## Санкт-Петербургский государственный политехнический университет Институт Информационных Технологий и Управления

## Кафедра компьютерных систем и программных технологий

Отчёт по расчётной работе № 2

по предмету «Системное программное обеспечение»

**Межпроцессное взаимодействие в ОС Windows**

## Работу выполнил студент гр. 53501/3 Мартынов С. А. Работу принял преподаватель Душутина Е. В.

## Санкт-Петербург 2014

**Оглавление**

[Постановка задачи 3](#_TOC_250009)

[Введение 4](#_TOC_250008)

[Анонимные каналы 7](#_TOC_250007)

[Именованные каналы 14](#_TOC_250006)

[Почтовые ящики 23](#_TOC_250005)

[Shared memory 31](#_TOC_250004)

[Сокеты 37](#_TOC_250003)

[Порты завершения 49](#_TOC_250002)

[Сигналы 62](#_TOC_250001)

[Заключение 65](#_TOC_250000)

Список литературы 67

# Постановка задачи

## В рамках данной работы необходимо ознакомиться с основными механизмами межпроцесс- ное взаимодействие в ОС Windows

## Анонимные каналы;

## Именованные каналы (локальная/сетевая реализация);

## Почтовые ящики;

## Shared memory;

## Сокеты;

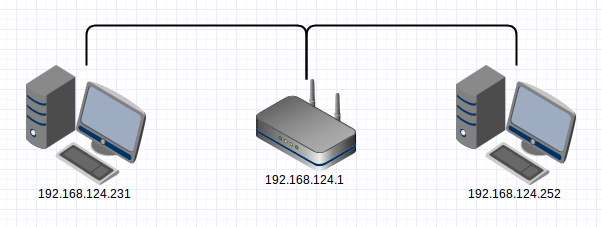
## Порты завершения;

## Сигналы.

## В процессе изучения предполагается разработать простой (консольный) мгновенный обмен сообщениями.

# Введение

## Для тестирования сетевых реализаций используются два виртуальные машины (Win7) под управлением гипервизора VirtualBox. Сетевое подключение осуществляется в режиме bridge. Топология представлена на рисунке 1. Разницы между виртуальной и физической средой быть не должно.



## Рис. 1: Топология сети.

## Все результаты, представленные в данном отчёте получены с использованием Microsoft Windows 7 Ultimate Service Pack 1 64-bit (build 7601). Для разработки использовалась Microsoft Visual Studio Express 2013 for Windows Desktops (Version 12.0.30723.00 Update 3). В качестве отладчика использовался Microsoft WinDbg (release 6.3.9600.16384).

## Для логирования использовался код, приведённый в листинге 1. Этот файл подключается в первых строках каждого проекта. Он создаёт файл, с именем идентичным исполняемой программе, и записывает туда все события.

## Листинг 1: Логер, используемый во всех проектах (src/InterProcessCommunication/AnonymousPipeServer/logger.h)

1 # pragma once

### 2

1. # include < stdio . h>
2. # include < stdlib . h>
3. # include < tchar . h>
4. # include < stdarg . h>
5. # include < time . h>

### 8

1. *// log*
2. FILE \* logfile ;

### 11

1. void initlog ( const \_ TCHAR \* prog );
2. void closelog ();
3. void writelog ( \_ TCHAR \* format , ...) ;

### 15

1. void initlog ( const \_ TCHAR \* prog ) {
2. \_ TCHAR logname [ 255 ];
3. wcscpy\_ s ( logname , prog );
4. *// replace extension*
5. \_ TCHAR \* extension ;
6. extension = wcsstr ( logname , \_T(". exe "));
7. wcsncpy\_ s ( extension , 5 , \_T(". log "), 4);
8. *// Try to open log file for append*
9. if ( \_ wfopen\_ s (& logfile , logname , \_T(" a+"))) {
10. \_ wperror (\_T(" The following error occurred "));
11. \_ tprintf (\_T(" Can ’t open log file % s\ n"), logname );

27 exit ( -1);

28 }

29 writelog (\_T("% s is starting ."), prog );

30 }

1. void closelog () {
2. writelog (\_T(" Shutting down .\ n"));
3. fclose ( logfile );

34 }

1. void writelog ( \_ TCHAR \* format , ...) {
2. \_ TCHAR buf [ 255 ];
3. va\_ list ap;
4. struct tm newtime ;
5. \_\_ time 64 \_ t long\_ time ;
6. *// Get time as 64 - bit integer .*
7. \_ time 64 (& long\_ time );
8. *// Convert to local time.*

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

\_ localtime 64 \_ s (& newtime , & long\_ time );

*// Convert to normal representation .*

swprintf\_ s ( buf , \_T("[% d/% d/% d % d:% d:% d] "), newtime . tm\_mday , newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour ,

newtime . tm\_min ,

*// Write date and*

fwprintf ( logfile ,

newtime . tm\_ sec );

*time*

\_T("% s"), buf );

*// Write all params*

va\_ start ( ap , format );

\_ vsnwprintf\_ s ( buf , sizeof ( buf ) - 1 , format , ap); fwprintf ( logfile , \_T("% s"), buf );

va\_ end ( ap);

*// New sting*

fwprintf ( logfile , \_T("\ n"));

}

## В данном файле содержатся следующие методы:

## initlog – принимает в качестве параметра имя приложения, заменяет exe на log и создаёт лог-файл;

## closelog – обеспечивает корректное завершение работы, освобождая дескриптор файла;

## writelog – функция получает форматированную строку и набор параметров, из которых формируется строка лога по аналогии с библиотечной функцией printf. Интересным моментом является тот факт, что строка дописывает текущее время, что упрощает анализ параллельных программ.

## Часть кода и лог-файлов приведена в листингах по ходу работы, однако более полная версия находится в папках src/InterProcessCommunication и logs/InterProcessCommunication.

# Анонимные каналы

## Анонимные каналы (anonymous channels) Windows обеспечивают однонаправленное (по- лудуплексное) посимвольное межпроцессное взаимодействие. Каждый канал имеет два дескриптора: дескриптор чтения (read handle) и дескриптор записи (write handle). Функция, с помощью которой создаются анонимные каналы, имеет следующий прототип [1, 2]:

## BOOL CreatePipe(PHANDLE phRead, PHANDLE phWrite,

## LPSECURITY\_ATTRIBUTES lpsa, DWORD cbPipe)

## Дескрипторы каналов часто бывают наследуемыми; причины этого станут понятными из приведенного ниже примера. Значение параметра cbPipe, указывающее размер канала в байтах, носит рекомендательный характер, причем значению 0 соответствует размер канала по умолчанию.

## Чтобы канал можно было использовать для IPC, должен существовать еще один процесс, и для этого процесса требуется один из дескрипторов канала. Предположим, например, что родительскому процессу, вызвавшему функцию CreatePipe, необходимо вывести дан- ные, которые нужны дочернему процессу. Тогда возникает вопрос о том, как передать дочернему процессу дескриптор чтения (phRead). Родительский процесс осуществляет это, устанавливая дескриптор стандартного ввода в структуре STARTUPINFO для дочерней процедуры равным \*phRead.

## Чтение с использованием дескриптора чтения канала блокируется, если канал пуст. В противном случае в процессе чтения будет воспринято столько байтов, сколько имеется в канале, вплоть до количества, указанного при вызове функции ReadFile. Операция записи в заполненный канал, которая выполняется с использованием буфера в памяти, также будет блокирована.

## Наконец, анонимные каналы обеспечивают только однонаправленное взаимодействие. Для двухстороннего взаимодействия необходимы два канала.

## В листинге 2 и 3 демонстрируются серверный и клиентский модуль программы, в которой используется передача дескрипторов через наследование. Анонимный канал является по-

## лудуплексным, поэтому для организации эхо-сервера было необходимо создавать 2 канала (для передачи от клиента к серверу и в обратном направлении). При этом ненужные дескрипторы каналов закрываются только на стороне сервера (т.к. клиент наследует 4 дескриптора, а явно передаются только 2). Дескрипторы каналов связываются со стандарт- ным вводом и выводом клиентского процесса. Для вывода информации клиенту остаётся только поток ошибок[1].

## Листинг 2: Сервер анонимных каналов (src/InterProcessCommunication/AnonymousPipeServer/main.cpp)

1. # include < windows . h>
2. # include < stdio . h>
3. # include < tchar . h>
4. # include " logger . h"

### 5

1. *// размер буфера для сообщений*
2. # define BUF\_ SIZE 100

### 8

9 int \_ tmain ( int argc , \_ TCHAR \* argv []) {

1. *// Init log*
2. initlog ( argv [0]) ;

### 12

1. *// буфер приема/ передачи*
2. \_ TCHAR buf [ BUF\_ SIZE ];
3. *// число прочитанных/ переданных байт*
4. DWORD readbytes , writebytes ;

### 17

1. *// дескрипторы канала для передачи от сервера клиенту*
2. HANDLE h Read Pipe From Serv To Client , h Write Pipe From Serv To Client ;
3. *// дескрипторы канала для передачи от клиента серверу*
4. HANDLE h Read Pipe From Client To Serv , h Write Pipe From Client To Serv ;

### 22

1. *// чтобы сделать дескрипторы наследуемыми*
2. SECURITY\_ ATTRIBUTES Pipe SA = { sizeof ( SECURITY\_ ATTRIBUTES ), NULL , TRUE };
3. *// создаем канал для передачи от сервера клиенту, сразу делаем дескрипторы наследуемыми*
4. if ( Create Pipe (& h Read Pipe From Serv To Client , & h Write Pipe From Serv To Client , & PipeSA , 0) == 0) {
5. double errorcode = Get Last Error ();
6. writelog (\_T(" Can ’t create anonymous pipe from server to client , GLE =% d."

), errorcode );

1. \_ tprintf (\_T(" Can ’t create anonymous pipe from server to client , GLE =% d."

), errorcode );

1. closelog ();
2. exit (1) ;

32 }

1. writelog (\_T(" Anonymous pipe from server to client created "));
2. \_ tprintf (\_T(" Anonymous pipe from server to client created \ n"));

### 35

1. *// создаем канал для передачи от клиента серверу, сразу делаем дескрипторы наследуемыми*
2. if ( Create Pipe (& h Read Pipe From Client To Serv , & h Write Pipe From Client To Serv , & PipeSA , 0) == 0) {
3. double errorcode = Get Last Error ();
4. writelog (\_T(" Can ’t create anonymous pipe from client to server , GLE =% d."

), errorcode );

1. \_ tprintf (\_T(" Can ’t create anonymous pipe from client to server , GLE =% d."

), errorcode );

1. closelog ();
2. exit (2) ;

43 }

1. writelog (\_T(" Anonymous pipe from client to server created "));
2. \_ tprintf (\_T(" Anonymous pipe from client to server created \ n"));

### 46

1. PROCESS\_ INFORMATION process Info\_ Client ; *// информация о процессе-клиенте*
2. *// структура, которая описывает внешний вид основного окна и содержит*
3. *// дескрипторы стандартных устройств нового процесса*
4. STARTUPINFO startup Info\_ Client ;

### 51

1. *// процесс-клиент будет иметь те же параметры запуска, что и сервер,*
2. *// за исключением дескрипторов ввода, вывода и ошибок*
3. Get Startup Info (& startup Info\_ Client );
4. *// устанавливаем поток ввода*
5. startup Info\_ Client . h Std Input = h Read Pipe From Serv To Client ;
6. *// установим поток вывода*
7. startup Info\_ Client . h Std Output = h Write Pipe From Client To Serv ;
8. *// установим поток ошибок*
9. startup Info\_ Client . h Std Error = Get Std Handle ( STD\_ ERROR\_ HANDLE );
10. *// устанавливаем наследование*
11. startup Info\_ Client . dw Flags = STARTF\_ USESTDHANDLES ;
12. *// создаем процесс клиента*
13. if (! Create Process (\_T(" Anonymous Pipe Client . exe "), *// имя исполняемого моду ля*
14. NULL , *// командная строка*
15. NULL , *// атрибуты безопасности процесса*
16. NULL , *// атрибуты безопасности потока*
17. TRUE , *// флаг наследования описателя*
18. CREATE\_ NEW\_ CONSOLE , *// флаги создания*
19. NULL , *// новый блок окружения*
20. NULL , *// имя текущей директории*
21. & startup Info\_ Client , *// STARTUPINFO*
22. & process Info\_ Client )) { *// PROCESS\_ INFORMATION*
23. writelog (\_T(" Can ’t create process , GLE =% d."), Get Last Error ());
24. \_ wperror (\_T(" Create process "));
25. closelog ();
26. exit (3) ;

78 }

1. writelog (\_T(" New process created "));
2. \_ tprintf (\_T(" New process created \ n"));

### 81

1. *// закрываем дескрипторы созданного процесса и его потока*
2. Close Handle ( process Info\_ Client . h Thread );
3. Close Handle ( process Info\_ Client . h Process );
4. *// закрываем ненужные дескрипторы каналов, которые не использует сервер*
5. Close Handle ( h Read Pipe From Serv To Client );
6. Close Handle ( h Write Pipe From Client To Serv );

### 88

89 *// Начинаем взаимодействие с клиентом через анонимный канал*

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

1. *// читаем данные из канала от клиента*
2. if (! Read File ( h Read Pipe From Client To Serv , buf , sizeof ( buf ), & readbytes , NULL )) {
3. double errorcode = Get Last Error ();
4. writelog (\_T(" Impossible to use readfile , GLE =% d."), errorcode );
5. \_ tprintf (\_T(" Impossible to use readfile , GLE =% d."), errorcode );
6. closelog ();
7. exit (4) ;

98 }

1. \_ tprintf (\_T(" Get from client : \"% s \"\ n"), buf );
2. writelog (\_T(" Get from client : \"% s\""), buf );
3. *// пишем данные в канал клиенту*
4. if (! Write File ( h Write Pipe From Serv To Client , buf , readbytes , & writebytes , NULL )) {
5. double errorcode = Get Last Error ();
6. writelog (\_T(" Impossible to use writefile , GLE =% d."), errorcode );
7. \_ tprintf (\_T(" Impossible to use writefile , GLE =% d."), errorcode );
8. closelog ();
9. exit (5) ;

### 108 }

### 109 }

1. *// закрываем HANDLE каналов*
2. Close Handle ( h Read Pipe From Client To Serv );
3. Close Handle ( h Write Pipe From Serv To Client );

### 113

114 closelog ();

115 exit (0) ;

116 }

## Клиент передаёт сообщение, например, вида: «message num 1», ждёт 1 секунду, и пытается прочитать его от сервера. Сервер, получив сообщение от клиента, передаёт его обратно. Процессы завершаются после передачи 10 сообщений.

