

Лекция 7. Межпроцессное взаимодействие

Операционные системы

18 ноября 2016 г.

Обзор средств

Средства межпроцессного взаимодействия

- Общие объекты ядра: файлы, примитивы синхронизации, ...;
- Общая память;
- Неименованные каналы;
- Именованные каналы.

Объекты синхронизации

Windows API	POSIX
События	Условные переменные
	Мьютексы
	Семафоры

Таблица 1: объекты синхронизации, используемые для взаимодействия между процессами

Способы доступа к объектам ядра в других процессах

Средства межпроцессного обмена дескрипторами

- Именованные объекты;
- Наследование дескрипторов дочерними процессами;
- Дублирование дескрипторов (`DuplicateHandle()`).

События Windows API

Windows API CreateEvent(), OpenEvent()

```
HANDLE WINAPI CreateEvent(  
    _In_opt_ LPSECURITY_ATTRIBUTES lpEventAttributes,  
    _In_      BOOL                  bManualReset,  
    _In_      BOOL                  bInitialState,  
    _In_opt_ LPCTSTR                lpctszName  
);
```

```
HANDLE WINAPI OpenEvent(  
    _In_      DWORD                dwDesiredAccess,  
    _In_      BOOL                  bInheritHandle,  
    _In_      LPCTSTR                lpctszName  
);
```

Мьютексы Windows API

Windows API CreateMutex(), OpenMutex()

```
HANDLE WINAPI CreateMutex(  
    _In_opt_ LPSECURITY_ATTRIBUTES lpMutexAttributes,  
    _In_     BOOL                  bInitialOwner,  
    _In_opt_ LPCTSTR               lpctszName  
);
```

```
HANDLE WINAPI OpenMutex(  
    _In_     DWORD                 dwDesiredAccess,  
    _In_     BOOL                  bInheritHandle,  
    _In_     LPCTSTR               lpctszName  
);
```

Семафоры Windows API

Windows API CreateSemaphore(), OpenSemaphore()

```
HANDLE WINAPI CreateSemaphore(  
    _In_opt_ LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,  
    _In_     LONG                   lInitialCount,  
    _In_     LONG                   lMaximumCount,  
    _In_opt_ LPCTSTR                lpctszName  
);
```

```
HANDLE WINAPI OpenSemaphore(  
    _In_     DWORD                   dwDesiredAccess,  
    _In_     BOOL                    bInheritHandle,  
    _In_     LPCTSTR                lpctszName  
);
```

Использование разделяемых событий в Windows API

Пример (event.h)

```
#ifndef EVENT_H__
#define EVENT_H__

#define MY_IPC_EVENT_NAME \
    "event_CB44CFBF-52C3-487A-95A0-1233F5A4393C"

#define MY_IPC_MUTEX_NAME \
    "mutex_0DF33C8C-71FF-4358-B10A-AD7F0B7484F7"

#endif    // EVENT_H__
```


Использование разделяемых событий (продолжение)

Пример (main_parent.cpp, продолжение)

```
#include "event.h"
// ...

int main()
{
    HANDLE hEvent = CreateEvent(
        NULL,                    // lpEventAttributes
        FALSE,                   // bManualReset
        FALSE,                   // bInitialState
        _T(MY_IPC_EVENT_NAME));  // lpctszName
    //
    // ...
}
```

Использование разделяемых событий (окончание)

Пример (main_child.cpp, окончание)

```
#include "event.h"
// ...

int main()
{
    HANDLE hEvent = OpenEvent(
        SYNCHRONIZE,           // dwDesiredAccess
        FALSE,                 // bInheritHandle
        _T(MY_IPC_EVENT_NAME)); // lpctszName
    //
    // ...
}
```

Семафоры POSIX

POSIX sem_init()

```
#include <fcntl.h>
#include <sys/stat.h>
#include <semaphore.h>

int      sem_init(
    sem_t *pSem, int nPShared, unsigned int uValue);
```

Семафоры POSIX (окончание)

