**Тема:     Обмен информацией между процессами с помощью каналов и почтовых ящиков**

**Лабораторная работа. Обмен информацией между процессами с помощью каналов и почтовых ящиков.**

**Цель работы**: изучить возможности применения механизма обмена данными между процессами в операционной системе Windows на основе каналов и почтовых ящиков.

**Темы для предварительной проработки**: процессы и нити в операционных системах, жизненный цикл процесса, средства взаимодействия процессов IPC, интерфейс прикладного программирования API WIN32.

**Теоретические сведения**

*Каналы.*

Через канал можно передавать данные только между двумя процессами. Один из процессов создает канал, другой открывает его. После этого оба процесса могут передавать данные через канал в одну или обе стороны.

*Именованные и анонимные каналы.*

Существуют две разновидности каналов Pipe – именованные (Named Pipes) и анонимные (Anonymous Pipes).

Именованным каналам при создании присваивается имя, которое доступно для других процессов. Зная имя какой-либо рабочей станции в сети, процесс может получить доступ к каналу, созданному на этой рабочей станции.

Анонимные каналы обычно используются для организации передачи данных между родительскими и дочерними процессами, запущенными на одной рабочей станции или на “отдельно стоящем” компьютере.

*Имена каналов.*

Имена каналов в общем случае имеют следующий вид:

**\\ИмяСервера\pipe\ИмяКанала**

Если процесс открывает канал, созданный на другой рабочей станции, он должен указать имя сервера. Если же процесс создает канал или открывает канал на своей рабочей станции, вместо имени указывается символ точки:

**\\.\pipe\ИмяКанала**

В любом случае процесс может создать канал только на той рабочей станции, где он запущен, поэтому при создании канала имя сервера никогда не указывается.

*Создание канала.*

Для создания именованных и анонимных каналов Pipes используются функции **CreatePipe** и **CreateNamedPipe**.

*Установка соединения с каналом со стороны сервера.*

После того как серверный процесс создал канал, он может перейти в режим соединения с клиентским процессом. Соединение со стороны сервера выполняется с помощью функции **ConnectNamedPipe**.

*Установка соединения с каналом со стороны клиента.*

Для создания канала клиентский процесс может воспользоваться функцией **CreateFile**.

*Отключение серверного процесса от клиентского процесса.*

Установив канал с клиентским процессом при помощи функции **ConnectNamedPipe**, серверный процесс может затем разорвать канал, вызвав для этого функцию **DisconnectNamedPipe**.

*Закрытие идентификатора канала.*

Если канал больше не нужен, после отключения от клиентского процесса серверный и клиентский процессы должны закрыть его идентификатор функцией **CloseHandle**.

*Запись и чтение данных в канале.*

Запись и чтение данных в открытом канале выполняется с помощью функциий **WriteFile** и **ReadFile**, аналогично записи и чтению в обычном файле.

Пример. Передача данных (синхронная) от программы-клиента программе-серверу с помощью именованного канала на одной рабочей станции.

Программа-сервер

#include <windows.h>

#include <stdio.h>

#include <conio.h>

BOOL   fConnected; // Флаг успешного создания канала

HANDLE hNamedPipe; // Идентификатор канала Pipe

LPSTR  lpszPipeName = "\\\\.\\pipe\\$MyFirstPipe"; // Имя создаваемого канала

char   szBuf[512]; // Буфер для приема данных из канала

DWORD  cbRead; // Количество байт данных, принятых через канал

int main()

{

// Создаем канал Pipe с именем lpszPipeName

hNamedPipe = CreateNamedPipe(

            lpszPipeName, PIPE\_ACCESS\_DUPLEX,

            PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT,

            PIPE\_UNLIMITED\_INSTANCES, 512, 512, 5000, NULL);

// Если возникла ошибка, завершаем работу приложения

if(hNamedPipe == INVALID\_HANDLE\_VALUE)

{

            printf("Error Pipe Creating!!!\n");

            getch();

            return 0;

}

// Ожидаем соединения со стороны клиента

fConnected = ConnectNamedPipe(hNamedPipe, NULL);

// Если возникла ошибка, завершаем работу приложения

if(!fConnected)

{

              printf("Error Pipe Connecting!!!\n");

