Лабораторная работа № 2

Процессы и потоки

Цель работы: получение практических навыков использования Win32 API для исследования процессов

Процесс - это исполняемый экземпляр приложения и набор ресурсов, которые выделяются данному исполняемому приложению. Ресурсами являются:

- виртуальное адресное пространство;
- системные ресурсы, такие как растровые изображения, файлы, области памяти и т.д.;
- модули процесса, то есть исполняемые модули, которые отображены (загружены) в его адресное пространство. Это могут быть динамические библиотеки (DLI), драйверы (DRV) и управляющие элементы (OCX), основной загрузочный модуль (EXE) процесса, который иногда называют собственно *модулем.* Модуль данных (или программный модуль) может находиться на диске, или может быть загружен в физическую память (RAM). Правда, термин «загружен» (loaded) имеет иное значение, относящееся к виртуальному адресному пространству процесса. Здесь больше подходит термин «отображен» (mapped), так как само отображение - это просто назначение виртуальным адресам физических адресов. После того как модуль загружен в физическую память, его физические адреса могут отображаться в различные виртуальные адресные пространства, при этом возможно использование в каждом процессе разных виртуальных адресов. Отображение не обязательно требует физического перемещения реальных данных или программ (хотя бывает и так);
- уникальный идентификационный номер, называемый идентификатором процесса;
- один или несколько потоков управления.

Поток - это внутренняя составляющая процесса, которой операционная система выделяет процессорное время. Каждый процесс должен иметь, по крайней мере, один поток. Поток включает:

- текущее состояние регистров процессора;
- два стека, один из которых используется при выполнении в режиме ядра, второй - при выполнении в пользовательском режиме;
- участок памяти для работы подсистем, библиотеки времени выполнения,
- динамические библиотеки;
- уникальный идентификатор, называемый идентификатором потока.

Состояние регистров, содержимое стека и области памяти называют контекстом потока.

Основное назначение потоков - дать процессу возможность поддерживать несколько ветвей управления, то есть выполнять больше «дел» одновременно. В многопроцессорной конфигурации

(компьютер с двумя и более процессорами) Windows NT может назначать разные потоки разным процессорам в различные моменты времени, обеспечивая действительно параллельную обработку. В однопроцессорной конфигурации процессор должен выделять кванты времени каждому исполняемому в данный момент потоку.

Состояния потоков

Потоки могут находиться в одном из нескольких состояний:

- *Ready* (готов) – находящийся в пуле (pool) потоков, ожидающих выполнения;
- *Running* (выполнение) - выполняющийся на процессоре;
- *Waiting* (ожидание), также называется idle или suspended, приостановленный - в состоянии ожидания, которое завершается тем, что поток начинает выполняться (состояние Running) или переходит в состояние *Ready*;
- *Terminated* (завершение) - завершено выполнение всех команд потока. Впоследствии его можно удалить. Если поток не удален, система может вновь установить его в исходное состояние для последующего использования.

Приоритет потоков

Термин многозадачность (мультипрограммирование) обозначает возможность управлять несколькими процессами (или несколькими потоками) на базе одного процессора. Многопроцессорной обработкой называется управление некоторым числом процессов или потоков на нескольких процессорах.

Компьютер может одновременно выполнять команды двух разных процессов только в многопроцессорной среде. Однако даже на одном процессоре с помощью переключения задач можно создать впечатление, что одновременно выполняются команды нескольких процессов.

Уровни приоритета потоков

Win32 является многопоточной (multithreaded) ОС, что определяет ее многозадачность. В ней реализована модель вытесняющей (preemptive) многозадачности, в которой операционная система решает, когда каждый поток получает процессорное время, выделяемое квантами времени (time slices), и сколько именно времени выделяется. Временной интервал, выделяемый потокам для выполнения, в Windows называется квантом.

Продолжительность кванта времени зависит от аппаратуры и может фактически меняться от потока к потоку. Например, базовое значение в Windows 95 со составляет 20 мс, для Windows NT Workstation (на базе процессора Pentium) -30 мс, для Windows NT Server - 180 мс.

Каждый поток в системе имеет *уровень приоритета* (priority level), который представляет собой число в диапазоне от 0 до 31. Ниже перечислено то самое основное, что нужно знать:

- если существуют какие-либо потоки с приоритетом 31, которые требуют процессорного времени, то есть не находятся в состоянии ожидания (not idle), операционная система перебирает эти потоки (независимо от того, каким процессам они принадлежат), выделяя по очереди каждому из них кванты времени. Потокам с более низким приоритетом кванты времени совсем не выделяются, поэтому они не выполняются. Если нет активных потоков с приоритетом 31, операционная система ищет активные потоки с уровнем приоритета 30 и т.д. Однако потоки довольно часто простаивают. Тот факт, что приложение загружено, не означает, что все его потоки активны. Поэтому у потоков с более низким приоритетом все же есть возможность работать. Более того, если пользователь нажимает клавишу, относящуюся к процессу, потоки которого простаивают, операционная система временно выделяет процессорное время соответствующему потоку, чтобы он мог обработать нажатие клавиши;
- приоритет 0 зарезервирован исключительно за специальным системным потоком, который называется потоком нулевой страницы (zero page thread). Он освобождает незадействованные области памяти. Существует также поток idle, который работает с уровнем приоритета 0, опрашивая систему в поисках какой-нибудь работы;
- если поток с произвольным приоритетом выполняется в тот момент, когда потоку с большим приоритетом потребовалось процессорное время (например, он получает сообщение о том, что пользователь щелкнул мышью), операционная система немедленно вытесняет поток с меньшим приоритетом и отдает процессорное время потоку с большим. Таким образом, поток может не успеть завершить выделенный ему квант времени;
- для того чтобы перейти с одного потока на другой, система осуществляет переключение контекста (context switch). Это процедура, сохраняющая состояние процессора (регистров и стека) и загрузки соответствующих значений другого потока.

Назначение приоритета потоку

Назначение потоку приоритета происходит в два этапа. Во-первых, каждому процессу в момент создания присваивается класс приоритета. Узнать класс приоритета можно с помощью функции *GetPriorityClass*, а изменить - с помощью функции *SetPriorityClass*. Ниже приведены имена классов приоритета процессов, уровни приоритета и константы, которые используются с этими вышеупомянутыми функциями (как и с функцией *CreateProcess*).

Имя класса приоритета	Уровень приоритета класса	Символьная константа

Idle	4	*IDLE_PRIORITY_CLASS=&H40*
Normal	8	*NORMAL_PRIORITY_CLASS=&H20*
High	13	*HIGH_PRIORITY_CLASS=&H80*
Realtime	24	*REALTIME_PRIORITY_CLASS=&H100*

Большинство процессов должно получать класс уровня приоритета Normal (обычный). Однако некоторым приложениям, таким как приложения мониторинга системы, возможно, более уместно назначать приоритет Idle (ожидания). Назначения приоритета *Realtime* (реального времени) обычно следует избегать, потому что в этом случае потоки изначально получают приоритет более высокий, чем системные потоки, такие как потоки ввода от клавиатуры и мыши, очистки кэша и обработки нажатия клавиш Ctrl+Alt+Del. Такой приоритет может быть подходящим для краткосрочных, критичных к времени выполнения процессов, которые относятся к взаимодействию с аппаратурой.

При создании уровень приоритета потока по умолчанию устанавливается равным уровню класса приоритета процесса, создавшего данный поток. Тем не менее, можно использовать функцию *SetThreadPriority*, чтобы изменить приоритет потока:

```
BOOL SetThreadPriority (
HANDLE hThread,              // Дескриптор потока
int nPriority                // Уровень приоритета потока
);
```

Параметр nPriority используется для изменения приоритета потока относительно приоритета процесса, которому принадлежит данный поток. Возможные значения параметра nPriority и эффект их воздействия на уровень приоритета потока приведены ниже:

Константа	Уровень приоритета потоков
THREAD_PRIORITY_NORMAL	Уровень приоритета класса
THREAD_PRIORITY_ABOVE_NORMAL	Уровень приоритета класса + 1
THREAD_PRIORITY_BELOW_N0RMAL	Уровень приоритета класса - 1
THREAD_PRIORITY_HIGHEST	Уровень приоритета класса + 2
THREAD_PRIORITY_LOWEST	Уровень приоритета класса - 2
THREAD_PRIORITY_IDLE	Устанавливает уровень приоритета 1 для всех классов приоритета процессов за исключением Realtime. В этом случае устанавливает уровень приоритета 16.

THREAD_PRIORITY_TIME_CRITICAL	Устанавливает уровень приоритета 15 для всех классов приоритета процессов за исключением Realtime. В этом случае устанавливает уровень приоритета 31.

Повышение приоритета потока и квант изменений приоритета

Диапазон приоритета от 1 до 15 известен как диапазон динамического приоритета, а диапазон от 16 до 31 - как диапазон приоритета реального времени .

В Windows NT приоритет потока, находящийся в динамическом диапазоне, может временно повышаться операционной системой. Нижний уровень приоритета потока (установленный программистом с помощью API функции) называется уровнем его базового приоритета. API функция Windows NT *SetProcessPriorityBoost* может использоваться для разрешения или запрещения временных изменений приоритета (priority boosting).

Бывают также случаи, когда кванты времени, выделяемые потоку, временно увеличиваются.

Стремясь плавно выполнять операции, Windows будет повышать приоритет потока или увеличивать продолжительность его кванта времени при следующих условиях:

- если поток принадлежит приоритетному процессу, то есть процессу, окно которого активно и имеет фокус ввода;
- если поток первым вошел в состояние ожидания;
- если поток выходит из состояния ожидания;
- если поток совсем не получает процессорного времени.

