Лекция12

Обмен данными между процессами (продолжение)

Отображение файлов в память

- передача данных через дисковый файл;
- проецирование файлов на память;
- передача данных через проекцию файла;
- передача данных через swap-файл.

Стандартная библиотека *C*

```
#include <stdio.h>
#include <math.h>
#define pi 3.141592
#define DATA_SIZE 1024
struct _data{
	int index;
	double x;
	double y;
};


int main() {
	struct _data Data[DATA_SIZE];
	FILE* fp;
	for (int i = 0; i < DATA_SIZE; i++) {
		Data[i].index = i;
		Data[i].x = i*(2.0*pi / DATA_SIZE);
		Data[i].y = sin(Data[i].x);
	}

	fp = fopen("test.dat", "wb");
	fwrite(Data, sizeof(struct _data),
					DATA_SIZE, fp);
	fclose(fp);
	return 0;
}
```

Windows API

```
#include <windows.h>
#include <stdio.h>
#include <math.h>
#define pi 3.141592
#define DATA_SIZE 1024
struct _data {
	int index;
	double x;
	double y;
};
int main() {
	struct _data Data[DATA_SIZE];
	HANDLE hFile;
	DWORD N;

	for (int i = 0; i < DATA_SIZE; i++) {
		Data[i].index = i;
		Data[i].x = i*(2.0*pi / DATA_SIZE);
		Data[i].y = sin(Data[i].x);
	}
	hFile = CreateFile("test.dat",
		GENERIC_READ | GENERIC_WRITE,
		0, NULL,OPEN_ALWAYS,
		FILE_ATTRIBUTE_NORMAL, NULL);

	WriteFile(hFile, Data, sizeof(struct _data)
				*DATA_SIZE, &N, NULL);
	CloseHandle(hFile);
	return 0;
}
```

```
#include <stdio.h>
#define DATA_SIZE 1024
struct _data {
        int index;
        double x;
        double y;
};

int main() {
  struct _data Data[64];
  FILE* fp;

  fp = fopen("test2.dat", "rb");

  for (int j = 0; j < DATA_SIZE / 64; j++) {
     fseek(fp, 64 * j * sizeof(struct _data),
                                        SEEK_SET);
     fread(Data, sizeof(struct _data), 64, fp);

     for (int i = 0; i < 64; i++)
        printf("%d\t%g\t%g\n", Data[i].index,
                              Data[i].x,   Data[i].y);
        getchar();
        }

        fclose(fp);
        return 0;
}
```

```
#include <windows.h>
#include <stdio.h>
#define DATA_SIZE 1024
struct _data {
        int index;
        double x;
        double y;
};
int main() {
  struct _data Data[64];
  HANDLE hFile;
  DWORD N;
  long n;
  hFile = CreateFile("test2.dat",
    GENERIC_READ | GENERIC_WRITE,
    0, NULL,OPEN_ALWAYS,
   FILE_ATTRIBUTE_NORMAL, NULL);

   for (int j = 0; j < DATA_SIZE / 64; j++) {
     SetFilePointer(hFile, 64 * j * sizeof(struct _data),
                                        NULL,FILE_BEGIN);
     ReadFile(hFile, Data, sizeof(struct _data)*64,&N, NULL);
     for (int i = 0; i < 64; i++)
        printf("%d\t%g\t%g\n", Data[i].index, Data[i].x,
                                                Data[i].y);
     getchar();
    }
    CloseHandle(hFile);
    return 0;
}
```

```
HANDLE CreateFile(
  LPCTSTR lpFileName,                    // имя файла
  DWORD dwDesiredAccess,                 // GENERIC_READ, GENERIC_WRITE
  DWORD dwShareMode,                     // FILE_SHARE_WRITE, FILE_SHARE_READ,
                                         // FILE_SHARE_DELETE
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,   // NULL – дескриптор файла не
                                                // наследуется и безопасность по умолчанию
  DWORD dwCreationDisposition,           //
  DWORD dwFlagsAndAttributes,            //
  HANDLE hTemplateFile                   // Копирование аттрибутов шаблона
);
```

dwCreationDisposition:

CREATE_ALWAYS	2	Всегда создает новый файл.
CREATE_NEW	1	Создает новый файл только, если он не существует.
OPEN_ALWAYS	4	Всегда открывает новый файл.
OPEN_EXISTING	3	Открывает файл, если он существует.
TRUNCATE_EXISTING	5	Открывает файл и стирает его содержимое только, если файл существует.

