Практическое задание №2

Программирование интерфейса RS-232C средствами Win32 API

Синхронный режим: Блокировка процессов и таймауты

1. Исследование работы функции чтения данных из последовательного порта

Показать, что операция чтения данных из последовательного порта в синхронном режиме может приводить к блокировке процесса.

- Изучить исходные коды GUI приложения работы с коммуникационным портом COMAPIc.
- Изменить параметры коммуникационного обмена в программе в соответствии с вариантом задания. Откомпилировать приложение.
- Запустить получившийся вариант COMAPIc.
- В качестве коммуникационного партнера использовать приложение Advanced Serial Port Terminal (Пуск►Программы (Все программы) ►Eltima Software►Advanced Serial Port Terminal). Запустить его, создать новую сессию, выбрать порт COM-порт и установить его параметры в соответствии с вариантом задания, открыть порт и установить режим Edit View.
- В COMAPIc открыть порт, считать данные, нажав кнопку «Read port». Данные передавать из Advanced Serial Port Terminal (количество считываемых данных определить на основании анализа исходных кодов COMAPIc).

ВНИМАНИЕ!

Аварийное завершение работы приложения, не освободившего системные ресурсы (COM-порт), сделает невозможным обращение к ним до перезагрузки операционной системы Microsoft Windows®

- Изменить приложение COMAPIc таким образом, чтобы операция чтения не блокировалась больше 10 с.

2. Исследование работы функции записи данных в последовательный порт

Показать, что операция записи данных в порт функцией WriteFile может приводить к блокировке процесса.

- Установить в настройках порта COMAPIc и Advanced Serial Port Terminal режим программного управления потоком (Xon/Xoff) и установить соединение.
- Из приложения COMAPIc передать данные и наблюдать их приём в Advanced Serial Port Terminal.
- Из Advanced Serial Port Terminal передать символ Xoff (13_{16}) и повторить предыдущий пункт.
- Из Advanced Serial Port Terminal передать символ Xon (11_{16}) и наблюдать происходящее в Advanced Serial Port Terminal и COMAPIc.
- Изменить приложение COMAPIc таким образом, чтобы операция записи не блокировалась больше 2 с, и при невозможности передать весь объем данных, генерировалось сообщение об этом.

Примечание

На компьютере должен быть установлен виртуальный NULL-модем, соединяющий виртуальные порты COM3 и COM4.

Вариант	Порт		Настройки портов
	Advanced Serial Port	COMAPIc	
1	COM3	COM4	38400/8o1
2	COM4	COM3	19200/8n2
3	COM4	COM3	9600/8s1
4	COM3	COM4	9600/8e1
5	COM3	COM4	38400/8o1
6	COM4	COM3	4800/8m1
7	COM4	COM3	4800/8e1
8	COM3	COM4	38400/8m1
9	COM4	COM3	9600/8o1
10	COM4	COM3	9600/8e1
11	COM3	COM4	38400/8n1
12	COM4	COM3	19200/8o1
все	Управление потоком приёма-передачи – отсутствует		

Программирование интерфейса RS-232C средствами Win32 API

Часть II – Синхронный режим

Открытие порта

При открытии коммуникационного порта в синхронном режиме параметр **dwFlagsAndAttributes** функции **CreateFile** должен быть установлен в **0**, то есть фрагмент кода программы, отвечающий за открытие порта COM2, будет следующим:

```
HANDLE hCom;

hCom = CreateFile("COM2",
                  GENERIC_READ|GENERIC_WRITE,
                  0,
                  NULL,
                  OPEN_EXISTING,
                  0,
                  NULL);
```

Запись в порт

Операция записи в порт в синхронном режиме (как, впрочем, и в асинхронном) осуществляется вызовом функции и **WriteFile**. Аргументы этой функции описаны ранее, и последним параметром должен быть **NULL**:

```
char Buffer[] = "Hello, World!";
DWORD dwBytes = 0;

WriteFile(hCom, Buffer, sizeof(Buffer), &dwBytes, NULL);
```

Чтение из порта

Операция чтения данных из последовательного порта в синхронном режиме (также как и в асинхронном) осуществляется вызовом функции и **ReadFile**. Аргументы этой функции описаны ранее, и последним параметром должен быть **NULL**:

```
unsigned char Buffer[1024];
DWORD dwBytes = 0;

ReadFile(hCom, Buffer, 1024, &dwBytes, NULL);
```

Однако особенностью синхронного режима является то, что функции чтения и записи блокирует выполнение процесса до тех пор, пока все **nNumOfBytesToRead** байт не будут прочитаны (или же **nNumOfBytesToWrite** байт не будут переданы). В ряде случаев, например при управлении потоком приёма-передачи, передача может быть заблокирована и процесс никогда не завершится, или же будет тянуться продолжительное время.

Разумным выходом из сложившейся ситуации является использование таймаутов (или же отказ от синхронного режима).