Дескрипторы и идентификаторы процессов

Разницу между дескрипторами и идентификаторами процессов можно установить, анализируя API-функцию *CreateProcess.* Объявление в VC++ записывается таким образом:

```
BOOL CreateProcess(
   LPCTSTR lpApplicationName,
   LPTSTR lpCommandLine,
   LPSECURITY_ATTRIBUTES lpProcessAttributes,
   LPSECURITY_ATTRIBUTES lpThreadAttributes,
   BOOL bInheritHandles,
   DWORD dwCreationFlags,
```

```
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation
   );
```

Параметры API-функцию *CreateProcess:*

lpApplicationName, lpCommandLine - определяют имя исполняемого файла, которым будет пользоваться новый процесс, и командную строку, передаваемую этому процессу.

lpProcessAttributes, lpThreadAttributes – эти параметры позволяют определить нужные атрибуты защиты для объектов «процесс» и «поток» соответственно. В эти параметры можно занести NULL, и система закрепит за данными объектами дескрипторы защиты по умолчанию. В качестве альтернативы можно объявить и инициализировать две структуры SECURITY_ATTRIBUTES; тем самым Вы создадите и присвоите объектам «процесс» и «поток» свои атрибуты защиты. Вид структуры SECURITY_ATTRIBUTES приведен ниже:

```
typedef struct _SECURITY_ATTRIBUTES { // sa
    DWORD  nLength;
    LPVOID lpSecurityDescriptor;
    BOOL   bInheritHandle;
} SECURITY_ATTRIBUTES;
```

bInheritHandles - указывает, наследовал ли новый процесс дескрипторы от процесса запроса. Если ИСТИНА, каждый наследственный открытый дескриптор в процессе запроса унаследован новым процессом.

dwCreationFlags - определяет флаги, влияющие на то, как именно создается новый процесс. Имеется десять флагов, которые могут комбинироваться булевым оператором ОК.

lpEnvironment – указывает на блок памяти, хранящий строки переменных окружения, которыми будет пользоваться новый процесс. Обычно вместо этого параметра передается NULL, в результате чего дочерний процесс наследует строки переменных окружения от родительского процесса.

lpCurrentDirectory – позволяет установить текущие диски каталог. Если его значение – NULL, рабочий каталог нового процесса будет тем же, что и у приложения, его породившего.

lpStartupInfo – этот параметр указывает на структуру STARTUPINFO:

```
typedef struct _STARTUPINFO { // si
    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;
    WORD    cbReserved2;
    LPBYTE  lpReserved2;
    HANDLE  hStdInput;
    HANDLE  hStdOutput;
    HANDLE  hStdError;
} STARTUPINFO, *LPSTARTUPINFO;
```

Элементы структуры STARTUPINFO используются Windows-функциями при создании процесса. При этом большинство приложений порождает процессы с атрибутами по умолчанию. Но в этом случае Вы должны инициализировать все элементы структуры STARTUPINFO хотя бы нулевыми значениями, а в элемент cb – заносить размер этой структуры.

lpProcessInformation – указывает на структуру PROCESS_INFORMATION:

```
typedef  struct  _PROCESS_INFORMATION (
    HANDLE hProcess;
    HANDLE hTHread;
    DWORD dwProcessID;
    DWORD dwThreadID;
 )  PROCESS_INFORMATION;
```

Ее члены в соответствии с документацией имеют следующие значения:

- HProcess возвращает дескриптор вновь созданного процесса;
- Hthread возвращает дескриптор первичного потока вновь созданного процесса.

Дескриптор используется для спецификации данного потока во всех функциях, которые выполняют операции с объектом Thread;

- DwProcessId возвращает глобальный идентификатор процесса. Время жизни идентификатора - с момента создания процесса до момента его завершения;
- *DwThreId* возвращает глобальный идентификатор потока. Время жизни идентификатора - с момент создания потока до момента его завершения.

Таким образом, между дескриптором и идентификатором процесса (или потока) существуют следующие основные различия:

- дескриптор действует в пределах «своего» процесса, в рамках которого он был создан, в то время как идентификатор работает на системном уровне;
- у каждого процесса есть только один идентификатор, но может быть несколько дескрипторов;
- некоторым API-функциям требуется идентификатор, в то время как другим - дескриптор процесса.

Следует подчеркнуть, что, хотя дескриптор является зависимым от процесса, один процесс может иметь дескриптор другого процесса. Иными словами, если процесс А имеет дескриптор процесса В, то этот дескриптор идентифицирует процесс В, но действует только в процессе А. Он может использоваться в процессе А для вызова некоторых API-функций, которые имеют отношение к процессу В (однако память процесса В остается недоступной для процесса А).