POSIX sem_open()

```
sem_t *sem_open(  
    const char *pcszName, int nOFlag,  
    /* unsigned long ulMode, unsigned int uValue */ ...);
```

nOFlag		ulMode			
O_CREAT	S_IRWXU	S_IRUSR	S_IWUSR	S_IXUSR	
O_EXCL	S_IRWXG	S_IRGRP	S_IWGRP	S_IXGRP	
	S_IRWXO	S_IROTH	S_IWOTH	S_IXOTH	

Таблица 2: возможные значения флагов параметров функции sem_open()

Мьютексы POSIX

POSIX pthread_mutex_init() и т. д.

```
int pthread_mutex_init(  
    pthread_mutex_t      *pMutex,  
    const pthread_mutexattr_t *pCAttr);
```

Мьютексы POSIX (окончание)

```
pthread_mutexattr_init()
```

```
int pthread_mutexattr_init(  
    pthread_mutexattr_t    *pAttr);
```

```
int pthread_mutexattr_destroy(  
    pthread_mutexattr_t    *pAttr);
```

```
#ifdef _POSIX_THREAD_PROCESS_SHARED  
int pthread_mutexattr_setpshared(  
    pthread_mutexattr_t    *pAttr,  
    int                     nPShared);  
#endif
```

nPShared

PTHREAD_PROCESS_PRIVATE

PTHREAD_PROCESS_SHARED

Таблица 3: значения параметра
функции
pthread_mutexattr_setpshared()

Условные переменные POSIX

POSIX pthread_condattr_init() и т. д.

```
int pthread_condattr_init(
    pthread_condattr_t      *pAttr);

int pthread_condattr_destroy(
    pthread_condattr_t      *pAttr);

#ifdef _POSIX_THREAD_PROCESS_SHARED
int pthread_condattr_setpshared(
    pthread_condattr_t      *pAttr,
    int                      nPShared);
#endif
```

Разделяемая память POSIX

POSIX shmget()

```
#include <sys/types.h>
```

```
#include <sys/shm.h>
```

```
int shmget(key_t nKey, int nSize, int nShmFlg);
```

nKey	nShmFlg
IPC_PRIVATE	IPC_CREAT
	IPC_EXCL
	младшие 9 бит

Таблица 4: возможные значения флагов параметров функции shmget()

Разделяемая память POSIX (продолжение)

POSIX shmat(), shmdt()

```
#include <sys/types.h>
```

```
#include <sys/shm.h>
```

```
void *shmat(int nShmId, const void *pvShmAddr, int nShmFlg);
```

```
int shmdt(const void *pvShmAddr);
```

nShmFlg
SHM_RND (SHMLBA)
SHM_RDONLY

Таблица 5: возможные значения флагов параметров функции shmat()

Разделяемая память POSIX (окончание)

Пример

```
const key_t g_cKey = 1917;
// ...
int nShmId = shmget(g_cKey, sizeof (struct connect), IPC_CREAT | 0644);
if (nShmId < 0)
{
    perror("shmget");
    exit(1);
}
struct connect *pConnect = (struct connect *) shmat(nShmId, NULL, 0);
// ...
shmdt(pConnect);
```

Решение проблемы дублирования ключей

Варианты

- Использование в качестве ключа константы `IPC_PRIVATE`.
- Генерирование ключа при помощи функции `ftok()`.

POSIX `ftok()`

```
#include <sys/types.h>
```

```
#include <sys/ipc.h>
```

```
key_t ftok(const char *pszPathName, int nProjId);
```

Разделяемая память Windows API

Windows API CreateFileMapping()

```
HANDLE WINAPI CreateFileMapping(  
    _In_      HANDLE          hFile,  
    _In_opt_ LPSECURITY_ATTRIBUTES lpAttributes,  
    _In_      DWORD           dwProtect,  
    _In_      DWORD           dwMaximumSizeHigh,  
    _In_      DWORD           dwMaximumSizeLow,  
    _In_opt_ LPCTSTR          lpctszName  
);
```

dwProtect

PAGE_READONLY

PAGE_READWRITE

PAGE_WRITECOPY

PAGE_EXECUTE_READ

...