DwFlagsAndAttributes:

FILE_ATTRIBUTE_ARCHIVE	32 (0x20) Архивный файл
FILE_ATTRIBUTE_ENCRYPTED	16384 (0x4000) Шифрованный файл
FILE_ATTRIBUTE_HIDDEN	2 (0x2) Скрытый файл
FILE_ATTRIBUTE_NORMAL	128 (0x80)
FILE_ATTRIBUTE_READONLY	1 (0x1) Файл только для чтения
.....	
FILE_FLAG_DELETE_ON_CLOSE	0x04000000 Файл удаляется, когда закрыты все дескрипторы
FILE_FLAG_NO_BUFFERING	0x20000000 Файл открывается без системного кэширования
FILE_FLAG_OVERLAPPED	0x40000000 Файл открывается для асинхронного ввода/вывода
.....	

Изменение владельца файла по умолчанию:

```
typedef struct _SECURITY_ATTRIBUTES {
  DWORD  nLength;
  LPVOID lpSecurityDescriptor;
  BOOL   bInheritHandle;
} SECURITY_ATTRIBUTES, *PSECURITY_ATTRIBUTES, *LPSECURITY_ATTRIBUTES;
```

```
GetSecurityDescriptorGroup
GetSecurityDescriptorOwner
……..
SetSecurityDescriptorGroup
SetSecurityDescriptorOwner(
  PSECURITY_DESCRIPTOR pSecurityDescriptor,
  PSID                 *pOwner,
  LPBOOL               lpbOwnerDefaulted
);
……
```

```
BOOL WINAPI LookupAccountName(
  LPCTSTR       lpSystemName,
  LPCTSTR       lpAccountName,
  PSID          Sid,
  LPDWORD       cbSid,
  LPTSTR        ReferencedDomainName,
  LPDWORD       cchReferencedDomainName,
  PSID_NAME_USE peUse
);
```

```
BOOL WINAPI WriteFile(
  HANDLE hFile,
  LPCVOID lpBuffer,
  DWORD nNumberOfBytesToWrite,
  LPDWORD lpNumberOfBytesWritten,
  LPOVERLAPPED lpOverlapped //Асинхронный вывод и
                                        //установка начальной позиции
);
```

```
BOOL WINAPI ReadFile(
  HANDLE hFile,
  LPVOID lpBuffer,
  DWORD nNumberOfBytesToRead,
  LPDWORD lpNumberOfBytesRead,
  LPOVERLAPPED lpOverlapped
);
```

```
typedef struct _OVERLAPPED {
  ULONG_PTR Internal;
  ULONG_PTR InternalHigh;
  union {
    struct {
      DWORD Offset;
      DWORD OffsetHigh;
    };
    PVOID  Pointer;
  };
  HANDLE    hEvent;
} OVERLAPPED, *LPOVERLAPPED;
```

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#define N 80
int main(int argc, char* argv[]) {
	HANDLE hFile;
	char buff[N] = { '\0' };
	DWORD n;

	hFile = CreateFile("test.txt", GENERIC_WRITE, FILE_SHARE_READ |
			FILE_SHARE_WRITE, NULL, TRUNCATE_EXISTING,
		FILE_ATTRIBUTE_NORMAL, NULL);
	if (hFile == INVALID_HANDLE_VALUE) {
		printf("Could not open file.");
		return -1;
	}
	do {
		gets(buff);
		WriteFile(hFile, buff, sizeof(buff), &n, NULL);
	} while (getch() != 27);
	CloseHandle(hFile);
	return 0;
}
```

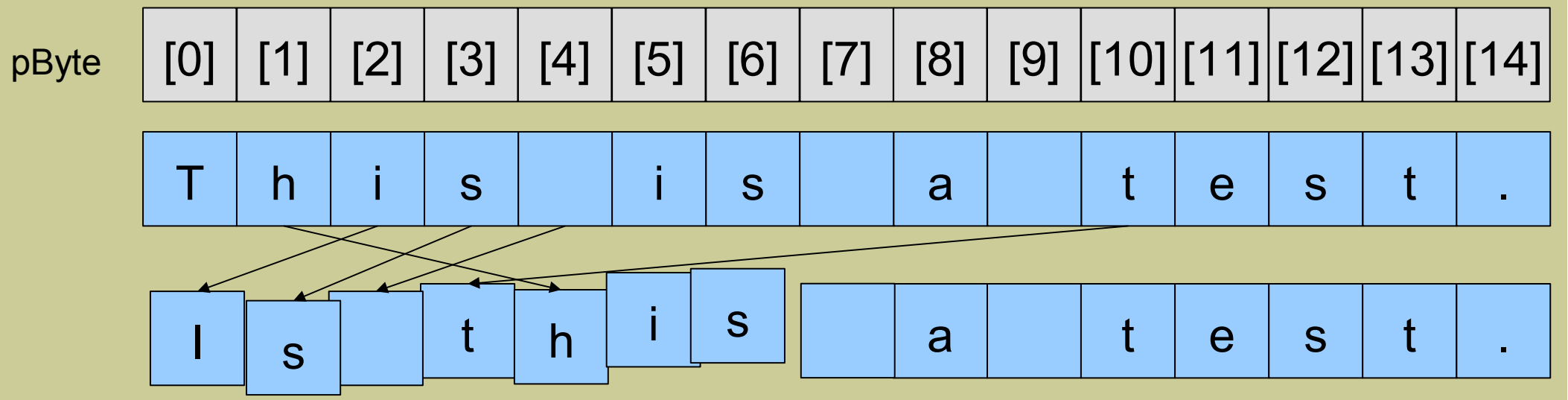
```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#define N 80

