Лекция 5

Содержание

I. Библиотеки динамической компоновки -
- Windows dll,
- разделяемые библиотеки Linux.

II. Создание процессов в MS Windows

Библиотеки динамической компоновки

td1.c

```
#include <windows.h>
extern  __declspec(dllexport)  int a=23;
__declspec(dllexport) int f(int b){
  return b*b;
}
__declspec(dllexport) int g(int b){
  return b*b*b;
}
/*
BOOL WINAPI DllMain(
          HANDLE hModule,
          DWORD  dwReason,
          LPVOID  lpReserved){
 return  TRUE;
}
*/
```

Компоновка библиотеки:

```
> cl /c td1.c
> link /DLL  td1.obj
```

td1.lib td1.dll

td2.c

```
#include <windows.h>
                                    int a=23;
                            int f(int b){
  return b*b;
}
                            int g(int b){
  return b*b*b;
}
/*
BOOL WINAPI DllMain(
          HANDLE hModule,
          DWORD  dwReason,
          LPVOID  lpReserved){
 return  TRUE;
}
*/
```

td2.def

```
LIBRARY  td1
EXPORTS
      a  @1
      f  @2
      g  @3
```

Компоновка библиотеки с помощью .def-файла:

```
>cl  /c  td2.c
>link /DLL  /DEF:td2.def td2.obj
```

Неявное связывание

t1.c

```
#include <windows.h>
extern __declspec(dllimport) int a;
int f(int);
int g(int);

int main(){
 printf("%i %i %i\n",a,f(3),g(3));

 return 0;
}
```

```
> cl  t1.c  td1.lib
```

Явное связывание

```
#include <windows.h>
typedef  int (*fun)(int);
int main(){
HINSTANCE hInst;
fun pf,pg;
int* pa;
      hInst=LoadLibrary("td2.dll");
      pa=(int*)GetProcAddress(hInst, "a");
      pf=(fun)GetProcAddress(hInst, "f");
      pg=(fun)GetProcAddress(hInst, "g");
      printf("%i %i %i\n",*pa,pf(3),pg(3));
      FreeLibrary(hInst);

       return 0;
}
```

```
> cl  t2.c  kernel32.lib
```

Разделяемые библиотеки

*Компилятор **gcc**, платформа **LINUX, компоновка библиотеки***

1d.c

```
double a=3.1415;
int sh_fun(int i){
 return i*i;
}
```

```
$gcc -shared -fPIC 1d.c -o lib1d.so
```

1.c

```
#include <stdio.h>
#include <stdlib.h>
extern double a;
int sh_fun(int);
int main(int argc, char* argv[])
{
  if(argc>1)
     a=atof(argv[1]);
  printf("%g\n",a);
  printf("%i\n", sh_fun(3));
 return 0;
}
```

Неявное связывание

```
$ gcc 1.c  -L.  -l1d -o 1
$ ./1
3.1415
9
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
typedef  int (*fun)(int);
extern double a;

int main(int argc, char* argv[]){
  fun f;
  printf("Dynamic library test\n");

 void* h=dlopen("lib1d.so",RTLD_LAZY);
  printf("%g\n",a);

  f=(fun)dlsym(h,"sh_fun");
  printf("%i\n", f(5));
  dlclose(h);
 return 0;
}
```

