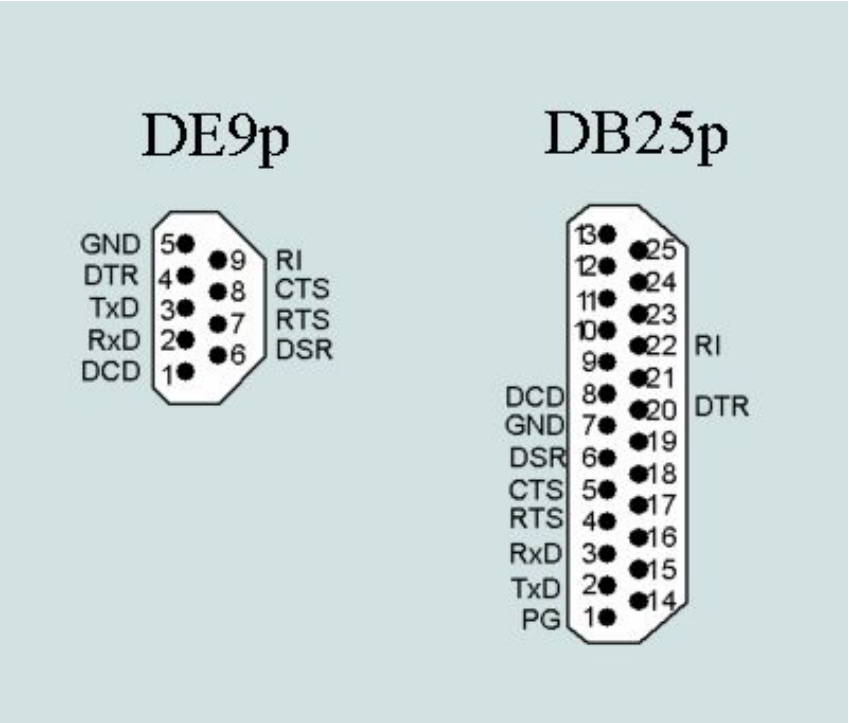
DE9p
GND 5
DTR 4
TxD 3
RxD 2
DCD 1
9 RI
8 CTS
7 RTS
6 DSR
DB25p
13
12
11
10
9
DCD 8
GND 7
DSR 6
CTS 5
RTS 4
RxD 3
TxD 2
PG 1
25
24
23
22 RI
21
20 DTR
19
18
17
16
15
14