              CloseHandle(hNamedPipe);

              getch();

              return 0;

}

// Получение данных из канала

if(ReadFile(hNamedPipe, szBuf, 512, &cbRead, NULL))

            printf("Received %d bytes: <%s>\n", cbRead, szBuf);

else printf("Error Data Transfer!!!\n");

CloseHandle(hNamedPipe);

getch();

return 0;

}

Программа-клиент

#include <windows.h>

#include <stdio.h>

#include <conio.h>

HANDLE hNamedPipe; // Идентификатор канала Pipe

DWORD  cbWritten; // Количество байт, переданных через канал

char   szBuf[256]; // Буфер для передачи данных

LPSTR  lpszPipeName = "\\\\.\\pipe\\$MyFirstPipe"; // Имя создаваемого канала

int main()

{

// Установка соединения с процессом PIPES

hNamedPipe = CreateFile(lpszPipeName, GENERIC\_READ | GENERIC\_WRITE,

                        0, NULL, OPEN\_EXISTING, 0, NULL);

// Если возникла ошибка, завершаем работу приложения

if(hNamedPipe == INVALID\_HANDLE\_VALUE)

{

              printf("Error Pipe Creating!!!\n");

              getch();

              return 0;

}

// Передача данных серверному процессу

strcpy(szBuf,"Test Pipe Connection");

if(!WriteFile(hNamedPipe, szBuf, strlen(szBuf) + 1, &cbWritten, NULL))

                        // Если произошла ошибка, выдаем сообщение

                        printf("Error Data Transfer!!!\n");

else printf("Transferred %d bytes: <%s>\n", cbWritten, szBuf);

// Закрываем идентификатор канала

CloseHandle(hNamedPipe);

getch();

return 0;

}

*Почтовые ящики.*

Почтовые ящики позволяют выполнять одностороннюю передачу данных от одного или нескольких клиентов к одному или нескольким серверам. Главная особенность почтовых ящиков заключается в том, что они, в отличие от каналов позволяют передавать данные в широковещательном режиме.

Это означает, что на компьютере или в сети могут работать несколько серверных процессов, способных получать сообщения через почтовые ящики. При этом один клиентский процесс может посылать сообщения сразу всем этим серверным процессам.

*Создание почтового ящика*

**Mailslot** создается серверным процессом с помощью специально предназначенной для этого функции **CreateMailslot**. После создания серверный процесс получает идентификатор **Mailslot**. Пользуясь этим идентификатором, сервер может читать сообщения, посылаемые в почтовый ящик клиентскими процессами. Однако сервер не может выполнять над **Mailslot** операцию записи, так как этот почтовый ящик предназначен только для односторонней передачи данных – от клиента к серверу.

Имя **Mailslot** задается аналогично имени канала **Pipe** (приведен синтаксис для создания почтового ящика на своей рабочей станции):

**\\.\mailslot\[Путь]ИмяПочтовогоЯщика**

Чтобы открыть **Mailslot**, созданный на другой рабочей станции в сети, строка имени канала должна иметь следующий вид:

**\\ИмяРабочейСтанции\mailslot\[Путь]ИмяПочтовогоЯщика**

Можно открыть канал для передачи сообщений всем рабочим станциям заданного домена. Для этого необходимо задать имя по следующему образцу:

**\\ИмяДомена\mailslot\[Путь]ИмяПочтовогоЯщика**

Для передачи сообщений одновременно всем рабочим станциям сети первичного домена имя задается следующим образом:

**\\\*\mailslot\[Путь]ИмяПочтовогоЯщика**

В последних двух случаях размер сообщений ограничивается 400 байтами.

Запись и чтение данных в почтовых ящиках осуществляется аналогично записи и чтению в каналах.

*Определение состояния Mailslot*

Серверный процесс может определить текущее состояние **Mailslot** по его идентификатору с помощью функции **GetMailslotInfo**.

Пример. Передача данных от программы-клиента программе-серверу с помощью почтового ящика на одной рабочей станции.