Таблица 6:
возможные
значения флагов
параметров
функции

Разделяемая память Windows API (окончание)

Windows API MapViewOfFile(), UnmapViewOfFile()

```
LPVOID WINAPI MapViewOfFile(  
    _In_     HANDLE           hFileMappingObject,  
    _In_     DWORD            dwDesiredAccess,  
    _In_     DWORD            dwFileOffsetHigh,  
    _In_     DWORD            dwFileOffsetLow,  
    _In_     SIZE_T           dwNumberOfBytesToMap  
);  
  
BOOL WINAPI UnmapViewOfFile(  
    _In_     LPCVOID          lpcvBaseAddress  
);
```

dwDesiredAccess

FILE_MAP_READ

FILE_MAP_WRITE

FILE_MAP_COPY

...

Таблица 7:

возможные
значения флагов
параметров
функции

Общая обработка ошибок Windows API

Пример (win_assert.h)

```
#ifndef WIN_ASSERT_H__
#define WIN_ASSERT_H__

#include <windef.h>    // LPCTSTR

void WinAssert(bool bSuccess, LPCTSTR lpctszMessage);

#endif    // WIN_ASSERT_H__
```

Общая обработка ошибок Windows API (продолжение)

Пример (win_assert.cpp)

```
#include <cstdio>
#include <cstdlib>

#include <windows.h>
#include <tchar.h>

using namespace std;
```

Пример (win_assert.cpp, продолжение)

```
void WinAssert(
    bool bSuccess, LPCTSTR lpctszMessage)
{
    if (bSuccess)
        return;
    //
    DWORD dwError = GetLastError();
    //
```

Общая обработка ошибок Windows API (продолжение)

Пример (win_assert.cpp, продолжение)

```
LPVOID lpMsgBuf;  
FormatMessage(  
    FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,  
    NULL, dwError,  
    MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),  
    (LPTSTR) &lpMsgBuf, 0, NULL);  
//  
static TCHAR s_tszMsg[1024], s_tszOem[1024];  
_stprintf(  
    s_tszMsg,  
    _T("%s - %08Xh: %s\n"),  
    lpctszMessage, dwError, (LPCTSTR) lpMsgBuf);
```


Общая обработка ошибок Windows API (окончание)

Пример (win_assert.cpp, окончание)

```
//  
CharTo0em(s_tszMsg, s_tsz0em);  
_tprintf(s_tsz0em);  
//  
exit(-1);  
}    // WinAssert()
```

Общая память Windows API

Пример (map_parent.cpp)

```
#include <iostream>
#include <cstring>

#include <windows.h>
#include <tchar.h>

#include "map_name.h"
#include "win_assert.h"
```

Пример (map_parent.cpp, продолжение)

```
using namespace std;

int main()
{
    char szData[] = "Sample data";
    //
```

Общая память Windows API (продолжение)

Пример (map_parent.cpp, продолжение)

```
HANDLE hMapping = CreateFileMapping(  
    INVALID_HANDLE_VALUE,           // hFile  
    NULL,                           // lpAttributes  
    PAGE_READWRITE,                 // dwProtect  
    0,                               // dwMaximumSizeHigh  
    sizeof (szData),                 // dwMaximumSizeLow  
    _T(MY_IPC_MAP_NAME));           // lpctszName  
//  
WinAssert(  
    hMapping != NULL,  
    _T("Ошибка создания общей памяти в род. процессе"));  
//
```

Общая память Windows API (продолжение)

Пример (map_parent.cpp, продолжение)

```
LPVOID lpvData = MapViewOfFile(  
    hMapping,                                // hFileMappingObject  
    FILE_MAP_WRITE,                          // dwDesiredAccess  
    0,                                       // dwFileOffsetHigh  
    0,                                       // dwFileOffsetLow  
    0);                                     // dwNumberOfBytesToMap  
//  
WinAssert(  
    lpvData != NULL,  
    _T("Ошибка подключения общей памяти в род. процессе"));  
//  
memcpy(lpvData, szData, sizeof (szData));  
//
```

Общая память Windows API (продолжение)