Дескрипторы модулей

Каждый модуль (DLL, OCX, DRV и т.д.), загруженный в пространство процесса, имеет свой дескриптор (module handle), называемый также логическим номером экземпляра (instance handle). В 16-разрядной Windows данными терминами обозначались разные объекты, в 32-разрядной системе это один и тот же объект.

Дескриптором исполняемого модуля является начальный или базовый адрес данного исполняемого модуля в адресном пространстве процесса. Это объясняет, почему такой дескриптор имеет смысл только в рамках процесса, включающего данный модуль.

Базовый адрес загрузочного (EXE) модуля, создаваемого VC++, по умолчанию имеет значение &H400000. Однако программист в VC++ может изменить этот адрес по умолчанию, и есть серьезные основания поступать именно так для того, чтобы избежать конфликта с другими модулями (перераспределение памяти, связанное с конфликтом базовых адресов, требует слишком большого объема физической памяти и временных затрат). Кроме того, модули, создаваемые другими программными средами, могут иметь другие базовые адреса, устанавливаемые по умолчанию.

Дескрипторы модулей часто используются для вызова API-функций, которые выделяют

ресурсы данному процессу. Например, функция LoadBitmap загружает ресурс растровой картинки из загрузочного файла данного процесса. Ее декларация записывается так:

```
HBITMAP LoadBitmap(
   HINSTANCE hInstance,        // Дескриптор экземпляра приложения.
   LPCTSTR  IpBitmapeName      //  Адрес  имени ресурса растровой картинки.
 );
```

Обратите внимание на то, что дескриптор модуля загрузочного файла процесса часто ошибочно называют дескриптором модуля процесса, а об имени загрузочного файла говорят как об имени модуля процесса.

Идентификация процесса

Следующие четыре объекта часто встречаются в программировании API, связанном с процессами:

- идентификатор процесса;
- дескриптор процесса;
- полное имя загрузочного файла;
- дескриптор модуля загрузочного файла процесса.

Получение дескриптора процесса по его идентификатору

Получить дескриптор процесса по его идентификатору относительно легко, но обратная операция вряд ли возможна (по крайней мере, прямо).

По идентификатору можно определить дескриптор любого процесса с помощью функции *OpenProcess:*

```
HAMDLE  OpenProcess(
DWORD dwDesiredAccess,       // Флаг доступа.
BOOL bInheritHandle,         // Флаг наследования дескриптора.
DWORD dwProcessID            // Идентификатор процесса.  .
);
```

dwDesiredAccess - имеет отношение к правам доступа и может принимать различные значения:

- *PROCESS_ALL_ACCESS* эквивалентно установке флагов полного доступа;
- *PROCESS_DUP_HANDLE* использует дескриптор как исходного процесса, так и принимающего в функции *DuplicateHandle* для копирования (дублирования) дескриптора;
- *PROCESS_QUERY_INFORMATTON* задействует дескриптор процесса для чтения информации из объекта *Process*;

- PROCESS_VM_OPERATION использует дескриптор процесса для модификации виртуальной памяти процесса;
- PROCESS_TERMINATE работает для завершения процесса с его дескриптором в функции *TerminateProcess*;
- process_vm_read применяет для чтения из виртуальной памяти процесса его дескриптор в функции *ReadProcessMemory*;
- PROCESS_VM_WRITE использует для записи в виртуальную память процесса его дескриптор в функции *WriteProcessMernory*;
- SYNCHRONIZE (Windows NT) работает с дескриптором процесса в любой из функций ожидания, таких как *WaitForSingleObject*, для ожидания завершения процесса.

bInheritHandle - установлен в значение *True*, для того чтобы позволить порожденным процессам наследовать дескриптор. Иначе говоря, порожденный процесс получает дескриптор родительского процесса. Отметим, что значение дескриптора может изменяться.

dwProcessID - должен иметь значение идентификатора того процесса, дескриптор которого нужно узнать. Функция *OpenProcess* возвращает дескриптор указанного процесса.

Полученный дескриптор обязательно должен быть закрыт, если он больше не нужен. Это делается путем вызова функции *CloseHandle*:

BOOL CloseHandle(
HANDLE hObject // Дескриптор закрываемого объекта.
);

Имена файлов и дескрипторы модулей

Перейти от имени файла модуля к дескриптору модуля и наоборот не составляет особого труда, по крайней мере, в пределах одного процесса. Функция *GetModuleFileName* принимает дескриптор модуля, чтобы вернуть полное имя (имя и путь) исполняемого файла:

DWORD GetModuleFileName(
HMODULE hModule, // Дескриптор модуля.
LPTSTR IpFilename, // Указатель на буфер-приемник пути к модулю.
DWORD nSize // Размер буфера в символах.
);

Эта функция работает только из того процесса, дескриптор модуля которого задается. Ее нельзя использовать из другого процесса.

Функция *GetModuleHandle*, которая так же применяется внутри одного процесса, выполняет обратное действие - возвращает дескриптор модуля по имени файла (без пути):

HMODULE GetModuleHandle(

LPCTSTR IpModuleName // Имя модуля.

