Лекция 13. Обмен данных между процессами

В лекции использован материал книги Побегайло А.П. Системное программирование в Windows и книги Албахари Д. и Албахари Б. C# 3.0 Справочник.

13.1Способы обмена данных между процессами

Под обменом данных между процессами понимается пересылка данных от одной нити к другой нити. При этом нити должны находиться в различных процессах (пересылку данных между нитями одного процесса мы рассматривали в модуле 2).

Нить, посылающая данные, называется отправителем, а нить, получающая данные, называется получателем или адресантом. Обмен данными между нитями реализуется с помощью специальных средств операционной системы – канала передачи данных.

Структурно канал передачи данных изображен на рисунке 13.1.

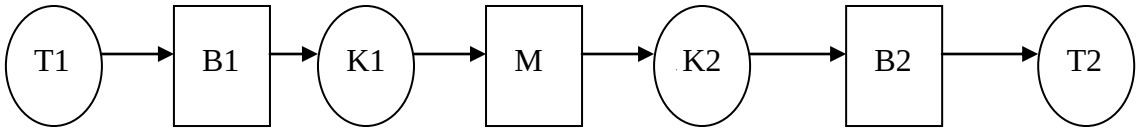

Рисунок 13.1 Структурная схема канала передачи данных

T1,T2 – нити различных процессов;
B1,B2 – буферы;
K1,K2 – потоки ядра операционной системы;
M – общая память.

При разработке платформы .NET очень большое внимание было уделено вопросам пересылке информации (одной из задач платформы является упрощения процессов программирования работы в сетях) и в технологии передачи данных появились новые понятия или изменилось назначение некоторых старых.

Определяющим является понятие потока в языке C#.

Поток в языке C# является стандартным классом платформы .NET и представлен набором методов для чтения, записи и позиционирования данных.

Потоки работают с данными последовательно, выполняя операции либо над байтами, либо над блоками небольшого размера. Поток «соединяет» источник и приемник данных. В качестве источника и приемника данных могут быть, например, файлы, процессы (нити) или сетевое соединение компьютера и т.д.

Различают потоки с резервным хранилищем и потоки декораторы.

Потоки с резервным хранилищем, например, FileStream, MemoryStream, NetworkStream это потоки, в которых данные только передаются (говорят, что передаются «сырые» данные) и поэтому для каждого типа данных необходимо использовать соответствующий поток (класс).

Потоки декораторы, например, DeflateStream или GZipStream это потоки которые обеспечивают некоторые преобразования данных, например их сжатие или шифрование.

Существует понятие адаптеров потоков, которые являются специальными классами, обеспечивающими поток специализированными методами преобразования типов данных, например, StreamReader и StreamWriter обеспечивают потоку работу с текстами (обычно потоки работают с байтами данных).

13.2 Связи между процессами

Перед передачей данных между процессами необходимо установить связь между этими процессами.

Кто звонил по номеру телефона, к которому подключен модем, тот знает, что модем пытается установить связь с вами посредством повторяющихся сигналов.

Связь между процессами может устанавливаться как на физическом (аппаратном), так и на логическом (программном) уровнях.

В зависимости от направления передачи данных различают полудуплексную связь – данные передаются только в одном направлении, и дуплексную связь – данные передаются в обоих направлениях.

Обмен данными между параллельными процессами возможен двумя способами: потоком и сообщением.

Если данные передаются непрерывной последовательностью байтов, то такой способ передачи данных называется передача данных потоком. При этом возможна передача данных непосредственно из буфера одной нити в буфер другой нити без потоков ядра операционной системы и общей памяти.

Если данные передаются блоками байтов, то такой блок данных называется сообщением, а передача данных называется передачей данных сообщениями.

Очень часто связь между процессами характеризуется топологией связей – возможной структурой связей между процессами-отправителями и процессами-получателями данных.

С точки зрения топологии связей, например, для полудуплексного вида связей различают следующие виды связей:

1<-->1 – между собой связаны два процесса;

1<-->N – один процесс связан с N процессами;

N<-->1 – каждый из N процессов связан с одним процессом;

N<-->M – каждый из N процессов связан с каждым из M процессов.

При разработке системы обмена данными между процессами первоначально определяется топология связей и направления передачи данных по этим связям. Это позволяет организовать связь на аппаратном или физическом уровне. После этого в программах реализуются выбранные связи между процессами с помощью специальных функций операционной системы, которые предназначены для установки связи между процессами. Эти функции совместно с функциями, предназначенными для обмена данными между процессами, находятся в системе передачи данных, которая является частью ядра операционной системы.