## Листинг 3: Клиент анонимных каналов (src/InterProcessCommunication/AnonymousPipeClient/main.cpp)

1. # include < stdio . h>
2. # include < Windows . h>
3. # include " logger . h"

### 4

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. *// Init log*
3. initlog ( argv [0]) ;

### 8

1. \_ TCHAR strtosend [ 100 ]; *// строка для передачи*
2. \_ TCHAR getbuf [ 100 ]; *// буфер приема*
3. *// число переданных и принятых байт*
4. DWORD bytessended , bytesreaded ;

### 13

14 *// Начинаем взаимодействие с сервером через анонимный канал*

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

1. *// формирование строки для передачи*
2. bytessended = swprintf\_ s ( strtosend , \_T(" Message num % d"), i + 1);
3. strtosend [ bytessended ++] = \_T(’\0 ’);

### 19

1. writelog (\_T(" Client sended : \"% s\""), strtosend );
2. fwprintf ( stderr , \_T(" Client sended : \"% s \"\ n"), strtosend );

### 22

1. *// передача данных*
2. if (! Write File ( Get Std Handle ( STD\_ OUTPUT\_ HANDLE ), strtosend , bytessended \* sizeof ( wchar\_ t ), & bytesreaded , NULL )) {
3. double errorcode = Get Last Error ();
4. writelog (\_T(" Error with writeFile , GLE =% d."), errorcode );
5. fwprintf ( stderr , \_T(" Error with writeFile , GLE =% d."), errorcode );
6. closelog ();
7. exit (1) ;

30 }

31 Sleep ( 1000 ) ;

1. *// прием ответа от сервера*
2. if (! Read File ( Get Std Handle ( STD\_ INPUT\_ HANDLE ), getbuf , 100 , & bytesreaded , NULL )) {

34

35

36

37

38

39

40

41

42

43

44

45

46

double errorcode = Get Last Error ();

writelog (\_T(" Error with readFile , GLE =% d."), errorcode );

fwprintf ( stderr , \_T(" Error with

closelog (); exit (2) ;

readFile , GLE =% d."), errorcode );

}

writelog (\_T(" Get msg from server : fwprintf ( stderr , \_T(" Get msg from

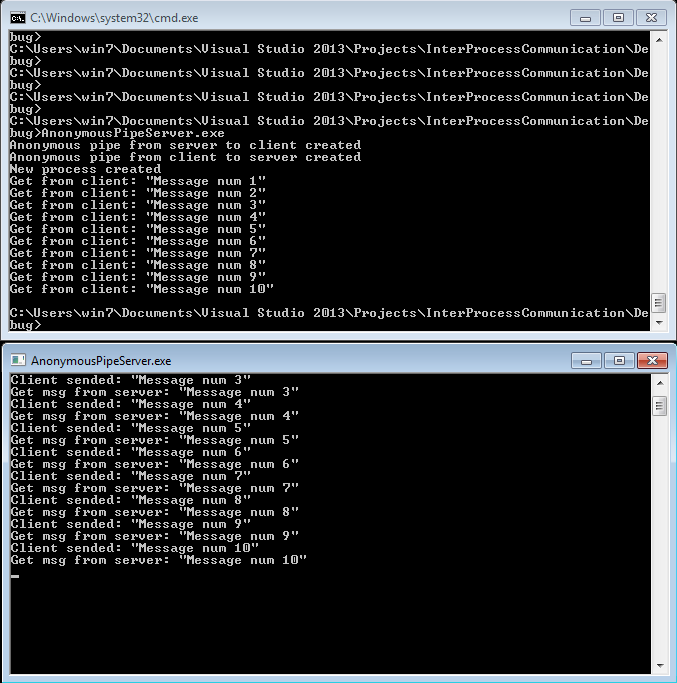
\"% s\""), getbuf );

server : \"% s \"\ n"), getbuf );

}

Sleep ( 10000 ) ; closelog (); return 0;

}



## Рис. 2: Работа с анонимными каналами.

## Результаты работы показаны на рисунке 2. Листинг 4 и листинг 5 содержат протоколы работы сервера и клиента.

## Листинг 4: Протокол работы серверного модуля программы работы с анонимными каналами

1 [ 14 / 2 / 2015

2 [ 14 / 2 / 2015

3 [ 14 / 2 / 2015

4 [ 14 / 2 / 2015

5 [ 14 / 2 / 2015

6 [ 14 / 2 / 2015

7 [ 14 / 2 / 2015

8 [ 14 / 2 / 2015

9 [ 14 / 2 / 2015

10 [ 14 / 2 / 2015

11 [ 14 / 2 / 2015

12 [ 14 / 2 / 2015

13 [ 14 / 2 / 2015

14 [ 14 / 2 / 2015

15 [ 14 / 2 / 2015

3 : 46 : 36 ]

3 : 46 : 36 ]

3 : 46 : 36 ]

3 : 46 : 36 ]

3 : 46 : 36 ]

3 : 46 : 37 ]

3 : 46 : 38 ]

3 : 46 : 39 ]

3 : 46 : 40 ]

3 : 46 : 41 ]

3 : 46 : 42 ]

3 : 46 : 43 ]

3 : 46 : 44 ]

3 : 46 : 45 ]

3 : 46 : 45 ]

Anonymous Pipe Server . exe is

Anonymous pipe from server Anonymous pipe from client

starting .

to

to

client created

server created

New

Get Get Get Get Get Get Get Get Get Get

process created

from

from from from from from from from from from

client :

client : client : client : client : client : client : client : client : client : down .

* Message
* Message " Message " Message " Message " Message " Message " Message " Message " Message

num 1"

num 2"

num 3"

num 4"

num 5"

num 6"

num 7"

num 8"

num 9"

num 10"

Shutting

## Листинг 5: Протокол работы клиентского модуля программы работы с анонимными кана- лами

1. [ 14 / 2 / 2015 3 : 46 : 36 ] Anonymous Pipe Client . exe is starting .
2. [ 14 / 2 / 2015 3 : 46 : 36 ] Client sended : " Message num 1"
3. [ 14 / 2 / 2015 3 : 46 : 37 ] Get msg from server : " Message num 1"
4. [ 14 / 2 / 2015 3 : 46 : 37 ] Client sended : " Message num 2"
5. [ 14 / 2 / 2015 3 : 46 : 38 ] Get msg from server : " Message num 2"
6. [ 14 / 2 / 2015 3 : 46 : 38 ] Client sended : " Message num 3"
7. [ 14 / 2 / 2015 3 : 46 : 39 ] Get msg from server : " Message num 3"
8. [ 14 / 2 / 2015 3 : 46 : 39 ] Client sended : " Message num 4"
9. [ 14 / 2 / 2015 3 : 46 : 40 ] Get msg from server : " Message num 4"
10. [ 14 / 2 / 2015 3 : 46 : 40 ] Client sended : " Message num 5"
11. [ 14 / 2 / 2015 3 : 46 : 41 ] Get msg from server : " Message num 5"
12. [ 14 / 2 / 2015 3 : 46 : 41 ] Client sended : " Message num 6"
13. [ 14 / 2 / 2015 3 : 46 : 42 ] Get msg from server : " Message num 6"
14. [ 14 / 2 / 2015 3 : 46 : 42 ] Client sended : " Message num 7"
15. [ 14 / 2 / 2015 3 : 46 : 43 ] Get msg from server : " Message num 7"
16. [ 14 / 2 / 2015 3 : 46 : 43 ] Client sended : " Message num 8"
17. [ 14 / 2 / 2015 3 : 46 : 44 ] Get msg from server : " Message num 8"
18. [ 14 / 2 / 2015 3 : 46 : 44 ] Client sended : " Message num 9"
19. [ 14 / 2 / 2015 3 : 46 : 45 ] Get msg from server : " Message num 9"

20 [ 14 / 2 / 2015 3 : 46 : 45 ] Client sended : " Message num 10"

21 [ 14 / 2 / 2015 3 : 46 : 46 ] Get msg from server : " Message num 10"

22 [ 14 / 2 / 2015 3 : 46 : 56 ] Shutting down .

# Именованные каналы

## Именованные каналы (named pipes) предлагают ряд возможностей, которые делают их полезными в качестве универсального механизма реализации приложений на основе IPC, включая приложения, требующие сетевого доступа к файлам, и клиент-серверные системы, хотя для реализации простых вариантов IPC, ориентированных на байтовые потоки, как в предыдущем примере, в котором взаимодействие процессов ограничивается рамками одной системы, анонимных каналов вам будет вполне достаточно. К числу упомянутых возможностей (часть которых обеспечивается дополнительно) относятся следующие[2]:

## Именованные каналы ориентированы на обмен сообщениями, поэтому процесс, вы- полняющий чтение, может считывать сообщения переменной длины именно в том виде, в каком они были посланы процессом, выполняющим запись.

## Именованные каналы являются двунаправленными, что позволяет осуществлять обмен сообщениями между двумя процессами посредством единственного канала.

## Допускается существование нескольких независимых экземпляров канала, имеющих одинаковые имена. Например, с единственной серверной системой могут связываться одновременно несколько клиентов, использующих каналы с одним и тем же именем. Каждый клиент может иметь собственный экземпляр именованного канала, и сервер может использовать этот же канал для отправки ответа клиенту.

## Каждая из систем, подключенных к сети, может обратиться к каналу, используя его имя. Взаимодействие посредством именованного канала осуществляется одинаковым образом для процессов, выполняющихся как на одной и той же, так и на разных машинах.

## Имеется несколько вспомогательных и связных функций, упрощающих обслуживание взаимодействия "запрос/ответ"и клиент-серверных соединений.

## Как правило, именованные каналы являются более предпочтительными по сравнению с анонимными, хотя существуют ситуации, когда анонимные каналы оказываются исключи- тельно полезными. Во всех случаях, когда требуется, чтобы канал связи был двунаправлен-

## ным, ориентированным на обмен сообщениями или доступным для нескольких клиентских процессов, следует применять именованные каналы.

## Реализация работы с именованными каналами представлена в листинге 8 (серверный модуль) и 9 (клиентский модуль). Сервер, как и ранее, создает все необходимые ресурсы и переходит в состояние ожидания соединений. Именованный канал создается для чтения и записи. Передача происходит сообщениями, функции передачи и приема блокируются до их окончания.

## Листинг 6: Сервер именованного каналов (src/InterProcessCommunication/NamedPipeServer/main.cpp)

1. # include < windows . h>
2. # include < stdio . h>
3. # include < conio . h>
4. # include < tchar . h>
5. # include < strsafe . h>
6. # include " logger . h"

### 7

8 # define BUFSIZE 512

### 9

10 DWORD WINAPI Instance Thread ( LPVOID );

### 11

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. *// Init log*
3. initlog ( argv [0]) ;
4. \_ tprintf (\_T(" Server is started .\ n\ n"));

### 16

1. BOOL f Connected = FALSE ; *// Флаг наличия подключенных клиентов*
2. DWORD dw Thread Id = 0; *// Номер обслуживающего потока*
3. HANDLE h Pipe = INVALID\_ HANDLE\_ VALUE ; *// Идентификатор канала*
4. HANDLE h Thread = NULL ; *// Идентификатор обслуживающего потока*
5. LPTSTR lpsz Pipename = \_T(" \\\\.\\ pipe \\ $$ My Pipe$$ "); *// Имя создаваемого к анала*

### 22

23 *// Цикл ожидает клиентов и созда e¨ т для них потоки обработки*

24 for (;;) {

1. writelog (\_T(" Try to create named pipe on % s"), lpsz Pipename );
2. \_ tprintf (\_T(" Try to create named pipe on % s\ n"), lpsz Pipename );

### 27

1. *// Создаем канал:*
2. if (( h Pipe = Create Named Pipe (
3. lpsz Pipename , *// имя канала,*
4. PIPE\_ ACCESS\_ DUPLEX , *// режим отрытия канала - двунаправленный,*
5. PIPE\_ TYPE\_ MESSAGE | *// данные записываются в канал в виде потока сообщ ений,*
6. PIPE\_ READMODE\_ MESSAGE | *// данные считываются в виде потока сообщений,*
7. PIPE\_WAIT , *// функции передачи и приема блокируются до их окончания,*
8. PIPE\_ UNLIMITED\_ INSTANCES , *// максимальное число экземпляров каналов не ограничено,*
9. BUFSIZE , *// размеры выходного и входного буферов канала,*
10. BUFSIZE ,
11. 5000 , *// 5 секунд - длительность для функции Wait Named Pipe ,*
12. NULL )) *// дескриптор безопасности по умолчанию.*

40 == INVALID\_ HANDLE\_ VALUE ) {

1. double errorcode = Get Last Error ();
2. writelog (\_T(" Create Named Pipe failed , GLE =% d."), errorcode );
3. \_ tprintf (\_T(" Create Named Pipe failed , GLE =% d.\ n"), errorcode );
4. closelog ();
5. exit (1) ;

46 }

1. writelog (\_T(" Named pipe created successfully !"));
2. \_ tprintf (\_T(" Named pipe created successfully !\ n"));

### 49

1. *// Ожидаем соединения со стороны клиента*
2. writelog (\_T(" Waiting for connect ... "));
3. \_ tprintf (\_T(" Waiting for connect ...\ n"));
4. f Connected = Connect Named Pipe ( hPipe , NULL ) ?
5. TRUE :
6. ( Get Last Error () == ERROR\_ PIPE\_ CONNECTED );

### 56

1. *// Если произошло соединение*
2. if ( f Connected ) {
3. writelog (\_T(" Client connected !"));
4. writelog (\_T(" Creating a processing thread ... "));
5. \_ tprintf (\_T(" Client connected !\ n Creating a processing thread ...\ n"));

### 62

1. *// Созда e¨ м поток для обслуживания клиента*
2. h Thread = Create Thread (
3. NULL , *// дескриптор защиты*
4. 0 , *// начальный размер стека*
5. Instance Thread , *// функция потока*
6. ( LPVOID ) hPipe , *// параметр потока*
7. 0 , *// опции создания*
8. & dw Thread Id ); *// номер потока*

### 71

1. *// Если поток создать не удалось - сообщаем об ошибке*
2. if ( h Thread == NULL ) {
3. double errorcode = Get Last Error ();
4. writelog (\_T(" Create Thread failed , GLE =% d."), errorcode );
5. \_ tprintf (\_T(" Create Thread failed , GLE =% d.\ n"), errorcode );
6. closelog ();
7. exit (1) ;

79 }

80 else Close Handle ( h Thread );

81 }

1. else {
2. *// Если клиенту не удалось подключиться, закрываем канал*
3. Close Handle ( hPipe );
4. writelog (\_T(" There are not connecrtion reqests ."));
5. \_ tprintf (\_T(" There are not connecrtion reqests .\ n"));

87 }

88 }

### 89

1. closelog ();
2. exit (0) ;

92 }

### 93

1. DWORD WINAPI Instance Thread ( LPVOID lpv Param ) {
2. writelog (\_T(" Thread % d started !"), Get Current Thread Id ());
3. \_ tprintf (\_T(" Thread % d started !\ n"), Get Current Thread Id ());
4. HANDLE h Pipe = ( HANDLE ) lpv Param ; *// Идентификатор канала*
5. *// Буфер для хранения полученного и передаваемого сообщения*
6. \_ TCHAR \* ch Buf = ( \_ TCHAR \*) Heap Alloc ( Get Process Heap () , 0 , BUFSIZE \* sizeof (

\_ TCHAR ));

1. DWORD readbytes , writebytes ; *// Число байт прочитанных и переданных*

### 101

1. while (1) {
2. *// Получаем очередную команду через канал Pipe*
3. if ( Read File ( hPipe , chBuf , BUFSIZE \* sizeof ( \_ TCHAR ), & readbytes , NULL )) {
4. *// Посылаем эту команду обратно клиентскому приложению*
5. if (! Write File ( hPipe , chBuf , ( lstrlen ( chBuf ) + 1)\* sizeof ( \_ TCHAR ), & writebytes , NULL ))
6. break ;
7. *// Выводим принятую команду на консоль*
8. writelog (\_T(" Thread % d: Get client msg : % s"), Get Current Thread Id () , ch Buf );
9. \_ tprintf ( TEXT (" Get client msg : % s\ n"), ch Buf );
10. *// Если пришла команда " exit", завершаем работу приложения*
11. if (! \_ tcsncmp ( chBuf , L" exit ", 4))
12. break ;
13. } else {
14. double errorcode = Get Last Error ();
15. writelog (\_T(" Thread % d: GRead File : Error % ld"), Get Current Thread Id () , errorcode );

|  |  |  |
| --- | --- | --- |
| 117 |  | \_ tprintf ( TEXT (" Read File : Error % ld\ n"), errorcode ); |
| 118 |  | \_ getch (); |
| 119 |  | break ; |
| 120 |  | } |
| 121  122 |  | } |
| 123 |  | *// Освобождение ресурсов* |
| 124 |  | Flush File Buffers ( hPipe ); |
| 125 |  | Disconnect Named Pipe ( hPipe ); |
| 126 |  | Close Handle ( hPipe ); |
| 127 |  | Heap Free ( Get Process Heap () , 0 , ch Buf ); |
| 128 |  |  |
| 129 |  | writelog (\_T(" Thread % d: Instance Thread exitting ."), Get Current Thread Id ()); |
| 130 |  | \_ tprintf ( TEXT (" Instance Thread exitting .\ n")); |
| 131 |  | return 0; |
| 132 | } |  |

## Клиент после соединения с сервером начинает чтение сообщений с консоли, пока не встретит слово «exit». По данному слову и клиент и сервер завершают свою работу.