);

С использованием приведенных API-функций можно разработать функцию (первое задание в лабораторной работе), которая принимая дескриптор, имя или полное имя модуля, возвращает другие два элемента в своих выходных параметрах. Возможны четыре варианта работы этой функции. Например:

1. Получить дескриптор, имя и полное имя EXE.

   Handle: &HFFFFFFFF

   Name: VB6.EXE

   FName: G:\Visual Studio\VB98\VB6.EXE

2. Получить имя и полное имя модуля.

   Задаем: Handle: &H77E70000

   Получаем:

   Name: User32.dll

   FName: G:\WINNT\system32\USER32.DLL

3. Получить дескриптор и полное имя.

   Задаем: Name: User32.dll

   Получаем:

   Handle: &H77E70000

   FName: G:\WINNT\system32\USER32.DLL

4. Получить дескриптор и имя.

   Задаем: FName: G:\WINNT\system32\USER32.DLL

   Получаем:

   Handle: &H77E70000

   Name: User32.dll

Получение идентификатора текущего процесса

Чтобы получить идентификатор текущего процесса, можно использовать функцию *GetCurrentProcessId*, объявление которой выглядит так:

DWORD GetCurrentProcess ID(VOID)

Эта функция возвращает идентификатор процесса. Значение идентификатора процесса может быть в верхнем диапазоне *unsigned long,* поэтому может потребоваться преобразование возвращаемого значения. Следует также учесть, что данная функция работает только в текущем процессе. Не существует способа определить идентификатор другого процесса, кроме как

получить список всех процессов и выбрать из него тот процесс, характеристики которого требуются.

Получение идентификатора процесса от окна

Существует функция *FindWindow*. Она объявляется следующим образом:

```
HWND FindWindow(
LPCTSTR IpClassName,                    // Указатель на имя класса.
LPCTSTR IpWindowName                    // Указатель на имя окна.
);
```

Функция использует имя класса или заголовок окна для получения дескриптора окна. Имея дескриптор, можно вызвать функцию *GetWindowThreadProcessId,* возвращающую идентификатор потока, который создал данное окно, и идентификатор процесса, которому принадлежит данный поток. Синтаксис выглядит так:

```
DWORD GetWindowThreadProcessId(
HMD hWnd,                               // Дескриптор окна.
LPDWORD IpdwProcessId                   // Адрес переменной для идентификатора
                                        // процесса.
);
```

Данная функция возвращает идентификатор потока. Кроме того, если ей передается указатель на *DWORD* в *IpdwProcessIdl(),* в целевой переменной возвращается идентификатор процесса.

Получение имен и дескрипторов модулей

Обычно одному процессу принадлежит много модулей, загруженных в его адресное пространство, и это усложняет задачу получения дескрипторов и имен модулей. Для Windows 9x и Windows NT необходимо использовать совершенно разные методы.

Windows NT требует использования динамической библиотеки, называемой *PSAPI.DLL,* что означает API состояния процесса (Process Status API). Эта библиотека, несовместимая с Windows 9x, экспортирует функции перечисления всех процессов в системе и всех драйверов устройств. Она предоставляет возможность получения информации обо все модулях, исполняемых данным процессом. Для перечисления потоков в операционной системе Windows NT нужно использовать динамическую библиотеку *PDH.DLL*, что означает «вспомогательная система для оценки характеристик производительности» (Performance Data Helper), которая поставляется в комплекте инструментальных средств NT Resource Toolkit.

С другой стороны Windows 9x поддерживает функции *Toolhelp* (вспомогательные средства) в своей версии динамической библиотеки *KERNEL32.DLL.* Они используются для фиксации состояния области памяти любою процесса. Используя этот «снимок» памяти, можно

получить любую информацию о текущих процессах, а также о модулях и потоках каждого процесса (в отличие от *PSAPI.DLL* здесь отсутствует информация о драйверах устройств).

Таким образом, для Windows NT и Windows 9x придется писать разные программы.

Псевдодескрипторы процессов

Функция *GetcurrentProcess* возвращает псевдодескриптор текущего процесса*:*

HANDLE GetCurrentProcess(VOID)

Псевдодескриптор (pseudohandle) представляет собой упрощенный вариант дескриптора. По определению, псевдодескриптор - это зависимое от процесса число, которое служит идентификатором процесса и может использоваться в вызовах тех API-функций, которым требуемся дескриптор процесса.

Хотя назначение псевдодескрипторов и обычных дескрипторов почти одно и то же, у них все же есть некоторые существенные различия. Псевдодескрипторы не могут наследоваться порожденными процессами, как настоящие дескрипторы (real handler). К тому же псевдодескрипторы ссылаются только на текущий процесс, а настоящие дескрипторы могут ссылаться и на внешний (foreign).