int main(int argc, char* argv[]) {
	HANDLE hFile;
	char buff[N] = { '\0' };
	DWORD n;

	hFile = CreateFile("test.txt",
		GENERIC_READ | GENERIC_WRITE,
		FILE_SHARE_READ | FILE_SHARE_WRITE, NULL,
		OPEN_ALWAYS,FILE_ATTRIBUTE_NORMAL, NULL);

	if (hFile == INVALID_HANDLE_VALUE) {
		printf("Could not open file.");
		return -1;
	}
	do{
		ReadFile(hFile, buff, sizeof(buff), &n, NULL);
		printf("%s\n", buff);
	} while (getch() != 'q');
	CloseHandle(hFile);
	return 0;
}
```

```
#include <windows.h>
#include <stdio.h>
#define N 20
int main(int argc, char* argv[]){
	HANDLE hFile, hFileMap;
	PBYTE pByte;
	int i;
	char dummy;

	hFile = CreateFile("testmap.txt",
		GENERIC_READ | GENERIC_WRITE,
		0,
		NULL,
		OPEN_ALWAYS,
		FILE_ATTRIBUTE_NORMAL,
		NULL);
	if (hFile == INVALID_HANDLE_VALUE){
		printf("Could not open file.");
	}

	hFileMap = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0, N, NULL);
	if (hFileMap == NULL){
		printf("Could not create mapping file.");
	}
```

```
HANDLE CreateFileMapping(
  HANDLE hFile,                       // дескриптор файла
  LPSECURITY_ATTRIBUTES lpAttributes, //
  DWORD flProtect,                            // PAGE_READWRITE, PAGE_READONLY
  DWORD dwMaximumSizeHigh,          // high-order DWORD объекта отображения
  DWORD dwMaximumSizeLow,             // low-order DWORD объекта отображения
  LPCTSTR lpName                    // имя объекта отображения
);
```

```
LPVOID MapViewOfFile(
  HANDLE hFileMappingObject,   // дескриптор объекта отображения
  DWORD dwDesiredAccess,       // FILE_MAP_WRITE, FILE_MAP_READ
  DWORD dwFileOffsetHigh,      // high-order DWORD смещения
  DWORD dwFileOffsetLow,       // low-order DWORD смещения
  SIZE_T dwNumberOfBytesToMap  // количество байт отображения
);
```

```
	pByte = (PBYTE)MapViewOfFile(hFileMap, FILE_MAP_WRITE, 0, 0, 0);
	if (pByte == NULL){
		printf("Could not map file.");
	}

	for (i = 0; i<N; i++)
		printf("%c\n", pByte[i]);

	dummy = pByte[1];
	pByte[0] = pByte[5] - 0x20;
	pByte[1] = pByte[3];
	pByte[2] = pByte[4];
	pByte[3] = pByte[10];
	pByte[4] = dummy;


	for (i = 0; i<N; i++)
		printf("%lx\t%c\n", (unsigned long)(pByte + i), *(pByte + i));

	UnmapViewOfFile(pByte);
	CloseHandle(hFile);

	return 0;
}
```




```
pByte[0] = pByte[2] — 0x20;
dummy = pByte[1];
pByte[1] = pByte[3];
pByte[2] = pByte[4];
pByte[3] = pByte[10];
pByte[4] = dummy;
```

```
#include <windows.h>

int main(){
 HANDLE hFileMap;
 LPVOID lpMapView;
 char* buff="Hello,students!";

hFileMap=CreateFileMapping((HANDLE)0xFFFFFFFF,NULL,PAGE_READWRITE,0,256,"MyCommonRe
gion");
 lpMapView=MapViewOfFile(hFileMap,FILE_MAP_WRITE,0,0,256);
 CopyMemory(lpMapView,buff, sizeof("Hello,students!"));
 getch();
 UnmapViewOfFile(lpMapView);
 CloseHandle(hFileMap);
 return 0;
}
```

```
#include <windows.h>
int main(){
 HANDLE hFileMap;
 LPVOID lpMapView;
 char buff[80];

 hFileMap=OpenFileMapping(FILE_MAP_READ,TRUE,"MyCommonRegion");
 lpMapView=MapViewOfFile(hFileMap,FILE_MAP_READ,0,0,256);
 CopyMemory(buff,lpMapView, 80);
 printf("%s\n",buff);
 UnmapViewOfFile(lpMapView);
  CloseHandle(hFileMap);
 return 0;
}
```