6.3v
3300μF
6.3v
3300μF
KZG

number	name	description	level	Dir.	Etc.
1	DCD	Data Carrier Detect	RS232	Input	Mandatory
2	RXD	Receive Data	RS232	Input	Mandatory
3	TXD	Transmit Data	RS232	Output	Mandatory
4	DTR	Data Terminal Ready	RS232	Output	Optional
5	GND	Ground	Ground	-	Mandatory
6	DSR	Data Set Ready	RS232	Input	Optional
7	RTS	Request To Send	RS232	Output	Optional
8	CTS	Clear To Send	RS232	Input	Optional
9	RI	Ring Indicator	RS232	Input	Optional

```
//функция открытия и инициализации порта
void COMOpen()
{
String portname;  //имя порта (например, "COM1", "COM2" и т.д.)
DCB dcb;  //структура для общей инициализации портаDCB
COMMTIMEOUTS timeouts;  //структура для установки таймаутов
portname = Form1->ComboBox1->Text;  //получить имя выбранного порта
//открыть порт, для асинхронных операций обязательно нужно указать флаг
FILE_FLAG_OVERLAPPED
COMport = CreateFile(portname.c_str(),GENERIC_READ | GENERIC_WRITE, 0, NULL,
OPEN_EXISTING, FILE_FLAG_OVERLAPPED, NULL);

if(COMport == INVALID_HANDLE_VALUE)  //если ошибка открытия порта
{
Form1->SpeedButton1->Down = false;  //отжать кнопку
Form1->StatusBar1->Panels->Items[0]->Text = "Не удалось открыть порт";  //вывести
сообщение в строке состояния
return;
}
```

```
//инициализация порта
dcb.DCBlength = sizeof(DCB);  //в первое поле структуры DCB необходимо занести её длину, она будет использоваться функциями настройки порта для контроля корректности структуры
//считать структуруDCB из порта
if(!GetCommState(COMport, &dcb)) //если не удалось- закрыть порт и вывести сообщение об ошибке в строке состояния
{
COMClose();
Form1->StatusBar1->Panels->Items[0]->Text = "Не удалось считатьDCB";
return;  }
//инициализация структуры DCB
dcb.BaudRate = StrToInt(Form1->ComboBox2->Text);  //задаём скорость передачи 115200 бод
dcb.fBinary = TRUE;  //включаем двоичный режим обмена
dcb.fOutxCtsFlow = FALSE;  //выключаем режим слежения за сигналомCTS
dcb.fOutxDsrFlow = FALSE;  //выключаем режим слежения за сигналомDSR
dcb.fDtrControl = DTR_CONTROL_DISABLE;  //отключаем использование линии DTR
dcb.fDsrSensitivity = FALSE;  //отключаем восприимчивость драйвера к состоянию линии DSR
dcb.fNull = FALSE;  //разрешить приём нулевых байтов
dcb.fRtsControl = RTS_CONTROL_DISABLE;  //отключаем использование линииRTS
dcb.fAbortOnError = FALSE;  //отключаем остановку всех операций чтения/записи при ошибке
dcb.ByteSize = 8;  //задаём8 бит в байте
dcb.Parity = 0;  //отключаем проверку чётности
dcb.StopBits = 0;  //задаём один стоп-бит
```

```
//загрузить структуру DCB в порт
if(!SetCommState(COMport, &dcb)) //если не удалось- закрыть порт и вывести сообщение об ошибке в строке состояния
{
COMClose();
Form1->StatusBar1->Panels->Items[0]->Text = "Не удалось установитьDCB";
return;
}
//установить таймауты
timeouts.ReadIntervalTimeout = 0;   //таймаут между двумя символами
timeouts.ReadTotalTimeoutMultiplier = 0; //общий таймаут операции чтения
timeouts.ReadTotalTimeoutConstant = 0;  //константа для общего таймаута операции чтения
timeouts.WriteTotalTimeoutMultiplier = 0;  //общий таймаут операции записи
timeouts.WriteTotalTimeoutConstant = 0;  //константа для общего таймаута операции записи
//записать структуру таймаутов в порт
if(!SetCommTimeouts(COMport, &timeouts)) //если не удалось- закрыть порт и вывести сообщение об ошибке в строке состояния
{
COMClose();
Form1->StatusBar1->Panels->Items[0]->Text = "Не удалось установить тайм-ауты";
return;
}
//установить размеры очередей приёма и передачи
SetupComm(COMport,2000,2000);
```

//создать или открыть существующий файл для записи принимаемых данных
handle = open("test.txt", O_CREAT | O_APPEND | O_BINARY | O_WRONLY, S_IREAD | S_IWRITE);
if(handle==-1) //если произошла ошибка открытия файла
{
Form1->StatusBar1->Panels->Items[1]->Text = "Ошибка открытия файла"; //вывести сообщение об этом в командной строке

}
else { Form1->StatusBar1->Panels->Items[0]->Text = "Файл открыт успешно"; } //иначе вывести в строке состояния сообщение об успешном открытии файла
PurgeComm(COMport, PURGE_RXCLEAR); //очистить принимающий буфер порта
reader = new ReadThread(false); //создать и запустить поток чтения байтов
reader->FreeOnTerminate = true; //установить это свойство потока, чтобы он автоматически уничтожался после завершения
}

ReadFile(COMport, bufrd, btr, &temp, &overlapped);

WriteFile(COMport, bufwr, strlen(bufwr), &temp, &overlappedwr);

Цикл читання даних з COM порту

0) при відкритті порту для асинхронних операцій функцією **CreateFile** потрібно використовувати прапорець **FILE_FLAG_OVERLAPPED**, а також необхідна структура **OVERLAPPED**;

1) функцією **CreateEvent** створюємо сигнальний об'єкт-подія з ручним скиданням в несигнальному стан;

2) функцією **SetCommMask** встановлюємо маску очікуваної події для відкритого порту;

