14장 운영체제의 정보 엿보기

* 예제 프로젝트 SysInfo - 운영체제 정보 얻기
* 플랫폼 중립적 설계
* 윈도우즈 95/98의 ToolHelp32 사용법
* 윈도우즈 NT의 PSAPI 사용법

이번 장에서는 운영체제에 대한 여러 가지 정보를 얻는 SysInfo라는 예제 프로그램을 만든다. 이 프로그램을 만드는 과정을 따라가다 보면, 비교적 덜 알려진 API들을 이용해서 운영체제에 관한 상세한 정보를 얻는 방법을 배우게 될 것이다. 또한, 윈도우즈 95/98에서 정보를 얻을 때와 윈도우즈 NT에서 정보를 얻을 때의 차이점도 나온다. 그리고 메모리 사용량이라든지, 윈도우즈 버전 정보, 환경 변수 설정, 적재된 모듈들의 목록을 얻는 방법도 배울 수 있다. 또한, 프로그래밍 기법의 측면에서 본다면 다양한 정보를 일관성 있고 편리한 사용자 인터페이스로 제공하는 기법도 배우게 될 것이다. 그와 더불어, 윈도우즈 3.x에서 애용했던 API 함수들이 Win32에서 어떤 함수들로 대치되었는지도 알게 될 것이다.

운영체제에 대한 정보를 얻어야 하는 이유는 다양하다. 메모리 사용량을 실시간으로 체크하다가 메모리가 부족할 것 같으면 사용자에게 경고를 보내는 간단한 유틸리티를 만들어 봐도 재미 있을 것이다. 또는, 메모리에 올라온 모듈을 직접 제거하는 유틸리티도 생각할 수 있다. 또는, 환경 변수(SET으로 설정하는 것들)의 값으로 디렉토리를 찾거나, 프로그램의 설정을 바꿀 수도 있다. 어쨌든, 운영체제에 대한 상세한 정보를 얻을 수 있다면 애플리케이션이 할 수 있는 일의 범위가 한층 확장될 것이다.

예제 프로젝트 SysInfo - 운영체제 정보 얻기

그럼 몸을 아니 머리를 푸는 의미에서, 이번 예제의 사용자 인터페이스 중 자식 폼부터 살펴보자. 이 애플리케이션은 InfoForm이라는 자식 폼을 이용해서 다양한 정보를 표시한다(그림 14.1). 이 자식 폼을 통해서 메모리나 하드웨어 정보, 운영체제의 버전, 환경 변수의 설정 등을 표시하게 된다.

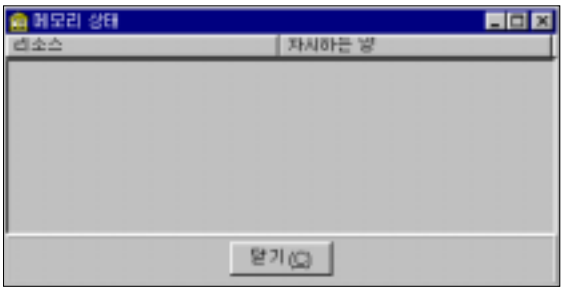


그림 14.1
이번 예제의 자식 폼인 InfoForm

폼의 구성은 매우 간단하다. 폼은 THeaderListbox(이 책이 제공하는 사용자 컴포넌트로, 21장에서 만든 TddgTabListBox를 개선한 것이다)와 TButton 하나로 이루어져 있다. THeaderListbox 컨트롤은 THeader 컴포넌트와 TListBox 컨트롤을 결합한 것이다. 이 컴포넌트는 부록 CD의 D5DDG 패키지를 설치하면 자동적으로 컴포넌트 팔레트에 등록된다. 헤더의 컬럼 너비를 변경하면 목록 상자의 내용도 그에 맞게 다시 그려진다. 앞에서 이야기한 다양한 정보들이 THeaderListbox 컴포넌트인 InfoLB에 표시되며, Close 버튼은 폼을 닫는 역할을 한다.

문자열 만들기

이 애플리케이션은 Format() 함수를 주로 이용해서 운영체제의 정보를 미리 정의한 문자열과 결합한다. 그 문자열들은 다음과 같이 자식 폼 유닛인 InfoU.pas의 const절에 정의되어 있다.

```
const
  { 메모리 상태 관련 문자열 }
  SMemUse   = 'Memory in useq%d%%';
  STotMem   = 'Total physical memoryq$%.8x bytes';
  SFreeMem  = 'Free physical memoryq$%.8x bytes';
  STotPage  = 'Total page file memoryq$%.8x bytes';
  SFreePage = 'Free page file memoryq$%.8x bytes';
  STotVirt  = 'Total virtual memoryq$%.8x bytes';
  SFreeVirt = 'Free virtual memoryq$%.8x bytes';

  { 운영체제 버전 정보 관련 문자열 }
  SOSVer    = 'OS Versionq%d.%d';
  SBuildNo  = 'Build Numberq%d';
  SOSPlat   = 'Platformq%s';
  SOSWin32s = 'Windows 3.1x running Win32s';
  SOSWin95  = 'Windows 95';
  SOSWinNT  = 'Windows NT';

  { 시스템 관련 문자열 }
```

```
SProc     = 'Processor Arhitectureq%s';
SPIntel   = 'Intel';
SPageSize = 'Page Sizeq$%.8x bytes';
SMinAddr  = 'Minimum Application Addressq$%p';
SMaxAddr  = 'Maximum Application Addressq$%p';
SNumProcs = 'Number of Processorsq%d';
SAllocGra = 'Allocation Granularityq$%.8x bytes';
SProcLevl = 'Processor Levelq%s';
SIntel3   = '80386';
SIntel4   = '80486';
SIntel5   = 'Pentium';
SIntel6   = 'Pentium Pro';
SProcRev  = 'Processor Revisionq%.4x';

{ 디렉토리 관련 문자열 }
SWinDir   = 'Windows directoryq%s';
SSysDir   = 'Windows system directoryq%s';
SCurDir   = 'Current directoryq%s';
```

각 문자열에 끼어 있는 q가 눈에 띌 것이다. q는 THeaderListbox 컴포넌트의 DelimChar 속성에 설정된 분리 문자로, THeaderListbox는 DelimChar 속성에 해당하는 문자를 만나면 그 이후의 문자열은 다음 컬럼에 표시한다.

문자열들을 개별적으로 포매팅하지 않고, 미리 정의된 문자열과 Format()을 사용하는 이유는 기본적으로 세 가지이다.

- Format()은 여러 가지 데이터형을 받아들일 수 있기 때문에 IntToStr()이나 IntToHex() 같은 문자열 변환 함수들을 지저분하게 끼워넣을 필요가 없다.
- 또한, Format()은 문자열 안에 여러 가지 데이터형 변수를 쉽게 포함시킬 수 있기 때문에(%s나 %d 등) 일일이 형 변환 함수들을 사용할 필요가 없다.
- 그리고 최종 문자열의 틀로 사용할 문자열과 실제 데이터를 분리해서 관리할 수 있기 때문에 유지/보수가 쉬워진다.



Format()으로 퍼센트 기호(%)를 표시하려면 퍼센트 기호를 두 번(%%) 써주면 된다.

메모리 상태 얻기

그럼, InfoLB에 표시할 첫 번째 항목인 메모리 상태를 얻는 방법을 알아보자. 운영체제의 현재 메모리 상태는 API 함수 GlobalMemoryStatus()로 얻는다. GlobalMemoryStatus()는 TMemoryStatus형의 var 인자를 하나 받아들이는 프로시저이다. 이 레코드의 정의는 다음과 같다.

```
type
  PMemoryStatus = ^TMemoryStatus;
  _MEMORYSTATUS = record
    dwLength: DWORD;
    dwMemoryLoad: DWORD;
    dwTotalPhys: DWORD;
    dwAvailPhys: DWORD;
    dwTotalPageFile: DWORD;
    dwAvailPageFile: DWORD;
    dwTotalVirtual: DWORD;
    dwAvailVirtual: DWORD;
  end;
  {$EXTERNALSYM _MEMORYSTATUS}
  TMemoryStatus = _MEMORYSTATUS;
```

- 첫 번째 인자 dwLength는 TMemoryStatus의 크기이다. 이후 버전의 윈도우즈에서는 이 레코드의 크기가 변할 수도 있으므로 dwLength에는 SizeOf(TMemoryStatus)를 통해서 얻은 값을 넣어야 한다.
- dwMemoryLoad 필드는 메모리 사용량을 뜻하는 것이다. 0은 메모리가 전혀 사용되지 않고 있다는 뜻이며, 100은 모든 메모리가 사용중이라는 뜻이다.
- dwTotalPhys는 물리적 메모리의 용량(컴퓨터에 설치된 램의 용량, 바이트 단위)을 뜻하며, dwAvailPhys는 현재 사용할 수 있는 물리적 메모리 용량을 뜻한다.

- dwTotalPageFile은 하드 디스크 페이지 파일에 담을 수 있는 총 용량(바이트 단위)을 뜻한다. 그리고 현재 사용할 수 있는 페이지 파일의 크기는 dwAvailPageFile이다.
- dwTotalVirtual은 호출한 프로세스의 가상 주소 공간의 전체 가상 메모리 용량(바이트 단위)이다. dwAvilVirtual은 그 중 사용할 수 있는 용량을 뜻한다.