Пример (map_parent.cpp, продолжение)

```
STARTUPINFO startup_info =  
{  
    sizeof (STARTUPINFO),  
    0  
};  
//  
PROCESS_INFORMATION process_info =  
{  
    INVALID_HANDLE_VALUE,           // hProcess  
    INVALID_HANDLE_VALUE,          // hThread  
    0,                             // dwProcessId  
    0                              // dwThreadId  
};
```

Общая память Windows API (продолжение)

Пример (map_parent.cpp, продолжение)

```
//  
BOOL bSuccess = CreateProcess(  
    NULL,                                // lpctszApplicationName  
    _T("map_child"),                    // lpctszCommandLine  
    NULL,                                // lpProcessAttributes  
    NULL,                                // lpThreadAttributes  
    FALSE,                               // bInheritHandles  
    0,                                   // dwCreationFlags  
    NULL,                                // lpEnvironment  
    NULL,                                // lpctszCurrentDirectory  
    &startup_info,                       // lpStartupInfo  
    &process_info);                     // lpProcessInformation  
//
```

Общая память Windows API (продолжение)

Пример (map_parent.cpp, окончание)

```
WinAssert(
    bSuccess,
    _T("Ошибка запуска дочернего процесса map_child"));
//
WaitForSingleObject(process_info.hProcess, INFINITE);
//
UnmapViewOfFile(lpvData);
//
CloseHandle(hMapping);
CloseHandle(process_info.hProcess);
CloseHandle(process_info.hThread);
//
}    // main()
```

Общая память Windows API (продолжение)

Пример (map_child.cpp)

```
#include <iostream>

#include <windows.h>
#include <tchar.h>

#include "map_name.h"
#include "win_assert.h"

using namespace std;
```


Общая память Windows API (продолжение)

Пример (map_child.cpp, продолжение)

```
int main()
{
    HANDLE hMapping = CreateFileMapping(
        INVALID_HANDLE_VALUE,           // hFile
        NULL,                           // lpAttributes
        PAGE_READWRITE,                 // dwProtect
        0,                               // dwMaximumSizeHigh
        12,                              // dwMaximumSizeLow
        _T(MY_IPC_MAP_NAME));           // lpctszName
    //
    WinAssert(
        hMapping != NULL,
        _T("Ошибка открытия общей памяти в дочернем процессе"));
```

Общая память Windows API (продолжение)

Пример (map_child.cpp, продолжение)

```
//
LPVOID lpvData = MapViewOfFile(
    hMapping,                                // hFileMappingObject
    FILE_MAP_WRITE,                          // dwDesiredAccess
    0,                                        // dwFileOffsetHigh
    0,                                        // dwFileOffsetLow
    0);                                       // dwNumberOfBytesToMap
//
WinAssert(
    lpvData != NULL,
    _T("Ошибка подключения памяти в дочернем процессе"));
```

Общая память Windows API (продолжение)

Пример (map_child.cpp, окончание)

```
//  
cout  
    << "map_child reports: "  
    << reinterpret_cast <const char *> (lpvData) << endl;  
//  
UnmapViewOfFile(lpvData);  
//  
CloseHandle(hMapping);  
//  
}    // main()
```

Общая память Windows API (окончание)

Пример (компиляция и запуск)

```
> g++ -o map_parent map_parent.cpp win_assert.cpp
```

```
> g++ -o map_child map_child.cpp win_assert.cpp
```

```
> map_parent
```

```
map_child reports: Sample data
```

Каналы POSIX

POSIX pipe()

```
#include <unistd.h>    // <io.h>

int    pipe(int anFD[2]);    // anFD[0] – чтение,
                             // anFD[1] – запись

int    dup2(int nOldFD, int nNewFD);
ssize_t read(int nFD, void *pvBuf, size_t uCount);
ssize_t write(int nFD, const void *pcvBuf, size_t uCount);
int    eof(int nFD);
int    close(int nFD);
```

Использование каналов в POSIX

Пример

```
$ ls | more
```

Реализация bash

- 1 pipe(anFD)
- 2 fork() (2 раза)
- 3 close(anFD[i])
(2 раза)