3) запускаємо цикл, який буде працювати весь час існування потоку;

4) функцією **WaitCommEvent** запускаємо операцію що перекривається в стан очікування цієї події, при цьому сигнальний об'єкт-подія перейде в несигнальному стан;

5) функцією **WaitForSingleObject** поміщаємо потік в стан ефективного очікування (Припиняємо) до тих пір, поки не відбудеться подія, і об'єкт-подія не встановиться в сигнальний стан;

6) коли подія відбулася (об'єкт-подія встановився в сигнальний стан), і потік активувався, функцією **GetOverlappedResult** перевіряємо результат операції **WaitCommEvent**. якщо результат успішний, виконуємо наступні кроки (7, 8 і 9), інакше-переходимо на початок циклу (крок 4);

7) по масці подій, яка передавалася в функцію **WaitCommEvent**, перевіряємо, що сталося саме подія приходу байта. Якщо маска в функції **SetCommMask** вказувала тільки на одну подію, перевірку можна не виконувати.

8) функцією **ClearCommError** заповнюємо структуру **COMSTAT** і з її поля **cbInQue** зчитуємо кількість доступних для читання байтів.

9) функцією **ReadFile** зчитуємо ці байти.

10) переходимо на початок циклу

```
CloseHandle(COMport);  //закрыть порт
COMport=0;  //обнулить переменную для дескриптора порта
close(handle);  //закрыть файл для записи принимаемых данных
handle=0;   //обнулить переменную для дескриптора файла
```

```
//---------------------------------------------------------------------------
//поток для чтения последовательности байтов из COM-порта в буфер
class ReadThread : public TThread
{
private:  void __fastcall Printing();  //вывод принятых байтов на экран и в файл
protected: void __fastcall Execute();  //основная функция потока
public:  __fastcall ReadThread(bool CreateSuspended);  //конструктор потока
};
//---------------------------------------------------------------------------
//..... потокReadThead .....
ReadThread *reader;  //объект потокаReadThread
//---------------------------------------------------------------------------
//конструктор потока ReadThread, по умолчанию пустой
__fastcall ReadThread::ReadThread(bool CreateSuspended) : TThread(CreateSuspended)
{}
//---------------------------------------------------------------------------
//главная функция потока, реализует приём байтов из COM-порта
void __fastcall ReadThread::Execute()
{
OVERLAPPED over;
COMSTAT comstat;   //структура текущего состояния порта, в данной программе используетс
для определения количества принятых в порт байтов
DWORD btr, temp, mask, signal;  //переменная temp используется в качестве заглушки
overlapped.hEvent = CreateEvent(NULL, true, true, NULL);  //создать сигнальный объект-соб
 для асинхронных операций
SetCommMask(COMport, EV_RXCHAR);   //установить маску на срабатывание по событию п
 байта в порт
```

```
while(!Terminated)   //пока поток не будет прерван, выполняем цикл
{
WaitCommEvent(COMport, &mask, &overlapped);  //ожидать события приёма байта(это
 и есть перекрываемая операция)
signal = WaitForSingleObject(overlapped.hEvent, INFINITE);  //приостановить поток до прихода байта
if(signal == WAIT_OBJECT_0)      //если событие прихода байта произошло
{
if(GetOverlappedResult(COMport, &overlapped, &temp, true)) //проверяем, успешно ли завершилась
перекрываемая операция WaitCommEvent
if((mask & EV_RXCHAR)!=0)      //если произошло именно событие прихода байта
{  ClearCommError(COMport, &temp, &comstat);   //нужно заполнить структуру COMSTAT
btr = comstat.cbInQue;  //и получить из неё количество принятых байтов
if(btr)    //если действительно есть байты для чтения
{
ReadFile(COMport, bufrd, btr, &temp, &overlapped);  //прочитать байты из порта в буфер программы
counter+=btr;  //увеличиваем счётчик байтов
Synchronize(Printing);   //вызываем функцию для вывода  данных на экран и в файл
}
}
}
}
CloseHandle(overlapped.hEvent);   //перед выходом из потока закрыть объект-событие
}
```

```
//---------------------------------------------------------------------------