다음 코드는 메모리 상태를 얻어서 목록 상자에 추가하는 코드이다.

```
procedure TInfoForm.ShowMemStatus;
var
  MS: TMemoryStatus;
begin
  InfoLB.DelimChar := 'q';
  MS.dwLength := SizeOf(MS);
  GlobalMemoryStatus(MS);
  with InfoLB.Items, MS do
  begin
    Clear;
    Add(Format(SMemUse, [dwMemoryLoad]));
    Add(Format(STotMem, [dwTotalPhys]));
    Add(Format(SFreeMem, [dwAvailPhys]));
    Add(Format(STotPage, [dwTotalPageFile]));
    Add(Format(SFreePage, [dwAvailPageFile]));
    Add(Format(STotVirt, [dwTotalVirtual]));
    Add(Format(SFreeVirt, [dwAvailVirtual]));
  end;
  InfoLB.Sections[0].Text := 'Resource';
  InfoLB.Sections[1].Text := 'Amount';
  Caption:= 'Memory Status';
end;
```



GlobalMemoryStatus()를 호출하기 전에 TMemoryStatus 레코드의 dwLength의 값을 제대로 설정해야 한다.

그림 14.2는 InfoForm에 메모리 상태 정보가 표시된 모습이다.

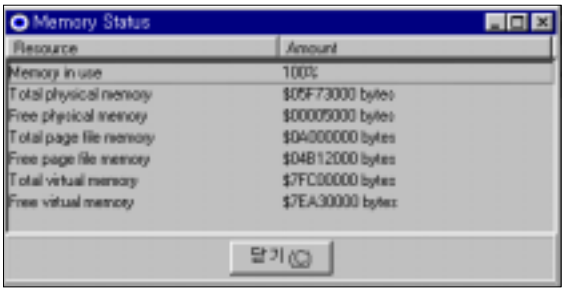


그림 14.2
메모리 상태 정보가 표시된 모습

운영체제의 버전 얻기

윈도우즈와 Win32 운영체제의 버전을 얻을 때에는 GetVersionEx() API 함수를 사용한다. GetVersionEx()는 TOSVersionInfo 레코드를 참조 전달로 받아들인다. 이 레코드의 정의는 다음과 같다.

```
type
  POSVersionInfoA = ^TOSVersionInfoA;
  POSVersionInfo = POSVersionInfoA;
  _OSVERSIONINFOA = record
    dwOSVersionInfoSize: DWORD;
    dwMajorVersion: DWORD;
    dwMinorVersion: DWORD;
    dwBuildNumber: DWORD;
    dwPlatformId: DWORD;
    szCSDVersion: array[0..127] of AnsiChar; { PSS 사용 위한 유지보수 문자열 }
  end;
```

- GetVersionEx()를 호출하기 전에 dwOSVersionInfoSize 필드를 SizeOf (TOSVersionInfo)값으로 설정해야 한다.
- dwMajorVersion은 운영체제의 메이저 릴리즈 번호이다. 즉, 운영체제의 버전이 4.0이면 이 필드의 값은 4이다.

- dwMinorVersion은 운영체제의 마이너 릴리즈 번호이다. 즉, 운영체제의 버전이 4.0이면 이 필드의 값은 0이다.
- dwBuildNumber 필드의 하위 워드는 운영체제의 빌드 번호이다.
- dwPlatformId는 현재 Win32 플랫폼을 뜻한다. 이 필드는 다음 중 하나의 값을 가진다.

값	플랫폼
VER_PLATFORM_WIN32s	윈도우즈 3.1상의 Win32s
VER_PLATFORM_WIN32_WINDOWS	윈도우즈 95상의 Win32
VER_PLATFORM_WIN32_NT	윈도우즈 NT상의 Win32

- szCSDVersion에는 추가적인 임의의 운영체제 정보가 들어간다. 이 값은 종종 빈 문자열이다.

다음은 InfoLB에 운영체제 버전 정보를 채우는 코드이다.

```
procedure TInfoForm.GetOSVerInfo;
var
  VI: TOSVersionInfo;
begin
  VI.dwOSVersionInfoSize := SizeOf(VI);
  GetVersionEx(VI);
  with InfoLB.Items, VI do
  begin
    Clear;
    Add(Format(SOSVer, [dwMajorVersion, dwMinorVersion]));

    Add(Format(SBuildNo, [LoWord(dwBuildNumber)]));
    case dwPlatformID of
      VER_PLATFORM_WIN32S: Add(Format(SOSPlat, [SOSWin32s]));
      VER_PLATFORM_WIN32_WINDOWS: Add(Format(SOSPlat, [SOSWin95]));
      VER_PLATFORM_WIN32_NT: Add(Format(SOSPlat, [SOSWinNT]));
    end;
  end;
end;
```



윈도우즈 3.x에서 운영체제의 버전 정보를 얻을 때에는 GetVersion() 함수를 사용했다. 그러나 Win32에서는 GetVersion() 대신 GetVersionEx()를 사용해야 한다.

디렉토리 정보 얻기

운영체제는 윈도우즈나 시스템 디렉토리에 공유 DLL이나 드라이버, 애플리케이션, INI 파일들을 저장한다. 그리고 Win32는 각 프로세스마다 "현재 디렉토리"를 관리한다. Win32 애플리케이션을 작성하다 보면 현재 디렉토리나 윈도우즈, 시스템 디렉토리의 경로를 얻어야 할 경우가 있다. 다행히 Win32의 API에는 그러한 정보를 제공하는 것들이 있다.

GetWindowsDirectory(), GetSystemDirectory(), GetCurrentDirectory()가 바로 그것들이다. 이름만 봐도 어떤 함수인지 알 수 있을 것이다. 이 세 함수 모두 첫 번째 인자는 디렉토리 위치를 뜻하는 문자열이 들어갈 버퍼의 포인터이며, 두 번째 인자는 그 버퍼의 크기이다. 함수를 호출하면 포인터로 지정한 버퍼에 경로를 뜻하는 널 종료 문자열이 채워진다.

다음 프로시저는 임시 char 배열로 세 가지 디렉토리의 경로를 얻은 그것을 다음 InfoLB에 채운다.

```
procedure TInfoForm.GetDirInfo;
var
  S: array[0..MAX_PATH] of char;
begin
  { 윈도우즈 디렉토리의 경로를 얻는다. }
  GetWindowsDirectory(S, SizeOf(S));
  InfoLB.Items.Add(Format(SWinDir, [S]));
  { 시스템 디렉토리의 경로를 얻는다.  }
  GetSystemDirectory(S, SizeOf(S));
  InfoLB.Items.Add(Format(SSysDir, [S]));
  { 현재 프로세스의 현재 디렉토리 경로를 얻는다. }
```

```
  GetCurrentDirectory(SizeOf(S), S);
  InfoLB.Items.Add(Format(SCurDir, [S]));
end;
```



윈도우즈 3.x에서 사용하던 GetWindowsDir()나 GetSystemDir() 함수는 Win32에서 사용할 수 없다.

시스템 정보 얻기

Win32 API 함수인 GetSystemInfo()는 운영체제가 실행중인 컴퓨터에 대한 정보를 제공한다. 이 함수는 TSystemInfo 레코드를 참조 전달로 받아들여서 레코드의 필드들에 정보를 채운다. TSystemInfo 레코드의 정의는 다음과 같다.

```
type
  PSystemInfo = ^TSystemInfo;
  _SYSTEM_INFO = record
    case Integer of
      0: (
        dwOemId: DWORD);
      1: (
        wProcessorArchitecture: Word;
        wReserved: Word;
        dwPageSize: DWORD;
        lpMinimumApplicationAddress: Pointer;
        lpMaximumApplicationAddress: Pointer;
        dwActiveProcessorMask: DWORD;
        dwNumberOfProcessors: DWORD;
        dwProcessorType: DWORD;
        dwAllocationGranularity: DWORD;
        wProcessorLevel: Word;
        wProcessorRevision: Word);
  end;
  {$EXTERNALSYM _SYSTEM_INFO}
  TSystemInfo = _SYSTEM_INFO;
```

- dwOemId는 윈도우즈 95를 위한 것이다. 이 값은 항상 0, 즉 PROCESSOR_ARCHITECTURE_INTEL로 설정된다.
- 윈도우즈 NT의 경우, dwOemId 대신 wProcessorArchitector 필드가 사용된다(가변 필드이다). 이 필드는 현재 실행중인 컴퓨터의 프로세서 종류를 나타낸다. 델파이는 인텔 기반의 개발 도구이므로 사실 이 필드의 값은 별로 의미가 없지만, 그래도 궁금한 독자를 위해 이 필드가 가질 수 있는 값들을 이야기하겠다. 다음과 같다.