Таймауты

Параметры временных задержек при приеме и передаче определяются управляющей структурой **COMMTIMEOUTS**. Значения, задаваемые полями этой структуры, оказывают большое влияние на работу функций чтения/записи.

```
typedef struct _COMMTIMEOUTS {
      DWORD ReadIntervalTimeout;
      DWORD ReadTotalTimeoutMultiplier;
      DWORD ReadTotalTimeoutConstant;
      DWORD WriteTotalTimeoutMultiplier;
      DWORD WriteTotalTimeoutConstant;
} COMMTIMEOUTS, *LPCOMMTIMEOUTS;
```

Поля структуры **COMMTIMEOUTS** имеют следующие значения:

ReadIntervalTimeout

Максимальное время, в миллисекундах, допустимое между двумя последовательными символами, считываемыми с коммуникационной линии. Во время операции чтения временной период начинает отсчитываться с момента приема первого символа.

Нулевое значение данного поля означает, что данный тайм-аут не используется.

Значение **MAXDWORD**, вместе с нулевыми значениями полей **ReadTotalTimeoutConstant** и **ReadTotalTimeoutMultiplier**, означает немедленный возврат из операции чтения с передачей уже принятого символа, даже если ни одного символа не было получено из линии.

ReadTotalTimeoutMultiplier

Задает множитель, в миллисекундах, используемый для вычисления общего тайм-аута операции чтения.

Для каждой операции чтения данное значение умножается на количество запрошенных для чтения символов.

ReadTotalTimeoutConstant	Задает константу, в миллисекундах, используемую для вычисления общего тайм-аута операции чтения. Для каждой операции чтения данное значение прибавляется к результату умножения **ReadTotalTimeoutMultiplier** на количество запрошенных для чтения символов. Нулевое значение полей **ReadTotalTimeoutMultiplier** и **ReadTotalTimeoutConstant** означает, что общий тайм-аут для операции чтения не используется.
WriteTotalTimeoutMultiplier	Задает множитель, в миллисекундах, используемый для вычисления общего тайм-аута операции записи. Для каждой операции записи данное значение умножается на количество записываемых символов.
WriteTotalTimeoutConstant	Задает константу, в миллисекундах, используемую для вычисления общего тайм-аута операции записи. Для каждой операции записи данное значение прибавляется к результату умножения **WriteTotalTimeoutMultiplier** на количество записываемых символов. Нулевое значение полей **WriteTotalTimeoutMultiplier** и **WriteTotalTimeoutConstant** означает, что общий тайм-аут для операции записи не используется.

Должным образом сконфигурировав структуру **COMMTIMEOUTS**, можно избежать блокировок процессов приложения, сохранив при этом простоту программирования работы последовательного порта в синхронном режиме.

Пусть мы считываем из порта 200 символов со скоростью 19 200 бит/с, и используется кодирование 8 бит на символ, дополнение до четности и один стоповый бит, то на один символ в физической линии приходится 11 бит (включая стартовый бит). Значит, 200 символов на указанной скорости будут приниматься в течении

$$200 \times \frac{11}{19200} = 0{,}11458(3)\ \text{с},$$

или примерно 114,6 мс, при условии нулевого интервала между приемом последовательных символов (стартовый бит очередного символа будет передаваться сразу же после стоп-бита предыдущего). Если в процессе чтения прошло более 115 мс, то можно предположить, что произошла ошибка в работе внешнего устройства и операция чтения может быть прекращена. Для организации такого решения нужно установить значение поля **ReadTotalTimeoutMultiplier** структуры **COMMTIMEOUTS**:

```
COMMTIMEOUTS ct = {0};
ct.ReadTotalTimeoutMultiplier = 115;
```

Однако, во время передачи могут происходить различного рода задержки. Если, скажем, интервал между символами составляет треть времени передачи одного символа, то есть

$$\frac{1}{3}\times\frac{11}{19200}\approx 0.2\text{ мс},$$

то в целом время приёма увеличится на

$$199\times 0.2 = 39.8\text{ мс},$$

а сама передача будет осуществляться в течении

$$114.6 + 39.8 = 154.4\text{ мс}.$$

Тогда для приведенного выше примера необходимо установить ещё и значение поля **ReadTotalTimeoutConstant**:

```
ct.ReadTotalTimeoutConstant = 40;
```

В этом случае операция чтения данных из последовательного порта завершится за

$$115 + 40 = 155\text{ мс}.$$

Таким образом, формула для вычисления общего таймаута операции чтения, выглядит следующим образом:
nNumOfBytesToRead × ReadTotalTimeoutMultiplier + ReadTotalTimeoutConstant,

где **nNumOfBytesToRead** – число символов, запрошенных для операции чтения.

В приведённом примере величина общего таймаута зависит от числа считываемых функцией **ReadFile** символов **nNumOfBytesToRead**, а значит, если в реальности будет передано меньшее количество символов, нежели 200, то в течение всего оставшегося времени принимающая сторона будет воспринимать входящие данные как продолжение предыдущего пакета. Поэтому необходимо не забывать про **ReadIntervalTimeout**, которое определяет максимальное время ожидания между символами. Если интервал между двумя последовательными символами превысит заданное значение (правильнее сказать, если период следования символов будет больше чем **ReadIntervalTimeout**), операция чтения завершится, и все данные, накопленные в буфере, передадутся в программу.