## Для данной программы были внесены изменения в логер. Теперь каждый клиентский процесс создаёт свой отдельный файл с протоколом.

## Листинг 7: Клиент именованного каналов (src/InterProcessCommunication/NamedPipeClient/main.cpp)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

# include

# include

# include

# include

# include

# include

< windows . h>

< stdio . h>

< conio . h>

< tchar . h>

< strsafe . h> " logger . h"

# define BUFSIZE 512

int \_ tmain ( int argc , \_ TCHAR \* argv []) {

*// Init log*

initlog ( argv [0]) ;

\_ tprintf (\_T(" Client is started !\ n\ n"));

HANDLE h Pipe = INVALID\_ HANDLE\_ VALUE ; *// Идентификатор канала*

1. LPTSTR lpsz Pipename = \_T(" \\\\.\\ pipe \\ $$ My Pipe$$ "); *// Имя создаваемого к анала Pipe*
2. \_ TCHAR chBuf [ BUFSIZE ]; *// Буфер для передачи данных через канал*
3. DWORD readbytes , writebytes ; *// Число байт прочитанных и переданных*

### 19

1. writelog (\_T(" Try to use Wait Named Pipe ... "));
2. \_ tprintf (\_T(" Try to use Wait Named Pipe ...\ n"));
3. *// Пытаемся открыть именованный канал, если надо - ожидаем его освобождени я*
4. while (1) {
5. *// Создаем канал с процессом- сервером:*
6. hPipe = Create File (
7. lpsz Pipename , *// имя канала,*
8. GENERIC\_ READ *// текущий клиент имеет доступ на чтение,*
9. | GENERIC\_ WRITE , *// текущий клиент имеет доступ на запись,*
10. 0 , *// тип доступа,*
11. NULL , *// атрибуты защиты,*
12. OPEN\_ EXISTING , *// открывается существующий файл,*
13. 0 , *// атрибуты и флаги для файла,*
14. NULL ); *// доступа к файлу шаблона.*

### 34

1. *// Продолжаем работу, если канал создать удалось*
2. if ( h Pipe != INVALID\_ HANDLE\_ VALUE )
3. break ;

### 38

1. *// Выход, если ошибка связана не с занятым каналом.*
2. double errorcode = Get Last Error ();
3. if ( errorcode != ERROR\_ PIPE\_ BUSY ) {
4. writelog (\_T(" Could not open pipe . GLE =% d\ n"), errorcode );
5. \_ tprintf (\_T(" Could not open pipe . GLE =% d\ n"), errorcode );
6. closelog ();
7. exit (1) ;

46 }

### 47

1. *// Если все каналы заняты, жд e¨ м 20 секунд*
2. if (! Wait Named Pipe ( lpsz Pipename , 20000 ) ) {
3. writelog (\_T(" Could not open pipe : 20 second wait timed out ."));
4. \_ tprintf (\_T(" Could not open pipe : 20 second wait timed out ."));
5. closelog ();
6. exit (2) ;

54 }

55 }

### 56

1. *// Выводим сообщение о создании канала*
2. writelog (\_T(" Successfully connected !"));
3. \_ tprintf (\_T(" Successfully connected !\ n\ n Input message ...\ n"));
4. *// Цикл обмена данными с серверным процессом*
5. while (1) {
6. *// Выводим приглашение для ввода команды*
7. \_ tprintf (\_T(" cmd >"));
8. *// Вводим текстовую строку*
9. \_ fgetts ( chBuf , BUFSIZE , stdin );
10. *// Заносим строку в протокол*
11. writelog (\_T(" Client sended : % s"), chBuf );
12. *// Передаем введенную строку серверному процессу в качестве команды*
13. if (! Write File ( hPipe , chBuf , ( lstrlen ( chBuf ) + 1)\* sizeof ( TCHAR ), & writebytes , NULL )) {
14. writelog (\_T(" connection refused \ n"));
15. \_ tprintf (\_T(" connection refused \ n"));
16. break ;

73 }

1. *// Получаем эту же команду обратно от сервера*
2. if ( Read File ( hPipe , chBuf , BUFSIZE \* sizeof ( TCHAR ), & readbytes , NULL )) {
3. writelog (\_T(" Received from server : % s"), ch Buf );
4. \_ tprintf (\_T(" Received from server : % s\ n"), ch Buf );
5. } else {
6. *// Если произошла ошибка, выводим ее код и завершаем работу приложения*
7. double errorcode = Get Last Error ();
8. writelog (\_T(" Read File : Error % ld\ n"), errorcode );
9. \_ tprintf (\_T(" Read File : Error % ld\ n"), errorcode );
10. \_ getch ();
11. break ;

85 }

1. *// В ответ на команду " exit" завершаем цикл обмена данными с серверным п роцессом*
2. if (! \_ tcsncmp ( chBuf , L" exit ", 4)) {
3. writelog (\_T(" Processing exit code "));
4. break ;

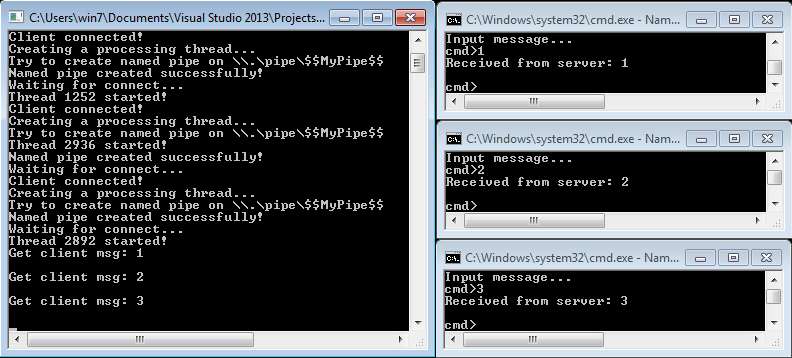
90 }

91 }

1. *// Закрываем идентификатор канала*
2. Close Handle ( hPipe );
3. closelog ();
4. return 0;

96 }

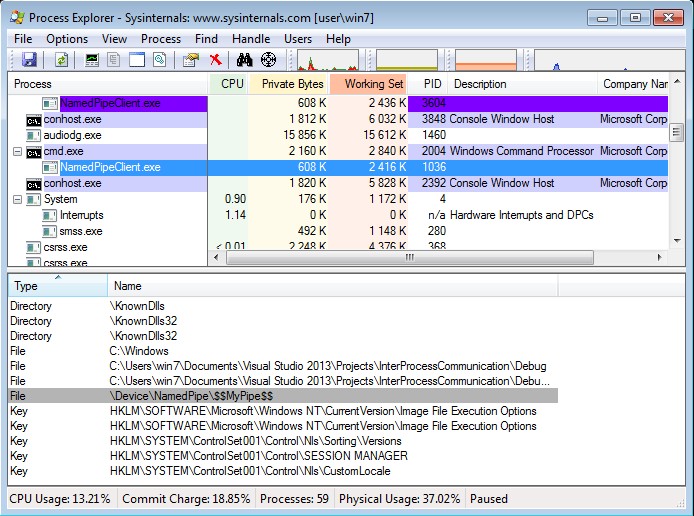
## Результат работы программы показан на рисунке 3. Запущен один сервер и три клиента, с которыми этот сервер взаимодействует.



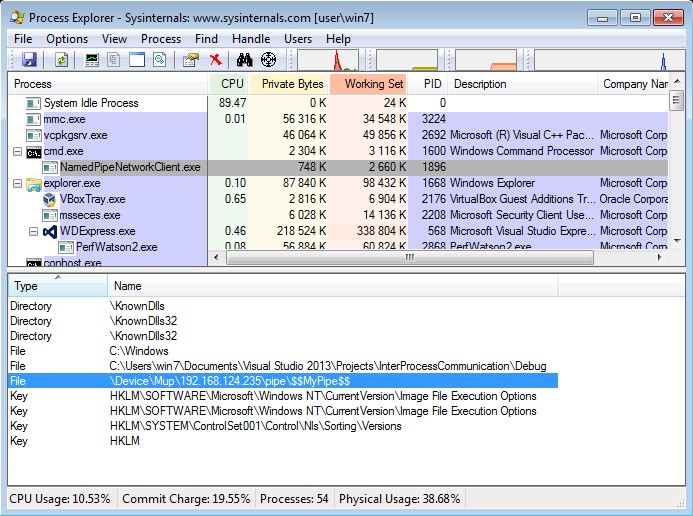
## Рис. 3: Работа нескольких клиентов с одним сервером по именованному каналу.

## Программа Process Explorer известного разработчика Марка Русиновича позволяет отсле- дить, кто использует именованные каналы. На рисунке 4 видно, что клиентский модуль использует канал MyPipe.

## Помимо локального обмена, именованные каналы могут использоваться и для сетевого взаимодействия. Это требует не большой доработки клиента в части указания пути к каналу и изменения настроек безопасности: клиент обращается к именованному каналу указывая имя серверного хоста или его IP адрес[1]. Работа сетевой версии программы показана на рисунке 5.



## Рис. 4: Отслеживания обращений к локальным (не сетевым) именованным каналам



## Рис. 5: Отслеживания обращений к сетевым именованным каналам в Process Explorer

# Почтовые ящики

## Как и именованные каналы, почтовые ящики (mailslots) Windows снабжаются именами, которые могут быть использованы для обеспечения взаимодействия между независимыми каналами[2]. Почтовые ящики представляют собой широковещательный механизм, и ведут себя иначе по сравнению с именованными каналами, что делает их весьма полезными в ряде ограниченных ситуаций, которые, тем не менее, представляют большой интерес. Из наиболее важных свойств почтовых ящиков можно отметить следующие:

## Почтовые ящики являются однонаправленными.

## С одним почтовым ящиком могут быть связаны несколько записывающих программ (writers) и несколько считывающих программ (readers), но они часто связаны между собой отношениями "один ко многим"в той или иной форме.

## Записывающей программе (клиенту) не известно достоверно, все ли, только некоторые или какая-то одна из программ считывания (сервер) получили сообщение.

## Почтовые ящики могут находиться в любом месте сети.

## Размер сообщений ограничен.

## Использование почтовых ящиков требует выполнения следующих операций:

## Каждый сервер создает дескриптор почтового ящика с помощью функции CreateMailSlot.

## После этого сервер ожидает получения почтового сообщения, используя функцию ReadFile.

## Клиент, обладающий только правами записи, должен открыть почтовый ящик, вызвав функцию CreateFile, и записать сообщения, используя функцию WriteFile. В случае отсутствия ожидающих программ считывания попытка открытия почтового ящика завершится ошибкой (наподобие "имя не найдено").

## Сообщение клиента может быть прочитано всеми серверами; все серверы получают одно и то же сообщение.

## Листинг 8 и 9 демонстрируют реализацию приложения, иллюстрирующую обмен инфор- мацией почтовыми слотами. В процессе экспериментов было протестировано локальное, сетевое взаимодействие[1]. Для широковещательной передач сообщений, адрес заменялся символом звездочки (\*).

## Листинг 8: Реализация серверной части почтового ящика (src/InterProcessCommunication/MailslotServer/main.cpp)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

# include

# include

# include

< windows . h>

< stdio . h> " logger . h"

LPTSTR Slot Name = TEXT (" \\\\.\\ mailslot \\ sample\_ mailslot ");

BOOL Write Slot ( HANDLE hSlot ,

{

BOOL f Result ; DWORD cb Written ;

LPTSTR lpsz Message )

writelog (\_T(" Text to send : % s"), lpsz Message );

\_ tprintf (\_T(" Text to send : % s\ n"), lpsz Message );

f Result = Write File ( hSlot ,

lpsz Message ,

( DWORD )( lstrlen ( lpsz Message ) + 1)\* sizeof ( TCHAR ),

& cbWritten ,

( LPOVERLAPPED ) NULL );

if (! f Result ) {

double errorcode = Get Last Error ();

writelog (\_T(" Write File failed , GLE =% d."), errorcode );

\_ tprintf (\_T(" Write File failed , GLE =% d."), errorcode ); return FALSE ;

}

writelog (\_T(" Slot written to successfully "));

\_ tprintf (\_T(" Slot written to successfully \ n"));

return TRUE ;

}

int \_ tmain ( int argc , \_ TCHAR \* argv []) {

*// Init log*

initlog ( argv [0]) ;

HANDLE h File ;

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

hFile = Create File ( SlotName ,

GENERIC\_ WRITE , FILE\_ SHARE\_ READ ,

( LPSECURITY\_ ATTRIBUTES ) NULL , OPEN\_ EXISTING , FILE\_ ATTRIBUTE\_ NORMAL ,

( HANDLE ) NULL );

if ( h File == INVALID\_ HANDLE\_ VALUE )

double errorcode = Get Last Error ();

{

writelog (\_T(" Create File failed , GLE =% d."), errorcode );

\_ tprintf (\_T(" Create File failed , GLE =% d."), errorcode ); closelog ();

exit (1) ;

}

writelog (\_T(" Mailslot created "));

\_ tprintf (\_T(" Mailslot created "));

*// for ( int i = 0; i != 100; ++ i) {*

Write Slot ( hFile , Write Slot ( hFile , Sleep ( 5000 ) ; Write Slot ( hFile ,

*//}*

Close Handle ( hFile );

\_T(" Message

\_T(" Message

one for mailslot ."));

two for mailslot ."));

\_T(" Message

three for mailslot ."));

closelog ();

exit (0) ;

}

## Листинг 9: Реализация клиентской части почтового ящика (src/InterProcessCommunication/MailslotClient/main.cpp)

1

2

3

4

5

6

7

8

9

# include

# include

# include

# include

# include

# include

< windows . h>

< tchar . h>

< stdio . h>

< strsafe . h>

< conio . h> " logger . h"

HANDLE h Slot ;

LPTSTR Slot Name = \_T(" \\\\.\\ mailslot \\ sample\_ mailslot ");

### 10

11 BOOL Read Slot ()

12 {

1. DWORD cbMessage , cMessage , cb Read ;
2. BOOL f Result ;
3. LPTSTR lpsz Buffer ;
4. TCHAR achID [ 80 ];
5. DWORD c All Messages ;
6. HANDLE h Event ;
7. OVERLAPPED ov;

### 20

21 cb Message = c Message = cb Read = 0;

### 22

1. h Event = Create Event ( NULL , FALSE , FALSE , \_T(" Example Slot "));
2. if ( NULL == h Event )
3. return FALSE ;
4. ov. Offset = 0;
5. ov. Offset High = 0;
6. ov. h Event = h Event ;

### 29

1. f Result = Get Mailslot Info ( hSlot , *// mailslot handle*
2. ( LPDWORD ) NULL , *// no maximum message size*
3. & cbMessage , *// size of next message*
4. & cMessage , *// number of messages*
5. ( LPDWORD ) NULL ); *// no read time - out*