  PROCESSOR_ARCHITECTURE_INTEL

  PROCESSOR_ARCHITECTURE_MIPS

  PROCESSOR_ARCHITECTURE_ALPHA

  PROCESSOR_ARCHITECTURE_PPC
- wReserved 필드는 현재 사용하지 않는다.
- dwPageSize는 KB 단위의 페이지 크기이며, 페이지 보호와 허용의 정밀도를 뜻한다. 인텔 x86 컴퓨터의 경우, 이 값은 4KB이다.
- lpMinumuApplicationAddress는 애플리케이션과 DLL들이 접근할 수 있는 최하위 메모리 주소를 뜻한다. 이 메모리 주소 이하의 주소를 접근하려고 하면 접근 위반(access violation) 오류가 일어날 가능성이 크다.

  lpMaximumApplicationAddress는 애플리케이션과 DLL들이 접근할 수 있는 최상위 주소를 뜻한다. 마찬가지로, 이 메모리 주소 이상의 주소를 접근하려고 하면 접근 위반(access violation) 오류가 일어날 가능성이 크다.
- dwActiveProcessorMask는 시스템에 설치된 프로세서들을 뜻하는 비트 마스크이다. 비트 0이 첫 번째 프로세서이고 비트, 31이 32번째 프로세서이다. 윈도우즈 95/98은 단일 프로세서 기반의 운영체제이기 때문에 비트 0만이 의미가 있다.
- dwNumberOfProcessor는 시스템에 설치된 프로세서의 개수이다.
- dwProcessorType은 하위 호환성을 위해 존재하는 것으로, 현재로서는 별로 의미가 없다. 이 필드는 다음과 같은 값들을 가진다.

  PROCESSOR_INTEL_386

  PROCESSOR_INTEL_486

  PROCESSOR_INTEL_PENTIUM

  PROCESSOR_MIPS_R4000

  PROCESSOR_ALPHA_21046

  물론, 윈도우즈 95에서는 PROCESSOR_INTEL_x값들만 가능하며, 그 나머지 것들은 윈도우즈 NT에서만 의미가 있다.

- dwAllocationGranularity는 메모리 할당의 정밀도를 뜻한다. 이전에는 이 값이 64KB로 고정되어 있었지만, 현재는 64KB 이외의 값을 가질 수 있게 되었다(다른 구조의 컴퓨터를 고려한 것임).
- wProcessorLevel 필드는 시스템의 구조 의존적 프로세서 수준을 뜻한다. 이 값은 프로세서의 종류에 따라 달라진다. 인텔 프로세서의 경우, 다음과 같은 값을 가진다.

값	의 미
3	80386 프로세서
4	80486 프로세서
5	펜티엄 프로세서

- wProcessorRevision은 구조 의존적 프로세서 리비전을 뜻한다. wProcessorLevel과 마찬가지로, 이 필드 역시 프로세서에 따라 다른 값을 가진다. 인텔 기반 컴퓨터이면 이 필드는 xxyy 형식의 값을 가진다. 인텔 386과 인텔 486 칩의 경우, 스테핑 수준(stepping level)은 xx + $0A이며, 스테핑은 yy이다(예를 들어서, 0300은 D0 칩). 인텔 펜티엄이나 Cyrix/NextGen 486 칩의 경우에 xx는 모델 번호이고, yy는 스테핑이다(예를 들어, 0201은 모델 2, 스테핑 1).

시스템 정보를 얻어서 InfoLB에 추가하는 프로시저는 다음과 같다(예제를 간단하게 하기 위해서 인텔 기반의 컴퓨터에만 적용되도록 만들었다).

```
procedure TInfoForm.GetSysInfo;
var
  SI: TSystemInfo;
begin
  GetSystemInfo(SI);
  with InfoLB.Items, SI do
  begin
    Add(Format(SProc, [SPIntel]));
    Add(Format(SPageSize, [dwPageSize]));
    Add(Format(SMinAddr, [lpMinimumApplicationAddress]));
    Add(Format(SMaxAddr, [lpMaximumApplicationAddress]));
    Add(Format(SNumProcs, [dwNumberOfProcessors]));
    Add(Format(SAllocGra, [dwAllocationGranularity]));
    case wProcessorLevel of
```

```
      3: Add(Format(SProcLevl, [SIntel3]));
      4: Add(Format(SProcLevl, [SIntel4]));
      5: Add(Format(SProcLevl, [SIntel5]));
      6: Add(Format(SProcLevl, [SIntel6]));
    else Add(Format(SProcLevl, [IntToStr(wProcessorLevel)]));
    end;
  end;
end;
```



Win32 API GetSystemInfo() 함수는 윈도우즈 3.x API의 GetWinFlags()에 해당한다.

그림 14.3은 InfoForm이 운영체제의 버전과 디렉토리 정보 등을 표시한 모습이다.

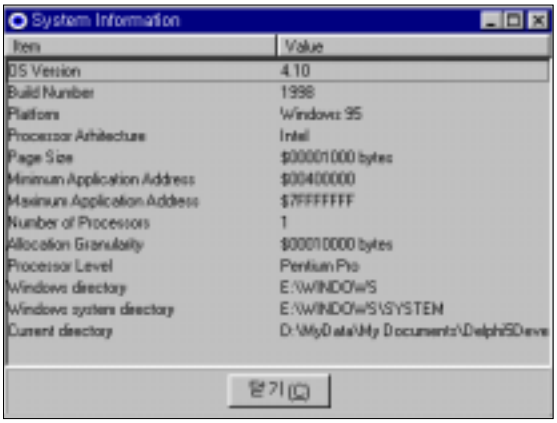


그림 14.3
시스템 정보를 표시한 모습

환경 변수 설정 얻기

프롬프트나 경로, 임시 디렉토리 위치 등 SET으로 설정하는 환경 변수들의 설정 내용을 얻으려면 GetEnvironmentString() API 함수를 사용하면 된다. 이 함수는 인자가 없으며, 널 문자로 분리된 환경 변수 설정 문자열 목록을 돌려준다. 리턴값의 형태는 문자열-널-문자열-널로 이어지는데, 널이 두 개가 나오면(#0#0) 끝이다. 다음은 SysInfo 프로그램 중 GetEnvironmentStrings() 함수를 이용해서 환경 변수 설정을 InfLB에 추가하는 프로시저이다.

```
procedure TInfoForm.ShowEnvironment;
var
  EnvPtr, SavePtr: PChar;
begin
  InfoLB.DelimChar := '=';
  EnvPtr := GetEnvironmentStrings;
  SavePtr := EnvPtr;
  InfoLB.Items.Clear;
  repeat
    InfoLB.Items.Add(StrPas(EnvPtr));
    inc(EnvPtr, StrLen(EnvPtr) + 1);
  until EnvPtr^ = #0;
  FreeEnvironmentStrings(SavePtr);
  InfoLB.Sections[0].Text := 'Environment Variable';
  InfoLB.Sections[1].Text := 'Value';
  Caption:= 'Current Environment';
end;
```



ShowEnvironment() 프로시저를 자세히 보면 PChar형의 문자열에 대한 포인터 연산을 수행해서 널로 분리된 문자열 목록의 각 문자열을 목록 상자에 추가하고 있다. 몇 줄 안되는 코드로 그러한 작업을 수행할 수 있는 것은 델파이의 포인터 연산 기능 덕분이다.

이 프로시저의 앞부분을 보면 InfoLB의 DelimChar 속성을 '='로 설정한다. 각 환경 변수 이름과 그 값은 모두 '='로 연결되어 있기 때문에 DelimChar를 '='로 하면 자연스럽게 환경 변수 이름과 값이 개별적인 컬럼으로 분리된다. 그리고 환경 변수 문자열 목록을 사용한 다음에는 FreeEnvironmentStrings() 함수를 이용해서 할당된 메모리 블록을 해제해 주어야 한다는 점도 잊지 말기 바란다.



GetEnvironmentStrings() 함수로 개별적인 환경 변수를 얻거나 설정할 수는 없다. 개별적인 환경 변수의 값을 얻거나 설정하려면 GetEnvironmentVariable() 이나 SetEnvironmentVariable() 함수를 사용해야 한다. 이 함수들에 대해서는 Win32 API 도움말을 참고하기 바란다.

그림 14.4는 InfoForm에 환경 변수 문자열들이 표시된 모습이다.

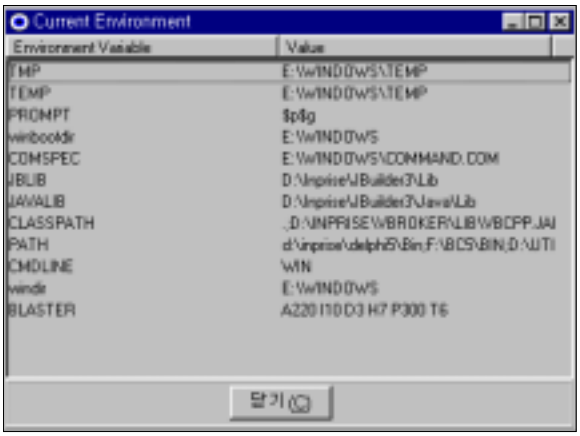


그림 14.4
환경 변수 문자열들

목록 14.1은 InfoForm이 정의되어 있는 InfoU.pas의 전체 코드이다.