Объект ядра операционной системы, обеспечивающий передачу данных между процессами, выполняющимися на одном компьютере, называется «анонимным каналом».

Объект ядра операционной системы, обеспечивающий передачу данных между процессами, выполняющимися на компьютерах в локальной сети, называется «именованным каналом».

Объект ядра операционной системы, обеспечивающий передачу сообщений от процессов-клиентов к процессам-серверам, выполняющимися на компьютерах в локальной сети, называется «почтовым ящиком».

13.3 Передача сообщений

Обмен сообщениями между процессами выполняется при помощи двух функций, которые условно будем называть send (послать сообщение) и receive (получить сообщение).

Само сообщение состоит из двух частей – заголовка и тела сообщения.

В заголовок включается следующая служебная информация:

- тип сообщения;
- имя адресата сообщения;
- имя отправителя сообщения;
- контрольная информация, обычно длина сообщения и контрольная сумма.

Тело сообщения содержит данные пересылаемого сообщения.

Условно будем считать, что функции обмена имеют следующий вид:

send(Process A, сообщение); - послать сообщение процессу A и

receive(Process B, сообщение); - получить сообщение от процессора B.

При передачи может быть использована прямая или косвенная (по адресу) адресация.

При прямой адресации процессов в функциях send и receive явно указываются имена процессов отправителя и получателя.

При косвенной адресации в функциях send и receive указываются не имена процессов отправителя и получателя, а имена связей, по которым необходимо передавать сообщения.

Адресация процессов может быть симметричной и асимметричной.

Если при обмене сообщениями между процессами используется только прямая или только косвенная адресация, то такая адресация процессов называется симметричной.

Если при обмене сообщениями между процессами используется как прямая, так и косвенная адресация, то такая адресация процессов называется асимметричной.

Асимметричная адресация часто используется в системах «клиент-сервер». Как правило «клиент» знает адрес «сервера» и посылает ему сообщения с использованием прямой адресации процесса. «Сервер» настраивается на канал связи, а не на конкретного «клиента» (говорят, что «сервер слушает» канал связи) – для этого используется косвенная адресация процессов.

Набор правил, по которым устанавливаются связи и передаются данные между процессами, называется протоколом.

По этим правилам, например, нельзя начинать передачу данных, не установив связь между отправителем и адресантом и т.д.

Передача данных возможна синхронным или асинхронным способами.

Если процесс, отправивший сообщение блокируется до получения этого сообщения адресантом (точнее до получения ответа от адресанта, что сообщение получено), то такая передача данных называется синхронной. Обычно отправитель сообщения должен получить информацию о готовности адресанта принять следующее сообщение.

Асинхронный способ передачи сообщений не требует подтверждения о приеме каждого сообщения. Такой способ передачи применяется, когда быстродействие получателя сообщений во много раз больше отправителя сообщений и очень низкая вероятность появления помех. В таких системах после установки связи сообщения пересылаются непрерывным потоком.

При синхронном способе передачи есть возможность проверить каждое сообщение и в случае обнаружения ошибки потребовать повторной передачи сообщения.

В системах с обнаружением ошибок используют избыточные коды. Например, для передачи одного бита информации используется два двоичных разряда: 0 заменяется кодом 00, а 1 – кодом 11. Коды 01 и 10 являются запрещенными. И если получатель сообщения в процессе приема данных получает код 01 или 10, то он определяет, что на сообщение воздействовала помеха и сообщение необходимо повторить.

Существуют системы с автоматическим исправлением ошибок при приеме сообщений. В таких системах для передачи одного бита информации используется трех или более разрядные двоичные коды. Например, для передачи 0 используется 000, а для передачи 1 – 111. Все остальные трех разрядные коды являются запрещенными. В случае приема кодов 011, 101 или 110 система с исправлением ошибки может предположить, что передавался код 111, но на него воздействовала одиночная помеха и принять полученных код за значение 111. Аналогично запрещенные коды 001, 010 и 100 могут быть приняты за код 000. В таких сложных системах

просчитывается частота появления одиночной ошибки, контрольные суммы сообщений и другие известные средства обнаружения и исправления ошибок – автоматический переход на дублирование сообщений и т.д.

Контрольная сумма может получаться различными алгоритмами. Наиболее простой алгоритм основан на суммировании всех передаваемых байтов сообщений с отбрасыванием переполнения. Приемник сообщения также суммирует все байты принимаемого сообщения, и полученное значение сравнивается с полученным значением контрольной суммы. Если они различаются, то приемник требует повторить передачу сообщения.

Синхронный обмен данными с использованием прямой адресации процессов называется рандеву (встреча).

Согласование работы различных процессов, имеющих разные скоростные характеристики, осуществляется с помощью буфера.