//выводим принятые байты на экран и в файл(если включено)
void __fastcall ReadThread::Printing()
{
Form1->Memo1->Lines->Add((char*)bufrd); //выводим принятую строку вMemo
Form1->StatusBar1->Panels->Items[2]->Text = "Всего принято" + IntToStr(counter) + " байт";\
//выводим счётчик в строке состояния
if(Form1->CheckBox3->Checked == true)  //если включен режим вывода в файл
{
write(handle, bufrd, strlen(bufrd)); //записать в файл данные из приёмного буфера
}  memset(bufrd, 0, BUFSIZE);   //очистить буфер(чтобы данные не накладывались)
}
//---------------------------------------------------------------------------
```

```
//---------------------------------------------------------------------------
//поток для записи последовательности байтов из буфера вCOM-порт
class WriteThread : public TThread
{
private:  void __fastcall Printing(); //вывод состояния на экран
protected: void __fastcall Execute();  //основная функция потока
public: __fastcall WriteThread(bool CreateSuspended);  //конструктор потока
};
//.............................. Поток WriteThead .............................
//---------------------------------------------------------------------------
WriteThread *writer;  //объект потокаWriteThread
//---------------------------------------------------------------------------
//конструктор потока WriteThread, по умолчанию пустой
__fastcall WriteThread::WriteThread(bool CreateSuspended) : TThread(CreateSuspended)
{}
//---------------------------------------------------------------------------
//главная функция потока, выполняет передачу байтов из буфера вCOM-порт
void __fastcall WriteThread::Execute()
{
DWORD temp, signal;  //temp - переменная-заглушка
overlappedwr.hEvent = CreateEvent(NULL, true, true, NULL);   //создать событие
WriteFile(COMport, bufwr, strlen(bufwr), &temp, &overlappedwr);  //записать байты
в порт(перекрываемая операция!)
```

```
signal = WaitForSingleObject(overlappedwr.hEvent, INFINITE);   //приостановить поток,
 пока не завершится перекрываемая операцияWriteFile
if((signal == WAIT_OBJECT_0) && (GetOverlappedResult(COMport,
&overlappedwr, &temp, true))) fl = true; //если
операция завершилась успешно, установить соответствующий флажок
else fl = false;
Synchronize(Printing);  //вывести состояние операции в строке состояния
CloseHandle(overlappedwr.hEvent);  //перед выходом из потока закрыть объект-событие
}
//---------------------------------------------------------------------------
//вывод состояния передачи данных на экран
void __fastcall WriteThread::Printing()
{
if(!fl)  //проверяем состояние флажка
{
Form1->StatusBar1->Panels->Items[0]->Text = "Ошибка передачи";
return;
}
Form1->StatusBar1->Panels->Items[0]->Text = "Передача прошла успешно";
}
```

```
writer->Resume();

writer->Suspend();
```

```
//функция закрытия порта
void COMClose()
{
//если поток записи существует, подать ему команду на завершение и запустить его, чтобы он выполнил завершение;
if(writer) //проверкаif(writer) обязательна, иначе возникают ошибки;
{
writer->Terminate();
writer->Resume();
}
if(reader) reader->Terminate();  //если поток чтения работает, завершить его; проверка if(reader) обязательна, иначе возникают ошибки
CloseHandle(COMport);  //закрыть порт
COMport=0;  //обнулить переменную для дескриптора порта
close(handle);  //закрыть файл для записи принимаемых данных
handle=0;   //обнулить переменную для дескриптора файла
}
```

```
HANDLE reader;  //дескриптор потока чтения из порта
DWORD WINAPI ReadThread(LPVOID);
//---------------------------------------------------------------------------
//..............................  потокReadThead  ..............................
//---------------------------------------------------------------------------
void ReadPrinting(void);
//---------------------------------------------------------------------------
//главная функция потока, реализует приём байтов изCOM-порта