Программа-сервер

#include <windows.h>

#include <stdio.h>

#include <conio.h>

BOOL   fReturnCode; // Код возврата из функций

DWORD  cbMessages; // Размер сообщения в байтах

DWORD  cbMsgNumber; // Количество сообщений в Mailslot

HANDLE hMailslot; // Идентификатор Mailslot

LPSTR  lpszMailslotName = "\\\\.\\mailslot\\$MyFirstMailslot"; // Имя Mailslot

char   szBuf[512]; // Буфер для передачи данных через Mailslot

DWORD  cbRead; // Количество байт данных, принятых через Mailslot

int main()

{

// Создаем Mailslot, с именем lpszMailslotName

hMailslot = CreateMailslot(lpszMailslotName, 0,

            MAILSLOT\_WAIT\_FOREVER, NULL);

// Если возникла ошибка, завершаем работу приложения

if(hMailslot == INVALID\_HANDLE\_VALUE)

{

            printf("Error MailSlot Creating!!!\n");

            getch();

            return 0;

}

while (1)

{

            // Определяем состояние канала Mailslot

            fReturnCode = GetMailslotInfo(hMailslot, NULL,

                        &cbMessages, &cbMsgNumber, NULL);

            if(!fReturnCode)

            {

                        printf("Get MailSlotInfo Error!!!\n");

                        getch();

                        return 0;

            }

            // Если в Mailslot есть сообщения, читаем первое и выводим на экран

            if(cbMsgNumber != 0)

            {

                        if(ReadFile(hMailslot, szBuf, 512, &cbRead, NULL))

                                   printf("Received %d bytes: <%s>\n", cbRead, szBuf);

                        else printf("Error Data Transfer!!!\n");

                        break;

            }

}

getch();

CloseHandle(hMailslot);

return 0;

}

Программа-клиент

#include <windows.h>

#include <stdio.h>

#include <conio.h>

HANDLE hMailslot; // Идентификатор Mailslot

char   szMailslotName[256]; // Буфер для имени Mailslot

char   szBuf[512]; // Буфер для передачи данных через Mailslot

DWORD  cbWritten; // Количество байт, переданных через Mailslot

LPSTR  lpszMailslotName = "\\\\.\\mailslot\\$MyFirstMailslot"; // Имя Mailslot

int main()

{

// Создаем Mailslot

hMailslot = CreateFile(

    lpszMailslotName, GENERIC\_WRITE,

    FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

// Если возникла ошибка, завершаем работу приложения

if(hMailslot == INVALID\_HANDLE\_VALUE)

{

            printf("CreateFile Error!!!\n");

            getch();

    return 0;

}

// Посылка данных через Mailslot

strcpy(szBuf,"Test MailSlot Connection");

if(!WriteFile(hMailslot, szBuf, strlen(szBuf) + 1, &cbWritten, NULL))

            // Если произошла ошибка, выдаем сообщение

            printf("Error Data Transfer!!!\n");

else printf("Transferred %d bytes: <%s>\n", cbWritten, szBuf);

CloseHandle(hMailslot);

getch();

return 0;

}

**Порядок выполнения работы**

1. Ознакомиться с постановкой задачи и исходными данными. В соответствии с номером по журналу определить вариант задачи.

2. Согласно рекомендациям, приведенным в исходных данных, сконструировать структуру программы.

3. Составить тексты программ и утвердить их у преподавателя.

4. Пункты 1 - 3 должны быть выполнены предварительно до проведения данной лабораторной работы. Утверждение преподавателем текстов программ является допуском к лабораторной работе.

5. Набрать текст программы.

6. Выполнить компиляцию программы.

7. Провести анализ и исправление обнаруженных синтаксических ошибок в тексте программы и повторить пункты 6. и 7. При устранении всех синтаксических ошибок перейти к выполнению пункта 8.

8. Получить решение и, в случае обнаружения логических ошибок, описать и устранить их. Продемонстрировать преподавателю окончательный вариант программы и ее работу.

**Содержание отчета**

1. Тема лабораторной работы.

2. Цель работы.

3. Индивидуальное задание.

4. Метод и алгоритм решения задачи.

5. Текст программы.

6. Результаты работы программы.

7. Выводы по работе.

**Индивидуальные задания**

Организовать передачу строковых данных, вводимых с клавиатуры, между процессами 1, 2, 3, …, N по заданной схеме (символ  означает направление передачи данных) с использованием указанного средства IPC. Завершение работы программ выполнять по команде с клавиатуры.

15. (12,3, 23, 31), почтовые ящики.