными  
    close() // клиент  
}  
close() // сервер
```

СОКЕТЫ

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
int bind(int sockfd, const struct sockaddr *addr,  
         socklen_t addrlen);
```

```
int listen(int socket, int backlog) ;
```

```
int accept(int socket, struct sockaddr *address,  
           socklen_t *address_len);
```

Форматы хранения данных

$$214259635_{10} = 0C C5 57 B3_{16}$$

□ *прямой* (от младшего к старшему, little-endian)

00	01	02	03
B3	57	C5	0C

□ *обратный* (от старшего к младшему, big-endian)

03	02	01	00
B3	57	C5	0C

Форматы хранения данных

- Порядок байтов, задаваемый компьютером или сервером, называется *серверным*
- Порядок байтов, определяемый сетевыми протоколами, называется *сетевым* и всегда является **обратным**

Функции преобразования порядка байтов

```
dest.sin_family /* серверный порядок байтов */  
dest.sin_port   /* сетевой порядок байтов */  
dest.sin_addr   /* сетевой порядок байтов */
```

```
uint32_t htonl(uint32_t hostlong); #include <arpa/inet.h>  
uint16_t htons(uint16_t hostshort);  
uint32_t ntohl(uint32_t netlong);  
uint16_t ntohs(uint16_t netshort);
```

htonl()	Из серверного порядка в сетевой длинный	32-разрядное число в обратном порядке
htons()	Из серверного порядка в сетевой короткий	16-разрядное число в обратном порядке
ntohl()	Из сетевого порядка в серверный длинный	32-разрядное число в серверном порядке
ntohs()	Из сетевого порядка в серверный короткий	16-разрядное число в серверном порядке

Функции преобразования порядка байтов

```
dest.sin_family /* серверный порядок байтов */
dest.sin_port   /* сетевой порядок байтов */
dest.sin_addr   /* сетевой порядок байтов */
```

```
#include <sys/socket.h>
```

```
#include <netinet/in.h>
```

```
#include <arpa/inet.h>
```

```
int inet_aton(const char *cp, struct in_addr *inp);
```

```
char *inet_ntoa(struct in_addr in);
```

127.0.0.1 

inet_aton() Преобразует адрес из точечной записи (###.###.###.###) в двоичную форму с сетевым порядком следования байтов; возвращает нуль в случае неудачи и ненулевое значение, если адрес допустимый

inet_ntoa() Преобразует IP-адрес, представленный в двоичном виде с сетевым порядком следования байтов, в точечную запись

Пример клиента

```
int sd, res;
struct sockaddr_in server;
char snd_line[] = "Hello, server!", recv_line[100];

sd = socket(PF_INET, SOCK_STREAM, 0);

bzero(&server, sizeof(struct sockaddr_in));
server.sin_family = AF_INET;
server.sin_port = htons(7777);
inet_aton("127.0.0.1", &(server.sin_addr));

connect(sd, (struct sockaddr *)&server, sizeof(server));

write(sd, snd_line, strlen(snd_line));
res = read(sd, recv_line, 100);

printf("Принято %d байт [%s].\n", res,
       recv_line);
close(sd);
```

Пример сервера

```
int sd, res, clientsd, client_size;
struct sockaddr_in server, client;
char snd_line[] = "Hello, client!", recv_line[100];

sd = socket(PF_INET, SOCK_STREAM, 0);
bzero(&server, sizeof(struct sockaddr_in));
server.sin_family = AF_INET;
server.sin_port = htons(7777);
server.sin_addr.s_addr = htonl(INADDR_ANY);
bind(sd, (struct sockaddr *)&server, sizeof(server));
listen(sd, 10);
client_size = sizeof(struct sockaddr_in);
clientsd = accept(sd, (struct sockaddr *)&client,
                 &client_size);

res = read(clientsd, recv_line, 100);
write(clientsd, snd_line, strlen(snd_line));
printf("Принято %d байт [%s].\n", res, recv_line);
close(clientsd);
close(sd);
```

Пример клиента и сервера



Терминал
сервера



Терминал
клиента

```
user@user-pc:~/sockets$ ./server
```

```
Принято 14 байт [Hello,  
server!].
```

```
user@user-pc:~/sockets$ ./client  
Принято 14 байт [Hello, client!].
```

Справочные материалы

- ✓ **Уолтон Ш.** Создание сетевых приложений в среде Linux. : Пер. с англ.— М.: Вильямс, 2001. — 464 с.
- ✓ **Иванов Н.Н.** Программирование в Linux. Самоучитель. – СПб.: БХВ-Петербург, 2007. – 416 с.
- ✓ man