### 35

1. if (! f Result ) {
2. double errorcode = Get Last Error ();
3. writelog (\_T(" Get Mailslot Info failed , GLE =% d."), errorcode );
4. \_ tprintf (\_T(" Get Mailslot Info failed , GLE =% d."), errorcode );
5. return FALSE ;

41 }

### 42

1. if ( cb Message == MAILSLOT\_ NO\_ MESSAGE ) {
2. writelog (\_T(" Waiting for a message ... "));
3. \_ tprintf (\_T(" Waiting for a message ...\ n"));
4. return TRUE ;

47 }

### 48

49 c All Messages = c Message ;

### 50

1. while ( c Message != 0) { *// retrieve all messages*
2. *// Create a message - number string.*
3. String Cch Printf (( LPTSTR ) achID ,

54 80 ,

1. \_T("\ n Message #% d of % d\ n"),
2. c All Messages - c Message + 1 ,
3. c All Messages );

### 58

59 *// Allocate memory for the message.*

### 60

1. lpsz Buffer = ( LPTSTR ) Global Alloc ( GPTR ,
2. lstrlen (( LPTSTR ) achID )\* sizeof ( TCHAR ) + cb Message );
3. if ( NULL == lpsz Buffer )
4. return FALSE ;
5. lpsz Buffer [0] = ’\0 ’;

### 66

1. f Result = Read File ( hSlot ,
2. lpsz Buffer ,
3. cbMessage ,
4. & cbRead ,

71 & ov);

### 72

1. if (! f Result ) {
2. double errorcode = Get Last Error ();
3. writelog (\_T(" Read File failed , GLE =% d."), errorcode );
4. \_ tprintf (\_T(" Read File failed , GLE =% d./ n"), errorcode );
5. Global Free (( HGLOBAL ) lpsz Buffer );
6. return FALSE ;

79 }

### 80

1. *// Concatenate the message and the message - number string.*
2. String Cb Cat ( lpsz Buffer ,
3. lstrlen (( LPTSTR ) achID )\* sizeof ( TCHAR ) + cbMessage ,
4. ( LPTSTR ) achID );

### 85

1. *// Display the message.*
2. writelog (\_T(" Contents of the mailslot : % s\ n"), lpsz Buffer );
3. \_ tprintf (\_T(" Contents of the mailslot : % s\ n"), lpsz Buffer );

### 89

90 Global Free (( HGLOBAL ) lpsz Buffer );

### 91

1. f Result = Get Mailslot Info ( hSlot , *// mailslot handle*
2. ( LPDWORD ) NULL , *// no maximum message size*
3. & cbMessage , *// size of next message*
4. & cMessage , *// number of messages*
5. ( LPDWORD ) NULL ); *// no read time - out*

### 97

1. if (! f Result ) {
2. double errorcode = Get Last Error ();
3. writelog (\_T(" Get Mailslot Info failed , GLE =% d."), errorcode );
4. \_ tprintf (\_T(" Get Mailslot Info failed , GLE =% d./ n"), errorcode );
5. return FALSE ;

### 103 }

### 104 }

1. Close Handle ( h Event );
2. return TRUE ;

### 107 }

### 108

109 BOOL WINAPI Make Slot ( LPTSTR lpsz Slot Name )

### 110 {

1. hSlot = Create Mailslot ( lpsz Slot Name ,
2. 0 , *// no maximum message size*
3. MAILSLOT\_ WAIT\_ FOREVER , *// no time - out for operations*
4. ( LPSECURITY\_ ATTRIBUTES ) NULL ); *// default security*

### 115

1. if ( h Slot == INVALID\_ HANDLE\_ VALUE ) {
2. double errorcode = Get Last Error ();
3. writelog (\_T(" Create Mailslot failed , GLE =% d."), errorcode );
4. \_ tprintf (\_T(" Create Mailslot failed , GLE =% d."), errorcode );
5. return FALSE ;

### 121 }

1. writelog (\_T(" Mailslot created "));
2. \_ tprintf (\_T(" Mailslot created "));

### 124

125 return TRUE ;

### 126 }

### 127

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. *// Init log*
3. initlog ( argv [0]) ;

### 131

132 Make Slot ( Slot Name );

### 133

1. while ( TRUE ) {
2. Read Slot ();

136 Sleep ( 3000 ) ;

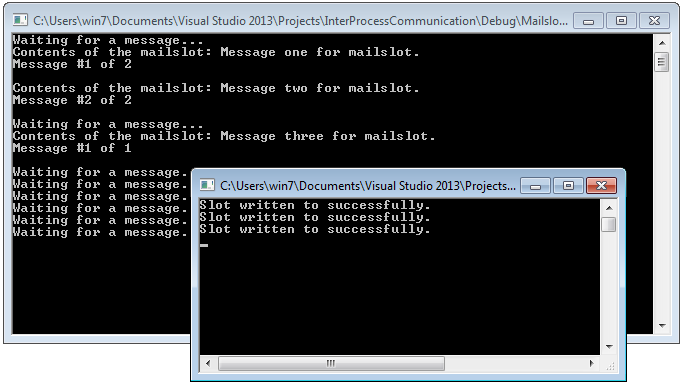
### 137 }

### 138

1. closelog ();
2. exit (0) ;

### 141 }

## Результат работы программы показан на рисунке 6.



## Рис. 6: Работа почтовыми ящиками.

## На рисунке 7 показана работа почтового ящика в Process Explorer. Листинг 10 и 11 содержит протокол работы программы.

## Листинг 10: Протокол работы серверного модуля программы работы с почтовыми ящиками

1 [ 14 / 2 / 2015 7 : 3 : 38 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \

Inter Process Communication \ Debug \ Mailslot Server . exe is starting .

2 [ 14 / 2 / 2015

3 [ 14 / 2 / 2015

4 [ 14 / 2 / 2015

5 [ 14 / 2 / 2015

6 [ 14 / 2 / 2015

7 [ 14 / 2 / 2015

8 [ 14 / 2 / 2015

9 [ 14 / 2 / 2015

7 : 3 : 38 ]

7 : 3 : 38 ]

7 : 3 : 38 ]

7 : 3 : 38 ]

7 : 3 : 38 ]

7 : 3 : 43 ]

7 : 3 : 43 ]

7 : 3 : 43 ]

Mailslot created

Text

Slot Text Slot Text Slot

to send : Message one for mailslot .

written to successfully

to send : Message two for mailslot . written to successfully

to send : Message three for mailslot .

written to successfully

Shutting down .

## Листинг 11: Протокол работы клиентского модуля программы работы с почтовыми ящика- ми

1 [ 14 / 2 / 2015 7 : 3 : 38 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \

Inter Process Communication \ Debug \ Mailslot Client . exe is starting .

2 [ 14 / 2 / 2015

3 [ 14 / 2 / 2015

4 [ 14 / 2 / 2015

5 Message #1

7 : 3 : 38 ]

7 : 3 : 38 ]

7 : 3 : 41 ]

of 2

Mailslot created

Waiting for a message ...

Contents of the mailslot : Message one for mailslot .

6 [ 14 / 2 / 2015

7 Message #2

8 [ 14 / 2 / 2015

9 Message #1

10 [ 14 / 2 / 2015

7 : 3 : 41 ]

of 2

7 : 3 : 44 ]

of 1

7 : 3 : 47 ]

Contents of the mailslot : Message two for mailslot .

Contents of

the mailslot : Message three for mailslot .

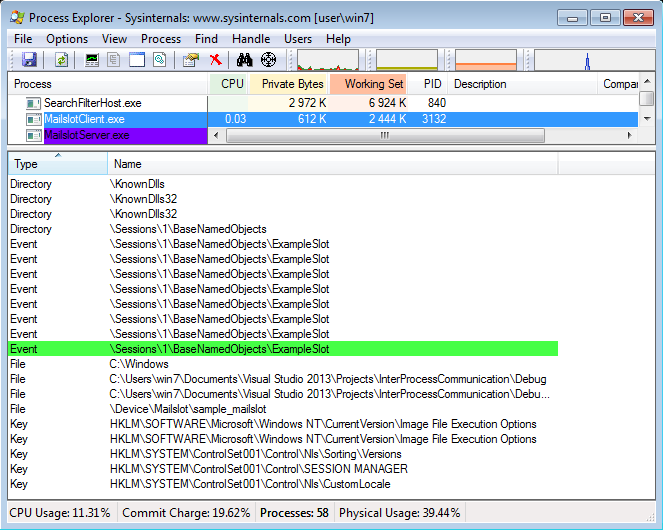
Waiting for

a message ...

## Существует еще одна возможность. В вызове функции CreateFile клиент может указать имя почтового ящика в следующем виде:

## \\\*\mailslot\mailslotname

## При этом символ звездочки (\*) действует в качестве группового символа (wildcard), и клиент может обнаружить любой сервер в пределах имени домена — группы систем, объединенных общим именем, которое назначается администратором сети[1].



## Рис. 7: Работа почтовыми ящиками.

# Shared memory

## Этот способ взаимодействия реализуется через технологию File Mapping - отображения файлов на оперативную память. Механизм позволяет осуществлять доступ к файлу таким образом, как будто это обыкновенный массив, хранящийся в памяти (не загружая файл в память явно). Можно создать объект file mapping, но не ассоциировать его с каким- то конкретным файлом[1]. Получаемая область памяти будет общей между процессами. Работая с этой памятью, потоки обязательно должны согласовывать свои действия с помощью объектов синхронизации.

## В листинге 12 и 13 представлен код двух программ, одна из которых генерирует случайные числа, а другая их читает и выводит на экран. Взаимодействие осуществляется через разделяемую память, защищённую мьютексом. Рисунок 8 показывает результат такого взаимодействия.

## Листинг 12: Программа, генерирующая случайные числа в разделяемую память (src/InterProcessCommunication/SharedMemoryServer/main.cpp)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

# include

# include

# include

# include

< windows . h>

< stdio . h>

< conio . h> " logger . h"

# define BUF\_ SIZE 256

TCHAR sz Name [] = \_T(" My File Mapping Object ");

TCHAR sz Msg [] = \_T(" Message from first process "); HANDLE mutex ;

int \_ tmain ( int argc , \_ TCHAR \* argv []) {

*// Init log*

initlog ( argv [0]) ;

HANDLE h Map File ; LPCTSTR pBuf ;

mutex = Create Mutex ( NULL , false , TEXT (" Sync Mutex "));

writelog (\_T(" Mutex created "));

1. *// create a memory , wicth two proccess will be working*
2. h Map File = Create File Mapping (
3. INVALID\_ HANDLE\_ VALUE , *// использование файла подкачки*
4. NULL , *// защита по умолчанию*
5. PAGE\_ READWRITE , *// доступ к чтению/ записи*
6. 0 , *// макс. размер объекта*
7. BUF\_SIZE , *// размер буфера*
8. sz Name ); *// имя отраженного в памяти объекта*

### 27

1. if ( h Map File == NULL || h Map File == INVALID\_ HANDLE\_ VALUE ) {
2. double errorcode = Get Last Error ();
3. writelog (\_T(" Create File Mapping failed , GLE =% d"), errorcode );
4. \_ tprintf (\_T(" Create File Mapping failed , GLE =% d"), errorcode );
5. closelog ();
6. exit (1) ;

34 }

35 writelog (\_T(" File Mapping Object created "));

### 36

1. pBuf = ( LPTSTR ) Map View Of File (
2. hMapFile , *// дескриптор проецируемого в памяти объекта*
3. FILE\_ MAP\_ ALL\_ ACCESS , *// разрешение чтения/ записи( режим доступа)*
4. 0 , *// Старшее слово смещения файла, где начинается отображение*
5. 0 , *// Младшее слово смещения файла, где начинается отображение*
6. BUF\_ SIZE ); *// Число отображаемых байтов файла*

### 43

1. if ( pBuf == NULL ) {
2. double errorcode = Get Last Error ();
3. writelog (\_T(" Map View Of File failed , GLE =% d"), errorcode );
4. \_ tprintf (\_T(" Map View Of File failed , GLE =% d"), errorcode );
5. closelog ();
6. exit (1) ;

50 }

### 51

1. int i = 0;
2. while ( true ) {
3. i = rand ();
4. \_ itow\_ s ( i, szMsg , sizeof ( szMsg ), 10) ;
5. writelog (\_T(" Wait For Mutex "));
6. Wait For Single Object ( mutex , INFINITE );
7. writelog (\_T(" Get Mutex "));
8. Copy Memory (( PVOID ) pBuf , szMsg , sizeof ( szMsg ));
9. \_ tprintf (\_T(" Write message : % s\ n"), pBuf );
10. writelog (\_T(" Write message : % s"), pBuf );
11. Sleep ( 1000 ) ; *// необходимо только для отладки - для удобства*
12. *// представления и анализа результатов*
13. Release Mutex ( mutex );
14. writelog (\_T(" Release Mutex "));

66 }

1. *// освобождение памяти и закрытие описателя handle*
2. Unmap View Of File ( pBuf );
3. Close Handle ( h Map File );
4. Close Handle ( mutex );

71

1. closelog ();
2. exit (0) ;

74

}

## Листинг 13: Программа, читающая случайные числа из разделяемой памяти (src/InterProcessCommunication/SharedMemoryClient/main.cpp)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

# include

# include

# include

# include

< windows . h>

< stdio . h>

< conio . h> " logger . h"

# define BUF\_ SIZE 256

# define TIME 150

*// number of reading operation in this process* TCHAR sz Name [] = \_T(" My File Mapping Object "); HANDLE mutex ;

int \_ tmain ( int argc , \_ TCHAR \* argv []) {

*// Init log* initlog ( argv [0]) ; Sleep ( 1000 ) ;

HANDLE h Map File ; LPCTSTR pBuf ;

mutex = Open Mutex (

MUTEX\_ ALL\_ ACCESS , *// request full access*

FALSE , *// handle not inheritable*

TEXT (" Sync Mutex ")); *// object name*

if ( mutex == NULL ) {

double errorcode = Get Last Error ();

writelog (\_T(" Open Mutex error , GLE =% d"), errorcode );

\_ tprintf (\_T(" Open Mutex error , GLE =% d\ n"), errorcode );

}

writelog (\_T(" Open Mutex successfully opened the mutex "));

29 \_ tprintf (\_T(" Open Mutex successfully opened the mutex .\ n"));

### 30

1. h Map File = Open File Mapping (
2. FILE\_ MAP\_ ALL\_ ACCESS , *// доступ к чтению/ записи*
3. FALSE , *// имя не наследуется*
4. sz Name ); *// имя " проецируемого " объекта*
5. if ( h Map File == NULL ) {
6. double errorcode = Get Last Error ();
7. writelog (\_T(" Open File Mapping failed , GLE =% d"), errorcode );
8. \_ tprintf (\_T(" Open File Mapping failed , GLE =% d"), errorcode );
9. closelog ();
10. exit (1) ;

41 }

1. pBuf = ( LPTSTR ) Map View Of File ( hMapFile ,
2. *// дескриптор " проецируемого" объекта*
3. FILE\_ MAP\_ ALL\_ ACCESS , *// разрешение чтения/ записи*
4. 0 , 0 , BUF\_ SIZE );
5. if ( pBuf == NULL ) {
6. double errorcode = Get Last Error ();
7. writelog (\_T(" Map View Of File failed , GLE =% d"), errorcode );
8. \_ tprintf (\_T(" Map View Of File failed , GLE =% d"), errorcode );
9. closelog ();
10. exit (1) ;

52 }

1. for ( int i = 0; i < TIME ; i ++) {
2. writelog (\_T(" Wait For Mutex "));
3. Wait For Single Object ( mutex , INFINITE );
4. writelog (\_T(" Get Mutex "));
5. \_ tprintf (\_T(" Read message : % s\ n"), pBuf );
6. writelog (\_T(" Read message : % s"), pBuf );
7. Release Mutex ( mutex );
8. writelog (\_T(" Release Mutex "));

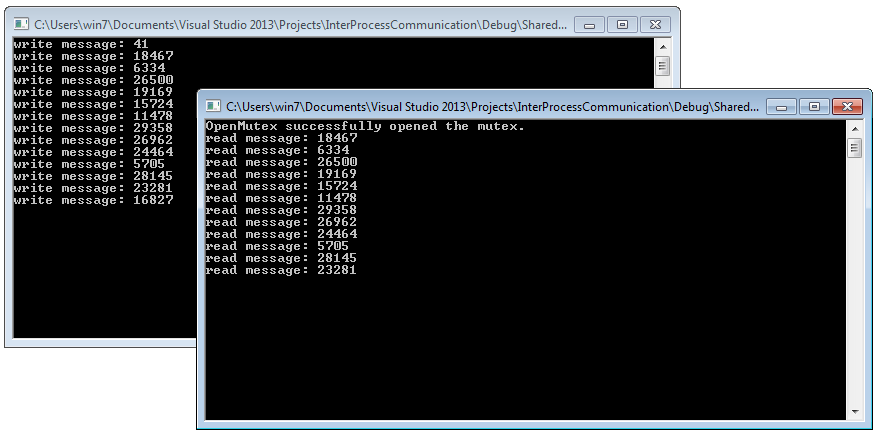
61 }

1. Unmap View Of File ( pBuf );
2. Close Handle ( h Map File );

### 64

1. closelog ();
2. exit (0) ;

67 }



## Рис. 8: Работа с разделяемой памятью.

## Листинг 14 и 15 содержит протокол (он не много подрезан для краткости) работы про- граммы. Интерес здесь представляет порядок захвата мьютекса.