Реализация ls

- 1 dup2(anFD[1], 1)
- 2 close(anFD[i])
(2 раза)
- 3 execve(
"ls", argv, envp)

Реализация more

- 1 dup2(anFD[0], 0)
- 2 close(anFD[i])
(2 раза)
- 3 execve(
"more", argv, envp)

Высокоуровневое управление каналами в POSIX

POSIX popen(), pclose()

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

```
FILE * popen(const char *pcszCommand, const char *pcszType);  
int    pclose(FILE *stream);
```

```
FILE * fdopen(int nFD, const char *pcszMode);  
size_t fread(  
    void *pvBuf, size_t uSize, size_t uCount, FILE *stream);  
size_t fwrite(  
    const void *pcvBuf, size_t uSize, size_t uCount, FILE *stream);  
int    feof(FILE *stream);  
int    fclose(FILE *stream);
```

Реализация popen()

Родительский процесс

- 1 pipe(anFD)
- 2 fork()
- 3 **если** pcszType ~ "r", **то**
 | close(anFD[1])
иначе *// pcszType ~ "w"*
 | close(anFD[0])
- 4 **если** pcszType ~ "r", **то**
 | **вернуть** fdopen(anFD[0], pcszType)
иначе *// pcszType ~ "w"*
 | **вернуть** fdopen(anFD[1], pcszType)

Реализация `pipe()` (окончание)

Дочерний процесс

- 1 **если** `pcszType ~ "r"`, **то**
 `dup2(anFD[1], 1)`
 иначе
 `dup2(anFD[0], 0)`
- 2 `close(anFD[i])` (2 раза)
- 3 `execve(pcszCommand, argv, envp)`

// pcszType ~ "w"

Реализация `pclose()`

Родительский процесс

- 1 `wait4()` для дочернего процесса.

Работа с каналами

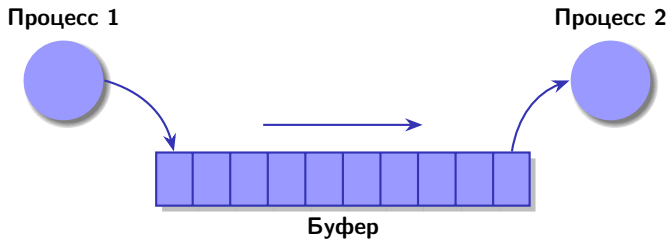


Рис. 1: принцип обмена данными при помощи канала

Именованные каналы POSIX

POSIX mkfifo()

```
#include <sys/types.h>  
#include <sys/stat.h>
```

```
int mkfifo(const char *pcszPathName, mode_t ulMode);
```

Каналы POSIX

Пример (fifo_server.c)

```
#include <sys/stat.h>
#include <fcntl.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define PIPE_NAME_1 "to-server"
#define PIPE_NAME_2 "from-server"
#define RESPONSE_ST "Response from server"
#define BUFFER_SIZE 50
```

Каналы POSIX (продолжение)

Пример (fifo_server.c, продолжение)

```
int main()
{
    int nResult, nFD1, nFD2;
    ssize_t i, nLength;
    char achBuffer[BUFFER_SIZE];
    const char *pcszResponse = RESPONSE_ST;
    //
```

Каналы POSIX (продолжение)

Пример (продолжение)

```
printf("Starting server...\n");  
if (unlink(PIPE_NAME_1) != 0)  
{  
    perror("Remove pipe 1");  
    exit(-1);  
}  
//
```

Пример (продолжение)

```
if (unlink(PIPE_NAME_2) != 0)  
{  
    perror("Remove pipe 2");  
    exit(-1);  
}  
//
```

Каналы POSIX (продолжение)

Пример (fifo_server.c, продолжение)

```
nResult = mkfifo(PIPE_NAME_1, S_IWUSR | S_IRUSR);
if (nResult != 0)
{
    perror("Create pipe 1");
    exit(-1);
}
//
nResult = mkfifo(PIPE_NAME_2, S_IWUSR | S_IRUSR);
if (nResult != 0)
{
    perror("Create pipe 2");
    exit(-1);
}
```


