

Выполнила: Белоусова Е., ИП-911

Задача

Цель: познакомиться с принципами обмена данными между процессами на основе неименованных и именованных каналов.

Задание:

- протестировать программы Лекции13.
- написать «чат» один-к-одному в локальной сети на основе именованных каналов.

Описание работы программы

Именованный канал — это именованный односторонний или дуплексный канал для обмена данными между сервером канала и одним или несколькими клиентами канала. Все экземпляры именованного канала имеют одинаковое имя канала, но каждый экземпляр имеет собственные буферы и дескрипторы, а также предоставляет отдельный канал для обмена данными между клиентом и сервером. Использование экземпляров позволяет нескольким клиентам каналов одновременно использовать один и тот же именованный канал.

Именованные каналы создаются процессом-сервером при помощи функции *CreateNamedPipe*, которая имеет следующий прототип:

```
HANDLE CreateNamedPipe (
    LPCTSTR      lpName,           // имя канала
    DWORD        dwOpenMode,       // атрибуты канала
    DWORD        dwPipeMode,       // режим передачи данных
    DWORD        nMaxInstances,    // максимальное количество экземпляров канала
    DWORD        nOutBufferSize,   // размер выходного буфера
    DWORD        nInBufferSize,    // размер входного буфера
    DWORD        nDefaultTimeout,  // время ожидания связи с клиентом
    LPSECURITY_ATTRIBUTES lpPipeAttributes // атрибуты защиты
);
```

После того, как сервер создал именованный канал, он должен дожидаться соединения клиента с этим каналом. Для этого сервер вызывает функцию

```
BOOL ConnectNamedPipe (
    HANDLE      hNamedPipe,        // дескриптор канала
    LPOVERLAPPED lpOverlapped      // асинхронная связь
);
```

которая возвращает значение TRUE в случае успеха или значение FALSE в случае неудачи. Сервер может использовать эту функцию для связи с клиентом по каждому новому экземпляру именованного канала.

После окончания обмена данными с клиентом, сервер может вызвать функцию

```
BOOL DisconnectNamedPipe (
    HANDLE      hNamedPipe        // дескриптор канала
);
```

Протестируем примеры.

file:///D:/семестр%205/os/лаб13/1.html

| | |
|-----|-----------|
| 0 | 0 |
| 0.1 | 0.0998334 |
| 0.2 | 0.198669 |
| 0.3 | 0.29552 |
| 0.4 | 0.389418 |
| 0.5 | 0.479426 |
| 0.6 | 0.564642 |
| 0.7 | 0.644218 |

PS D:\семестр 5\os\лаб13> ./np2.exe
PS D:\семестр 5\os\лаб13> █

PS D:\семестр 5\os\лаб13> ./np1.exe
A message from client
PS D:\семестр 5\os\лаб13> █

Написать «чат» один-к-одному в локальной сети на основе именованных каналов

```
PS D:\семестр 5\os\лаб13> ./1.exe
Name of server: 192.168.1.152
192.168.1.152 looking forward to your messages.
(Type "Quit" to leave chat)
hey
192.168.1.152 message:  hello
how r u
?
192.168.1.152 message:  good
192.168.1.152 message:  and u?
goood
bye
Quit
Для продолжения нажмите любую клавишу . . . █
```

E:\2.exe

```
Waiting for the client to connect.
The client is connected! Waiting for your messages...
    (Type "Quit" to leave the chat)
Client's message:  hey
hello
Client's message:  how r u
Client's message:  ?
good
and u?
Client's message:  gooood
Client's message:  bye
The client has left the chat.
Для продолжения нажмите любую клавишу . . .
```

cmd. Командная строка

```
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.
C:\Users\79965>ipconfig

Настройка протокола IP для Windows

Адаптер Ethernet Ethernet:

    DNS-суффикс подключения . . . . . : sib.mts.ru
    IPv6-адрес. . . . . : fd01::6572:e3ba:1733:9b90
    Временный IPv6-адрес. . . . . : fd01::f010:ac37:5e98:97db
    Локальный IPv6-адрес канала . . . : fe80::6572:e3ba:1733:9b90%6
    IPv4-адрес. . . . . : 192.168.1.152
```

