

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»
(БГТУ им. В.Г.Шухова)**

Лабораторная работа №1
дисциплина «Сети ЭВМ и телекоммуникации»
по теме «Протокол сетевого уровня IPX»

Выполнил: студент группы ВТ-31
Проверил:

Макаров Д.С.
Федотов Е.А.

Белгород 2020

Лабораторная работа №1

«Протокол сетевого уровня IPX»

Цель работы:изучить протокол сетевого уровня IPX, основные функции API драйвера IPX и разработать программу для приема/передачи данных.

Вариант 6

Содержание отчета

1. Краткие теоретические сведения.
2. Основные функции API, использованные в данной работе.
3. Разработка программы. Блок-схемы программы.
4. Анализ функционирования разработанных программ.
5. Выводы.
6. Тексты программ. Скриншоты программ.

Ход работы

1. Краткие теоретические сведения.

Протокол IPX(*internetwork packet exchange*) — протокол сетевого уровня модели OSI в стеке протоколов IPX/SPX. Предназначен для передачи датаграмм. Для передачи данных установки соединения не требуется (так же, как для IP).

Роль компьютера в сети определяется программным обеспечением.

Сетям присваивается адрес от 00:00:00:01 до FF:FF:FF:FE. Адрес 00:00:00:00 - адрес локальной сети, адрес FF:FF:FF:FF - широковещательный адрес (пакеты адресованные этой сети отправляются всей сети).

2. Основные функции API, использованные в данной работе.

- Функция `int IPXOpenSocket(int SocketType, unsigned *Socket)`.

Входные параметры: тип сокета (долгосрочный, краткосрочный), номер сокета(от 4000).

Выходные параметры: если Socket пустой то автоматически назначает сокет.

Назначение: открывает сокет с номером Socket, типа Type.

- Функция `void IPXCloseSocket(unsigned *Socket)`.

Входные параметры: номер сокета.

Назначение: закрывает сокет с номером Socket.

- Функция `void IPXListenForPacket(struct ECB *RxECB)`.

Входные параметры: структура ECB

Назначение: запрос к драйверу IPX на получение входящего пакета, и запись результата в ECB.

- Функция `void IPXSendPacket(struct ECB *TxECB)`.

Входные параметры: структура ECB

Назначение: запрос к драйверу IPX на отправку пакета из структуры ECB

3. Разработка программы. Блок-схемы программы.