Ch14\SysInfor\InfoU.pas

```
unit InfoU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  HeadList, StdCtrls, ExtCtrls, SysMain;

type
  TInfoVariety = (ivMemory, ivSystem, ivEnvironment);

  TInfoForm = class(TForm)
    InfoLB: THeaderListbox;
    Panel1: TPanel;
    OkBtn: TButton;
  private
    procedure GetOSVerInfo;
    procedure GetSysInfo;
    procedure GetDirInfo;
  public
    procedure ShowMemStatus;
    procedure ShowSysInfo;
    procedure ShowEnvironment;
  end;

procedure ShowInformation(Variety: TInfoVariety);

implementation

{$R *.DFM}

procedure ShowInformation(Variety: TInfoVariety);
begin
  with TInfoForm.Create(Application) do
```

```
    try
      Font := MainForm.Font;
      case Variety of
        ivMemory: ShowMemStatus;
        ivSystem: ShowSysInfo;
        ivEnvironment: ShowEnvironment;
      end;
      ShowModal;
    finally
      Free;
    end;
end;

const
  { 메모리 상태 관련 문자열 }
  SMemUse   = 'Memory in useq%d%%';
  STotMem   = 'Total physical memoryq$%.8x bytes';
  SFreeMem  = 'Free physical memoryq$%.8x bytes';
  STotPage  = 'Total page file memoryq$%.8x bytes';
  SFreePage = 'Free page file memoryq$%.8x bytes';
  STotVirt  = 'Total virtual memoryq$%.8x bytes';
  SFreeVirt = 'Free virtual memoryq$%.8x bytes';

  { 운영체제 버전 정보 관련 문자열 }
  SOSVer    = 'OS Versionq%d.%d';
  SBuildNo  = 'Build Numberq%d';
  SOSPlat   = 'Platformq%s';
  SOSWin32s = 'Windows 3.1x running Win32s';
  SOSWin95  = 'Windows 95';
  SOSWinNT  = 'Windows NT';

  { 시스템 관련 문자열 }
  SProc     = 'Processor Arhitectureq%s';
  SPIntel   = 'Intel';
  SPageSize = 'Page Sizeq$%.8x bytes';
  SMinAddr  = 'Minimum Application Addressq$%p';
  SMaxAddr  = 'Maximum Application Addressq$%p';
  SNumProcs = 'Number of Processorsq%d';
  SAllocGra = 'Allocation Granularityq$%.8x bytes';
```

```
  SProcLevl = 'Processor Levelq%s';
  SIntel3   = '80386';
  SIntel4   = '80486';
  SIntel5   = 'Pentium';
  SIntel6   = 'Pentium Pro';
  SProcRev  = 'Processor Revisionq%.4x';

  { 디렉토리 관련 문자열 }
  SWinDir   = 'Windows directoryq%s';
  SSysDir   = 'Windows system directoryq%s';
  SCurDir   = 'Current directoryq%s';

procedure TInfoForm.ShowMemStatus;
var
  MS: TMemoryStatus;
begin
  InfoLB.DelimChar := 'q';
  MS.dwLength := SizeOf(MS);
  GlobalMemoryStatus(MS);
  with InfoLB.Items, MS do
  begin
    Clear;
    Add(Format(SMemUse, [dwMemoryLoad]));
    Add(Format(STotMem, [dwTotalPhys]));
    Add(Format(SFreeMem, [dwAvailPhys]));
    Add(Format(STotPage, [dwTotalPageFile]));
    Add(Format(SFreePage, [dwAvailPageFile]));
    Add(Format(STotVirt, [dwTotalVirtual]));
    Add(Format(SFreeVirt, [dwAvailVirtual]));
  end;
  InfoLB.Sections[0].Text := 'Resource';
  InfoLB.Sections[1].Text := 'Amount';
  Caption:= 'Memory Status';
end;

procedure TInfoForm.GetOSVerInfo;
var
  VI: TOSVersionInfo;
begin
```

```
  VI.dwOSVersionInfoSize := SizeOf(VI);
  GetVersionEx(VI);
  with InfoLB.Items, VI do
  begin
    Clear;
    Add(Format(SOSVer, [dwMajorVersion, dwMinorVersion]));
    Add(Format(SBuildNo, [LoWord(dwBuildNumber)]));
    case dwPlatformID of
      VER_PLATFORM_WIN32S: Add(Format(SOSPlat, [SOSWin32s]));
      VER_PLATFORM_WIN32_WINDOWS: Add(Format(SOSPlat, [SOSWin95]));
      VER_PLATFORM_WIN32_NT: Add(Format(SOSPlat, [SOSWinNT]));
    end;
  end;
end;

procedure TInfoForm.GetSysInfo;
var
  SI: TSystemInfo;
begin
  GetSystemInfo(SI);
  with InfoLB.Items, SI do
  begin
    Add(Format(SProc, [SPIntel]));
    Add(Format(SPageSize, [dwPageSize]));
    Add(Format(SMinAddr, [lpMinimumApplicationAddress]));
    Add(Format(SMaxAddr, [lpMaximumApplicationAddress]));
    Add(Format(SNumProcs, [dwNumberOfProcessors]));
    Add(Format(SAllocGra, [dwAllocationGranularity]));
    case wProcessorLevel of
      3: Add(Format(SProcLevl, [SIntel3]));
      4: Add(Format(SProcLevl, [SIntel4]));
      5: Add(Format(SProcLevl, [SIntel5]));
      6: Add(Format(SProcLevl, [SIntel6]));
    else Add(Format(SProcLevl, [IntToStr(wProcessorLevel)]));
    end;
  end;
end;

procedure TInfoForm.GetDirInfo;
```

```
var
  S: array[0..MAX_PATH] of char;
begin
  { 윈도우즈 디렉토리의 경로를 얻는다. }
  GetWindowsDirectory(S, SizeOf(S));
  InfoLB.Items.Add(Format(SWinDir, [S]));
  { 시스템 디렉토리의 경로를 얻는다.  }
  GetSystemDirectory(S, SizeOf(S));
  InfoLB.Items.Add(Format(SSysDir, [S]));
  { 현재 프로세스의 현재 디렉토리 경로를 얻는다. }
  GetCurrentDirectory(SizeOf(S), S);
  InfoLB.Items.Add(Format(SCurDir, [S]));
end;

procedure TInfoForm.ShowSysInfo;
begin
  InfoLB.DelimChar := 'q';
  GetOSVerInfo;
  GetSysInfo;
  GetDirInfo;
  InfoLB.Sections[0].Text := 'Item';
  InfoLB.Sections[1].Text := 'Value';
  Caption:= 'System Information';
end;

procedure TInfoForm.ShowEnvironment;
var
  EnvPtr, SavePtr: PChar;
begin
  InfoLB.DelimChar := '=';
  EnvPtr := GetEnvironmentStrings;
  SavePtr := EnvPtr;
  InfoLB.Items.Clear;
  repeat
    InfoLB.Items.Add(StrPas(EnvPtr));
    inc(EnvPtr, StrLen(EnvPtr) + 1);
  until EnvPtr^ = #0;
  FreeEnvironmentStrings(SavePtr);
  InfoLB.Sections[0].Text := 'Environment Variable';
```

```
  InfoLB.Sections[1].Text := 'Value';
  Caption:= 'Current Environment';
end;

end.
```

플랫폼 중립적 설계

같은 Win32라도 윈도우즈 NT와 윈도우즈 95/98의 실제적인 구현은 약간의, 경우에 따라서는 상당한 차이가 있다. 그렇지만 SysInfo는 윈도우즈 95/98과 윈도우즈 NT에서 모두 작동하도록 만들어져 있다. 이처럼 한 애플리케이션이 플랫폼에 무관하게 작동하는 것을 플랫폼 중립성이라고 한다. SysInfo는 프로세스 관련 정보를 얻을 때 인터페이스를 사용해서 플랫폼 중립성을 구현한다. 인터페이스에는 프로세스 정보를 얻고 표시하는 데 관련된 메소드들이 정의되어 있으며, 그 메소드들은 운영체제에 따라 다르게 구현된다. SysInfo가 정의한 인터페이스는 IWin32Info인데, 이 인터페이스의 정의는 다음과 같다.

```
type
  IWin32Info = interface
    procedure FillProcessInfoList(ListView: TListView; ImageList:
TImageList);
    procedure ShowProcessProperties(Cookie: Pointer);
  end;
```

- FillProcessInfoList()는 실행중인 프로세스들과 관련 아이콘들을 TListView와 TImageList에 추가하는 작업을 한다.
- ShowProcessProperties()는 TListView에서 선택된 프로세스에 대한 좀더 자세한 정보를 얻을 때 사용한다.

윈도우즈 95에 대해 이 인터페이스를 구체적으로 구현하는 것은 TWin95Info 클래스(W95Info.pas)인데, 이 클래스는 윈도우즈 95의 ToolHelp32 API를 사용한다. 그리고 윈도우즈 NT에 대해 이 인터페이스를 구체적으로 구현하는 것은 TWinNTInfo 클래

스(WNTInfo.pas)이며, 이 클래스는 윈도우즈 NT의 PSAPI를 사용한다. 다음 코드는 SysMain(프로젝트의 메인 유닛)의 소스 코드 중 운영체제에 따라서 적절한 클래스를 생성하는 부분이다.

```
if Win32Platform = VER_PLATFORM_WIN32_WINDOWS then
  FWinInfo := TWin95Info.Create
else if Win32Platform = VER_PLATFORM_WIN32_NT then
  FWinInfo := TWinNTInfo.Create
```

윈도우즈 95/98의 ToolHelp32 사용법

TooHelp32는 운영체제의 몇몇 저수준 작동 상태에 대한 정보를 얻는 데 필요한 Win32 API 함수/프로시저들을 모아둔 것이다. 특히, 시스템에서 현재 실행중인 모든 프로세스들과 각 프로세스의 스레드, 모듈, 힙 메모리에 대한 정보를 얻는 데 유용하다. 물론, 그러한 정보들은 디버거 같은 프로그램을 만드는 것이 아니라면 굳이 필요하지는 않은 것들이지만, Win32 운영체제의 내부를 들여다 보는 일은 Win32 프로그래머로서 한 번쯤 경험해 볼 만한 것이라 생각한다.



