ЛАБОРАТОРНАЯ РАБОТА. Порты завершения.

**Цель**. Изучить механизм синхронизации – порт завершения.

**Теория**.

Порт завершения – это системная очередь, в которую один процесс посылает пакеты, а второй считывает. Порт завершения связывается с файлом. Посылка пакетов связывается с завершением операции ввода-вывода. Итак, один процесс создает порт завершения и связывает его с файлом. Этот процесс выполняет операцию ввода-вывода над файлом. По завершении операции в порт направляется пакет с информацией о числе записанных (прочитанных) байтов и др. Ожидающий процесс (поток) может считать этот пакет и среагировать на завершение операции ввода-вывода.

Рассмотрим следующую программу. В ней главная функция main создает порт завершения, создает поток и запускает его. Привязывает порт завершения к файлу. После выполн6ения операции записи в порт направляется пакет, который считывает поток, выдает сообщение и завершает свою работу.

#include <windows.h>

#include <iostream>

HANDLE hCompletionPort; // порт завершения

DWORD WINAPI thread(LPVOID)

{

int i = 0; // количество полученных пакетов

DWORD dwNumberOfBytes; // количество переданных байтов

ULONG ulCompletionKey; // ключ файла

LPOVERLAPPED lpOverlapped; // указатель на структуру типа OVERLAPPED

cout « "The thread is started." « endl;

// подключаем поток к порту

while (GetQueuedCompletionStatus(

hCompletionPort, // дескриптор порта завершения

&dwNumberOfBytes, // количество переданных байтов

&ulCompletionKey, // ключ файла

&lpOverlapped, // указатель на структуру типа OVERLAPPED

INFINITE)) // бесконечное ожидание

// { тело цикла

// проверяем пакет на завершение вывода

if (ulCompletionKey == 0)

{

cout « endl « "The thread is finished." « endl;

break; }

else

cout « "\tPacket: " « ++i « endl

« "Number of bytes: " « dwNumberOfBytes « endl « "Completion key: " « ulCompletionKey « endl;

// конец тела цикла

return 0;

}

int main()

{

HANDLE hFile;

OVERLAPPED ovl;

ULONG ulKey;

HANDLE hThread;

// дескриптор файла

// структура управления асинхронным доступом к файлу

// ключ файла

// массив для дескрипторов потоков

DWORD dwThreadID; // массив для идентификаторов потоков

// инициализируем структуру OVERLAPPED

ovl.Offset =0; // младшая часть смещения равна 0

ovl.OffsetHigh =0; // старшая часть смещения равна 0

ovl.hEvent =0; // события нет

// запрашиваем ключ файла

cout « "Input a number for file key (not zero)

с in » ulKey;

if (ulKey == 0)

{

cout « "The file key can't be equal to zero.

« "Press any key to exit." « endl;

« endl;

return 0;

}

// создаем файл для записи данных

hFile = CreateFile(

TEXT("С:\my.dat"),

GENERIC\_WRITE,

FILE\_SHARE\_WRITE,

NULL,

OPEN ALWAYS,

// имя файла

// запись в файл

// совместный доступ к файлу

// защиты нет

// открываем или создаем новый файл

FILE\_FLAG\_OVERLAPPED, // асинхронный доступ к файлу

NULL // шаблона нет

);

// проверяем на успешное создание

if (hFile == INVALID\_HANDLE\_VALUE)

{

cerr « "Create file failed." « endl

« "The last error code: " « GetLastError() « endl;

cout « "Press any key to finish." « endl;

cin.getO ;

return 0;

}

// создаем порт завершения и подключаем к нему файл

hCompletionPort = CreateloCompletionPort(

hFile, // дескриптор файла

NULL, // новый порт

ulKey, // ключа файла

1 // один поток

);

// проверяем на успешное создание

if (hCompletionPort == NULL)

{

cerr « "Create completion port failed." « endl

« "The last error code: " « GetLastError() « endl;

cout « "Press any key to finish." « endl;

cin.getO ;

return 0;

}

// запускаем поток

hThread = CreateThread(NULL, 0, thread, NULL, 0, fcdwThreadlD);

// пишем данные в файл

for (int i = 0; i < 10; ++i)

{

DWORD dwBytesWri te;

DWORD dwRet;

if (!WriteFile(

hFile, // дескриптор файла

&i, // адрес буфера, откуда идет запись

sizeof(i), // количество записываемых байтов

&dwBytesWrite, // количество записанных байтов

&ovl // запись асинхронная

))