Листинг

//1.c

```
#include <windows.h>
```

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
int main() {
```

```
    char c;
```

```
    HANDLE hPipe;
```

```
    char MessageIn[256] = "", MessageOut[256] = "", machineName[80], pipeName[80],
    MessageIn1[256] = "";
```

```
    DWORD BytesWrite, BytesRead;
```

```
    int delayedQuit = 0;
```

```
    printf("Name of server: ");
```

```
    scanf("%s", &machineName);
```

```
    sprintf(pipeName, "\\\\"%s\\pipe\\MyPipe", machineName);
```

```

hPipe = CreateFile(
    pipeName,
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL);

if (hPipe == INVALID_HANDLE_VALUE) {
    printf("CreatePipe failed: error code %d\n", (int)GetLastError());
    return 0;
}

printf("%s looking forward to your messages.\n \t (Type \"Quit\" to leave chat)\n",
machineName);

while (1) {
    if (!ReadFile(
        hPipe,
        MessageIn,
        sizeof(MessageIn),
        &BytesRead,
        NULL)) {
        printf("%s was close.\nPress any key to out.", machineName);
        break;
    }
    else {
        if (strcmp(MessageIn1, MessageIn) != 0) {
            printf("%s message: %s\n", machineName, MessageIn);
            memset(MessageIn1, 0, 256);
            strcpy(MessageIn1, MessageIn);
        }
    }
}

```

```

        if (kbhit()) {
            gets(MessageOut);
            if (strcmp(MessageOut,"Quit") == 0)
            {
                sprintf(MessageOut,"<Quit>");
                delayedQuit = 1;
            }
        }

        WriteFile(
            hPipe,
            MessageOut,
            sizeof(MessageOut),
            &BytesWrite,
            NULL);

        if (delayedQuit)
        {
            break;
        }

    }

    //scanf("%c", &c);
    CloseHandle(hPipe);
    system("PAUSE");
    return 0;
}

//2.c
#include <stdio.h>
#include <windows.h>
#include <conio.h>
#include <string.h>

```

```

int main(){
    HANDLE hPipe;

    char MessageIn[256]="", MessageOut[256]="",MessageIn1[256]="";

    DWORD BytesRead, BytesWrite;


    SECURITY_ATTRIBUTES sa;
    SECURITY_DESCRIPTOR sd;


    sa.nLength = sizeof(sa);
    sa.bInheritHandle = FALSE;
    InitializeSecurityDescriptor(&sd,SECURITY_DESCRIPTOR_REVISION);
    SetSecurityDescriptorDacl(&sd,TRUE,NULL,FALSE);
    sa.lpSecurityDescriptor = &sd;


    hPipe = CreateNamedPipe(
        "\\.\pipe\\MyPipe",
        PIPE_ACCESS_DUPLEX,
        PIPE_TYPE_MESSAGE| PIPE_WAIT,
        1,
        0,
        0,
        INFINITE,
        &sa);


    if (hPipe == INVALID_HANDLE_VALUE){
        printf("CreatePipe failed: error code %d\n", (int)GetLastError());
        return 0;
    }


    printf("Waiting for the client to connect.\n");
    if ((ConnectNamedPipe(hPipe,NULL)) == 0){
        printf("Client could not connect\n");
        return 0;
    }
}

```

```

    }
    else
        printf("The client is connected! Waiting for your messages...\n \t (Type \"Quit\" to leave
the chat)\n");
    while(1){

        if(kbhit()){
            gets(MessageOut);
            if (strcmp(MessageOut,"Quit") == 0)
            {
                break;
            }
        }

        if (MessageOut != NULL) WriteFile(
            hPipe,
            MessageOut,
            sizeof(MessageOut),
            &BytesWrite,
            NULL);

        ReadFile(
            hPipe,
            MessageIn,
            sizeof(MessageIn),
            &BytesRead,
            NULL);

        if (strcmp(MessageIn1, MessageIn) != 0){
            if (strcmp(MessageIn, "<Quit>") == 0)
            {
                printf("The client has left the chat.\n");
                break;
            }
        }
    }
}

```

```
        printf("Client's message: %s\n",MessageIn);
        memset(MessageIn1,0,256);
        strcpy(MessageIn1,MessageIn);
    }

}

CloseHandle(hPipe);
system("PAUSE");
return 0;
}
```