## Листинг 14: Протокол работы серверного модуля программы работы с разделяемой памятью

1 [ 14 / 2 / 2015 8 : 33 : 21 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \ Inter Process Communication \ Debug \ Shared Memory Server . exe is starting .

2 [ 14 / 2 / 2015 8 : 33 : 21 ] Mutex created

3 [ 14 / 2 / 2015 8 : 33 : 21 ] File Mapping Object created

4 [ 14 / 2 / 2015 8 : 33 : 21 ] Wait For Mutex

5 [ 14 / 2 / 2015 8 : 33 : 21 ] Get Mutex

6 [ 14 / 2 / 2015 8 : 33 : 21 ] Write message : 41

7 [ 14 / 2 / 2015 8 : 33 : 22 ] Release Mutex

8 [ 14 / 2 / 2015 8 : 33 : 22 ] Wait For Mutex

9 [ 14 / 2 / 2015 8 : 33 : 22 ] Get Mutex

## Листинг 15: Протокол работы клиентского модуля программы работы с разделяемой памятью

1. [ 14 / 2 / 2015 8 : 33 : 21 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \ Inter Process Communication \ Debug \ Shared Memory Client . exe is starting .
2. [ 14 / 2 / 2015 8 : 33 : 22 ] Open Mutex successfully opened the mutex

3 [ 14 / 2 / 2015 8 : 33 : 22 ] Wait For Mutex

4 [ 14 / 2 / 2015 8 : 33 : 22 ] Get Mutex

5 [ 14 / 2 / 2015 8 : 33 : 22 ] Read message : 41

6 [ 14 / 2 / 2015 8 : 33 : 22 ] Release Mutex

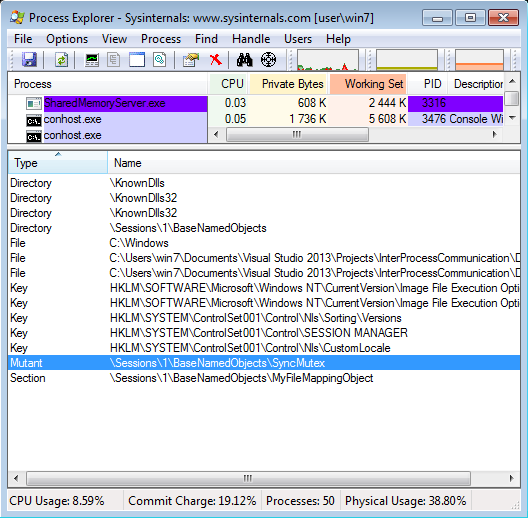
7 [ 14 / 2 / 2015 8 : 33 : 22 ] Wait For Mutex

8 [ 14 / 2 / 2015 8 : 33 : 23 ] Get Mutex

9 [ 14 / 2 / 2015 8 : 33 : 23 ] Read message : 18467

10 [ 14 / 2 / 2015 8 : 33 : 23 ] Release Mutex

## На рисунке 9 видно, как программа клиент работает с ресурсами. Две последние строчки, это мьютекс и общая память.



## Рис. 9: Мьютекс и общая память

# Сокеты

## Winsock API разрабатывался как расширение Berkley Sockets API для среды Windows и поэтому поддерживается всеми системами Windows. К особенностям Winsock можно отнести следующее[2]:

## Перенос уже имеющегося кода, написанного для Berkeley Sockets API, осуществляется непосредственно.

## Системы Windows легко встраиваются в сети, использующие как версию IPv4 про- токола TCP/IP, так и постепенно распространяющуюся версию IPv6. Помимо всего остального, версия IPv6 допускает использование более длинных IP-адресов, преодо- левая существующий 4-байтовый адресный барьер версии IPv4.

## Сокеты могут использоваться совместно с перекрывающимся вводом/выводом Windows, что, помимо всего прочего, обеспечивает возможность масштабирования серверов при увеличении количества активных клиентов.

## Сокеты можно рассматривать как дескрипторы (типа HANDLE) файлов при ис- пользовании функций ReadFile и WriteFile и, с некоторыми ограничениями, при использовании других функций, точно так же, как в качестве дескрипторов файлов сокеты применяются в UNIX. Эта возможность оказывается удобной в тех случаях, когда требуется использование асинхронного ввода/вывода и портов завершения ввода/вывода.

## Существуют также дополнительные, непереносимые расширения.

## Работа с сокетами демонстрируется в листинге 16 и 17. Рисунок 6 показывает работу программ, из этих листингов.

## Листинг 16: Сервер для работы с Win-сокетами (src/InterProcessCommunication/WinSockServer/main.cpp)

1. # define \_ WINSOCK\_ DEPRECATED\_ NO\_ WARNINGS
2. # include < winsock 2 . h>
3. # include " logger . h"

### 4

5 # pragma comment ( lib , " Ws 2 \_ 32 . lib ")

### 6

7 struct CLIENT\_ INFO

8 {

1. SOCKET h Client Socket ;
2. struct sockaddr\_ in client Addr ;

11 };

### 12

1. \_ TCHAR sz Server IPAddr [] = \_T(" 127 . 0 . 0 . 1 "); *// server IP*
2. int n Server Port = 5050; *// server port*
3. *// clients to talk with the server*

### 16

1. bool Init Win Sock 2 \_ 0 ();
2. BOOL WINAPI Client Thread ( LPVOID lp Data );

### 19

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. *// Init log*
3. initlog ( argv [0]) ;

### 23

1. if (! Init Win Sock 2 \_ 0 ()) {
2. double errorcode = WSAGet Last Error ();
3. writelog (\_T(" Unable to Initialize Windows Socket environment , GLE =% d"), errorcode );
4. \_ tprintf (\_T(" Unable to Initialize Windows Socket environment , GLE =% d"), errorcode );
5. closelog ();
6. exit (1) ;

30 }

31 writelog (\_T(" Windows Socket environment ready "));

### 32

1. SOCKET h Server Socket ;
2. h Server Socket = socket (
3. AF\_INET , *// The address family. AF\_ INET specifies TCP/ IP*
4. SOCK\_ STREAM , *// Protocol type. SOCK\_ STREM specified TCP*
5. 0 *// Protoco Name. Should be 0 for AF\_ INET address family*

38 );

### 39

1. if ( h Server Socket == INVALID\_ SOCKET ) {
2. writelog (\_T(" Unable to create Server socket "));
3. \_ tprintf (\_T(" Unable to create Server socket "));
4. *// Cleanup the environment initialized by WSAStartup ()*
5. WSACleanup ();
6. closelog ();
7. exit (2) ;

47 }

48 writelog (\_T(" Server socket created "));

### 49

1. *// Create the structure describing various Server parameters*
2. struct sockaddr\_ in server Addr ;

### 52

1. server Addr . sin\_ family = AF\_ INET ; *// The address family. MUST be AF\_ INET*
2. size\_ t convtd ;
3. char \* p MBBuffer = new char [20];
4. wcstombs\_ s (& convtd , pMBBuffer , 20 , sz Server IPAddr , 20) ;
5. *// server Addr . sin\_ addr . s\_ addr = inet\_ addr ( p MBBuffer );*
6. server Addr . sin\_ addr . s\_ addr = INADDR\_ ANY ;
7. delete [] p MBBuffer ;
8. server Addr . sin\_ port = htons ( n Server Port );

### 61

1. *// Bind the Server socket to the address & port*
2. if ( bind ( h Server Socket , ( struct sockaddr \*) & server Addr , sizeof ( server Addr

)) == SOCKET\_ ERROR ) {

1. writelog (\_T(" Unable to bind to % s on port % d"), sz Server IPAddr , n Server Port );
2. \_ tprintf (\_T(" Unable to bind to % s on port % d"), sz Server IPAddr , n Server Port );
3. *// Free the socket and cleanup the environment initialized by WSAStartup ()*
4. closesocket ( h Server Socket );
5. WSACleanup ();
6. closelog ();
7. exit (3) ;

71 }

72 writelog (\_T(" Bind "));

### 73

1. *// Put the Server socket in listen state so that it can wait for client connections*
2. if ( listen ( h Server Socket , SOMAXCONN ) == SOCKET\_ ERROR ) {
3. writelog (\_T(" Unable to put server in listen state "));
4. \_ tprintf (\_T(" Unable to put server in listen state "));
5. *// Free the socket and cleanup the environment initialized by WSAStartup ()*
6. closesocket ( h Server Socket );
7. WSACleanup ();
8. closelog ();
9. exit (4) ;

83 }

1. writelog (\_T(" Ready for connection "));
2. \_ tprintf (\_T(" Ready for connection \ n"));

### 86

1. *// Start the infinite loop*
2. while ( true ) {
3. *// As the socket is in listen mode there is a connection request pending*

*.*

1. *// Calling accept( ) will succeed and return the socket for the request.*
2. CLIENT\_ INFO \* p Client Info = new CLIENT\_ INFO ;
3. int n Size = sizeof ( p Client Info -> client Addr );

### 93

1. p Client Info -> h Client Socket = accept ( h Server Socket , ( struct sockaddr \*) & p Client Info -> client Addr , & n Size );
2. if ( p Client Info -> h Client Socket == INVALID\_ SOCKET ) {
3. writelog (\_T(" accept () failed "));
4. \_ tprintf (\_T(" accept () failed \ n"));

98 }

99 else {

1. HANDLE h Client Thread ;
2. DWORD dw Thread Id ;

### 102

1. wchar\_ t \* sin\_ addr = new wchar\_ t [20];
2. size\_ t convtd ;
3. mbstowcs\_ s (& convtd , sin\_addr , 20 , inet\_ ntoa ( p Client Info -> client Addr . sin\_ addr ), 20) ;
4. writelog (\_T(" Client connected from % s:% d"), sin\_addr , p Client Info -> client Addr . sin\_ port );
5. \_ tprintf (\_T(" Client connected from % s:% d\ n"), sin\_addr , p Client Info -> client Addr . sin\_ port );
6. delete [] sin\_ addr ;

### 109

1. *// Start the client thread*
2. h Client Thread = Create Thread ( NULL , 0 ,
3. ( LPTHREAD\_ START\_ ROUTINE ) Client Thread ,
4. ( LPVOID ) p Client Info , 0 , & dw Thread Id );
5. if ( h Client Thread == NULL ) {
6. writelog (\_T(" Unable to create client thread "));
7. \_ tprintf (\_T(" Unable to create client thread \ n"));

### 117 }

1. else {
2. Close Handle ( h Client Thread );

### 120 }

### 121 }

### 122 }

### 123

1. closesocket ( h Server Socket );
2. WSACleanup ();
3. closelog ();
4. exit (0) ;

### 128 }

### 129

1. bool Init Win Sock 2 \_ 0 () {
2. WSADATA wsa Data ;
3. WORD w Version = MAKEWORD (2 , 0);

### 133

1. if (! WSAStartup ( wVersion , & wsa Data ))
2. return true ;

### 136

137 return false ;

### 138 }

### 139

1. BOOL WINAPI Client Thread ( LPVOID lp Data ) {
2. CLIENT\_ INFO \* p Client Info = ( CLIENT\_ INFO \*) lp Data ;
3. \_ TCHAR sz Buffer [ 1024 ];
4. int n Length ;

### 144

1. while (1) {
2. n Length = recv ( p Client Info -> h Client Socket , ( char \*) szBuffer , sizeof ( sz Buffer ), 0);
3. wchar\_ t \* sin\_ addr = new wchar\_ t [20];
4. size\_ t convtd ;
5. mbstowcs\_ s (& convtd , sin\_addr , 20 , inet\_ ntoa ( p Client Info -> client Addr . sin\_ addr ), 20) ;
6. if ( n Length > 0) {
7. sz Buffer [ n Length ] = ’\0 ’;
8. writelog (\_T(" Received % s from % s:% d"), szBuffer , sin\_addr , p Client Info

-> client Addr . sin\_ port );

1. \_ tprintf (\_T(" Received % s from % s:% d\ n"), szBuffer , sin\_addr , p Client Info -> client Addr . sin\_ port );

### 154

1. *// Convert the string to upper case and send it back , if its not QUIT*
2. *// \_ wcsdup( sz Buffer );*
3. if ( wcscmp ( szBuffer , \_T(" QUIT ")) == 0) {
4. closesocket ( p Client Info -> h Client Socket );
5. delete p Client Info ;

|  |  |  |
| --- | --- | --- |
| 168 | if ( n Cnt Send == -1) { |  |
| 169 | writelog (\_T(" Error sending the data to % s:% d"), | sin\_addr , |

## Клиент отправляет серверу сообщения, и получает эхо-ответ. Слово QUIT зарезервировано для завершения работы.

160

161

162

163

164

165

166

167

return TRUE ;

}

*// send () may not*

*// So try sending*

int n Cnt Send = 0;

\_ TCHAR \* p Buffer =

*be able to send the complete data in one go.*

*the data in multiple requests*

sz Buffer ;

while (( n Cnt Send = send ( p Client Info -> h Client Socket , ( char \*) pBuffer , nLength , 0) != n Length )) {

170

171

172

173

174

175

176

177

178

179

180

181

p Client Info -> client Addr . sin\_ port );

\_ tprintf (\_T(" Error sending the data to % s:% d\ n"), sin\_addr , p Client Info -> client Addr . sin\_ port );

break ;

}

if ( n Cnt Send == n Length ) break ;

p Buffer += n Cnt Send ; n Length -= n Cnt Send ;

}

182

183

184

185

186

187

188

}

else {

writelog (\_T(" Error reading the data from % s:% d"), sin\_addr , p Client Info -> client Addr . sin\_ port );

\_ tprintf (\_T(" Error reading the data from % s:% d\ n"), sin\_addr , p Client Info -> client Addr . sin\_ port );

}

delete [] sin\_ addr ;

}

return TRUE ;

}

## Листинг 17: Клиент для работы с Win-сокетами (src/InterProcessCommunication/WinSockClient/main.cpp)

1

2

3

4

5

# define \_ WINSOCK\_ DEPRECATED\_ NO\_ WARNINGS

# include < winsock 2 . h>

# include " logger . h"

# pragma comment ( lib , " Ws 2 \_ 32 . lib ")

### 6

* 1. \_ TCHAR sz Server IPAddr [ 20 ]; *// server IP*
  2. int n Server Port ; *// server port*