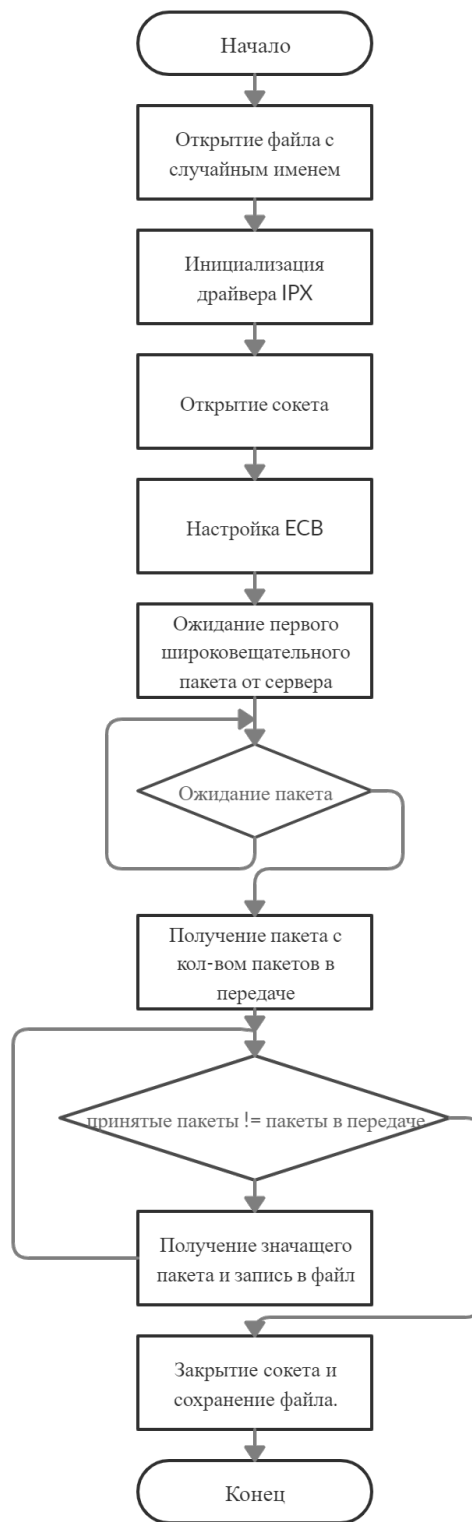


Рис. 1: Блок схема программы клиент

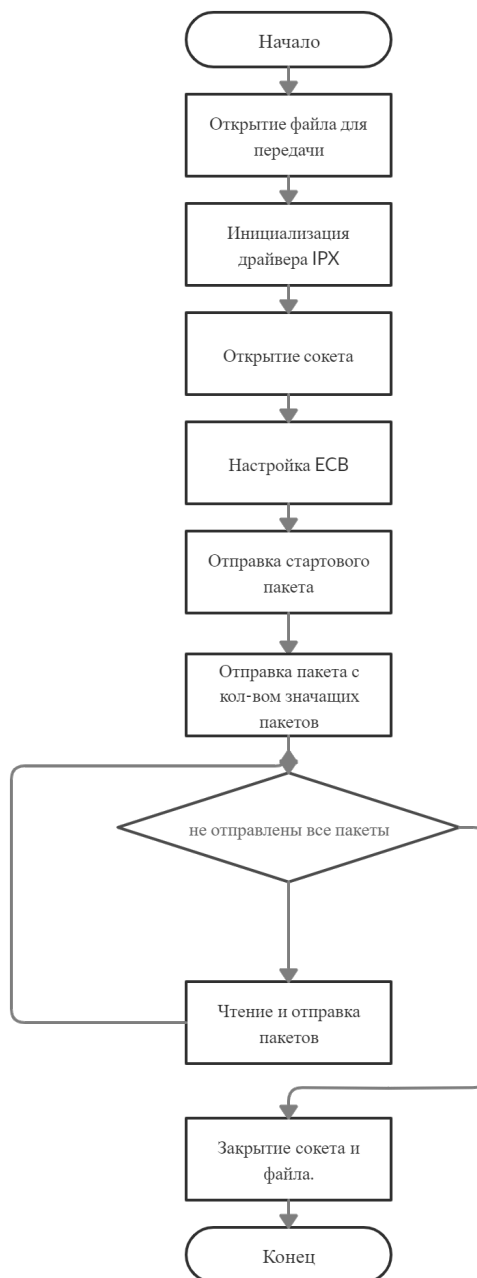


Рис. 2: Блок схема программы сервер

4. Анализ функционирования разработанных программ.

При отправке изображения блоками по 504 байта, без задержки между пакетами со стороны сервера, полученное изображения имеет сильные дефекты. С добавлением задержки количество дефектов уменьшается, но уменьшается скорость передачи.



Рис. 3: Исходное изображение



Рис. 4: Изображение переданное без задержки между пакетами

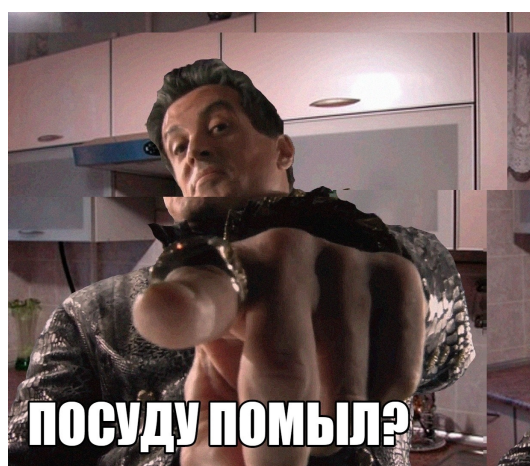


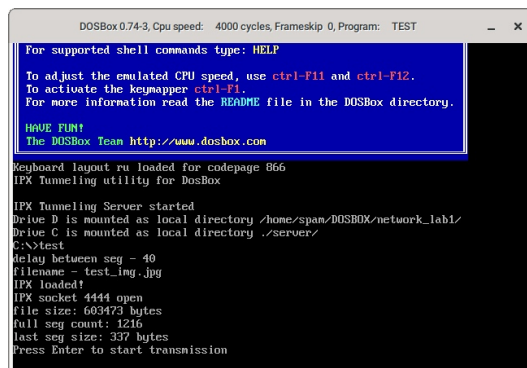
Рис. 5: Изображение переданное с задержкой между пакетами 50 мс