ToolHelp32 API는 윈도우즈 95/98에서만 사용할 수 있다. 이런 종류의 기능은 윈도우즈 NT의 견고한 프로세스 보호와 보안 기능에는 맞지 않기 때문이다. 따라서, ToolHelp32의 함수들을 사용하는 애플리케이션은 윈도우즈 NT에서는 제대로 작동하지 않는다.

ToolHelp32는 윈도우즈 3.x에 포함된 ToolHelp와는 다르다. ToolHelp의 대부분의 함수들은 Win32에 적용할 수 없으며, 따라서 Win32는 그런 함수들을 지원하지 않는다. 그리고 윈도우즈 3.1.x에서 ToolHelp 함수들은 TOOLHELP.DLL이라고 하는 개별적인 DLL에 들어 있었지만, Win32의 ToolHelp32 함수들은 커널에 포함되어 있다.

ToolHelp32의 데이터형과 함수 정의는 TlHelp32 유닛에 있으므로 ToolHelp32 API를 사용할 때에는 uses절에 TlHelp32를 포함시켜야 한다. SysInfo 프로젝트는 ToolHelp32, 즉 TlHlep32에 정의된 함수들과 데이터형을 사용해서 윈도우즈 95 환경에서의 프로세스 정보를 얻는다.

그림 14.5는 SysInfo의 메인 폼이다. 메인 폼의 THeaderListBox는 11장, "멀티스레드 애플리케이션의 작성"에서 만든 오너 드로우 목록 상자이다. 목록에는 프로세스들이 나타나며, 목록의 프로세스를 하나 더블클릭하면 그 프로세스에 대한 자세한 정보가 나타난다. 자세한 정보는 메인 폼과 비슷한 형태의 자식 폼으로 표시된다.

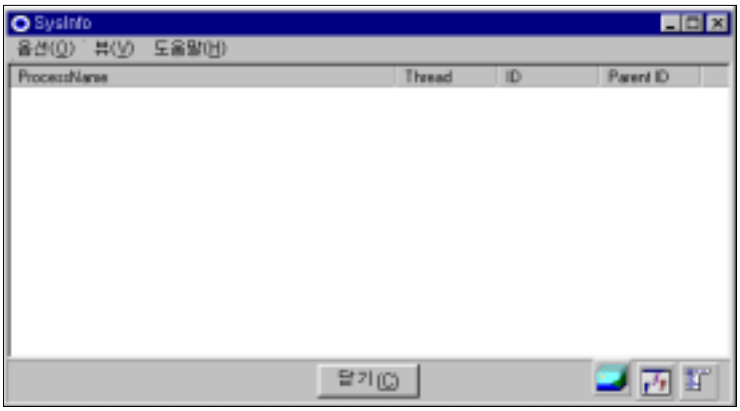


그림 14.5
SysInfo의 메인 폼인 TMainForm

스냅샷

Win32 환경은 멀티태스킹 환경이므로 프로세스나 스레드, 모듈, 힙 같은 객체들은 끊임 없이 생성, 변경, 파괴된다. 시스템의 상태는 항상 유동적이므로 현재의 시스템 정보는 1초 후에는 아무런 의미가 없을 수도 있다. 예를 들어서, 시스템에 적재된 모든 모듈들을 나열하는 함수가 실행되는 중에도 모듈이 제거되거나, 새로운 모듈이 적재될 수 있는 것이다.

이러한 동적인 환경을 고려할 때 일정 기간 동안 시스템의 활동을 정지시킨 다음 시스템의 정보를 얻는 방법이 있으면 좋을 것이다. ToolHelp32는 시스템을 정지시키는 기능을 제공하지는 않지만, ToolHelp32의 함수를 통해서 시스템의 한 순간을 포착한

스냅샷(snapshot)을 얻을 수는 있다. CrateToolhelp32Snapshot()이라는 함수가 바로 그 함수인데, 선언은 다음과 같다.

```
function CreateToolhelp32Snapshot(dwFlags, th32ProcessID: DWORD): THandle;
stdcall;
```

- dwFlags 인자는 스냅샷에 포함할 정보의 종류를 결정하는 것이다. 이 인자에 넣을 수 있는 값들은 다음과 같다.

값	의 미
TH32CS_INHERIT	스냅샷 핸들을 상속할 수 있음.
TH32CS_SNAPALL	TH32CS_SNAPHEAPLIST와 TH32CS_SNAPPROCESS, TH32CS_SNAPTHREAD, TH32CS_SNAPMODULE을 모두 적용한 것과 같음.
TH32CS_SNAPHEAPLIST	지정된 Win32 프로세스의 힙 목록을 스냅샷에 포함시킨다.
TH32CS_SNAPMODULE	지정된 Win32 프로세스의 모듈 목록을 스냅샷에 포함시킨다.
TH32CS_SNAPPROCESS	Win32 프로세스 목록을 스냅샷에 포함시킨다.
TH32CS_SNAPTHREAD	Win32 스레드 목록을 스냅샷에 포함시킨다.

- th32ProcessID 인자는 정보를 얻고자 하는 프로세스를 뜻한다. 현재 프로세스에 대한 정보를 얻을 때에는 이 인자에 0을 넣는다. 프로세스에 속한 정보는 모듈과 힙 목록이므로 이 인자는 모듈과 힙 목록을 얻을 때에만 의미가 있다. 프로세스나 스레드 목록은 시스템 전반에 관계된 것이므로 이 인자는 의미가 없다.
- CreateToolhelp32Snapshot() 함수는 생성된 스냅샷의 핸들의 돌려준다. 에러가 생기면 -1을 돌려준다. 스냅샷의 핸들은 다른 Win32 핸들과 동일한 데이터형이므로 사용 방법도 크게 다르지 않다.

다음은 현재 시스템에 로드된 모든 프로세스들에 대한 정보를 담고 있는 스냅샷 핸들을 생성하는 코드이다(EToolHelpError는 기본 예외 클래스가 아니라, 필자가 정의한 예외 클래스이다).

```
var
  Snap: THanlde;
begin
  Snap := CrateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
  if Snap = -1 then
      raise EToolHelpError.Create('CreateToolHelp32Snapshot failed');
end;
```



스냅샷 핸들도 표준 Win32 핸들이므로 사용한 후에는 Win32 API ClassHandle() 함수로 닫아줘야 한다.

프로세스 목록 다루기

만들어진 스냅샷 핸들에 들어 있는 프로세스 정보를 나열할 때 유용한 함수가 ToolHelp32에 들어 있다. 바로 Process32First()와 Process32Next()인데, 이 두 함수의 선언은 다음과 같다.

```
function Process32First(hSnapshot: THandle;
      var lppe: TProcessEntry32): BOOL; stdcall;
function Process32Next(hSnapshot: THandle;
      var lppe: TProcessEntry32): BOOL; stdcall;
```

두 함수의 첫 번째 인자인 hSnapshot에는 CrateToolhelp32Snapshot()으로 얻은 스냅샷 핸들을 넣는다.

두 번째 인자 lppe는 참조 전달로 넘겨주는 TProcessEntry32 레코드이다. 프로세스들을 나열할 때마다 이 레코드에는 목록의 다음 프로세스에 대한 정보들이 채워진다. TProcessEntry32 레코드의 정의는 다음과 같다(역주 : 아래의 주석은 원래 볼랜드 사가 붙인 영문 주석을 번역한 것임).

```
type
  tagPROCESSENTRY32 = packed record
    dwSize: DWORD;
    cntUsage: DWORD;
    th32DefaultHeapID: DWORD;        // 이 프로세스
    th32ModuleID: DWORD;       // 관련된 실행 파일
    cntThreads: DWORD;
    th32ParentProcessID: DWORD; // 이 프로세스의 부모 프로세스
```

```
    pcPriClassBase: Longint;// 프로세스의 스레드들의 기본 우선순위
    dwFlags: DWORD;
    szExeFile: array[0..MAX_PATH - 1] of Char;// 경로
  end;
```

- dwSize 필드에는 TProcessEntry32 레코드의 크기를 넣는다. 크기는 SizeOf(TProcessEntry32)로 얻어야 한다.
- cntUsage 필드에는 프로세스에 대한 참조 횟수가 들어간다. 참조 횟수가 0이 되면 운영체제는 프로세스를 제거한다.
- th32ProcessID 필드에는 프로세스의 ID가 들어간다.
- th32DefaultHeapID에는 프로세스의 기본 힙의 ID가 들어간다. 이 ID는 ToolHelp32에만 의미가 있는 것이며, 다른 Win32 함수에 이 필드의 값을 사용할 수는 없다.
- thModuleID 필드에는 프로세스에 관련된 모듈의 ID가 들어간다. 이 필드 역시 ToolHelp32 함수들에만 의미가 있다.
- cntThreads 필드는 프로세스가 띄운 스레드의 개수가 들어간다.
- th32ParentProcessID에는 이 프로세스의 부모 프로세스의 ID가 들어간다.
- pcPriClassBase 필드에는 프로세스의 기본 우선순위가 들어간다. 운영체제는 기본 우선순위를 기반으로 해서 스레드들의 실행을 전환한다.
- dwFlags 필드는 예약만 되어 있는 것으로, 현재는 사용하지 않는다.
- szExeFile 필드에는 프로세스에 관련된 EXE나 드라이버의 경로 이름을 담은 널 종료 문자열이 들어간다.