Windows предоставляет возможность получения настоящего дескриптора по псевдодескриптору при помощи API-функции *DuplicateHandle*. Она определяется так:

```
BOOL DuplicateHandle(
HANDLE hSourceProcessHandle,    // Дескриптор процесса-источника.
HANDLE hSourceHandle,           // Копируемый дескриптор.
HANDLE hTargetProcessHandle,    // Дескриптор процесса-приемника.
LPHANDLE IpTargetHandle,        // Указатель на копию дескриптора.
DWORD dwDesiredAccess,          // Доступ к копии дескриптора.
BOOL bInheritHandle,            // Флаг наследования дескриптора.
DWORD dwOptions                 // Необязательные опции.
)
```

С использованием выше приведенных API-функций можно разработать программу (второе задание в лабораторной работе), которая будет выполнять следующее:

1. Используя функцию *GetCurrentProcessId* определяет идентификатор текущего процесса.

2. Используя функцию *GetCurrentProcess* определяет псевдодескриптор текущего процесса.

3. Используя функцию *DuplicateHandl и значение* псевдодескриптора определяет дескриптора текущего процесса.

4. Используя функцию *OpenProcess* определяет копию дескриптора текущего

процесса.

5. Закрывает дескриптор, полученный функцией *DuplicateHandl.*

6. Закрывает дескриптор, полученный функцией *OpenProcess.*

Настоящий дескриптор обязательно должен быть закрыт, а псевдодескриптор закрывать не нужно. Вывод на компьютере может быть таким:

Идентификатор текущего процесса: 183

Псевдодескриптор: 4294967295

Дескриптор, полученный функцией DuplicateHandle: 412

Дескриптор, полученный функцией OpenProcess: 392

Закрываем дескриптор, полученный функцией OpenProcess: 392

Закрываем дескриптор, полученный функцией DuplicateHandle: 412

Перечисление процессов

Рассмотрим проблему перечисления процессов, потоков и модулей в операционной системе Windows (это третье задание в этой работе). Способы ее решения в Windows NT и Windows 9x различны.

Перечисление процессов в Windows 9x (использование ToolHelp32)

ToolHelp32 — это семейство функций и процедур, составляющих подмножество Win32 API, которые позволяют получить сведения о некоторых низкоуровневых аспектах работы ОС, В частности, сюда входят функции, с помощью которых можно получить информацию обо всех процессах, выполняющихся в системе в данный момент, а также потоках, модулях, принадлежащих каждому процессу. Большинство данных, получаемых от функций ToolHelp32, используется главным образом приложениями, которые должны заглядывать "внутрь" ОС.

Семейство процедур и функций ToolHelp32 API доступно только в варианте реализации Win32 для Windows 95/98. В среде Windows NT вызов их приведет к нарушению системы защиты и безопасности NT-процессов. Поэтому приложения, которые используют функции ToolHelp32, работоспособны только под управлением Windows 95/98, но не Windows NT.

Типы и определения функций ToolHe1p32 размещаются в модуле TlHelp32, поэтому при работе с этими функциями не забудьте включить его имя в список инструкции uses (в Delphi).

Моментальные снимки

Благодаря многозадачной природе среды Win32 такие объекты, как процессы, потоки, модули и т.п., постоянно создаются, разрушаются и модифицируются. И поскольку состояние компьютера непрерывно изменяется, системная информация, которая, возможно, будет иметь значение в данный момент, через секунду уже никого не заинтересует. Например,

предположим, что вы хотите написать программу для регистрации всех модулей, загруженных в систему. Поскольку операционная система в любое время может прервать выполнение потока, отрабатывающего вашу программу, чтобы предоставить какие-то кванты времени другому потоку в системе, модули теоретически могут создаваться и разрушаться даже в момент выборки информации о них.

В этой динамической среде имело бы смысл на мгновение заморозить систему, чтобы получить такую системную информацию. В ToolHe1p32 не предусмотрено средств замораживания системы, но есть функция, с помощью которой можно сделать "снимок" системы в заданный момент времени. Эта функция называется CreateToolhelp32Snapshot(), и ее объявление (в Delphi) выглядит следующим образом:

function CreateToolhelp32Snapshot(dwFlags, th32ProcesslD: DWORD): THandle; stdcall;

• Параметр dwFlags означает тип информации, подлежащий включению в моментальный снимок. Этот параметр может иметь одно из перечисленных в таблице значений.

Значение	Описание
TH32CS_INHERIT	Означает, что дескриптор снимка будет наследуемым
TH32CS_SNAPALL	Эквивалентно заданию значений TH32CS_SNAPHEAPLIST, TH32CS_SNAPMODULE, TH32CS_SNAPPROCESS И TH32CS_SNAPTHREAD
TH32CS_SNAPHEAPLIST	Включает в снимок список куч заданного процесса Win32
TH32CS_SNAPMODULE	Включает в снимок список модулей заданного процесса Win32
TH32CS_SNAPPROCESS	Включает в снимок список процессов Win32
TH32CS_ SNAPTHREAD	Включает в снимок список потоков Win32

• Функция CreateToolhelp32Snapshot() возвращает дескриптор созданного снимка или -1 в случае ошибки. Возвращаемый дескриптор работает подобно другим дескрипторам Win32 относительно процессов и потоков, для которых он действителен.

