Объекты ядра

Для работы с важными системными ресурсами ОС Windows создает объекты, управление которыми осуществляет менеджер объектов.

- Ядро поддерживает базовые объекты двух видов: объекты диспетчера (события, мьютексы, семафоры, потоки ядра, таймеры и др.) и управляющие (DPC, APC, прерывания, процессы, профили и др.)
- Над объектами ядра находятся объекты исполнительной системы, каждый из которых инкапсулирует один или более объектов ядра. Объекты исполнительной системы предназначены для управления памятью, процессами и межпроцессным обменом. К ним относятся такие объекты, как: процесс, поток, открытый файл, семафор, мьютекс, маркер доступа и ряд других (MSDN). Эти объекты и называются объектами ядра в руководствах по программированию.

Объекты ядра

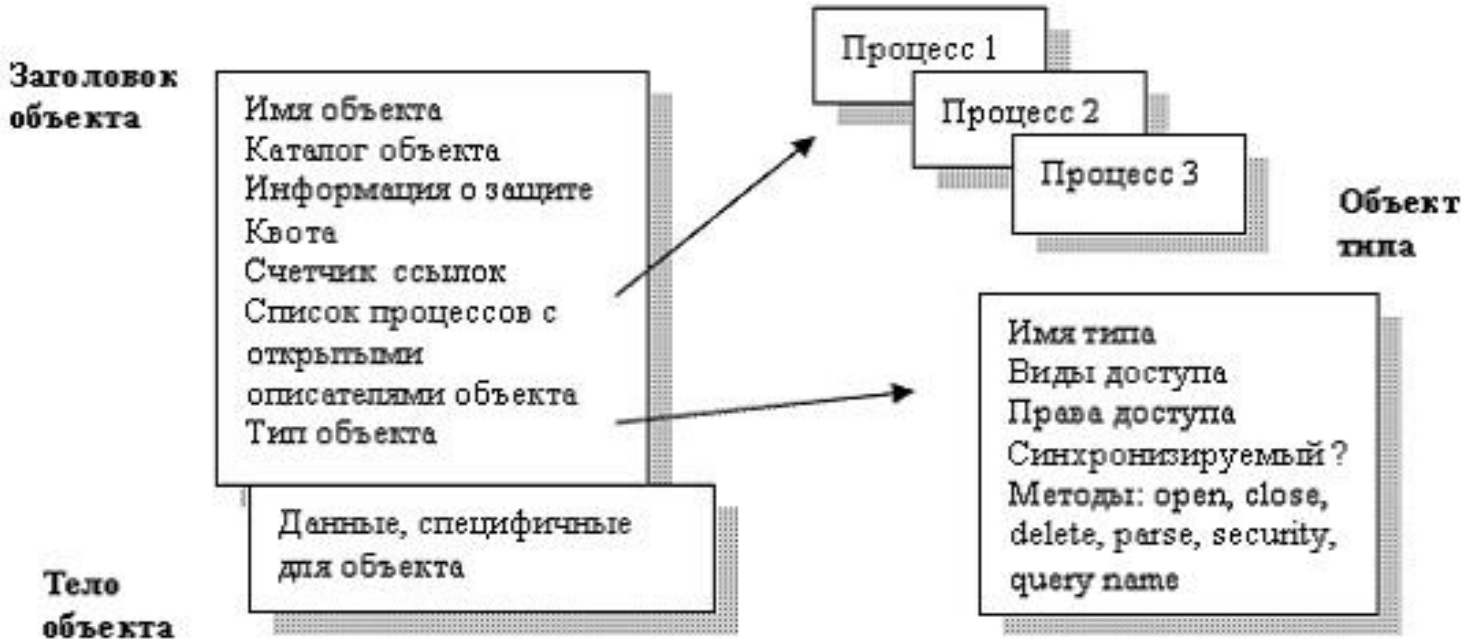
- Внешнее отличие объектов ядра (объектов исполнительной системы) от объектов User и GDI состоит в наличии у первых атрибутов защиты, Далее эти объекты ядра (объекты исполнительной системы) будут называться просто объектами.
- Объект представляет собой блок памяти в виртуальном адресном пространстве ядра. Этот блок содержит информацию об объекте в виде структуры данных. Структура содержит как общие, так и специфичные для каждого объекта элементы. Объекты создаются в процессе загрузки и функционирования ОС и теряются при перезагрузке и выключении питания.
- Содержимое объектов доступно только ядру, приложение не может модифицировать его непосредственно. Доступ к объектам можно осуществить только через его функции-методы (инкапсуляция данных), которые инициируются вызовами некоторых библиотечных Win32-функций.