### 9

10 bool Init Win Sock 2 \_ 0 ();

### 11

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. *// Init log*
3. initlog ( argv [0]) ;

### 15

1. \_ tprintf (\_T(" Enter the server IP Address : "));
2. wscanf\_ s (\_T(" %19 s"), sz Server IPAddr , \_ countof ( sz Server IPAddr ));
3. \_ tprintf (\_T(" Enter the server port number : "));
4. wscanf\_ s (\_T("% i"), & n Server Port );

### 20

1. if (! Init Win Sock 2 \_ 0 ()) {
2. double errorcode = WSAGet Last Error ();
3. writelog (\_T(" Unable to Initialize Windows Socket environment , GLE =% d"), errorcode );
4. \_ tprintf (\_T(" Unable to Initialize Windows Socket environment , GLE =% d"), errorcode );
5. closelog ();
6. exit (1) ;

27 }

28 writelog (\_T(" Windows Socket environment ready "));

### 29

1. SOCKET h Client Socket ;
2. h Client Socket = socket (
3. AF\_INET , *// The address family. AF\_ INET specifies TCP/ IP*
4. SOCK\_ STREAM , *// Protocol type. SOCK\_ STREM specified TCP*
5. 0); *// Protoco Name. Should be 0 for AF\_ INET address family*

### 35

1. if ( h Client Socket == INVALID\_ SOCKET ) {
2. writelog (\_T(" Unable to create Server socket "));
3. \_ tprintf (\_T(" Unable to create Server socket "));
4. *// Cleanup the environment initialized by WSAStartup ()*
5. WSACleanup ();
6. closelog ();
7. exit (2) ;

43 }

44 writelog (\_T(" Client socket created "));

### 45

1. *// Create the structure describing various Server parameters*
2. struct sockaddr\_ in server Addr ;

### 48

1. server Addr . sin\_ family = AF\_ INET ; *// The address family. MUST be AF\_ INET*
2. size\_ t convtd ;
3. char \* p MBBuffer = new char [20];
4. wcstombs\_ s (& convtd , pMBBuffer , 20 , sz Server IPAddr , 20) ;
5. server Addr . sin\_ addr . s\_ addr = inet\_ addr ( p MBBuffer );
6. delete [] p MBBuffer ;
7. server Addr . sin\_ port = htons ( n Server Port );

### 56

1. *// Connect to the server*
2. if ( connect ( h Client Socket , ( struct sockaddr \*) & server Addr , sizeof ( server Addr )) < 0) {
3. writelog (\_T(" Unable to connect to % s on port % d"), sz Server IPAddr , n Server Port );
4. \_ tprintf (\_T(" Unable to connect to % s on port % d"), sz Server IPAddr , n Server Port );
5. closesocket ( h Client Socket );
6. WSACleanup ();
7. closelog ();
8. exit (3) ;

65 }

66 writelog (\_T(" Connect "));

### 67

68 \_ TCHAR sz Buffer [ 1024 ] = \_T("");

### 69

1. while ( wcscmp ( szBuffer , \_T(" QUIT ")) != 0) {
2. \_ tprintf (\_T(" Enter the string to send ( QUIT ) to stop : "));
3. wscanf\_ s (\_T(" %1023 s"), szBuffer , \_ countof ( sz Buffer ));

### 73

74 int n Length = ( wcslen ( sz Buffer ) + 1) \* sizeof ( \_ TCHAR );

### 75

1. *// send( ) may not be able to send the complete data in one go.*
2. *// So try sending the data in multiple requests*
3. int n Cnt Send = 0;
4. \_ TCHAR \* p Buffer = sz Buffer ;

### 80

1. while (( n Cnt Send = send ( h Client Socket , ( char \*) pBuffer , nLength , 0) != n Length )) {
2. if ( n Cnt Send == -1) {
3. writelog (\_T(" Error sending the data to server "));
4. \_ tprintf (\_T(" Error sending the data to server \ n"));
5. break ;

86 }

1. if ( n Cnt Send == n Length )
2. break ;

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

p Buffer += n Cnt Send ;

n Length -= n Cnt Send ;

}

\_ wcsdup ( sz Buffer );

if ( wcscmp ( szBuffer , \_T(" QUIT ")) == 0) { break ;

}

n Length = recv ( h Client Socket , ( char \*) szBuffer , sizeof ( sz Buffer ), if ( n Length > 0) {

sz Buffer [ n Length ] = ’\0 ’;

writelog (\_T(" Received % s from server "), sz Buffer );

0);

\_ tprintf (\_T(" Received % s from server \ n"),

sz Buffer );

}

}

closesocket ( h Client Socket );

WSACleanup (); closelog (); exit (0) ;

}

bool Init Win Sock 2 \_ 0 () {

WSADATA wsa Data ;

WORD w Version = MAKEWORD (2 , 0);

if (! WSAStartup ( wVersion , & wsa Data )) return true ;

return false ;

}

## Листинг 18 и 19 содержит протокол работы программы.

## Листинг 18: Протокол работы серверного модуля программы работы с сокетами Windows

1 [ 14 / 2 / 2015 11 : 32 : 39 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \

Inter Process Communication \ Debug \ Win Sock Server . exe is starting .

2

3

4

5

6

7

[ 14 / 2 / 2015

[ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015

11 : 32 : 39 ] Windows Socket environment ready

11 : 32 : 39 ] Server socket created 11 : 32 : 39 ] Bind

11 : 32 : 39 ] Ready for connection

11 : 33 : 6 ] Client connected from 192 . 168 . 124 . 235

11 : 33 : 10 ] Received 1 from 192 . 168 . 124 . 235

8 [ 14 / 2 / 2015 11 : 33 : 14 ] Received Hello from 192 . 168 . 124 . 235

9 [ 14 / 2 / 2015 11 : 33 : 22 ] Received QUIT from 192 . 168 . 124 . 235

10 [ 14 / 2 / 2015 11 : 33 : 28 ] Shutting down .

## Листинг 19: Протокол работы клиентского модуля программы работы с сокетами Windows

1

2

3

4

5

6

7

[ 14 / 2 / 2015

[ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015

11 : 32 : 41 ] Win Sock Client . exe is starting .

11 : 33 : 6 ]

11 : 33 : 6 ]

11 : 33 : 6 ]

11 : 33 : 10 ]

11 : 33 : 14 ]

11 : 33 : 22 ]

Windows Socket environment ready

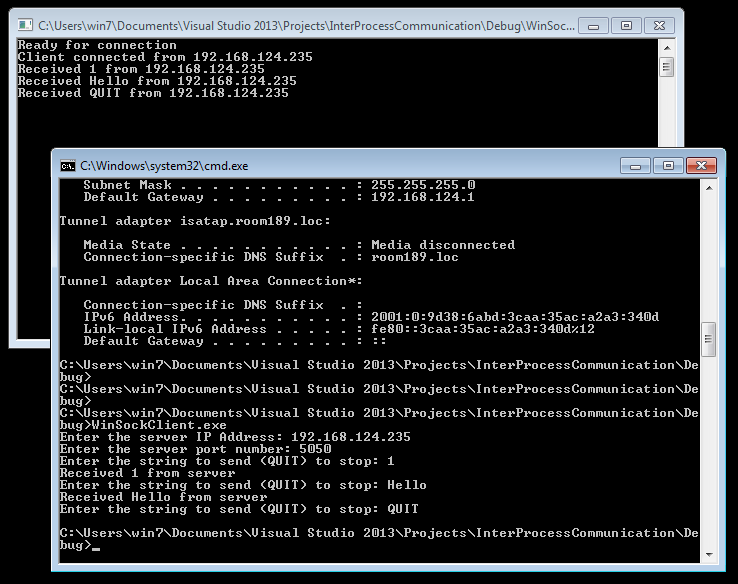
Client socket created Connect

Received

Received Shutting

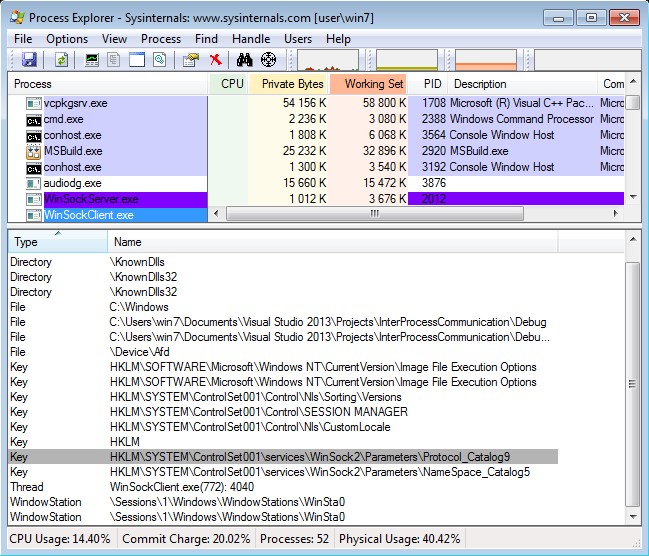
1 from server

Hello from server down .



## Рис. 10: Работа с сокетами.

## Результаты работы программы показаны на рисунке 10. Клиент соединяется с сервером по локальному IP адресу. Серверный слушающий сокет связан с адресом INADDR\_ANY, т.е. прослушивает все сетевые интерфейсы.



## Рис. 11: Сокеты в списке ресурсов

## На рисунке 11 показано как сокеты видны системе.

## Библиотека Winsock поддерживает два вида сокетов – синхронные (блокируемые) и асин- хронные (неблокируемые). Синхронные сокеты задерживают управление на время выпол- нения операции, а асинхронные возвращают его немедленно, продолжая выполнение в фоновом режиме, и, закончив работу, уведомляют об этом вызывающий код.

## Сокеты позволяют работать со множеством протоколов и являются удобным средством межпроцессорного взаимодействия[3], но в данной работе рассматриваются только сокеты семейства протоколов TCP/IP, использующихся для обмена данными между узлами сети Интернет.

## Независимо от вида, сокеты делятся на два типа – потоковые и дейтаграммные. Потоко- вые сокеты работают с установкой соединения, обеспечивая надежную идентификацию

## обоих сторон и гарантируют целостность и успешность доставки данных. Дейтаграмные сокеты работают без установки соединения и не обеспечивают ни идентификации отпра- вителя, ни контроля успешности доставки данных, зато они заметно быстрее потоковых. Дейтаграммные сокеты опираются на протокол UDP, а потоковые на TCP.

## Выбор того или иного типа сокетов определяется транспортным протоколом, на котором работает сервер, – клиент не может по своему желанию установить с дейтаграммным сервером потоковое соединение.

# Порты завершения

## Операциям ввода и вывода присуща более медленная скорость выполнения по сравнению с другими видами обработки. Причиной такого замедления являются следующие факторы[4]:

## Задержки, обусловленные затратами времени на поиск нужных дорожек и секторов на устройствах произвольного доступа (диски, компакт-диски).

## Задержки, обусловленные сравнительно низкой скоростью обмена данными между физическими устройствами и системной памятью.

## Задержки при передаче данных по сети с использованием файловых, серверов, хра- нилищ данных и так далее.

## Во всех предыдущих примерах операции ввода/вывода выполняются синхронно с потоком, поэтому весь поток вынужден простаивать, пока они не завершатся.

## В этом примере показано, каким образом можно организовать продолжение выполнения потока, не дожидаясь завершения операций ввода/вывода, что будет соответствовать выполнению потоками асинхронного ввода/вывода.

## Порты завершения оказываются чрезвычайно полезными при построении масштабируемых серверов, способных обеспечивать поддержку большого количества клиентов без создания для каждого из них отдельного потока.

## Листинг 20 показывает реализацию порта завершения. Для работы с ним использовался клиент из предыдущего примера.

## Листинг 20: Порт завершения (src/InterProcessCommunication/CompletionPortServer/main.cpp)

1

2

3

4

5

# include

# include

# include

# include

< winsock 2 . h>

< windows . h>

< stdio . h> " logger . h"

6 # pragma comment ( lib , " Ws 2 \_ 32 . lib ")

### 7

1. \_ TCHAR sz Server IPAddr [] = \_T(" 127 . 0 . 0 . 1 "); *// server IP*
2. int n Server Port = 5050; *// server port*
3. *// clients to talk with the server*

### 11

1. # define DATA\_ BUFSIZE 1024
2. # define EMPTY\_ MSG \_T(" ... ")

### 14

1. typedef struct {
2. OVERLAPPED Overlapped ;

17 *// //////////////////*

1. WSABUF Data Buf ;
2. CHAR Buffer [ DATA\_ BUFSIZE ];
3. DWORD Bytes Send ;
4. DWORD Bytes Recv ;
5. DWORD Total Bytes ;
6. SOCKADDR\_ IN client ;
7. } PER\_ IO\_ OPERATION\_ DATA , \* LPPER\_ IO\_ OPERATION\_ DATA ;

### 25

### 26

1. typedef struct {
2. SOCKET Socket ;
3. } PER\_ HANDLE\_ DATA , \* LPPER\_ HANDLE\_ DATA ;

### 30

### 31

32 DWORD WINAPI Client Thread ( LPVOID Completion Port ID );

### 33

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. *// Init log*
3. initlog ( argv [0]) ;

### 37

1. SOCKADDR\_ IN server ;
2. SOCKADDR\_ IN client ;

### 40

1. SOCKET Socket ;
2. SOCKET Accept ;

### 43

1. HANDLE Completion Port ;
2. SYSTEM\_ INFO Sys Info ;
3. HANDLE Thread ;

### 47

1. LPPER\_ HANDLE\_ DATA Per Handle Data ;
2. LPPER\_ IO\_ OPERATION\_ DATA Per Io Data ;

### 50

1. DWORD Send Bytes ;
2. DWORD Flags ;
3. DWORD Thread ID ;
4. WSADATA wsa Data ;

### 55

1. *// инициализируем Win Sock:*
2. if (( WSAStartup (0 x0202 , & wsa Data )) != 0){
3. double errorcode = WSAGet Last Error ();
4. writelog (\_T(" Unable to Initialize Windows Socket environment , GLE =% d"), errorcode );
5. \_ tprintf (\_T(" Unable to Initialize Windows Socket environment , GLE =% d"), errorcode );
6. closelog ();
7. exit (1) ;

63 }

64 writelog (\_T(" Windows Socket environment ready "));

### 65

1. *// Созда e¨ м порт завершения:*
2. if (( Completion Port = Create Io Completion Port ( INVALID\_ HANDLE\_ VALUE , NULL , 0 , 0)) == NULL ){
3. double errorcode = WSAGet Last Error ();
4. writelog (\_T(" Create Io Completion Port failed , GLE =% d"), errorcode );
5. \_ tprintf (\_T(" Create Io Completion Port failed , GLE =% d"), errorcode );
6. closelog ();
7. exit (2) ;

73 }

74 writelog (\_T(" Io Completion Port created "));

### 75

1. *// Получаем информацию о системы:*
2. Get System Info (& Sys Info );
3. *// созда e¨ м два потока на процессор:.*
4. for ( size\_ t i = 0; i < Sys Info . dw Number Of Processors \* 2; i ++) {
5. *// созда e¨ м рабочий поток, в качестве параметра переда e¨ м ей порт завершен ия*
6. if (( Thread = Create Thread ( NULL , 0 , Client Thread , Completion Port , 0 , & Thread ID )) == NULL ) {
7. double errorcode = WSAGet Last Error ();
8. writelog (\_T(" Create Thread () failed , GLE =% d"), errorcode );
9. \_ tprintf (\_T(" Create Thread () failed , GLE =% d"), errorcode );
10. closelog ();
11. exit (3) ;