5. Выводы.

Протокол IPX не гарантирует порядок и успешность доставки пакетов, поэтому использовать его без надстроек для передачи файлов нецелесообразно.

6. Тексты программ. Скриншоты программ.

Тексты программ см. в приложении.



```
DOSBox 0.74-3, Cpu speed: 4000 cycles, Frameskip 0, Program: TEST

For supported shell commands type: HELP

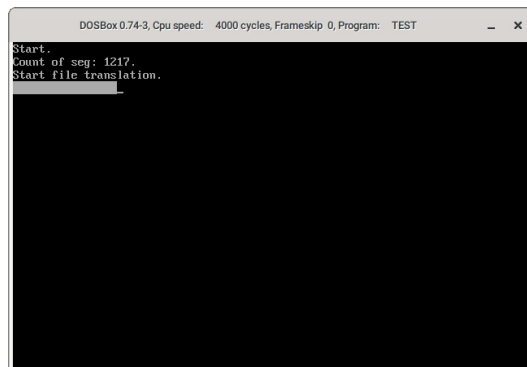
To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Keyboard layout ru loaded for codepage 866
IPX Tunneling utility for DosBox

IPX Tunneling Server started
Drive D is mounted as local directory /home/pan/DOSBOX/network_lab1/
Drive C is mounted as local directory ./server/
C:\>test
Delay between seg - 40
filename - test_img.jpg
IPX loaded!
IPX socket 4444 open
file size: 603473 bytes
full seg count: 1216
last seg size: 337 bytes
Press Enter to start transmission
```

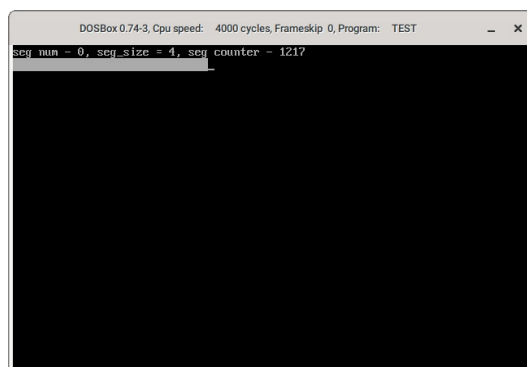
Рис. 6: Программа-сервер в ожидании начала передачи



```
DOSBox 0.74-3, Cpu speed: 4000 cycles, Frameskip 0, Program: TEST

Start.
Count of seg: 1217.
Start file translation.
```

Рис. 7: Программа-сервер в процессе передачи



```
DOSBox 0.74-3, Cpu speed: 4000 cycles, Frameskip 0, Program: TEST

seg num = 0, seg_size = 4, seg counter = 1217
```

Рис. 8: Программа-клиент в процессе передачи

Приложение

Содержимое файла CLIENT.C

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include "ipx.h"
#include <time.h>

#define BUFFER_SIZE 504
#define RANDOM_CHAR() rand()%35+55
typedef struct _file_segment
{
    long int segment_size;
    long int segment_num;
} file_segment;
unsigned char segment_buffer[BUFFER_SIZE];

void main(void) {
    //объявление переменных до вызова процедур.
    //размер файла в байтах
    long int file_size;
    //кол-во полных сегментов, всего сегментов
    long int full_segments_count, seg_counter;
    //размер неполного последнего сегмента (если 0 то сегмента нету)
    long int last_segment_size;
    int i,j;
    FILE* file;
    file_segment temp_segment;
    int progress_step;

    static unsigned Socket = 0x4444;
    struct ECB RxECB;
    struct IPX_HEADER RxHeader;
    unsigned char RxBuffer[BUFFER_SIZE];

    unsigned int sval;
    time_t t;
    char name[9];

    sval=(unsigned)time(&t);
    srand(sval);