Создание и мониторинг объектов ядра

```
void __fastcall TForm1::SmCreateClick(TObject *Sender)
{ try {
   HANDLE  r=CreateSemaphore(NULL,1,5,"MySem");
   if (r) {AnsiString msg="Semaphore was created";
           LBConsole->Items->Add(msg);} else
   LBConsole->Items->Add("Error");
 } catch ( ... ) { }
}
void __fastcall TForm1::SmOpenClick(TObject *Sender)
{ try
 {HANDLE  r=OpenSemaphore(SEMAPHORE_ALL_ACCESS, false,"MySem");
   if (r) {
           AnsiString msg="Semaphore was opened.";
           LBConsole->Items->Add(msg);
   } else
    LBConsole->Items->Add("Error opended");
  }
  catch ( ... ) {}
}
```

Приложение SemExample Приложение Winobj

Структура объекта. Методы объекта

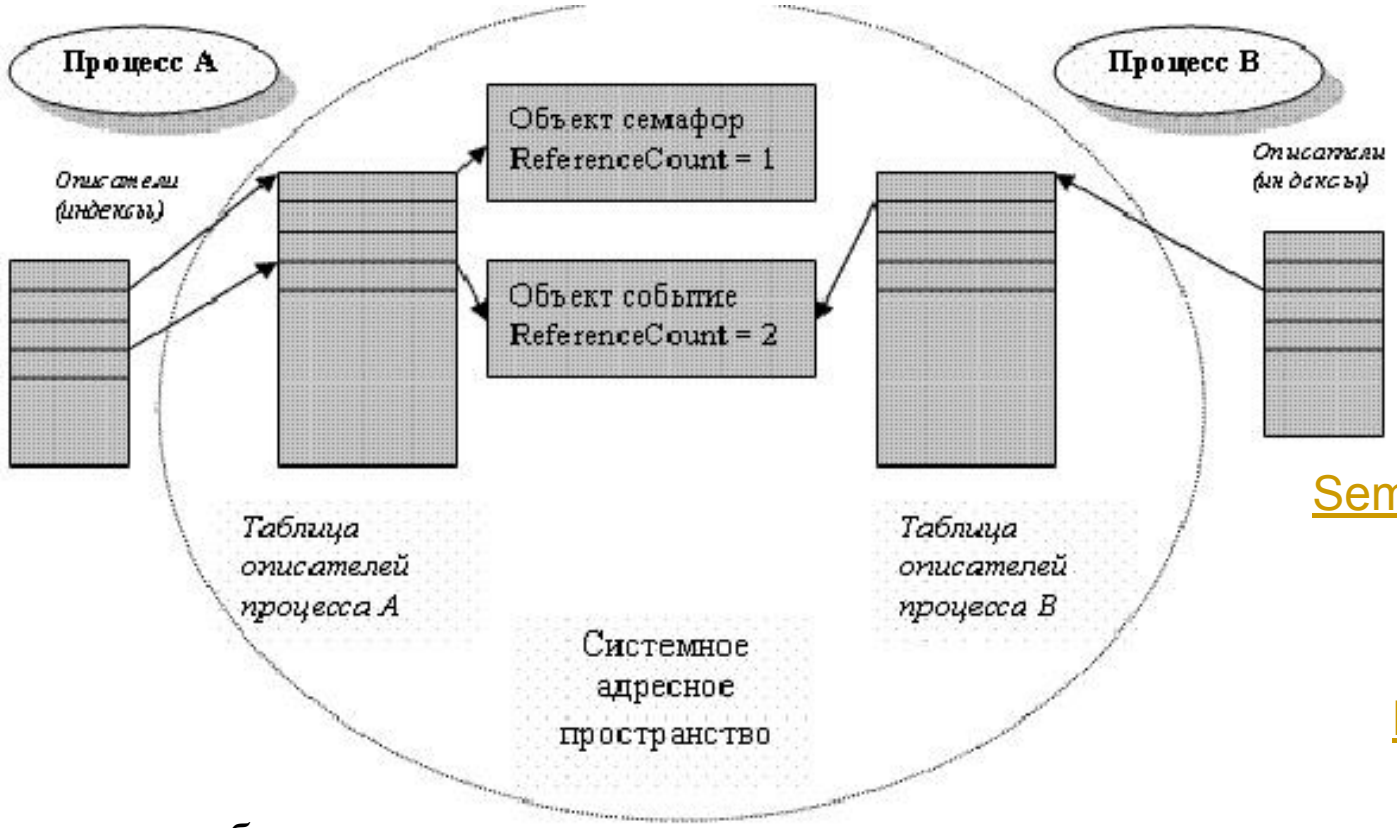


- Каждый объект имеет заголовок с информацией, общей для всех объектов, а также данные, специфичные для объекта. Например, в поле заголовка имеется список процессов, открывших данный объект, и информация о защите, определяющая, кто и как может использовать объект.
- Счетчик ссылок на объект увеличивается на 1 при открытии объекта и уменьшается на 1 при его закрытии.

Структура объекта. Методы объекта

- Квота устанавливает ограничения на объемы ресурсов. Например, по умолчанию лимит на открытые объекты для процесса - 230. Множество объектов делится на типы, а у каждого из объектов есть атрибуты, неизменные для объектов данного типа. Ссылка на тип объекта также входит в состав заголовка.
- В состав компонентов объекта типа входит атрибут методы - указатели на внутренние процедуры для выполнения стандартных операций. Методы вызываются диспетчером объектов при создании и уничтожении объекта, открытии и закрытии описателя объекта, изменении параметров защиты. Система позволяет динамически создавать новые типы объектов. В этом случае предполагается регистрация его методов у диспетчера объектов. Например, метод open вызывается всякий раз, когда создается или открывается объект и создается его новый описатель.