Следующий код создает дескриптор снимка, который содержит информацию обо всех процессах, загруженных в настоящий момент (EToolHelpError — это исключительная ситуация, определенная программистом):

```
var
Snap: THandle;
begin
Snap := CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
if Snap = -1 then
```

raise EToolHelpError.Create('CreateToolHelp32Snapshot failed');

end;

По завершении работы с созданным функцией CreateToolhelp32Snapshot() дескриптором, для освобождения связанных с ним ресурсов используйте функцию CloseHandle().

Обработка информации о процессах

Имея дескриптор снимка, содержащий информацию о процессах, можно воспользоваться двумя функциями ToolHelp32, которые позволяют последовательно просмотреть сведения обо всех процессах в системе. Функции Process32First() и Process32Next() определены следующим образом:

function Process32First(hSnapshot: THandle; var Ippe: TProcessEntry32): BOOL; stdcall;

function Process32Next(hSnapshot: THandle; var Ippe: TProcessEntry32): BOOL; stdcall;

Первый параметр у обеих функций является дескриптором снимка, возвращаемым функцией CreateToolhelp32Snapshot().

Второй параметр, Ippe, представляет собой запись TProcessEntry32, которая передается по ссылке. По мере прохождения по элементам перечисления функции будут заполнять эту запись информацией о следующем процессе. Запись TProcessEntry32 определяется так:

type TProcessEntry32 = record

dwSize: DWORD;

cntUsage: DWORD;

th32ProcessID: DWORD;

th32DefaultHeapID: DWORD;

th32ModuleID: DWORD;

cntThreads: DWORD;

th32ParentProcessID: DWORD;

pcPriClassBase: Longint;

dwFlags; DWORD;

szExeFile: array[Q..MAX_PATH - 1] of Char;

end;

- В поле dwSize содержится размер записи TProcessEntry32. До использования этой записи поле dwSize должно быть инициализировано значением SizeOf (TProcessEntry32).
- В поле cntUsage хранится значение счетчика ссылок процесса. Когда это значение станет равным нулю, операционная система выгрузит процесс.
- В поле th32ProcessID содержится идентификационный номер процесса.
- Поле th32DefaultHeapID предназначено для хранения идентификатора {ID} для кучи процесса, действующей по умолчанию. Этот ID имеет значение только для функций ToolHelp32, и его нельзя использовать с другими функциями Win32.

• Поле thHodulelD идентифицирует модуль, связанный с процессом. Это поле имеет значение только для функций ToolHelp32.

• По значению поля cntThreads можно судить о том, сколько потоков начало выполняться в данном процессе.

• Поле th32ParentProcessID идентифицирует родительский процесс для данного процесса.

• В поле pcPriClassBase хранится базовый приоритет процесса. Операционная система использует это значение для управления работой потоков,

• Поле dwFlags зарезервировано.

• В поле szExeFile содержится строка с ограничивающим нуль-символом, которая представляет собой путь и имя файла EXE-программы или драйвера, связанного с данным процессом.

После создания снимка, содержащего информацию о процессах, для опроса данных по каждому процессу следует вызвать сначала функцию Process32First(). а затем вызывать функцию Process32Next() до тех пор, пока она не вернет значение False.

Для отображения значка вместе с именем приложения, что придаст программе более профессиональный вид, можно использовать API-функцию ExtractIcon из модуля ShellAPI.

Обработка информации о потоках

Для составления списка потоков некоторого процесса в ToolHelp32 предусмотрены две функции, которые аналогичны функциям, предназначенным для регистрации процессов: Thread32First() и Thread32Next (). Эти функции объявляются следующим образом;

function Thread32First(hSnapshot: THandle; var Ipte: TThreadEntry32): BOOL; stdcall;

Помимо обычного параметра hSnapshot, этим функциям также передается по ссылке параметр типа TThreadEntry32. Как и в случае функций, работающих с процессами, каждая нз них заполняет запись TThreadEntry32, объявление которой имеет вид

```
type
TThreadEntry32 = record
dwSize: DWORD;
cntUsage: DWORD;
th32ThreadID:  DWORD;
th32OwnerprocessiD: Dword;
tpBasePri:  Longint;
tpDeltaPri: Longint;
dwFlags: DWORD;
end;
```

• Поле dwSize опрелеляет размер записи, и поэтому оно должно быть инициализировано значением SizeOf (TThreadEntry32) до использования этой записи.

• В поле cntUsage содержится счетчик ссылок данного потока. При обнулении этого счетчика поток выгружается операционной системой.