Process32First()와 Process32Next()의 함수 이름을 보면 짐작할 수 있겠지만, 이 두 함수를 이용해서 마치 테이블의 레코드들을 열거하듯이 프로세스들의 목록에서 개별 프로세스 정보를 얻으면 된다.

프로세스들의 목록을 열거하는 코드는 TWin95Info 클래스에 캡슐화되어 있다. 앞에서 말했지만, TWin95Info 클래스는 IWin32Info 인터페이스를 구현하는 클래스이다. 다음 코드는 TWin95Info 클래스의 private 메소드인 Refresh()인데, 이 메소드는 시스템 프로세스들을 열거해서 각각의 프로세스 정보를 목록에 추가한다.

```
procedure TWin95Info.Refresh;
var
  PE: TProcessEntry32;
  PPE: PProcessEntry32;
begin
  FProcList.Clear;
  if FSnap > 0 then CloseHandle(FSnap);
  FSnap := CreateToolHelp32Snapshot(TH32CS_SNAPPROCESS, 0);
  if FSnap = INVALID_HANDLE_VALUE then
    raise Exception.Create('CreateToolHelp32Snapshot failed');
  PE.dwSize := SizeOf(PE);
  if Process32First(FSnap, PE) then   // 첫 번째 프로세스 정보를 얻는다.
    repeat
      New(PPE);                        // 새 PPE를 생성하고
      PPE^ := PE;                      // 채운 다음,
      FProcList.Add(PPE);              // 목록에 추가한다.
    until not Process32Next(FSnap, PE);  // 다음 프로세스 정보를 얻는다.
end;
```

Refresh() 메소드를 호출하는 것은 FillProcessInfoList()이다. 앞에서도 말했듯이 이 메소드는 실행중인 모든 프로세스들에 대한 정보를 TListView와 TImageList에 추가한다. 다음은 FillProcessInfoList()의 코드이다.

```
procedure TWin95Info.FillProcessInfoList(ListView: TListView;
  ImageList: TImageList);
var
  I: Integer;
  ExeFile: string;
  PE: TProcessEntry32;
  HAppIcon: HIcon;
begin
  Refresh;
  ListView.Columns.Clear;
  ListView.Items.Clear;
  for I := Low(ProcessInfoCaptions) to High(ProcessInfoCaptions) do
    with ListView.Columns.Add do
```

```
    begin
      if I = 0 then Width := 285
      else Width := 75;
      Caption := ProcessInfoCaptions[I];
    end;
  for I := 0 to FProcList.Count - 1 do
  begin
    PE := PProcessEntry32(FProcList.Items[I])^;
    HAppIcon := ExtractIcon(HInstance, PE.szExeFile, 0);
    try
      if HAppIcon = 0 then HAppIcon := FWinIcon;
      ExeFile := PE.szExeFile;
      if ListView.ViewStyle = vsList then
        ExeFile := ExtractFileName(ExeFile);
      // 새 항목을 추가하고, 제목을 설정한 후 부 항목들을 추가한다.
      with ListView.Items.Add, SubItems do
      begin
        Caption := ExeFile;
        Data := FProcList.Items[I];
        Add(IntToStr(PE.cntThreads));
        Add(IntToHex(PE.th32ProcessID, 8));
        Add(IntToHex(PE.th32ParentProcessID, 8));
        if ImageList <> nil then
          ImageIndex := ImageList_AddIcon(ImageList.Handle, HAppIcon);
      end;
    finally
      if HAppIcon <> FWinIcon then DestroyIcon(HAppIcon);
    end;
  end;
end;
```

그림 14.6은 이 메소드가 실행된 결과를 나타낸 것이다. 윈도우즈 95 시스템에 떠 있는 모든 프로세스들의 정보가 나열되어 있다.

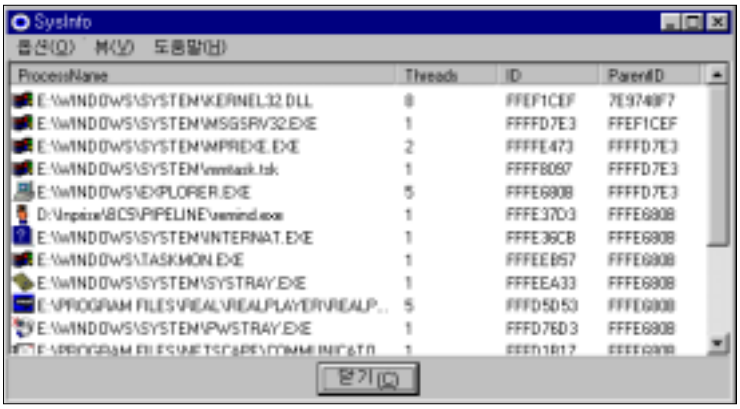


그림 14.6
윈도우즈 95의 프로세스 정보

그런데 목록을 보면 프로세스의 아이콘들이 표시되어 있다. 아이콘이 표시되니까 뭔가 윈도우즈 95 표준적인 프로그램처럼 보인다. 앞의 설명을 보면 알겠지만, 프로세스 정보 중에는 실행 파일의 경로가 포함되어 있다. 실행 파일의 경로를 얻을 수 있으므로 ShellAPI 유닛에 정의되어 있는 Win32 API 함수 ExtractIcon()을 사용해서 아이콘을 뽑아내면 된다. ExtractIcon()의 수행이 실패하면 FWinIcon을 사용해서 아이콘을 표시한다. FWinIcon은 표준 윈도우즈 아이콘으로, 이 클래스의 생성자인 Create()에서 LoadImage() API 함수를 통해 로드한 것이다.

```
FWinIcon := LoadImage(0, IDI_WINLOGO, IMAGE_ICON, LR_DEFAULTSIZE,
  LR_DEFAULTSIZE, LR_DEFAULTSIZE or LR_DEFAULTCOLOR or LR_SHARED);
```

사용자가 메인 폼의 목록에서 어떤 프로세스를 더블클릭하면 IWin32Info 인터페이스의 ShowProcessProperties() 메소드가 실행된다. 윈도우즈 95의 경우, ShowProcessProperties() 메소드는 Detail95 유닛에 있는 ShowProcessDetails() 메소드에게 자신이 받은 인자를 넘겨준다. ShowProcessProperties()의 정의는 다음과 같다.

```
procedure TWin95Info.ShowProcessProperties(Cookie: Pointer);
begin
  ShowProcessDetails(PProcessEntry32(Cookie));
end;
```

ShowProcessDetails()의 인자는 선택된 프로세스의 정보를 담고 있는 TProcessEntry32 레코드에 대한 포인터이다.

이 포인터를 넘겨 받은 ShowProcessDetails()는 CreateToolHelp32Snapshot()을 호출해서 현재 프로세스에 대한 스냅샷 핸들을 얻는다. 선택된 프로세스에 대한 스냅샷 핸들을 얻을 때에는 선택된 프로세스의 TProcessEntry32 레코드의 th32ProcessID 필드(프로세스 ID)를 CreateToolHelp32Snapshot()의 두 번째 인자에 넣고, 첫 번째 인자 dwFlags에는 TH32CS_SNAPALL을 넣어서 호출한다. 다음과 같다.

```
{ 현재 프로세스에 대한 스냅샷 핸들을 얻는다. }
FCurSnap := CreateToolhelp32Snapshot(TH32CS_SNAPALL, P^.th32ProcessID);
  if FCurSnap = INVALID_HANDLE_VALUE then
  raise Exception.Create('CreateToolHelp32Snapshot failed');
```

TDetailForm 객체는 한 번에 하나의 목록만을 표시한다. 다음은 어떤 종류의 목록을 표시하는지를 나타내는 열거형이다.

```
type
  TListType = (ltThread, ltModuel, ltHeap);
```

TDetailForm은 또한 스레드, 모듈, 힙의 정보를 개별적인 TStringList들로 관리한다. DeatilLists라는 배열이 바로 그 TStringList들이다.

```
DetailLists: array[TListType] of TStringList;
```

스레드 목록 다루기

프로세스의 스레드 목록을 나열할 때에는 ToolHelp32의 Thread32First()와 Thread32Next()를 사용한다. 이 두 함수의 선언은 다음과 같다.

```
function Thread32First(hSnapshot: THandle;
      var lpte: TThreadEntry32): BOOL; stdcall;
function Thread32Next(hSnapshot: THandle;
      var lpte: TThreadENtry32): BOOL; stdcall;
```

hSnapshot은 스레드 목록이 포함되어 있는 스냅샷 핸들이다. 그리고 lpte는 TThreadEntry32 레코드에 대한 참조인데, 프로세스를 다루는 함수와 마찬가지로, 이 레코드에 스레드에 대한 정보가 채워진다. TThreadEntry32 레코드의 정의는 다음과 같다.

```
type
  TThreadEntry32 = record
    dwSize: DWORD;
    cntUsage: DWORD;
    th32ThreadID: DWORD;
    th32OwnerProcessID: DWORD;
    tpBasePri: Longint;
    tpDeltaPri: Longint;
    dwFlags: DWORD;
  end;
```

- dwSize 필드에는 TThreadEntry32 레코드의 크기를 넣는다. 크기는 SizeOf(TThread Entry32)로 얻어야 한다.
- cntUsage 필드에는 스레드에 대한 참조 횟수가 들어간다. 참조 횟수가 0이 되면 운영체제는 스레드를 제거한다.
- th32ThreadID 필드에는 ID가 들어간다. 이 필드의 값은 ToolHelp32 함수들에만 의미가 있다.
- th32OwnerProcessID는 이 스레드를 소유한 프로세스의 ID이다. 이 값은 다른 Win32 함수들도 사용할 수 있다.
- tpBasePri는 스레드의 기본 우선순위 등급이다. 주어진 프로세스의 모든 스레드는 동일한 기본 우선순위 등급을 가진다. 이 필드에 들어가는 값은 일반적으로 4에서 24 사이의 값이다. 다음은 각 값들의 의미이다.