DWORD WINAPI ReadThread(LPVOID)
{
COMSTAT comstat;   //структура текущего состояния порта, в данной программе используется для определения количества принятых в порт байтов
DWORD btr, temp, mask, signal;  //переменная temp используется в качестве заглушки
overlapped.hEvent = CreateEvent(NULL, true, true, NULL);  //создать сигнальный объект-событие для асинхронных операций
SetCommMask(COMport, EV_RXCHAR);   //установить маску на срабатывание по событию приёма байта в порт
while(1)  //пока поток не будет прерван, выполняем цикл
{
WaitCommEvent(COMport, &mask, &overlapped);  //ожидать события приёма байта(это и есть перекрываемая операция)
signal = WaitForSingleObject(overlapped.hEvent, INFINITE);  //приостановить поток до прихода байта
```

```
if(signal == WAIT_OBJECT_0)      //если событие прихода байта произошло
{
if(GetOverlappedResult(COMport, &overlapped, &temp, true)) //проверяем, успешно ли
завершилась перекрываемая операцияWaitCommEvent
if((mask & EV_RXCHAR)!=0)      //если произошло именно событие прихода байта
{  ClearCommError(COMport, &temp, &comstat);   //нужно заполнить структуру COMSTAT
btr = comstat.cbInQue;  //и получить из неё количество принятых байтов
if(btr)    //если действительно есть байты для чтения
{
ReadFile(COMport, bufrd, btr, &temp, &overlapped);  //прочитать байты из порта в
буфер программы
counter+=btr;  //увеличиваем счётчик байтов
ReadPrinting();   //вызываем функцию для
 вывода данных на экран и в файл
} } } }
//---------------------------------------------------------------------------
//выводим принятые байты на экран и в файл(если включено)
void ReadPrinting()
{
Form1->Memo1->Lines->Add((char*)bufrd);   //выводим принятую строку вMemo
Form1->StatusBar1->Panels->Items[2]->Text = "Всего принято" + IntToStr(counter) + " байт"; //
строке состояния
if(Form1->CheckBox3->Checked == true)  //если включен режим вывода в файл
{
write(handle, bufrd, strlen(bufrd)); //записать в файл данные из приёмного буфера
}  memset(bufrd, 0, BUFSIZE);   //очистить буфер(чтобы данные не накладывались друг на д
}
```

```
HANDLE writer;  //дескриптор потока записи в порт
DWORD WINAPI WriteThread(LPVOID);
//главная функция потока, выполняет передачу байтов из буфера в COM-порт
DWORD WINAPI WriteThread(LPVOID)
{
DWORD temp, signal;  //temp - переменная-заглушка
overlappedwr.hEvent = CreateEvent(NULL, true, true, NULL);   //создать событие
while(1)
{WriteFile(COMport, bufwr, strlen(bufwr), &temp, &overlappedwr);  //записать байты в
порт(перекрываемая
операция!)
signal = WaitForSingleObject(overlappedwr.hEvent, INFINITE);   //приостановить
поток, пока не завершится перекрываемая операцияWriteFile
if((signal == WAIT_OBJECT_0) && (GetOverlappedResult(COMport, &overlappedwr,
&temp, true)))  //если операция завершилась успешно
{
Form1->StatusBar1->Panels->Items[0]->Text = "Передача прошла успешно";
//вывести сообщение об этом в строке состояния
}
else {Form1->StatusBar1->Panels->Items[0]->Text = "Ошибка передачи";}  //иначе
вывести в строке состояния сообщение об ошибке
SuspendThread(writer);
}
}
```

ResumeThread()

SuspendThread()

ResumeThread(writer); //активировать поток записи данных в порт

```
//функция закрытия порта
void COMClose()
{
if(writer)  //если поток записи работает, завершить его; проверка if(writer) обязательна, иначе возникают ошибки
{TerminateThread(writer,0);
CloseHandle(overlappedwr.hEvent);  //нужно закрыть объект-событие
CloseHandle(writer);
}
if(reader)   //если поток чтения работает, завершить его; проверкаif(reader) обязательна, иначе возникают ошибки
{TerminateThread(reader,0);
CloseHandle(overlapped.hEvent);  //нужно закрыть объект-событие
CloseHandle(reader);
}
CloseHandle(COMport);  //закрыть порт
COMport=0;   //обнулить переменную для дескриптора порта
close(handle);   //закрыть файл, в который велась запись принимаемых данных
handle=0;    //обнулить переменную для дескриптора файла
}
```