    //рандом имени файла
    name[0]=RANDOM_CHAR(); name[1]=RANDOM_CHAR(); name[2]=RANDOM_CHAR();
    ↪ name[3]=RANDOM_CHAR();
    name[4]='.'; name[5]='j'; name[6]='p'; name[7]='g'; name[8]='\0';

    printf("temp filename - %s\n",name);
    file = fopen(name,"wb+");
    if(file==NULL){
        exit(-1);
    }
    if(ipx_init() != 0xff) {
        printf("IPX not loaded!\n"); exit(-1);
    }
}
```



```

if(IPXOpenSocket(SHORT_LIVED, &Socket)) {
    printf("Socket open error\n");
    exit(-1);
};

memset(&RxECB, 0, sizeof(RxECB));
RxECB.Socket = IntSwap(Socket);
RxECB.FragmentCnt= 3;
RxECB.Packet[0].Address = &RxHeader;
RxECB.Packet[0].Size = sizeof(RxHeader);
RxECB.Packet[1].Address = RxBuffer;
RxECB.Packet[1].Size = BUFFER_SIZE;
RxECB.Packet[2].Address = &temp_segment;
RxECB.Packet[2].Size = 8;

//ожидание первого пакета
IPXListenForPacket(&RxECB);
printf("Await server response..");
while(1) {
    if(RxECB.InUse){
        printf(".");
        printf("\b");
    }else{
        printf("\nResponse accepted.\n");
        break;
    };
}
clrscr();

//ожидание пакета с размером файла
IPXListenForPacket(&RxECB);
while(RxECB.InUse){}
memcpy(&seg_counter,RxBuffer,temp_segment.segment_size);
printf("seg num - %ld, seg_size = %ld, seg counter -
↪ %ld\n",temp_segment.segment_num,temp_segment.segment_size,seg_counter);
progress_step=seg_counter/70-1;

for(i=0;i<=seg_counter;i++){
    IPXListenForPacket(&RxECB);
    while(RxECB.InUse){}
    for(j=0;j<temp_segment.segment_size;j++){
        fputc(RxBuffer[j],file);
    };
    if(i%progress_step==0) printf("%c",219);
    if(temp_segment.segment_num==seg_counter){
        break;
    };
}
fclose(file);
printf("ready.\n");
IPXCloseSocket(&Socket);

exit(0);
}

```

Содержимое файла IPX.C

```

#include <stdio.h>
#include <stdlib.h>

```

```

#include <dos.h>
#include "ipx.h"

unsigned IntSwap(unsigned i) {
    return((i>>8) | (i & 0xff)<<8);
}

int IPXOpenSocket(int SocketType, unsigned *Socket) {

    struct IPXSPX_REGS iregs;

    iregs.bx = IPX_CMD_OPEN_SOCKET;
    iregs.dx = IntSwap(*Socket);
    iregs.ax = SocketType;
    ipxspx_entry( (void far *)&iregs );

    *Socket = IntSwap(iregs.dx);
    return(iregs.ax);
}

void IPXCloseSocket(unsigned *Socket) {

    struct IPXSPX_REGS iregs;

    iregs.bx = IPX_CMD_CLOSE_SOCKET;
    iregs.dx = IntSwap(*Socket);
    ipxspx_entry( (void far *)&iregs );
}

void IPXListenForPacket(struct ECB *RxECB) {

    struct IPXSPX_REGS iregs;

    iregs.es = FP_SEG((void far*)RxECB);
    iregs.si = FP_OFF((void far*)RxECB);
    iregs.bx = IPX_CMD_LISTEN_FOR_PACKET;
    ipxspx_entry( (void far *)&iregs );
}

void IPXSendPacket(struct ECB *TxECB) {

    struct IPXSPX_REGS iregs;

    iregs.es = FP_SEG((void far*)TxECB);
    iregs.si = FP_OFF((void far*)TxECB);
    iregs.bx = IPX_CMD_SEND_PACKET;
    ipxspx_entry( (void far *)&iregs );
}

void IPXRelinquishControl(void) {

    struct IPXSPX_REGS iregs;

    iregs.bx = IPX_CMD_RELINQUISH_CONTROL;
    ipxspx_entry( (void far *)&iregs );
}

```

Содержимое файла IPX.H