• Поле th32ThreadID представляет собой идентификационный номер потока, который имеет значение только для функций ToolHelp32.

• В поле th320wnerProcessID содержится идентификатор (ID) процесса, которому принадлежит данный поток. Этот ID можно использовать с другими функциями Win32.

• Поле tpBasePri представляет собой базовый класс приоритета потока. Это значение одинаково для всех потоков данного процесса. Возможные значения этого поля обычно лежат в диапазоне от 4 до 24. Описания этих значений приведены в следующей таблице.

Значение	Описание
4	Ожидающий
8	Нормальный
13	Высокий
24	Реальное время

• Поле tpDeltaPri представляет собой дельта-приоритет (разницу), определяющий величину отличия реального приоритета от значения tpBasePri. Это число со знаком, которое в сочетании с базовым классом приоритета отображает общий приоритет потока. Константы, определенные для всех возможных значений дельта-приоритета, перечислены в следующей таблице.

Константа	Значение
THREAD_PRIORITY_IDLE	-15
THREAD_PRIOR ITY_LOWEST	-2
THREAD PRIORITY BELOW NORMAL	-1
THREAD_PRIOR IT Y_NORMAL	0
THREAD_PRIOR I TY_ABOVE_NORHAL	1
THREAD_PRIOR ITY_H IGHEST	2
THREAD_PRI OR ITY_TIME_CRIT ICAL	15

• Поле dwFlags в данный момент зарезервировано и не должно использоваться.

Списки потоков, полученные с помощью функций ToolHelp32 не связываются с определенным процессом. Поэтому при сканировании потоков нужно обязательно проверять

результат, так чтобы потоки были связаны с интересующим вас процессом.

Обработка информации о модулях

Опрос модулей выполняется практически так же, как опрос процессов или потоков. Для этого в ToolHe1p32 предусмотрены две функции: Module32First() и Module32Next(), которые определяются следующим образом:

function Module32First(hSnapshot: THandle; var Ipme: TModuleEtitry32): BOOL; stdcall;

function Module32Next(hSnapshot: THandle; var Ipme: TModuleEntry32): BOOL; stdcall;

Первым параметром в обеих функциях является дескриптор снимка, а вторым var-параметром — запись TModuleEntry32. Ее определение имеет следующий вид:

```
type TModuleEntry32 = record
dwSize: DWORD;
th32HoduleID:  DWORD;
th32ProcessID:  DWORD;
GlblcntUsage:  DWORD;
ProccntUsage:  DWORD;
modBaseAddr: PBYTE;
modBaseSize: DWORD;
hHodule: HMODULE;
szModule: array[0 .. MAX_MODULE_NAME32 + 1] of Char;
szExePath: array[0 .. MAX_PATH  + 1] of Char;
end;
```

• Поле dwSize определяет размер записи и поэтому должно быть инициализировано значением SizeOf (TModuleEntry32) до использования этой записи.

• Поле th32ModuleID представляет собой идентификатор модуля, который имеет значение только для функций Too1He1p32.

• Поле th32processID содержит идентификатор (ID) опрашиваемого процесса. Этот ID можно использовать с другими функциями Win32.

• Поле GlblcntUsage содержит глобальный счетчик ссылок данного модуля.

• Поле ProccntUsage содержит счетчик ссылок модуля в контексте процесса-владельца.

• Поле modBaseAddr представляет собой базовый адрес модуля в памяти. Это значение действительно только в контексте идентификатора процесса th32ProcessID.

• Поле modBaseSize определяет размер (в байтах) модуля в памяти.

• В поле hHodule содержится дескриптор модуля. Это значение действительно только в контексте идентификатора процесса th32ProcessID.

• В поле szHodole содержится строка с именем модуля, завершающаяся нуль-символом.

• Поле szExepath предназначено для хранения строки с ограничивающим нуль-

символом, содержащей полный путь модуля.

СОДЕРЖАНИЕ ОТЧЕТА

1. Наименование лабораторной работы, ее цель.
2. Программа, которая
   - принимая дескриптор, имя или полное имя модуля, возвращает другие два элемента в своих выходных параметрах (выполнить задание для своей программы и для любой известной библиотеки).
   - будет выполнять последовательно по шагам следующее:
     - Используя функцию *GetCurrentProcessId* определит идентификатор текущего процесса.
     - Используя функцию *GetCurrentProcess* определит псевдодескриптор текущего процесса.
     - Используя функцию *DuplicateHandl и значение* псевдодескриптора определит дескриптора текущего процесса.
     - Используя функцию *OpenProcess* определит копию дескриптора текущего процесса.
     - Закроет дескриптор, полученный функцией *DuplicateHandl.*
     - Закроет дескриптор, полученный функцией *OpenProcess.*

   - выдает список перечисления всех процессов, потоков, модулей и их свойства в системе:
     - для Windows NT (и список загруженных драйверов устройств).

3. Примеры разработанных приложений (результаты и тексты программ).