Буфер характеризуется «вместимостью связи между процессами» - количество сообщений, которое может одновременно пересылаться по этой связи.

Различают три типа буферизации:

- нулевой вместимости связи (нет буфера). В этом случае возможен только синхронный обмен данными между процессами.

- ограниченная вместимость связи (ограниченный буфер). В этом случае если буфер полон, то отправитель сообщения ждет очистки буфера хотя бы от одного сообщения.

- неограниченная вместимость связи (неограниченный буфер). В этом случае отправитель никогда не ждет при отправки сообщений.

Возможности буфера тесно связаны с синхронизацией передачи данных и должны учитываться при разработке систем обмена данными между процессами.

13.4 Обмен данными между процессами с помощью файла

Простейший способ передачи данных между процессами (нитями процессов) можно организовать, если общую память (см. рисунок 10.1) заменить файлом. Работа нитей с потоками (в нашем случае это потоки с резервным хранилищем) обычно не является безопасной. Действительно, если нитям разрешить писать и читать данные в потоке одновременно, то нет гарантии, что не возникнет ошибки. Поэтому работа с потоком нескольких процессов требует выполнение определенных правил. Например, при работе с потоком с резервным хранилищем, представленным файлом, необходимо, выполнив некоторое действие с файлом, его закрывать. Желательно не допускать одновременной работы с файлом нескольких процессов. Рекомендуется выполнять работу с файлом поочередно.

В качестве учебного примера рассмотрим работу двух процессов с одним текстовым файлом, через который первый процесс передает во второй процесс массив байтов, сформированный случайным образом. Второй процесс забирает содержимое файла в массив байтов, сортирует его и

возвращает в файл. Первый процесс забирает из файла отсортированный массив и печатает его. Задача реализована двумя программами – основной программой и дополнительной программой.

Код основной программы:

```
using System;
using System.IO;
using System.Diagnostics;

namespace ConsoleApplication1
{
 class Program
 {
  static void Main()
  {
   byte[] a = new byte[10];
   int k = 0;
   string buf;
   while (k < 6)
   {
    Console.WriteLine("1 - Создать файл ");
    Console.WriteLine("2 - Создать и напечатать массив из 10
чисел");
    Console.WriteLine("3 - Сортировка файла ");
    Console.WriteLine("4 - Запись массива в файл");
    Console.WriteLine("5 - Прочитать массив из файла и
напечатать его");
    Console.WriteLine("6 - Выход из программы");
    Console.WriteLine("Введите пункт меню программы");
    buf = Console.ReadLine();
    k = Convert.ToInt32(buf);
    switch (k)
    {
     case 1:
      using (Stream fmi = new FileStream("FiMaIn.txt",
FileMode.Create))
     { fmi.Write(a, 0, a.Length); fmi.Close(); }; break;
     case 2: SozdMas(a); break;
     case 3: Process.Start("ConsoleApplication2"); break;
     case 4:
      using (Stream fmi = new FileStream("FiMaIn.txt",
FileMode.Open))
     { fmi.Write(a, 0, a.Length); fmi.Close(); }; break;
     case 5:
      using (Stream fmi = new FileStream("FiMaIn.txt",
FileMode.Open))
     { fmi.Read(a, 0, a.Length); fmi.Close(); }; ReadMas(a);
break;
     default: break;
    }
   }
  }
  public static void SozdMas(byte[] ma)
```

```
   {
    Random rnd = new Random();
    Console.WriteLine("Массив создан !!");
    for (int i = 0; i < 10; i++)
    {
     ma[i] = (byte)(rnd.Next() % 101);
     Console.Write(ma[i]+"\t");
    }
    Console.WriteLine();
   }
   public static void ReadMas(byte[] ma)
   {
    for (int i = 0; i < 10; i++)
     Console.Write(ma[i] + "\t");
    Console.WriteLine();
   }
  }
}
```

Дополнительная программа сортировки массива:

```
using System;
using System.IO;
using System.Diagnostics;

namespace ConsoleApplication2
{
 class Program
 {
  static void Main()
  {
   byte b;
   byte[] a = new byte[10];
   Console.WriteLine("Cortirovka nacinaetcj");
   Console.ReadLine();
   using (Stream fmi = new FileStream("FiMaIn.txt",
FileMode.Open))
   {
    fmi.Read(a, 0, a.Length);
    for (int i = 0; i < 10; i++)
     Console.Write(a[i] + "\t");
    Console.WriteLine();
    for (int i = 0; i < 9; i++)
     for (int j = i+1; j < 10; j++)
      if(a[i]<a[j])
      {
       b = a[i]; a[i] = a[j]; a[j] = b;
      }
    for (int i = 0; i < 10; i++)
     Console.Write(a[i] + "\t");
    Console.WriteLine();
    fmi.Close();
   }
   using (Stream fmi = new FileStream("FiMaIn.txt",
FileMode.Create))
```

```
    {
     fmi.Write(a, 0, a.Length);
     fmi.Close();
    }
    Console.WriteLine("Cortirovka zakoncena");
    Console.ReadLine();
   }
  }
}
```