Пусть **ReadTotalTimeoutMultiplier = 2**, **ReadTotalTimeoutConstant = 1**, **ReadIntervalTimeout = 1**, считывается 250 символов. Если операция чтения завершится за $250\times 2+1 = 501$ мс, то будет считано всё сообщение. Если операция чтения не завершится за 501 мс, то она всё равно будет завершена, однако функцией **ReadFile** будут возвращены только те символы, приём которых завершился до истечения таймаута. Остальные символы могут быть получены во время следующей операции чтения. Если между стартовыми битами двух последовательных символов пройдёт более 1 мс, то операция чтения так же будет завершена, и функцией **ReadFile** также буду возвращены только те символы, приём которых завершился до истечения таймаута.

Если поля **ReadIntervalTimeout** и **ReadTotalTimeoutMultiplier** установлены в **MAXDWORD**, а **ReadTotalTimeoutConstant** больше нуля и меньше **MAXDWORD**, то выполнение операции чтения подчиняется следующим правилам:

- если в буфере есть символы, то чтение немедленно завершается и возвращается символ из буфера;
- если в буфере нет символов, то операция чтения будет ожидать появления любого символа, после чего она немедленно завершится;
- если в течении времени, заданного полем **ReadTotalTimeoutConstant**, не будет принято ни одного символа, то операция чтения завершится по таймауту.

Внешнее устройство может использовать программное или аппаратное управление потоком приёма-передачи. И в этом случае, если оно сигнализирует о необходимости приостановить передачу, то до тех пор, пока не будет восстановлено разрешение на передачу, передающая программа будет находиться в состоянии ожидания. Поэтому дабы обеспечить работоспособность приложения (исключить зависание программы) необходимо контролировать и таймауты записи. Общий таймаут записи определяется аналогично общему таймауту чтения.

Сформировать структуру **COMMTIMEOUTS** можно различными методами. Разумеется, можно установить все поля вручную. Или же воспользоваться функций считывания установленных в системе значений **GetCommTimeouts**:

```
BOOL GetCommTimeouts(
      HANDLE          hFile,
      LPCOMMTIMEOUTS lpCommTimeouts
);
```

hFile	Дескриптор открытого коммуникационного порта, полученный функцией **CreateFile**.
lpCommTimeouts	Указатель на структуру **COMMTIMEOUTS**. Для **COMMTIMEOUTS** предварительно должен быть выделен блок памяти.

```
      COMMTIMEOUTS ct = {0};

      if (!GetCommTimeouts(hCom, &ct))
      {
          // ошибка получения параметров COMMTIMEOUTS
      }
      else
      {
          // структура COMMTIMEOUTS инициализирована,
          // можно устанавливать свои значения

          ct.ReadIntervalTimeout = 100;
      }
```

Также можно воспользоваться функцией **BuildCommDCBAndTimeouts**, которая позволяет заполнить не только структуру **DCB**, но и структуру **COMMTIMEOUTS**. Её прототип:

```
BOOL BuildCommDCBAndTimeouts(
      LPCTSTR         lpDef,
      LPDCB           lpDCB,
      LPCOMMTIMEOUTS lpCommTimeouts
);
```

lpDef — Указатель на строку с конфигурационной информацией в формате команды **mode** (см. выше).

lpDCB — Указатель на структуру **DCB**. При этом структура должна быть уже создана, и поле **DCBlength** должно содержать корректное значение.

lpCommTimeouts — Указатель на структуру **COMMTIMEOUTS**. Для **COMMTIMEOUTS** предварительно должен быть выделен блок памяти.

То есть эта функция аналогична **BuildCommDCB**, но имеет дополнительный (третий) параметр – указатель на структуру **COMMTIMEOUTS**. Конфигурационная строка при вызове этой функции должна задаваться в новом формате. Если эта строка содержит подстроку **"TO=ON"**, то устанавливаются общие таймауты для операций чтения и записи. Эти значения устанавливаются на основе информации о скорости передачи и формате посылки. Если конфигурационная строка содержит подстроку **"TO=OFF"**, то устанавливается режим работы без таймаутов. Если конфигурационная строка не содержит подстроки **"TO=…"** или эта подстрока имеет неверное значение, то указатель на структуру **COMMTIMEOUTS** просто игнорируется, а вызов функции **BuilCommDCBAndTimeouts** оказывается идентичным вызовом функции **BuildCommDCB**.

За установку параметров отвечает функция SetCommTimeouts:

```
BOOL SetCommTimeouts(
      HANDLE          hFile,
      LPCOMMTIMEOUTS lpCommTimeouts
);
```

hFile — Дескриптор открытого коммуникационного порта, полученный функцией **CreateFile**.

lpCommTimeouts — Указатель на сконфигурированную структуру **COMMTIMEOUTS**.

Установку таймаутов можно производить как до установки параметров порта, так и после – последовательность вызова функций **SetCommState** и **SetCommTimeouts** не имеет никакого значения.

```
if(!SetCommTimeouts(hCom, &ct))
{
  // ошибка установки параметров COMMTIMEOUTS
}
else
{
  // параметры COMMTIMEOUTS установлены
}
```