```

#define IPX_CMD_OPEN_SOCKET          0x00
#define IPX_CMD_CLOSE_SOCKET        0x01

```

```

#define IPX_CMD_GET_LOCAL_TARGET                0x02
#define IPX_CMD_SEND_PACKET                    0x03
#define IPX_CMD_LISTEN_FOR_PACKET              0x04
#define IPX_CMD_SCHEDULE_IPX_EVENT            0x05
#define IPX_CMD_CANCEL_EVENT                  0x06
#define IPX_CMD_GET_INTERVAL_MARKER            0x08
#define IPX_CMD_GET_INTERNETWORK_ADDRESS      0x09
#define IPX_CMD_RELINQUISH_CONTROL             0x0a
#define IPX_CMD_DISCONNECT_FROM_TARGET        0x0b

#define NO_ERRORS                               0
#define ERR_NO_IPX                             1
#define ERR_NO_SPX                             2
#define NO_LOGGED_ON                           3
#define UNKNOWN_ERROR    0xff

#define SHORT_LIVED        0
#define LONG_LIVED         0xff
#define IPX_DATA_PACKET_MAXSIZE 546

void far    ipxspx_entry(void far *ptr);
int         ipx_init(void);

struct IPXSPX_REGS {
    unsigned int    ax;
    unsigned int    bx;
    unsigned int    cx;
    unsigned int    dx;
    unsigned int    si;
    unsigned int    di;
    unsigned int    es;
};

struct IPX_HEADER {
    unsigned int    Checksum;
    unsigned int    Length;
    unsigned char   TransportControl;
    unsigned char   PacketType;
    unsigned char   DestNetwork[4];
    unsigned char   DestNode[6];
    unsigned int    DestSocket;
    unsigned char   SourceNetwork[4];
    unsigned char   SourceNode[6];
    unsigned int    SourceSocket;
};

struct ECB {
    void far        *Link;
    void far        (*ESRAddress)(void);
    unsigned char    InUse;
    unsigned char    CCode;
    unsigned int     Socket;
    unsigned int     ConnectionId;
    unsigned int     RestOfWorkspace;
    unsigned char    DriverWorkspace[12];
    unsigned char    ImmAddress[6];
    unsigned int     FragmentCnt;
    struct {
        void far        *Address;
        unsigned int     Size;
    };
};

```

```

    } Packet[2];
};

unsigned IntSwap(unsigned i);
int IPXOpenSocket(int SocketType, unsigned *Socket);
void IPXCloseSocket(unsigned *Socket);
void IPXListenForPacket(struct ECB *RxECB);
void IPXRelinquishControl(void);
void IPXSendPacket(struct ECB *TxECB);

```

Содержимое файла SERVER.C

```

#include <STDIO.H>
#include <STDLIB.H>
#include <CONIO.H>
#include <MEM.H>
#include <STRING.H>
#include <DOS.H>
#include "IPX.H"

#define BUFFER_SIZE 504

// -1 сегмент - начало передачи
// 0 сегмент - кол-во значащих сегментов
// 1-n сегмент - значащие сегменты
// n+1 сегмент - конец передачи

typedef struct _file_segment
{
    long int segment_size;
    long int segment_num;
} file_segment;

unsigned char segment_buffer[BUFFER_SIZE];

long int get_file_size(FILE* stream){
    long int size;
    fseek(stream,0,SEEK_END);
    size = ftell(stream);
    fseek(stream,0,SEEK_SET);
    return size;
};

void main(void) {
    //размер файла в байтах
    long int file_size;
    //кол-во полных сегментов, всего сегментов
    long int full_segments_count, seg_counter;
    //размер неполного последнего сегмента (если 0 то сегмента нету)
    long int last_segment_size;
    FILE* file;
    FILE* check_file;
    file_segment temp_segment;

    static unsigned Socket = 0x4444;
    struct ECB ServerECB;
    struct IPX_HEADER InHeader, OutHeader;
    unsigned char OutBuffer[BUFFER_SIZE];
    int i;
    int progress_step;

```