값	의 미
4 8 13 24	Idle(공전) Normal(보통) High(높음) Real Time(최고)

- tpDeltaPri는 tpBasePri에 대한 우선순위 변화치이다. 이 필드의 값은 부호를 가진다. 이 필드의 값과 tpBasePri의 값을 더하면 스레드의 최종 우선순위가 된다. 다음은 이 필드에 들어갈 수 있는 상수들과 그 상수들의 값이다.

상 수	값
THREAD_PRIORITY_IDLE	-15*
THREAD_PRIORITY_LOWEST	-2
THREAD_PRIORITY_BELOW_NORMAL	-1
THREAD_PRIORITY_NORMAL	0
THREAD_PRIORITY_ABOVE_NORMAL	1
THREAD_PRIORITY_HIGHEST	2
THREAD_PRIORITY_TIME_CRITICAL	15*

- dwFlags는 예약되어 있으며, 현재는 사용하지 않는다.

TDetailForm의 WalkThreads() 메소드는 스레드들을 나열해서 스레드 관련 정보를 DetailLists 배열에 추가한다. 이 메소드의 코드는 다음과 같다.

```
procedure TWin95DetailForm.WalkThreads;
{ ToolHelp32 함수들을 이용해서 스레드들을 나열한다. }
var
  T: TThreadEntry32;
begin
  DetailLists[ltThread].Clear;
  T.dwSize := SizeOf(T);
  if Thread32First(FCurSnap, T) then
    repeat
      { 현재 프로세스의 스레드인지 확인한다. }
      if T.th32OwnerProcessID = FCurProc.th32ProcessID then

        DetailLists[ltThread].Add(Format(SThreadStr, [T.th32ThreadID,
          GetClassPriorityString(T.tpBasePri),
          GetThreadPriorityString(T.tpDeltaPri), T.cntUsage]));
    until not Thread32Next(FCurSnap, T);
end;
```



ToolHelp32의 스레드 목록은 프로세스별로 구성되는 것이 아니기 때문에 WalkThreads() 메소드 중 다음 코드가 매우 중요하다.

```
if T.th32OwnerProcessID = FCurProc.th32ProcessID then
```

WalkThreads()는 현재 프로세스의 스레드들만 처리하기 위해서 위와 같이 스레드를 소유한 프로세스와 현재 프로세스를 비교한다.

그림 14.7은 스레드 목록이 표시된 DetailForm 폼이다.

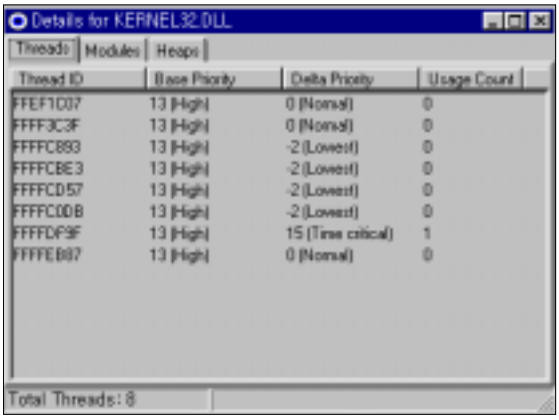


그림 14.7
윈도우즈 95 환경에서 스레드 정보를 표시한 모습

모듈 목록 다루기

모듈들을 나열하는 것도 프로세스나 스레드를 나열하는 것과 거의 비슷하다. ToolHelp32에 있는 Module32First()와 Module32Next()라는 함수를 사용하는데, 두 함수의 선언은 다음과 같다.

```
function Module32First(hSnapshot: THandle;
      var lpme: TModuleEntry32): BOOL; stdcall;
function Module32Next(hSnapshot: THandle;
      var lpme: TModuleEntry32): BOOL; stdcall;
```

이 두 함수 역시 첫 번째 인자는 스냅샷의 핸들이다. 두 번째 var 인자 lpme는 TModuleEntry32 레코드인데, 이 레코드의 정의는 다음과 같다.

```
type
  TModuleEntry32 = record
    dwSize: DWORD;
    th32ModuleID: DWORD;
    th32ProcessID: DWORD;
    GlblcntUsage: DWORD;
    ProccntUsage: DWORD;
    modBaseAddr: PBYTE;
    modBaseSize: DWORD;
    hModule: HMODULE;
    szModule: array[0..MAX_MODULE_NAME32] of Char;
    szExePath: array[0..MAX_PATH - 1] of Char;
  end;
```

- dwSize 필드에는 TModuleEntry32 레코드의 크기를 넣는다. 크기는 SizeOf(TModuleEntry32)로 얻어야 한다.
- th32ModuleID 필드에는 모듈의 ID가 들어간다. 이 ID는 ToolHelp32에만 의미가 있는 값이다.
- th32ProcessID 필드에는 현재 프로세스의 ID가 들어간다. 이 값은 다른 Win32 함수들도 사용할 수 있다.
- GlblcntUsage는 모듈에 대한 전역 참조 횟수가 들어간다.

- ProccntUsage는 모듈을 소유한 프로세스의 문맥 안에서의 모듈에 대한 참조 횟수가 들어간다.
- modBaseAddr은 메모리 안에서의 모듈의 기준 주소이다. 이 값은 th32ProcessID에 해당하는 프로세스의 문맥하에서만 유효한 값이다.
- modBaseSize는 모듈이 차지하고 있는 메모리의 크기(바이트)이다.
- hModule은 모듈의 핸들이다. 이 값은 th32ProcessID에 해당하는 프로세스의 문맥하에서만 유효한 값이다.
- szModule은 모듈 이름이 담긴 널 종료 문자열이다.
- szExepath는 모듈의 전체 경로 이름이 담긴 널 종료 문자열이다.

TDetailForm의 WalkModules() 메소드는 WalkThread()와 매우 비슷한 작업을 한다. 다음 코드를 보면 알겠지만, 이 메소드는 모듈 목록에서 각 모듈의 정보를 얻어서 DetailLists 배열의 모듈 목록에 추가한다.

```
procedure TWin95DetailForm.WalkModules;
{ ToolHelp32 함수들을 이용해서 모듈들을 나열한다. }
var
  M: TModuleEntry32;
begin
  DetailLists[ltModule].Clear;
  M.dwSize := SizeOf(M);
  if Module32First(FCurSnap, M) then

    repeat
      DetailLists[ltModule].Add(Format(SModuleStr, [M.szModule,
                M.ModBaseAddr, M.ModBaseSize, M.ProcCntUsage]));
    until not Module32Next(FCurSnap, M);
end;
```

그림 14.8은 모듈 목록이 표시된 DetailForm 폼이다.

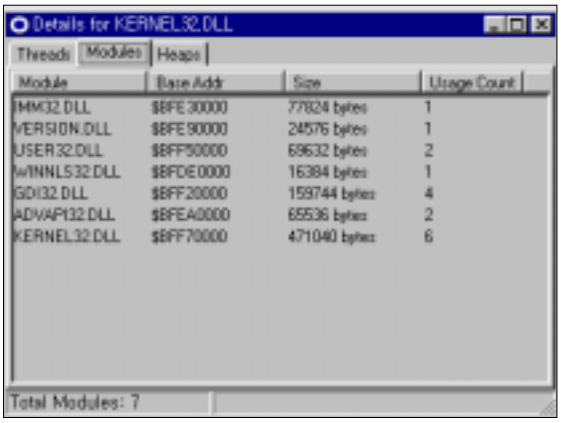


그림 14.8
윈도우즈 95 환경에서 모듈 정보를 표시한 모습

힙 목록 다루기

힙 목록을 나열하는 것은 앞에서 나온 프로세스나 스레드, 모듈을 나열하는 것보다는 조금 복잡하다. 힙 목록을 다룰 때 사용하는 것은 ToolHelp32의 Heap32ListFirst()와 Heap32ListNext()이다.

그리고 힙 목록의 개별 힙에 대한 정보를 다룰 때에는 Heap32First()와 Heap32Next()를 사용한다.