Каналы POSIX (продолжение)

Пример (продолжение)

```
//  
printf("Opening...\n");  
nFD1 = open(  
    PIPE_NAME_1, O_WRONLY);  
if (nFD1 < 0)  
{  
    perror("Open pipe 1");  
    exit(-1);  
}
```

Пример (продолжение)

```
//  
nFD2 = open(  
    PIPE_NAME_2, O_RDONLY);  
if (nFD2 < 0)  
{  
    perror("Open pipe 2");  
    exit(-1);  
}  
printf("Opened!\n");  
//
```

Каналы POSIX (продолжение)

Пример (fifo_server.c, продолжение)

```
do
{
    nLength = read(nFD2, achBuffer, BUFFER_SIZE);
    if (nLength < 0)
    {
        perror("read");
        exit(-1);
    }
    for (i = 0; i < nLength; ++ i)
        putchar(achBuffer[i]);
}
while (nLength == BUFFER_SIZE);
putchar('\n');
```

Каналы POSIX (продолжение)

Пример (fifo_server.c, продолжение)

```
//  
printf("Writing server...\n");  
write(nFD1, pcszResponse, strlen(pcszResponse));  
printf("Wrote server!\n");  
//  
close(nFD1);  
close(nFD2);  
}    // main()
```

Каналы POSIX (продолжение)

Пример (fifo_client.c)

```
#include <sys/stat.h>
#include <fcntl.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define PIPE_NAME_1 "to-server"
#define PIPE_NAME_2 "from-server"
#define RESPONSE_ST "Response from client"
#define BUFFER_SIZE 50
```

Каналы POSIX (продолжение)

Пример (fifo_client.c, продолжение)

```
int main()
{
    int nResult, nFD1, nFD2;
    ssize_t i, nLength;
    char achBuffer[BUFFER_SIZE];
    const char *pcszResponse = RESPONSE_ST;
    //
    printf("Starting client...\n");
    //
```

Каналы POSIX (продолжение)

Пример (продолжение)

```
printf("Opening...\n");
nFD2 = open(
    PIPE_NAME_1, O_RDONLY);
if (nFD2 < 0)
{
    perror("Open pipe 2");
    exit(-1);
}
//
```

Пример (продолжение)

```
nFD1 = open(
    PIPE_NAME_2, O_WRONLY);
if (nFD1 < 0)
{
    perror("Open pipe 1");
    exit(-1);
}
printf("Opened!\n");
//
```

Каналы POSIX (продолжение)

Пример (fifo_client.c, продолжение)

```
printf("Writing client...\n");  
write(nFD1, pcszResponse, strlen(pcszResponse));  
printf("Wrote client!\n");  
//
```

Каналы POSIX (продолжение)

Пример (fifo_client.c, продолжение)

```
do
{
    nLength = read(nFD2, achBuffer, BUFFER_SIZE);
    if (nLength < 0)
    {
        perror("read");
        exit(-1);
    }
    for (i = 0; i < nLength; ++ i)
        putchar(achBuffer[i]);
}
while (nLength == BUFFER_SIZE);
putchar('\n');
```


Каналы POSIX (продолжение)

Пример (fifo_client.c, окончание)

```
//  
close(nFD1);  
close(nFD2);  
}    // main()
```

Неименованные каналы Windows API

Windows API CreatePipe()

```
BOOL WINAPI CreatePipe(  
    _Out_      PHANDLE      phReadPipe,  
    _Out_      PHANDLE      phWritePipe,  
    _In_opt_   LPSECURITY_ATTRIBUTES lpPipeAttributes,  
    _In_       DWORD         dwSize  
);
```

Неименованные каналы Windows API (окончание)

Windows API WriteFile()

```
BOOL WINAPI WriteFile(  
    _In_      HANDLE          hFile,  
    _In_      LPCVOID         lpvBuffer,  
    _In_      DWORD           dwNumberOfBytesToWrite,  
    _Out_opt_ LPDWORD         lpdwNumberOfBytesWritten,  
    _Inout_opt_ LPOVERLAPPED lpOverlapped  
);
```