Описатели объектов



SemExample

WinObj

ProcExp

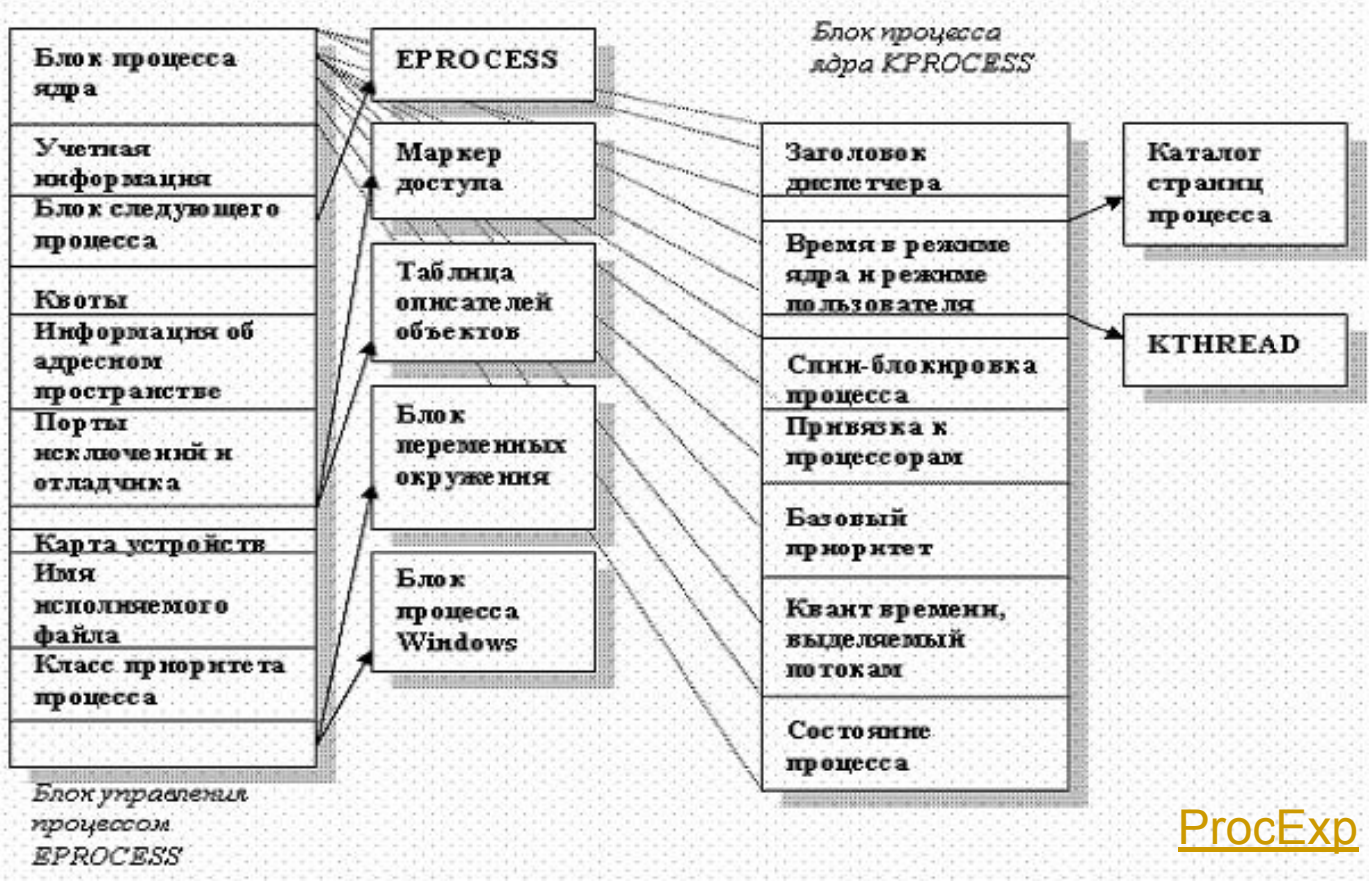
- Создание новых объектов, или открытие по имени уже существующих, приложение может осуществить при помощи Win32-функций, таких, как CreateFile, CreateSemaphore, OpenSemaphore и т.д.
- Это библиотечные процедуры, за которыми стоят сервисы Windows и методы объектов. В случае успешного выполнения создается 64-битный описатель в таблице описателей процесса в памяти ядра. На эту таблицу есть ссылка из блока управления процессом EPROCESS

Процессы и потоки в ОС Windows

Для описания процесса ОС поддерживает набор структур, главную из которых принято называть блоком управления процессом (PCB, Process control block). Состав PCB:

- идентификатор процесса;
- токен доступа - исполняемый объект, содержащий информацию о безопасности;
- базовый приоритет - основа для исполнительного приоритета нитей процесса;
- процессорная совместимость - набор процессоров, на которых могут выполняться нити процесса;
- предельные значения квот - максимальное количество страничной и нестраничной системной памяти, дискового пространства, предназначенного для выгрузки страниц, процессорного времени - которые могут быть использованы процессами пользователя;
- время исполнения - общее количество времени, в течение которого выполняются все нити процесса.

Внутреннее устройство процессов в ОС Windows



Блок управления процессом (PCB) реализован в виде набора связанных структур, главная из которых называется блоком процесса EPROCESS.