{

dwRet = GetLastError();

if (dwRet == ERROR\_IO\_PENDING)

cout « "Write file pending." « endl;

else

{

cout « "Write file failed." « endl

« "The last error code: " « dwRet « endl;

return 0;

}

}

// ждем, пока завершится асинхронная операция записи

WaitForSingleObject(hFile, INFINITE);

// увеличивает смещение в файле

ovl.Offset += sizeof(i);

}

// посылаем пакет с командой на завершение потока

PostQueuedCompletionStatus(

hCompletionPort, // дескриптор потока

0, // нет передачи

0, // ключ завершения

NULL); // нет структуры типа OVERLAPPED

// ждем завершения потока

WaitForSingleObject(hThread, INFINITE);

// закрываем дескрипторы

CloseHandle(hFile);

CloseHandle(hCompletionPort);

CloseHandle(hThread);

cout « "The file is written." « endl;

return 0;

}

Для понимания этой программы опишем некоторые функции для работы с портами завершения.

Для посылки приложением пакета в порт завершения ввода-вывода

используется функция PostQueuedCompletionStatus, которая имеет следующий прототип:

BOOL PostQueuedCompletionStatus(

HANDLE hCompletionPort, // дескриптор порта завершения

WORD lpNumberOfBytesTransferred, // количество переданных байтов

ULONG \*ulCompletionKey, // ключ завершения

LPOVERLAPPED \*IpOveriapped, // асинхронный доступ

);

В случае успешного завершения функция PostQueuedCompletionStatus возвращает ненулевое значение, а в случае неудачи - false.

В параметре hCompietionPort должен быть установлен дескриптор порта,

в который приложение посылает пакет. Остальные параметры могут

принимать произвольные значения. Эти значения будут переданы через

соответствующие выходные параметры функции GetQueuedCompletionStatus в поток, который получит пакет, вызвав эту функцию.

Для получения пакета из порта завершения ввода-вывода используется

функция GetQueuedCompletionStatus, которая имеет следующий формат

BOOL GetQueuedCompletionStatus(

HANDLE hCompletionPort, // дескриптор порта завершения

LPDWORD lpNumberOfBytes, // количество переданных байтов

ULONG \*ulCompletionKey, // ключ завершения

LPOVERLAPPED \*lpOverlapped, // адрес структур управления

// асинхронным доступом

DWORD dwMilliseconds // интервал ожидания

);

В случае успешного завершения функция GetQueuedCompletionStatus возвращает ненулевое значение, а в случае неудачи — false. Параметры этой

функции имеют следующее назначение.

В параметре hcompletionPort должен быть установлен дескриптор порта,

к которому поток обращается за пакетом.

В параметре lpNumberOfBytes должен быть установлен адрес переменной,

в которую в случае успешного завершения функция поместит количество

переданных байт.

В параметре uicompletionKey должен быть задан адрес переменной типа

unsigned long, в которую функция GetQueuedCompletionStatus поместит

ключ завершения, т. е. номер файла, для которого завершилась асинхронная

операции ввода-вывода. Посылку пакетов в порт завершения ввода-вывода

могут инициировать следующие функции: ConnectNamedPipe, DeviceloControl, LockFileEx, ReadDirectoryChangesW, ReadFile, TransactNamedPipe,waitcommEvent, writeFile.

В параметре IpOveriapped должен быть задан адрес на переменную, в

которую функция GetQueuedCompletionStatus Поместит адрес структуры

overlapped, для завершенной асинхронной операции ввода-вывода.

Параметр dwMilliseconds должен содержать количество миллисекунд,

которые определяют время ожидания функцией GetQueuedCompletionStatus

завершение асинхронной операции ввода-вывода. Если в течение этого

времени в очередь не поставлен ни один пакет, то функция завершается

неудачей, а по адресу, заданному параметром ipOverlapped, будет записано

значение null.

**ЗАДАНИЕ**.

Используя программу в данном методическом пособии написать приложение, в котором с одним и тем же портом приложения работают два потока. Один поток пишет в порт завершения строку текста и посылает в порт завершения пакет для второго потока, который прочитает строку из файла, выведет ее на консоль и завершится. Порт завершения создать в главной функции main, откуда запустить на выполнение два потока – один для записи, а второй - для чтения.