Работа программ:

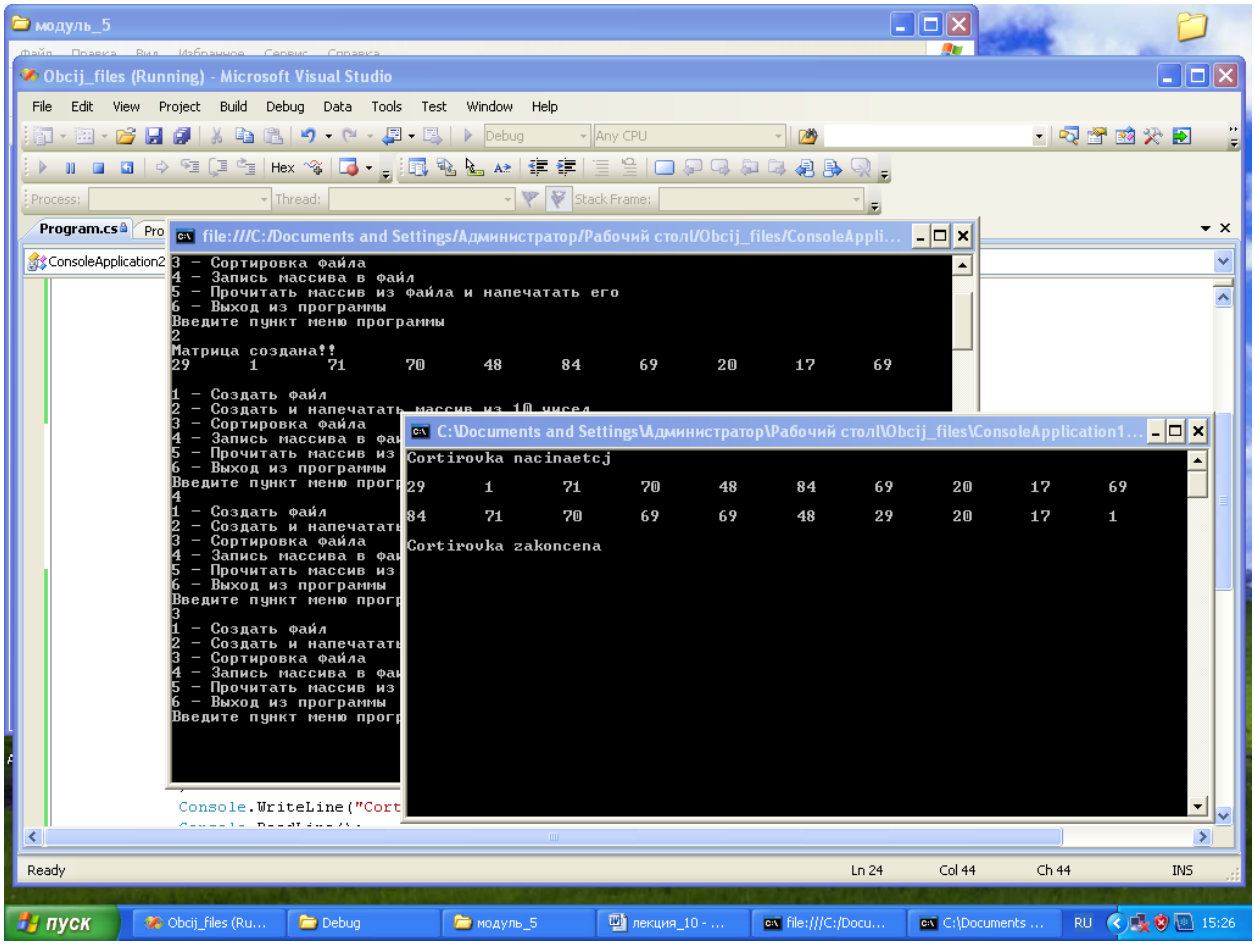

Рисунок 13.2 Обмен данными между процессами с помощью файла

В примерах, перед записью данных в файл они как бы создаются заново. Это нужно, чтобы выполнить очистку файла. Естественно, эту операцию можно выполнить и с помощью метода очистки.