```

char filename[255] = "test_img.bmp";
int value_delay = 0;

printf("delay between seg - ");
scanf("%i",&value_delay);

printf("filename - ");
scanf("%s",filename);
if(ipx_init() != 0xff) {
    printf("IPX not loaded!\n");
    exit(-1);
}else{
    printf("IPX loaded!\n");
}

if(IPXOpenSocket(SHORT_LIVED, &Socket)) {
    printf("IPX socket open error\n");
    exit(-1);
}else{
    printf("IPX socket %x open\n",Socket);
};
memset(&ServerECB, 0, sizeof(ServerECB));

//открытие файла и расчет кол-ва сегментов
file=fopen(filename,"rb+");
if(file!=NULL){
    file_size = get_file_size(file);
    full_segments_count = file_size / (BUFFER_SIZE-8);
    last_segment_size = file_size % (BUFFER_SIZE-8);
    seg_counter = full_segments_count+1;
    printf("file size: %ld bytes\n",file_size);
    printf("full seg count: %ld\n",full_segments_count);
    printf("last seg size: %ld bytes\n",last_segment_size);
    fclose(file);
    file=fopen(filename,"rb+");
    check_file=fopen("IMG.bmp","wb+");
}else{
    exit(1);
};

printf("Press Enter to start transmission\n");
getch();

clrscr();
//Пакет "начало передачи"
temp_segment.segment_size=0;
temp_segment.segment_num=-1;
segment_buffer[0]=0;
//Тело
memcpy(OutBuffer,&segment_buffer,BUFFER_SIZE);
//заголовок
OutHeader.PacketType = 4;
memset(OutHeader.DestNetwork, 0, 4);
memset(OutHeader.DestNode, 0xFF, 6);
OutHeader.DestSocket = IntSwap(Socket);
//заполнение ECB
ServerECB.Socket= IntSwap(Socket);
ServerECB.FragmentCnt= 3;
ServerECB.Packet[0].Address = &OutHeader;
ServerECB.Packet[0].Size = sizeof(OutHeader);

```

```

ServerECB.Packet[1].Address = OutBuffer;
ServerECB.Packet[1].Size = BUFFER_SIZE;
ServerECB.Packet[2].Address = &temp_segment;
ServerECB.Packet[2].Size = 8;
IPXSendPacket(&ServerECB);
printf("Start.\n");

//Пакет содержащий кол-во сегментов
temp_segment.segment_num=0;
temp_segment.segment_size=sizeof(seg_counter);
memcpy(segment_buffer,&seg_counter,sizeof(seg_counter));
memcpy(OutBuffer,&segment_buffer,BUFFER_SIZE);
IPXSendPacket(&ServerECB);
printf("Count of seg: %ld.\n",seg_counter);

progress_step=seg_counter/70-1;
printf("Start file translation.\n");
for(i=1;i<=full_segments_count;i++){
    //установка номера сегмента файла
    temp_segment.segment_num=i;
    //установка размера сегмента файла
    temp_segment.segment_size=BUFFER_SIZE;
    //отправка пакета
    //шаг прогресса
    fread(OutBuffer,1,BUFFER_SIZE,file);
    fwrite(OutBuffer,1,BUFFER_SIZE,check_file);
    if(i%progress_step==0) printf("%c",219);
    IPXSendPacket(&ServerECB);
    delay(value_delay);
};

    delay(value_delay);
    temp_segment.segment_num=seg_counter;
    temp_segment.segment_size=last_segment_size;
    fread(OutBuffer,1,last_segment_size,file);
    fwrite(OutBuffer,1,last_segment_size,check_file);
    IPXSendPacket(&ServerECB);
    printf("%i\n",i);
printf("Transmit end.\n",seg_counter);

fclose(file);
fclose(check_file);
IPXCloseSocket(&Socket);
exit(0);
}

```