Создание процесса

Обычно процесс создается другим процессом вызовом Win32-функции CreateProcess. Создание процесса осуществляется в несколько этапов:

1) на диске отыскивается нужный файл-образ, после чего создается объект "раздел" памяти для проецирования на адресное пространство нового процесса (kernel32.dll);
2) выполняется обращение к системному сервису NtCreateProcess для создания объекта "процесс". Формируются блоки EPROCESS, KPROCESS и блок переменных окружения PEB. Менеджер процессов инициализирует в блоке процесса маркер доступа (копируя аналогичный маркер родительского процесса), идентификатор и другие поля;
3) создание первичного потока (сервис NtCreateThread, библиотека kernel32.dll);
4) kernel32.dll посылает подсистеме Win32 сообщение, которое содержит информацию, необходимую для выполнения нового процесса. Данные о процессе и потоке помещаются, в список процессов и список потоков данного процесса, затем устанавливается приоритет процесса, создается структура, используемая частью Win32, которая работает в режиме ядра, и т.д.;
5) запускается первичный поток, для чего формируются его начальный контекст и стек, и выполняется запуск стартовой процедуры потока режима ядра KiThreadStartup. После этого стартовый код из библиотеки C/C++ передает управление функции main() запускаемой программы.

Создание процесса: функция CreateProcess

```
BOOL CreateProcess(
    PCTSTR pszApplicationName,  //имя программы;
    PTSTR pszCommandLine,  //параметры командной строки;
    SECURITY_ATTRIBUTES psaProcess, //атрибуты безопасности процесса;
    PSECURITY_ATTRIBUTES psaThread,  //атрибуты безопасности главного потока;
    BOOL bInheritHandles, //- если bInheritHandles == TRUE, то созданный процесс
    (запущенная программа), наследует дескрипторы (handles) запускающей
    программы;
    DWORD fdwCreationFlags,  //- параметры создания. Здесь можно указать класс
    приоритета создаваемого процесса и некоторые дополнительные
    параметры;
    PVOID pvEnvironment, // - указатель на блок окружения или NULL, тогда
    используется блок окружения родителя;
    PCTSTR pszCurDir, // текущая директория или NULL, тогда используется текущая
    директория родителя;
    PSTARTUPINFO psiStartInfo,  //-   указатель на структуру STARTUPINFO, которая
    определяет положение главного окна;
    PPROCESS_INFORMATION ppiProcInfo //информация о созданном процессе.
);
```