87 }

88 Close Handle ( Thread );

89 }

### 90

1. *// Созда e¨ м слушающий сокет:*
2. if (( Socket = WSASocket ( AF\_INET , SOCK\_ STREAM , 0 , NULL , 0 , WSA\_ FLAG\_ OVERLAPPED )) == INVALID\_ SOCKET ) {
3. double errorcode = WSAGet Last Error ();
4. writelog (\_T(" WSASocket () failed , GLE =% d"), errorcode );
5. \_ tprintf (\_T(" WSASocket () failed , GLE =% d"), errorcode );
6. closelog ();
7. exit (4) ;

98 }

### 99

1. server . sin\_ family = AF\_ INET ;
2. server . sin\_ addr . s\_ addr = htonl ( INADDR\_ ANY );
3. server . sin\_ port = htons ( n Server Port );

### 103

1. if ( bind ( Socket , ( PSOCKADDR )& server , sizeof ( server )) == SOCKET\_ ERROR ){
2. double errorcode = WSAGet Last Error ();
3. writelog (\_T(" bind () failed , GLE =% d"), errorcode );
4. \_ tprintf (\_T(" bind () failed , GLE =% d"), errorcode );
5. closelog ();
6. exit (5) ;

### 110 }

### 111

1. if ( listen ( Socket , 5) == SOCKET\_ ERROR ){
2. double errorcode = WSAGet Last Error ();
3. writelog (\_T(" listen () failed , GLE =% d"), errorcode );
4. \_ tprintf (\_T(" listen () failed , GLE =% d"), errorcode );
5. closelog ();
6. exit (6) ;

### 118 }

1. writelog (\_T(" Ready for connection "));
2. \_ tprintf (\_T(" Ready for connection \ n"));

### 121

1. *// принимаем соединения и переда e¨ м их порту завершения:*
2. while ( TRUE ){
3. *// принимаем соединение:*
4. if (( Accept = WSAAccept ( Socket , ( PSOCKADDR )& client , NULL , NULL , 0)) == SOCKET\_ ERROR ){
5. double errorcode = WSAGet Last Error ();
6. writelog (\_T(" WSAAccept () failed , GLE =% d"), errorcode );
7. \_ tprintf (\_T(" WSAAccept () failed , GLE =% d"), errorcode );
8. continue ;

### 130 }

### 131

1. *// Выделяем память под структуру, которая будет хранить информацию о сок ете:*
2. if (( Per Handle Data = ( LPPER\_ HANDLE\_ DATA ) Global Alloc ( GPTR , sizeof ( PER\_ HANDLE\_ DATA ))) == NULL ){
3. double errorcode = WSAGet Last Error ();
4. writelog (\_T(" Global Alloc () failed with error % d"), errorcode );
5. \_ tprintf (\_T(" Global Alloc () failed with error % d"), errorcode );
6. closelog ();
7. exit (7) ;

### 139 }

1. writelog (\_T(" Socket % d connected \ n"), Accept );
2. \_ tprintf (\_T(" Socket % d connected \ n"), Accept );

### 142

143 Per Handle Data -> Socket = Accept ; *// сохраняем описатель сокета*

### 144

1. *// привязываем сокет к порту завершения:*
2. if ( Create Io Completion Port (( HANDLE ) Accept , Completion Port , ( DWORD ) Per Handle Data , 0) == NULL ){
3. double errorcode = WSAGet Last Error ();
4. writelog (\_T(" Create Io Completion Port () failed with error % d"), errorcode );
5. \_ tprintf (\_T(" Create Io Completion Port () failed with error % d"), errorcode );
6. closelog ();
7. exit (8) ;

### 152 }

### 153

1. *// выделяем память под данные операции ввода вывода:*
2. if (( Per Io Data = ( LPPER\_ IO\_ OPERATION\_ DATA ) Global Alloc ( GPTR , sizeof ( PER\_ IO\_ OPERATION\_ DATA ))) == NULL ){
3. double errorcode = WSAGet Last Error ();
4. writelog (\_T(" Global Alloc () failed with error % d"), errorcode );
5. \_ tprintf (\_T(" Global Alloc () failed with error % d"), errorcode );
6. closelog ();
7. exit (9) ;

### 161 }

### 162

163 Zero Memory (&( PerIoData -> Overlapped ), sizeof ( OVERLAPPED ));

### 164

1. *// зада e¨ м изначальные данные для операции ввода вывода:*
2. PerIoData -> Bytes Send = 0;
3. PerIoData -> Bytes Recv = 0;
4. PerIoData -> Data Buf . len = ( wcslen ( EMPTY\_ MSG ) + 1) \* sizeof ( \_ TCHAR );
5. PerIoData -> Data Buf . buf = ( char \*) EMPTY\_ MSG ;
6. PerIoData -> client = client ;
7. PerIoData -> Total Bytes = 0;

### 172

173 Flags = 0;

### 174

1. *// отправляем welcome message*
2. *// остальные операции будут выполняться в рабочем потоке*
3. if ( WSASend ( Accept , &( PerIoData -> Data Buf ), 1 , & SendBytes , 0 , &( PerIoData

-> Overlapped ), NULL ) == SOCKET\_ ERROR ){

1. if ( WSAGet Last Error () != ERROR\_ IO\_ PENDING ){
2. double errorcode = WSAGet Last Error ();
3. writelog (\_T(" WSASend () failed with error % d"), errorcode );
4. \_ tprintf (\_T(" WSASend () failed with error % d\ n"), errorcode );
5. closelog ();

183 exit (10) ;

### 184 }

### 185 }

### 186 }

1. closelog ();
2. exit (0) ;

### 189 }

### 190

1. DWORD WINAPI Client Thread ( LPVOID Completion Port ID ) {
2. HANDLE Completion Port = ( HANDLE ) Completion Port ID ;
3. DWORD Bytes Transferred ;
4. LPPER\_ HANDLE\_ DATA Per Handle Data ;
5. LPPER\_ IO\_ OPERATION\_ DATA Per Io Data ;

### 196

1. DWORD SendBytes , Recv Bytes ;
2. DWORD Flags ;

### 199

1. while ( TRUE ){
2. *// ожидание завершения ввода- вывода на любом из сокетов*
3. *// которые связанны с портом завершения:*
4. if ( Get Queued Completion Status ( Completion Port , & Bytes Transferred ,
5. ( LPDWORD )& Per Handle Data , ( LPOVERLAPPED \*)& PerIoData , INFINITE ) == 0){
6. double errorcode = Get Last Error ();
7. writelog (\_T(" WSASend () failed with error % d"), errorcode );
8. \_ tprintf (\_T(" WSASend () failed with error % d\ n"), errorcode );
9. return 0;

### 209 }

### 210

1. *// проверяем на ошибки. Если была - значит надо закрыть сокет и очистить память за собой:*
2. if ( Bytes Transferred == 0){
3. *// тк не было переданно ни одного байта - значит сокет закрыли на той стороне*
4. *// мы должны сделать то же самое:*

|  |  |  |
| --- | --- | --- |
| 215 |  | writelog (\_T(" Closing socket % d"), Per Handle Data -> Socket ); |
| 216 |  | writelog (\_T(" Total bytes :% d\ n"), PerIoData -> Total Bytes ); |
| 217 |  | \_ tprintf (\_T(" Closing socket % d\ n Total bytes :% d\ n"), Per Handle Data -> |
| 218 |  | Socket , PerIoData -> Total Bytes ); |
| 219 |  | *// закрываем сокет:* |
| 220 |  | if ( closesocket ( Per Handle Data -> Socket ) == SOCKET\_ ERROR ){ |
| 221 |  | double errorcode = WSAGet Last Error (); |
| 222 |  | writelog (\_T(" closesocket () failed with error % d"), errorcode ); |
| 223 |  | \_ tprintf (\_T(" closesocket () failed with error % d\ n"), errorcode ); |
| 224 |  | return 0; |
| 225 |  | } |
| 226 |  |  |
| 227 |  | *// очищаем память:* |
| 228 |  | Global Free ( Per Handle Data ); |
| 229 |  | Global Free ( Per Io Data ); |
| 230 |  |  |
| 231 |  | *// жд e¨ м следующую операцию* |
| 232 |  | continue ; |
| 233 | } |  |
| 234 |  |  |
| 235 | Pe | rIoData -> Total Bytes += Bytes Transferred ; |
| 236 |  |  |
| 237 | *//* | *Проверим значение Bytes Recv - если оно равно нулю - значит мы получил* |

*и данные от клиента:*

* 1. if ( PerIoData -> Bytes Recv == 0){
  2. PerIoData -> Bytes Recv = Bytes Transferred ;
  3. PerIoData -> Bytes Send = 0;

### 241 }

1. else {
2. PerIoData -> Bytes Send += Bytes Transferred ;

### 244 }

### 245

1. *// мы должны отослать все принятые байты назад:*
2. if ( PerIoData -> Bytes Recv > PerIoData -> Bytes Send ){
3. *// Шл e¨ м данные через WSASend - тк вс e¨*
4. *// необходимо слать до упора.*

*сразу может не отослаться*

1. *// Теоретически, за один вызов WSASend все данные могут не отправится!*
2. Zero Memory (&( PerIoData -> Overlapped ), sizeof ( OVERLAPPED ));

### 252

1. PerIoData -> Data Buf . buf = PerIoData -> Buffer + PerIoData -> Bytes Send ;
2. PerIoData -> Data Buf . len = PerIoData -> Bytes Recv - PerIoData -> Bytes Send ;

### 255

1. *// Convert the string to upper case and send it back , if its not QUIT*
2. if ( wcscmp (( \_ TCHAR \*) PerIoData -> Buffer , \_T(" QUIT ")) == 0) {
3. \_ tprintf (\_T(" RCV % s\ n"), PerIoData -> Buffer );
4. closesocket ( Per Handle Data -> Socket );
5. return TRUE ;

### 261 }

1. if ( WSASend ( Per Handle Data -> Socket , &( PerIoData -> Data Buf ), 1 , & SendBytes , 0 ,
2. &( PerIoData -> Overlapped ), NULL ) == SOCKET\_ ERROR ) {
3. if ( WSAGet Last Error () != ERROR\_ IO\_ PENDING ) {
4. double errorcode = WSAGet Last Error ();
5. writelog (\_T(" WSASend () failed with error % d"), errorcode );
6. \_ tprintf (\_T(" WSASend () failed with error % d\ n"), errorcode );
7. return 0;

### 269 }

### 270 }

### 271 }

1. else {
2. PerIoData -> Bytes Recv = 0;

### 274

1. *// ожидаем ещ e¨*
2. Flags = 0;

*данные от пользователя:*

1. Zero Memory (&( PerIoData -> Overlapped ), sizeof ( OVERLAPPED ));

### 278

1. PerIoData -> Data Buf . len = DATA\_ BUFSIZE ;
2. PerIoData -> Data Buf . buf = PerIoData -> Buffer ;

### 281

1. if ( WSARecv ( Per Handle Data -> Socket , &( PerIoData -> Data Buf ), 1 , & RecvBytes , & Flags ,
2. &( PerIoData -> Overlapped ), NULL ) == SOCKET\_ ERROR ){
3. if ( WSAGet Last Error () != ERROR\_ IO\_ PENDING ){
4. double errorcode = WSAGet Last Error ();
5. writelog (\_T(" WSARecv () failed with error % d"), errorcode );
6. \_ tprintf (\_T(" WSASend () failed with error % d\ n"), errorcode );
7. return 0;

### 289 }

### 290 }

1. writelog (\_T(" Get task from % d: % s"), Per Handle Data -> Socket , PerIoData

-> Buffer );

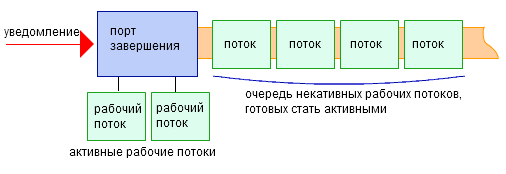
1. \_ tprintf (\_T(" Get task from % d: % s\ n"), Per Handle Data -> Socket , PerIoData -> Buffer );

### 293 }

### 294 }

### 295 }

## Объект порт, по сути, представляет собой очередь событий ядра, из которой извлекаются и в которую добавляются сообщения об операциях ввода/вывода. Туда добавляются не все текущие операции, а только те, которые указаны порту. Делается это путем связыва- ния дескриптора файла (сокета, именованного канала, мэйлслота и т.д.) с дескриптором порта. Когда над файлом инициируется асинхронная операция ввода/вывода, то после ее завершения соответствующая запись добавляется в порт.



## Рис. 12: Схема работы с портом завершения

## Для обработки результатов используется пул потоков, количество которых выбирается пользователем, и, как правило, коррелирует с количеством ядер центрального процессора. Когда поток присоединяют к пулу, он извлекает из очереди один результат операции и обрабатывает его. Если на момент присоединения очередь пуста, то поток засыпает до тех пор, пока не появится сообщение для обработки (см. рис. 12).

## Порт создаётся командой[5]

## HANDLE WINAPI CreateIoCompletionPort(

## \_In\_ HANDLE FileHandle,

## \_In\_opt\_ HANDLE ExistingCompletionPort,

## \_In\_ ULONG\_PTR CompletionKey,

## \_In\_ DWORD NumberOfConcurrentThreads

## );

## Параметрами являются:

## FileHandle – дескриптор файла открывается для завершения асинхронной операции ввода-вывода (Если FileHandle установлен в INVALID\_HANDLE\_VALUE, функция CreateIoCompletionPort создает порт завершение ввода - вывода, не связывая его с файлом);

## ExistingCompletionPort – дескриптор порта завершения ввода - вывода (Если этот параметр определяет существующий порт завершения I/O, функция связывает его с файлом, указанным параметром FileHandle);

## CompletionKey – код завершения для каждого файла, который включается в каждый блок завершения ввода - вывода для указанного файла;

## NumberOfConcurrentThreads – максимальное число потоков, которым операционная система дает возможность одновременно работать с блоками завершения ввода - вывода для порта завершения ввода - вывода.

## Если функция завершается успешно, возвращаемое значение - дескриптор порта завершения ввода-вывода (I/O), который связан с указанным файлом.

## Теперь рассмотрим функцию API, которая присоединяет вызывающий ее поток к пулу[6]:

## BOOL WINAPI GetQueuedCompletionStatus(

## \_In\_ HANDLE CompletionPort,

## \_Out\_ LPDWORD lpNumberOfBytes,

## \_Out\_ PULONG\_PTR lpCompletionKey,

## \_Out\_ LPOVERLAPPED \*lpOverlapped,

## \_In\_ DWORD dwMilliseconds

## );

## Параметры:

## CompletionPort – дескриптор ранее созданного порта завершения;

## lpNumberOfBytes – указатель на переменную, которая получает число байтов, пере- мещенных в ходе операции ввода-вывода (I/O), которая завершилась;

## lpCompletionKey – указатель на переменную, которая получает значение кода завер- шения, связанного с дескриптором файла, операция ввода-вывода (I/O) которого завершилась;

## lpOverlapped – указатель на переменную-буфер;

## dwMilliseconds –число миллисекунд, которое вызывающая программа будет ждать пакет завершения, чтобы появиться в порте завершения.

## Если функция исключает из очереди пакет окончания работы в следствие успешной операции I/O порта завершения, возвращаемое значение – не нуль.