Явное связывание

```
$ gcc 2.c -L. -l1d -ldl -o 2
$ ./2
   3.1415
   25
```

Создание процессов в Windows, часть II

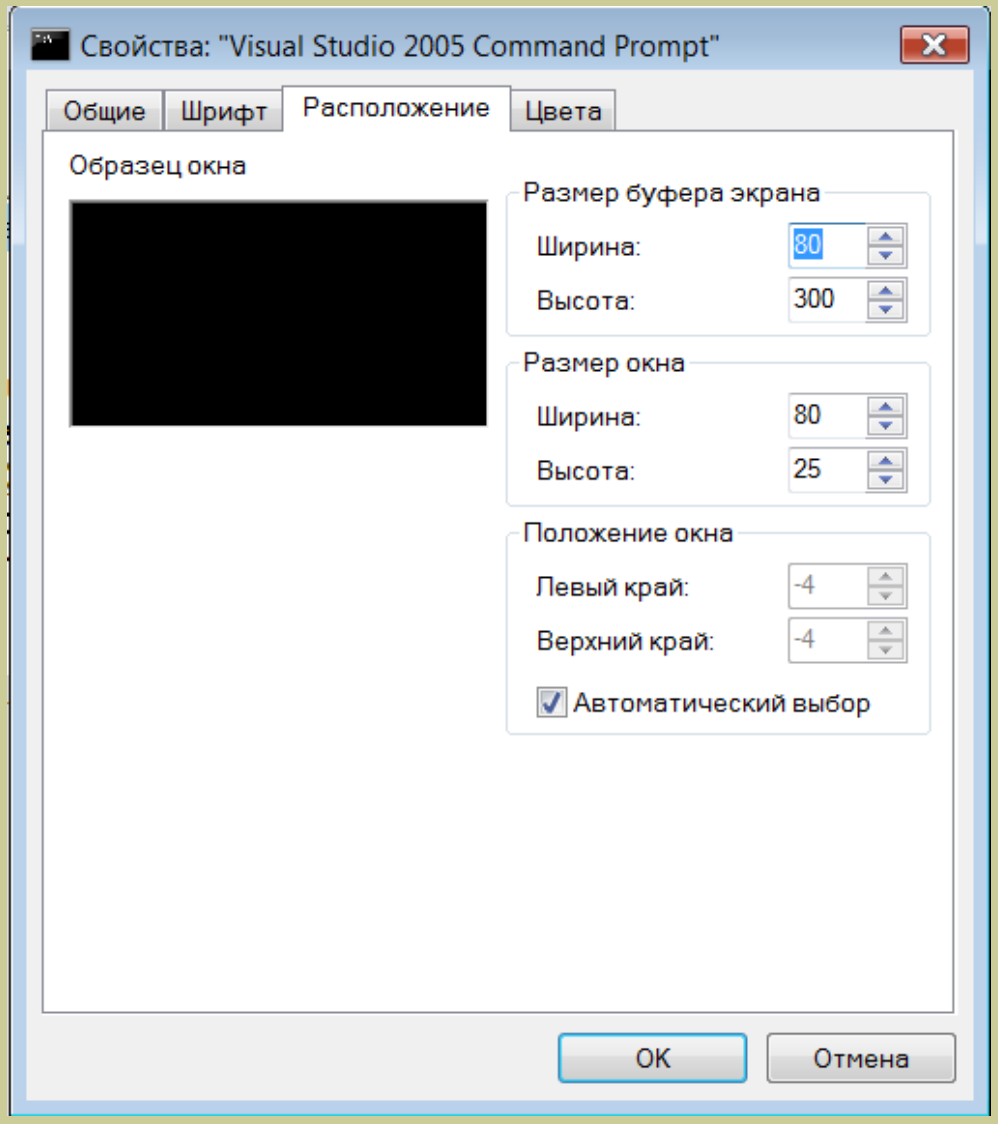
```
BOOL CreateProcess(
LPCTSTR lpApplicationName, // имя исполняемого модуля
LPTSTR lpCommandLine, // команда командной строки
LPSECURITY_ATTRIBUTES lpProcessAttributes,
// указатель на дескриптор безопасности процесса
LPSECURITY_ATTRIBUTES lpThreadAttributes,
// указатель на дескриптор безопасности потока
BOOL bInheritHandles, // бит, управляющий наследованием
дескрипторов
DWORD dwCreationFlags,
// флаги приоритета, консоли и т.д.
LPVOID lpEnvironment, // строки окружения
LPCTSTR lpCurrentDirectory, // имя текущей директории
LPSTARTUPINFO lpStartupInfo,
// описатель начального состояния окна
LPPROCESS_INFORMATION lpProcessInformation
// структура, содержащая информацию о процессе );
```

Descriptor – индекс записи в структуре данных на уровне ядра или сама эта запись.