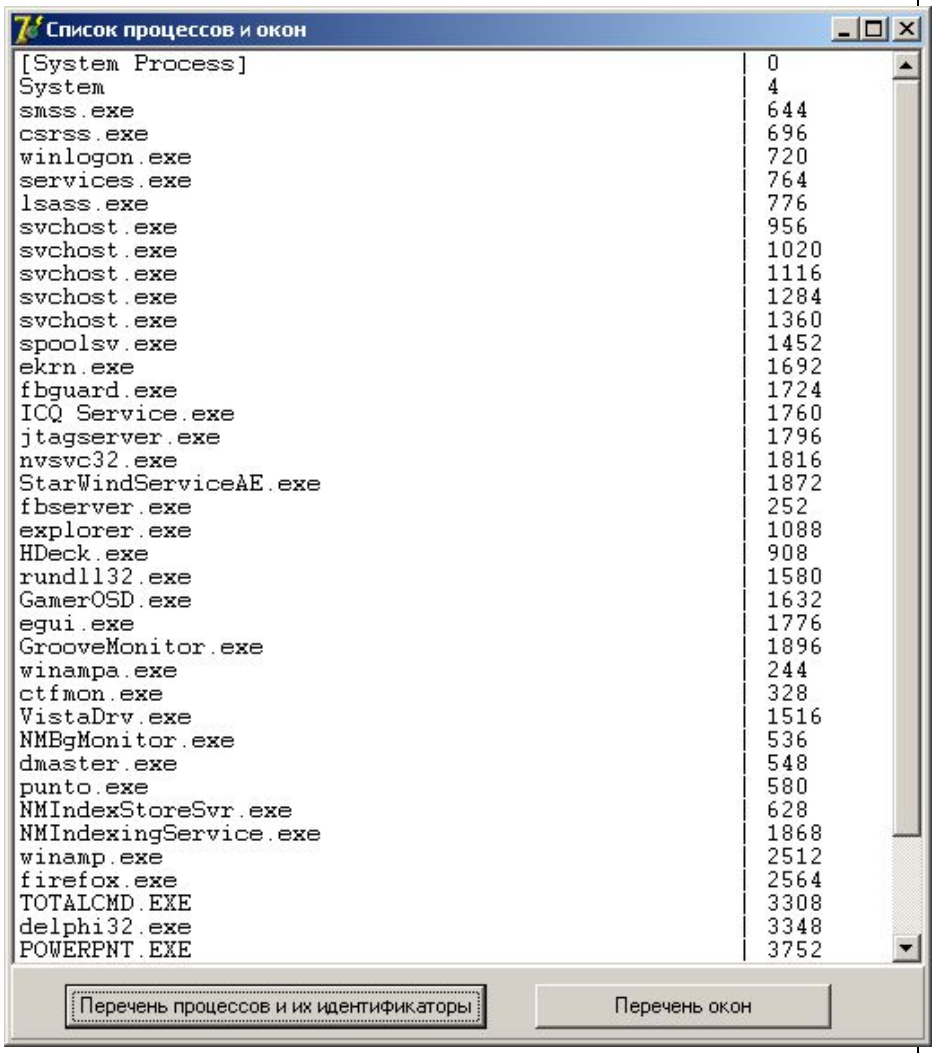
Создание процесса (пример)

```
#include <windows.h>
#include <stdio.h>
void main( VOID )
{
    STARTUPINFO StartupInfo;
    PROCESS_INFORMATION ProcInfo;
    TCHAR CommandLine[] = TEXT("format c:");

    ZeroMemory( &StartupInfo, sizeof(StartupInfo) );
    StartupInfo.cb = sizeof(StartupInfo);
    ZeroMemory( &ProcInfo, sizeof(ProcInfo) );

    if( !CreateProcess( NULL, // Не используется имя модуля
        CommandLine,          // Командная строка
        NULL,                 // Дескриптор процесса не наследуется.
        NULL,                 // Дескриптор потока не наследуется.
        FALSE,                // Установка описателей наследования
        0,                    // Нет флагов создания процесса
        NULL,                 // Блок переменных окружения родит. процесса
        NULL,                 // Использовать текущий каталог родит. процесса
        &StartupInfo,         // Указатель на структуру  STARTUPINFO.
        &ProcInfo )           // Указатель на структуру информации о
процессе.
      )

   printf( "CreateProcess failed." );
  // Ждать окончания дочернего процесса
  WaitForSingleObject( ProcInfo.hProcess, INFINITE );
  // Закрыть описатели процесса и потока
    CloseHandle( ProcInfo.hProcess );
    CloseHandle( ProcInfo.hThread );
}
```

Приложение createprocess

Получение идентификатора процесса

```
procedure TForm1.Button1Click(Sender: TObject);
var
      han : THandle;
      pi : TProcInfo; // from "tlhelp32" in uses clause
      sID : string;
begin
      LB.Items.Clear;
      // Get a snapshot of the system
      han := CreateToolhelp32Snapshot( TH32CS_SNAPALL, 0
      );
      if han = 0 then exit;
      pi:=TProcInfo.Create;
      pi.ProcStruct.dwSize := sizeof( PROCESSENTRY32 );
      if Process32First( han, pi.ProcStruct ) then
         repeat
          sID := ExtractFileName( pi.ProcStruct.szExeFile );
          while length(sID)<50 do sID:=sId+' '; sID:=sID+'|';
          sID:=sID+' '+IntToStr(pi.ProcStruct.th32ProcessID);
          LB.Items.AddObject(sId,pi);
          pi:=TProcInfo.Create;
          pi.ProcStruct.dwSize := sizeof( PROCESSENTRY32 );
        until not Process32Next( han, pi.ProcStruct);
end;
```



Приложение WinProcList