## Теперь рассмотрим внутреннюю структура порта завершения. Фактически, он представляет собой следующую структуру:

## typedef stuct \_IO\_COMPLETION { KQUEUE Queue; } IO\_COMPLETION;

## Это просто очередь событий ядра. Вот описание структуры KQUEUE:

## typedef stuct \_KQUEUE { DISPATCHER\_HEADER Header;

## LIST\_ENTRY EnrtyListHead; //очередь пакетов DWORD CurrentCount;

## DWORD MaximumCount;

## LIST\_ENTRY ThreadListHead; //очередь ожидающих потоков

## } KQUEUE;

## При создании порта функцией CreateIoCompletionPort вызывается внутренний сервис NtCreateIoCompletion. Затем происходит его инициализация с помощью функции KeInitializeQueue. Когда происходит связывание порта с объектом «файл», Win32-функция CreateIoCompletionPort вызывает NtSetInformationFile.

## NtSetInformationFile( HANDLE FileHandle,

## PIO\_STATUS\_BLOCK IoStatusBlock, PVOID FileInformation,

## ULONG Length,

## FILE\_INFORMATION\_CLASS FileInformationClass);

## Для этой функции FILE\_INFORMATION\_CLASS устанавливается как FileCompletionInformation, а в качестве параметра FileInformation передается указатель на структуру IO\_COMPLETION\_CON или FILE\_COMPLETION\_INFORMATION.

## typedef struct \_IO\_COMPLETION\_CONTEXT { PVOID Port;

## PVOID Key; } IO\_COMPLETION\_CONTEXT;

## typedef struct \_FILE\_COMPLETION\_INFORMATION { HANDLE IoCompletionHandle;

## ULONG CompletionKey; } FILE\_COMPLETION\_INFORMATION, \*PFILE\_COMPLETION\_INFORMATION

## После завершения асинхронной операции ввода/вывода для ассоциированного файла диспетчер ввода/вывода создает пакет запроса из структуры OVERLAPPED и ключа завершения и помещает его в очередь с помощью вызова KeInsertQueue. Когда поток вызывает функцию GetQueuedCompletionStatus, на самом деле вызывается функция NtRemoveIoCompletion. NtRemoveIoCompletion проверяет параметры и вызывает функцию

## KeRemoveQueue, которая блокирует поток, если в очереди отсутствуют запросы, или поле CurrentCount структуры KQUEUE больше или равно MaximumCount. Если запросы есть, и число активных потоков меньше максимального, KeRemoveQueue удаляет вызвавший ее поток из очереди ожидающих потоков и увеличивает число активных потоков на 1. При занесении потока в очередь ожидающих потоков поле Queue структуры KTHREAD устанавливается равным адресу порта завершения. Когда запрос помещается в порт за- вершения функцией PostQueuedCompletionStatus, на самом деле вызывается функция NtSetIoCompletion, которая после проверки параметров и преобразования хендла порта в указатель, вызывает KeInsertQueue.

## Листинги 21 и 22 содержат протокол работы программы. Стоит обратить внимание, что в отличии от предыдущего примера, когда клиент ожидал данные сразу после отправки, здесь клиент получает ответ асинхронно.

## Листинг 21: Протокол работы серверного модуля программы работы с портом завершения

1 [ 14 / 2 / 2015 14 : 36 : 25 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \

Inter Process Communication \ Debug \ Completion Port Server . exe is starting .

2

3

4

5

6

7

8

9

10

11

12

[ 14 / 2 / 2015

[ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015

14 : 36 : 25 ] Windows Socket environment ready

14 : 36 : 25 ] Io Completion Port created 14 : 36 : 26 ] Ready for connection 14 : 36 : 54 ] Socket 136 connected

[ 14 / 2 / 2015

[ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015

14 : 36 : 54 ]

Get task from 136:

14 : 37 : 1 ] Get task from 136: Hi!

14 : 37 : 6 ] Get task from 136: Salut ! 14 : 37 : 10 ]

14 : 37 : 12 ]

14 : 37 : 13 ]

Get

Get Get

task

task task

from

from from

136:

136:

136:

1

2

3

## Листинг 22: Протокол работы клиентского модуля программы работы с портом завершения

1 [ 14 / 2 / 2015 14 : 36 : 25 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \

Inter Process Communication \ Debug \ Win Sock Client . exe is starting .

2

3

4

5

6

7

8

9

10

[ 14 / 2 / 2015

[ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015 [ 14 / 2 / 2015

14 : 36 : 54 ] Windows Socket environment ready

14 : 36 : 54 ] Client socket created 14 : 36 : 54 ] Connect

14 : 37 : 1 ] Received ... from server

14 : 37 : 6 ] Received Hi! from server 14 : 37 : 10 ]

14 : 37 : 12 ]

14 : 37 : 13 ]

Received

Received Received

Salut !

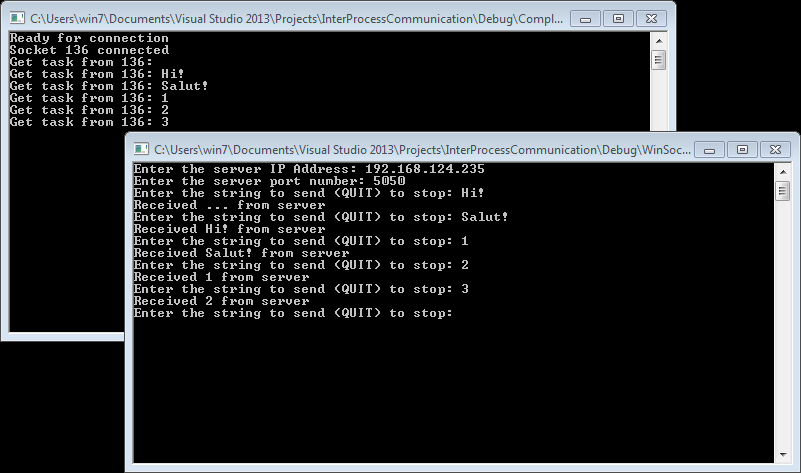
1. from
2. from

from server

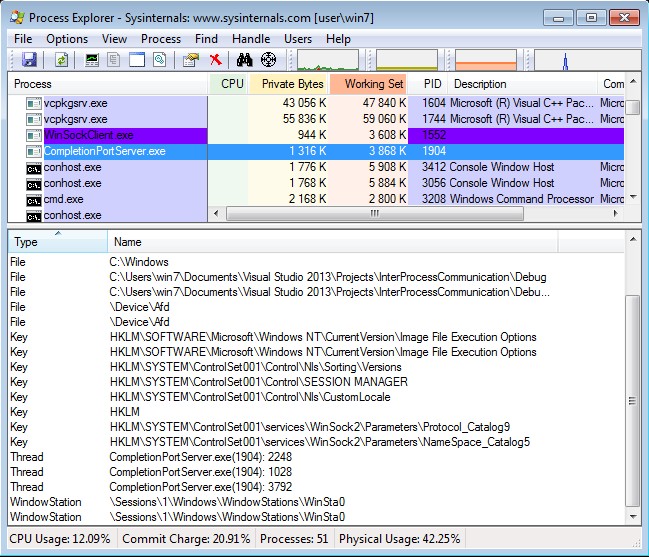
server server

14 : 39 : 1 ] Shutting down .

## Рисунок 13 показывает работу программы, а на рисунке 14 видны несколько процессов серверного модуля (в клиентском модуле никаких изменений, по сравнению с обычными сокетами нет).



## Рис. 13: Программа работы с портом завершения



## Рис. 14: Процессы серверного модуля

# Сигналы

## В отличии от Linux, сигналы в Windows имеют сильно усеченные возможности. Наиболее сложной задаче при работе с сигналами было придумать, что можно с ними сделать. В листинге 23 по сигналу меняется цвет консоли, это видно на рисунке 15.

## Листинг 23: Сигналы в Windows (src/InterProcessCommunication/Signals/main.cpp)

1. # include < windows . h>
2. # include < stdio . h>
3. # include < iostream >
4. # include " logger . h"

### 5

6 BOOL Ctrl Handler ( DWORD fdw Ctrl Type )

7 {

1. HANDLE h Stdout = Get Std Handle ( STD\_ OUTPUT\_ HANDLE );
2. if ( h Stdout == INVALID\_ HANDLE\_ VALUE )

10 {

1. std :: cout << " Error while getting input handle " << std :: endl ;
2. writelog (\_T(" Error while getting input handle "));
3. return EXIT\_ FAILURE ;

14 }

### 15

16 switch ( fdw Ctrl Type ) *// тип сигнала*

17 {

1. *// Handle the CTRL - C signal.*
2. case CTRL\_ C\_ EVENT :
3. Set Console Text Attribute ( hStdout , FOREGROUND\_ RED | BACKGROUND\_ BLUE | FOREGROUND\_ INTENSITY );
4. std :: cout << " Ctrl - C event \ n\ n" << std :: endl ;
5. writelog (\_T(" Ctrl - C event "));
6. Set Console Text Attribute ( hStdout , FOREGROUND\_ RED | FOREGROUND\_ GREEN | FOREGROUND\_ BLUE );
7. return ( TRUE );
8. *// CTRL - CLOSE: confirm that the user wants to exit.*
9. case CTRL\_ CLOSE\_ EVENT :
10. Set Console Text Attribute ( hStdout , FOREGROUND\_ RED | BACKGROUND\_ BLUE | FOREGROUND\_ INTENSITY );
11. std :: cout << " Ctrl - Close event \ n\ n" << std :: endl ;
12. writelog (\_T(" Ctrl - Close event "));
13. Set Console Text Attribute ( hStdout , FOREGROUND\_ RED | FOREGROUND\_ GREEN | FOREGROUND\_ BLUE );
14. return ( TRUE );
15. *// Pass other signals to the next handler.*
16. case CTRL\_ BREAK\_ EVENT :
17. Set Console Text Attribute ( hStdout , FOREGROUND\_ RED | BACKGROUND\_ BLUE | FOREGROUND\_ INTENSITY );
18. std :: cout << " Ctrl - Break event \ n\ n" << std :: endl ;
19. writelog (\_T(" Ctrl - Break event "));
20. Set Console Text Attribute ( hStdout , FOREGROUND\_ RED | FOREGROUND\_ GREEN | FOREGROUND\_ BLUE );
21. return FALSE ;
22. case CTRL\_ LOGOFF\_ EVENT :
23. Set Console Text Attribute ( hStdout , FOREGROUND\_ RED | BACKGROUND\_ BLUE | FOREGROUND\_ INTENSITY );
24. std :: cout << " Ctrl - Logoff event \ n\ n" << std :: endl ;
25. writelog (\_T(" Ctrl - Logoff event "));
26. Set Console Text Attribute ( hStdout , FOREGROUND\_ RED | FOREGROUND\_ GREEN | FOREGROUND\_ BLUE );
27. return FALSE ;
28. case CTRL\_ SHUTDOWN\_ EVENT :
29. Set Console Text Attribute ( hStdout , FOREGROUND\_ RED | BACKGROUND\_ BLUE | FOREGROUND\_ INTENSITY );
30. std :: cout << " Ctrl - Shutdown event \ n\ n" << std :: endl ;
31. writelog (\_T(" Ctrl - Shutdown event "));
32. Set Console Text Attribute ( hStdout , FOREGROUND\_ RED | FOREGROUND\_ GREEN | FOREGROUND\_ BLUE );
33. return FALSE ;
34. default :
35. return FALSE ;

53 }

54 }

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. *// Init log*
3. initlog ( argv [0]) ;

### 58

1. if ( Set Console Ctrl Handler (( PHANDLER\_ ROUTINE ) Ctrl Handler , TRUE )) {
2. std :: cout << " The Control Handler is installed ." << std :: endl
3. << " -- Now try pressing Ctrl + C or Ctrl + Break , or" << std :: endl
4. << " try logging off or closing the console ... " << std :: endl << std

:: endl

1. << " (... waiting in a loop for events ...) " << std :: endl << std :: endl ;
2. while (1) {}

65 }

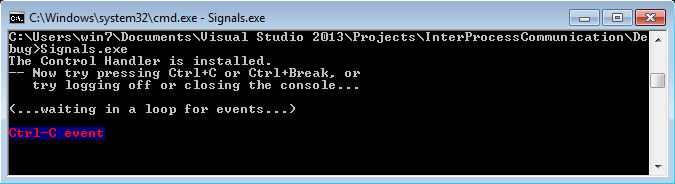
1. else {
2. std :: cout << " ERROR : Could not set control handler " << std :: endl ;
3. writelog (\_T(" ERROR : Could not set control handler "));
4. return EXIT\_ FAILURE ;

70 }

1. closelog ();
2. return EXIT\_ SUCCESS ;

73

}



## Рис. 15: Работа с сигналами в Windows

# Заключение

## В данной работе были рассмотрены основные механизмы межпроцессорного взаимодей- ствия, от самых простых, типа анонимных каналов, до самых сложных, таких как сокеты и порты завершения. Каждый механизм имеет свою нишу для использования.

## Анонимные каналы – достаточно слабый инструмент. Представляют собой полудуп- лексное средство потоковой передачи байтов между родственными процессами. Они функционируют в пределах локальной вычислительной системы и хорошо подходят для перенаправления выходного потока одной программы на вход другой. Минусом является то, что использование этого средства в некоторых случаях приводит к невозможности использования стандартных механизмов ввода/вывода (т.к. они уже заняты каналом).

## Именованные каналы – являясь объектами ядра ОС Windows, они позволяют органи- зовать межпроцессный обмен не только в изолированной вычислительной системе, но и в локальной сети. Они обеспечивают дуплексную связь и позволяют использовать как потоковую модель, так и модель, ориентированную на сообщения. Обмен данными может быть синхронным и асинхронным.

## Почтовые ящики – хотя этот механизм обеспечивают одностороннее взаимодействие процессов, любой процесс может выступать одновременно как в роли сервера, так и в роли клиента, вследствие чего возможно двустороннее межпроцессное взаимодействие (путем создания по крайней мере двух почтовых ящиков). Сообщения хранятся в почтовом ящике до тех пор, пока сервер их не прочтет.

## Shared memory – очень быстрый и очень опасный механизм взаимодействия. Общая память должна быть защищена каким-то механизмом типа блокера, в противном случае один процесс может испортить то, что перед этим записал другой.

## Сокеты – представляет собой независимый от протокола интерфейс, который да- ёт возможность использовать преимущества базовых протоколов. Сокет Windows Sockets 2 представляет собой дескриптор, который может быть использован и как

## дескриптор файла в стандартных файловых функциях ввода и вывода. Приложение, использующее сокеты Windows, может взаимодействовать с сокетами других ОС.

## Порты завершения – обеспечивают эффективную обработку асинхронных запросов ввода/вывода на многопроцессорных системах за счёт использования очередей и пула потоков обработки.

## Сигналы – самый слабый инструмент, которому сложно придумать практическое применение. Механизм сигналов в Windows значительно уступает аналогичному механизму из мира \*nix.

## Наиболее интересным средством взаимодействия оказался сокет. Он не имеет больших отличий от классического сокета Беркли, что упрощает его изучение. Работа в асинхронном режиме (порты завершения) оказывает драматическое влияние на скорость работы системы, и должна применяться в высоко нагруженных системах.

**Литература**

## Душутина Е.В. Межпроцессные взаимодействия в операционных системах. Учебное пособие – СПб.: 2014 – 135 стр.

## Джонсон М. Харт. Системное программирование в среде Windows – М.: Издательский дом "Вильямс 2005 – 587 стр.

## Александр Шаргин. Программирование сокетов в Linux – <http://rsdn.ru/article/unix/sockets.xml>

## Бердников Алексей: Использование портов завершения для клиент-серверных прило- жений. – <http://lexlunnyi.blogspot.ru/2013/08/blog-post.html>

## MSDN: CreateIoCompletionPort function – https://msdn.microsoft.com/en- us/aa363862.aspx

## MSDN: GetQueuedCompletionStatus function – https://msdn.microsoft.com/aa364986.aspx