Каналы Windows API

Пример (pipe_parent.cpp)

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

#include "win_assert.h"

int main()
{
    HANDLE hReadPipe = INVALID_HANDLE_VALUE;
    HANDLE hWritePipe = INVALID_HANDLE_VALUE;
```

Каналы Windows API (продолжение)

Пример (pipe_parent.cpp, продолжение)

```
SECURITY_ATTRIBUTES security_attributes =  
{  
    sizeof (SECURITY_ATTRIBUTES),  
    NULL,                                     // lpSecurityDescriptor  
    TRUE                                    // bInheritHandle  
};  
BOOL bSuccess = CreatePipe(  
    &hReadPipe,                               // phReadPipe  
    &hWritePipe,                             // phWritePipe  
    &security_attributes,                     // lpPipeAttributes  
    05);                                     // dwSize  
WinAssert(bSuccess, _T("Ошибка создания канала"));  
//
```

Каналы Windows API (продолжение)

Пример (pipe_parent.cpp, продолжение)

```
SetHandleInformation(  
    hWritePipe,                                // hObject  
    HANDLE_FLAG_INHERIT,                       // dwMask  
    0);                                         // dwFlags  
//  
HANDLE hOutput = GetStdHandle(STD_OUTPUT_HANDLE);  
HANDLE hErrors = GetStdHandle(STD_ERROR_HANDLE);  
//
```

Каналы Windows API (продолжение)

Пример (pipe_parent.cpp, продолжение)

```
STARTUPINFO startup_info =  
{  
    sizeof (STARTUPINFO),  
    0  
};  
startup_info.dwFlags = STARTF_USESTDHANDLES;  
startup_info.hStdInput = hReadPipe;  
startup_info.hStdOutput = hOutput;  
startup_info.hStdError = hErrors;
```

Каналы Windows API (продолжение)

Пример (pipe_parent.cpp, продолжение)

```
PROCESS_INFORMATION process_info =  
{  
    INVALID_HANDLE_VALUE,  
    INVALID_HANDLE_VALUE,  
    0,  
    0,  
};
```


Каналы Windows API (продолжение)

Пример (pipe_parent.cpp, продолжение)

```
bSuccess = CreateProcess(
    NULL,                                // lpctszApplicationName
    _T("pipe_child"),                   // lpctszCommandLine
    NULL,                                // lpProcessAttributes
    NULL,                                // lpThreadAttributes
    TRUE,                                // bInheritHandles
    0,                                    // dwCreationFlags
    NULL,                                // lpvEnvironment
    NULL,                                // lpctszCurrentDirectory
    &startup_info,                       // lpStartupInfo
    &process_info);                     // lpProcessInformation
WinAssert(bSuccess, _T("Ошибка запуска pipe_child"));
//
```

Каналы Windows API (продолжение)

Пример (pipe_parent.cpp, продолжение)

```
const char cszData[] = "Test data\n";
DWORD dwBytesWritten;
for (int i = 0; i < 30; ++ i)
{
    bSuccess = WriteFile(
        hWritePipe,                                // hFile
        cszData,                                    // lpvBuffer
        sizeof (cszData) - sizeof (char),          // dwNumberOfBytesToWrite
        &dwBytesWritten,                             // lpdwBytesWritten
        NULL);                                       // lpOverlapped
    WinAssert(bSuccess, _T("Ошибка записи в канал"));
}
```

Каналы Windows API (продолжение)

Пример (pipe_parent.cpp, окончание)

```
//  
CloseHandle(hWritePipe);  
CloseHandle(process_info.hProcess);  
CloseHandle(process_info.hThread);  
//  
}    // main()
```

Каналы Windows API (окончание)

Пример (pipe_child.cpp)

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

int main()
{
    string s_buf;
    int n = 0;
```

Пример (pipe_child.cpp, окончание)

```
    while (getline(cin, s_buf))
    {
        cout <<
            setw(2) << ++ n <<
            ". " << s_buf << endl;
    }
    cout << "child finished" << endl;
    //
}    // main()
```