Handle – описатель (дескриптор) объекта, целое число, используемое при манипуляции объектом на уровне пользователя. В зависимости от объекта смысл значения этого целого числа различен. Файловый описатель (file handle, FILE* fp;) является указателем на структуру, содержащую поле имени файла, поле указателя на буфер потока ввода/вывода и т.д. Описатель модуля (module handle) равен базовому адресу данного исполняемого модуля в адресном пространстве процесса. *В случае объекта ядра, его описатель преобразуется в указатель на этот объект диспетчером объектов.*

Структура STARTUPINFO (информация о начальных свойствах окна):

```
typedef struct _STARTUPINFO {
DWORD cb;
LPTSTR lpReserved;
LPTSTR lpDesktop;
LPTSTR lpTitle;
DWORD dwX;
DWORD dwY;
DWORD dwXSize;
DWORD dwYSize;
DWORD dwXCountChars;
DWORD dwYCountChars;
DWORD dwFillAttribute;
DWORD dwFlags;
WORD wShowWindow;
……
HANDLE hStdInput;
HANDLE hStdOutput;
HANDLE hStdError; } STARTUPINFO, *LPSTARTUPINFO;
```


Свойства: "Visual Studio 2005 Command Prompt"
Общие
Шрифт
Расположение
Цвета
Образец окна
Размер буфера экрана
Ширина:
80
Высота:
300
Размер окна
Ширина:
80
Высота:
25
Положение окна
Левый край:
-4
Верхний край:
-4
Автоматический выбор
OK
Отмена

Структура PROCESS_INFORMATION, содержащая идентификаторы процесса и начального потока:

```
typedef struct _PROCESS_INFORMATION {
HANDLE hProcess; // описатель процесса
HANDLE hThread; // описатель потока
DWORD dwProcessId;//глобальный идентификатор процесса
DWORD dwThreadId; //глобальный идентификатор потока
 } PROCESS_INFORMATION;
```

Дескриптор (описатель) процесса может использоваться для манипулирования процессом, в частности он может использоваться для его прекращения:

```
BOOL TerminateProcess(
HANDLE hProcess, // описатель процесса
UINT uExitCode // код выхода );
```

Пример создания процесса в Windows:

```
#include <windows.h>
void main( void )
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

 //char* cmdLine="cmd /C dir";
//Задание процесса с использованием командной строки
    ZeroMemory( &si, sizeof(si) );
//Выделение памяти для si и ее инициализация нулями
    si.cb = sizeof(si);
// инициализация первого поля структуры STARTUPINFO
    ZeroMemory( &pi, sizeof(pi) );
 //Выделение памяти для pi и ее инициализация нулями
```

```
// Запускается дочерний процесс
     if( !CreateProcess(
       "C:\\Windows\\notepad.exe", //Исполняемый модуль
        NULL,  //cmdLine, // Командная строка не используется
        NULL,        // Дескриптор процесса не наследуется
        NULL,       // Дескриптор потока не наследуется
        FALSE,
 // Открытые дескрипторы родительского процесса не
//наследуется
        CREATE_NO_WINDOW,
// Не создавать окна (консольного окна)
        NULL,
//  Используются переменные окружения родителя
        NULL,   // Используется текущая директория родителя
        &si,      // Указатель на структуру STARTUPINFO
        &pi )
// Указатель на структуру PROCESS_INFORMATION
) {
```

```
        printf("System error code: %i\n",GetLastError());
    }
    //Ожидание окончания дочернего процесса
    WaitForSingleObject( pi.hProcess, INFINITE );
    // Закрытие дескрипторов процесса и начального потока
    CloseHandle( pi.hProcess );
    CloseHandle( pi.hThread );
}
```

Упражнение 1: протестировать вызов функции CreateProcess с флагами, перечисленными в таблице:

Флаг	Значение
CREATE_NO_WINDOW	Не создавать новое окно
CREATE_NEW_CONSOLE	Создать новое консольное окно
ABOVE_NORMAL_PRIORITY_CLASS	Приоритет выше обычного
IDLE_PRIORITY_CLASS	Фоновый приоритет
HIGH_PRIORITY_CLASS	Высший приоритет