Создание именованного канала Windows API

Windows API CreateNamedPipe()

```
HANDLE WINAPI CreateNamedPipe(  
    _In_      LPCTSTR          lpctszName,  
    //        \\.\pipe\<имя>  
    _In_      DWORD            dwOpenMode,  
    _In_      DWORD            dwPipeMode,  
    _In_      DWORD            uMaxInstances,  
    //        PIPE_UNLIMITED_INSTANCES  
    _In_      DWORD            uOutBufferSize,  
    _In_      DWORD            uInBufferSize,  
    _In_      DWORD            uDefaultTimeOut,  
    _In_opt_  LPSECURITY_ATTRIBUTES lpSecurityAttributes  
);
```

Создание именованного канала (окончание)

dwOpenMode		dwPipeMode
PIPE_ACCESS_INBOUND	PIPE_TYPE_BYTE	PIPE_TYPE_MESSAGE
PIPE_ACCESS_OUTBOUND	PIPE_READMODE_BYTE	PIPE_READMODE_MESSAGE
PIPE_ACCESS_DUPLEX		
...		

Таблица 8: значения флагов параметров функции CreateNamedPipe()

Ожидание подключения клиента Windows API

Windows API ConnectNamedPipe(), DisconnectNamedPipe()

```
BOOL WINAPI ConnectNamedPipe(  
    _In_      HANDLE          hNamedPipe,  
    _Inout_opt_ LPOVERLAPPED lpOverlapped  
);  
  
BOOL WINAPI FlushFileBuffers(  
    _In_      HANDLE          hFile  
);  
  
BOOL WINAPI DisconnectNamedPipe(  
    _In_      HANDLE          hNamedPipe  
);
```

Открытие именованного канала Windows API

Windows API CreateFile()

```
HANDLE WINAPI CreateFile(  
    _In_          LPCTSTR          lpctszFileName,  
    _In_          DWORD             dwDesiredAccess,  
    //            GENERIC_READ | GENERIC_WRITE  
    _In_          DWORD             dwShareMode,  
    _In_opt_      LPSECURITY_ATTRIBUTES lpSecurityAttributes,  
    _In_          DWORD             dwCreationDisposition,  
    //            OPEN_EXISTING  
    _In_opt_      DWORD             dwFlagsAndAttributes,  
    _In_opt_      HANDLE            hTemplateFile  
);
```


Ожидание доступности именованного канала

Windows API `WaitNamedPipe()`

```
// GetLastError() == ERROR_PIPE_BUSY
```

```
BOOL WINAPI WaitNamedPipe(
```

```
    _In_ LPCTSTR lpNamedPipeName,
```

```
    _In_ DWORD uTimeout
```

```
);
```

`uTimeout`

`NMPWAIT_USE_DEFAULT_WAIT` `NMPWAIT_WAIT_FOREVER`

Таблица 9: значения флагов параметра функции `WaitNamedPipe()`

Изменение параметров именованного канала

Windows API SetNamedPipeHandleState()

```
BOOL WINAPI SetNamedPipeHandleState(  
    _In_      HANDLE          hNamedPipe,  
    _In_opt_  LPDWORD         lpdwMode,  
    _In_opt_  LPDWORD         lpuMaxCollectionCount,  
    _In_opt_  LPDWORD         lpuCollectDataTimeout  
);
```

Чтение из именованного канала

Windows API ReadFile()

```
BOOL WINAPI ReadFile(  
    _In_      HANDLE          hFile,  
    _Out_     LPVOID          lpvBuffer,  
    _In_      DWORD           uNumberOfBytesToRead,  
    _Out_opt_ LPDWORD         lpuNumberOfBytesRead,  
    _Inout_opt_ LPOVERLAPPED lpOverlapped  
);
```

Запись в именованный канал

Windows API WriteFile()

```
BOOL WINAPI WriteFile(  
    _In_      HANDLE          hFile,  
    _In_      LPCVOID         lpvBuffer,  
    _In_      DWORD           uNumberOfBytesToWrite,  
    _Out_opt_ LPDWORD         lpNumberOfBytesWritten,  
    _Inout_opt_ LPOVERLAPPED lpOverlapped  
);
```