Heap32ListFirst()와 Heap32ListNext()는 다음과 같이 선언되어 있다.

```
function Heap32ListFirst(hSnapshot: THandle;
       var lphl: THeapList32): BOOL; stdcall;
function Heap32ListNext(hSnapshot: THandle;
       var lphl: THeapList32): BOOL; stdcall;
```

두 함수의 첫 번째 인자는 이전과 마찬가지로 스냅샷 핸들이다. 두 번째 인자 lphl은 참조로 전달되는 THeapList32 레코드이다. 이 레코드의 정의는 다음과 같다.

```
type
  THeapList32 = record
    dwSize: DWORD;
    th32ProcessID: DWORD;
    th32HeapID: DWORD;
    dwFlags: DWORD;
  end;
```

- dwSize 필드에는 THeapLists32 레코드의 크기를 넣는다. 크기는 SizeOf(THeapLists32)로 얻어야 한다.
- th32ProcessID는 힙을 소유한 프로세스의 ID이다.
- th32HeapID는 힙의 ID이다. 이 값은 ToolHelp32에만 의미가 있다.
- dwFlags에는 힙 종류를 뜻하는 플래그가 들어간다. 이 필드의 값은 현재 힙이 프로세스의 기본 힙임을 뜻하는 HF32_DEFAULT이거나, 현재 힙이 일반 공유 힙임을 뜻하는 HF32_SHARED이다.

Heap32First()와 Heap32Next() 함수의 선언은 다음과 같다.

```
function Heap32First(var lphe: THeapEntry32; th32ProcessID,
  th32HeapID: DWORD): BOOL; stdcall;
function Heap32Next(var lphe: THeapEntry32): BOOL; stdcall;
```

이 함수들의 인자 구성은 프로세스나 스레드, 모듈, 그리고 힙 목록을 다루는 함수들과는 다르다. 이 함수들은 주어진 프로세스에 속한 주어진 힙의 블록들을 나열할 때 사용한다. Heap32First()를 호출할 때 th32ProcessID와 th32HeapID 인자에는 Heap32ListFirst()나 Heap32ListNext()를 호출해서 채운 THeapList32 레코드의 같은 이름의 필드들의 값을 넣어야 한다. Heap32First()와 Heap32Next() 함수의 var 인자 lphe는 THeapEntry32 레코드이다. 이 레코드에는 힙 블록에 관한 상세한 정보가 채워지는데, 정의는 다음과 같다.

```
type
  THeapEntry32 = record
    dwSize: DWORD;
    hHandle: THandle;     // 이 힙 블록에 대한 핸들
    dwAddress: DWORD;     // 블록 시작점의 선형 주소
    dwBlockSize: DWORD;   // 블록의 크기(바이트)
    dwFlags: DWORD;
    dwLockCount: DWORD;
    dwResvd: DWORD;
    th32ProcessID: DWORD; // 소유한 프로세스
    th32HeapID: DWORD;    // 블록이 속한 힙의 ID
  end;
```

- dwSize 필드에는 THeapEntry32 레코드의 크기를 넣는다. 크기는 SizeOf(THeapEntry32)로 얻어야 한다.
- hHandle에는 힙 블록의 핸들이 들어간다.
- dwAddress는 힙 블록의 시작점의 선형 주소(linear address)가 들어간다.
- dwBlockSize는 힙 블록의 바이트 단위 크기이다.
- dwFlags는 힙 블록의 종류를 뜻한다. 이 필드에 들어갈 수 있는 값은 다음과 같다.

값	의 미
LF32_FIXED	메모리 블록의 위치가 고정됨(움직일 수 없음).
LF32_FREE	메모리 블록이 사용되지 않았음.
LF32_MOVEABLE	메모리 블록의 위치를 이동할 수 있음.

- dwLockCount에는 메모리 블록의 잠금 횟수가 들어간다. 프로세스가 이 블록에 대해 GlobalLock()이나 LocalLock()을 수행할 때마다 잠금 횟수가 늘어난다.
- dwResvd는 현재 사용되지 않는다(예약됨).
- th32ProcessID는 소유한 프로세스의 ID이다.
- th32HeapID는 블록이 속한 힙의 ID이다.

개별 힙 블록에 접근하려면 우선 힙 목록의 개별 힙에 접근해야 하므로 힙을 다루는 코드는 이전 것들보다는 좀 복잡하다. TDetailForm.WalkHeaps() 메소드를 보면

Heap32ListFirst()/Heap32ListNext() 루프 안에 Heap32First()/Heap32Next() 루프가 포함되어 있다. 루프의 제일 안쪽에서는 개별 힙 블록에 대한 정보가 담긴 THeapEntry32 레코드에 대한 포인터(PHeapEntry32)를 DetailLists 배열의 힙 목록 부분에 추가한다. 나중에 개별 힙에 대한 정보를 보여줄 때 DetailLists 배열의 목록을 사용하게 된다.

```
procedure TWin95DetailForm.WalkHeaps;
{ ToolHelp32의 함수들을 이용해서 힙 목록을 나열한다. }
var
  HL: THeapList32;
  HE: THeapEntry32;
  PHE: PHeapEntry32;
begin
  DetailLists[ltHeap].Clear;
  HL.dwSize := SizeOf(HL);
  HE.dwSize := SizeOf(HE);
  if Heap32ListFirst(FCurSnap, HL) then
    repeat
      if Heap32First(HE, HL.th32ProcessID, HL.th32HeapID) then
        repeat

          New(PHE);      // 개별 힙 블록에 대한 정보를 보여주기 위해
          PHE^ := HE;    // THeapList32 레코드를 DetailLists 배열에 추가한다.
          DetailLists[ltHeap].AddObject(Format(SHeapStr, [HL.th32HeapID,
            Pointer(HE.dwAddress), HE.dwBlockSize,
            GetHeapFlagString(HE.dwFlags)]), TObject(PHE));
        until not Heap32Next(HE);
    until not Heap32ListNext(FCurSnap, HL);
  HeapListAlloc := True;
end;
```

그림 14.9는 힙 블록 목록이 표시된 DetailForm 폼이다.

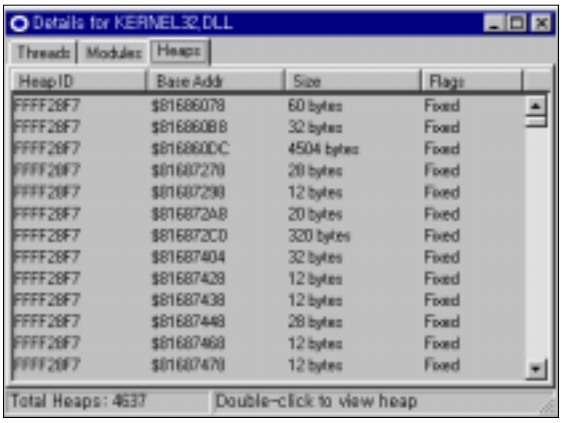


그림 14.9
윈도우즈 95 환경에서 힙 블록들이 표시된 모습

힙 블록 보기

지금까지 총 11개의 ToolHelp32 함수들을 이야기했는데, 이제 ToolHelp32의 마지막 함수인 ToolHelp32ReadProcessMemory()에 대해 알아보자.

ToolHelp32ReadProcessMemory()는 다음과 같이 선언되어 있다.

```
function Toolhelp32ReadProcessMemory(th32ProcessID: DWORD;
  lpBaseAddress: Pointer; var lpBuffer; cbRead: DWORD;
  var lpNumberOfBytesRead: DWORD): BOOL; stdcall;
```

이 함수는 ToolHelp32의 함수들 중 가장 강력하고, 가장 재미 있는 함수이다. 이 함수를 이용하면 다른 프로세스의 주소 공간에 있는 메모리 내용을 훔쳐볼 수 있다. 이 함수의 인자들은 다음과 같다.

- th32ProcessID는 읽고자 하는 메모리를 가진 프로세스의 ID이다. 이 값은 앞에서 나온 ToolHelp32의 여러 열거 함수들을 통해 얻는다. 현재 프로세스라면 0을 넣으면 된다.
- lpBaseAddress는 th32ProcessID로 지정한 프로세스의 주소 공간에서 읽고자 하는 메모리의 첫 번째 바이트의 선형 주소이다. Win32는 프로세스마다 고유한 주

소 공간을 가지므로 이 필드의 값은 th32ProcessID로 지정한 프로세스에서만 의미를 가진다.

- lpBuffer는 th32ProcessID로 지정한 프로세스의 주소 공간에서 읽어온 메모리가 들어갈 버퍼이다. 이 버퍼는 미리 메모리가 할당되어 있어야 한다.
- cbRead는 th32ProcessID로 지정한 프로세스의 주소 공간에서 lpBaseAddress로부터 읽어 들인 바이트 수이다.
- lpNumberOfBytesRead는 함수가 설정하는 값으로, th32ProcessID로 지정한 프로세스의 주소 공간에서 실제로 읽어들인 바이트 수를 뜻한다.

SysInfo 프로젝트는 선택된 힙 블록으로부터 읽어들인 메모리의 내용을 HeapViewForm이라는 모달 폼으로 표시한다. HeapViewForm은 TddgMemView라는 커스텀 컴포넌트를 이용해서 메모리의 내용을 표시한다. TddgMemView의 내부적 구현에 대한 내용은 이번 장의 주제와는 맞지 않으므로(그리고 별로 복잡하지도 않으므로) 따로 설명은 하지 않겠으니 부록 CD의 소스 코드를 보고 스스로 분석하기 바란다. 다음에 나온 TDetailForm.DetailLBDblClick()에서는 메소드 사용자가 THeaderListbox 컴포넌트인 DetailLB를 더블클릭하면 실행된다.

```
procedure TWin95DetailForm.DetailLBDblClick(Sender: TObject);
{ 사용자가 DetailLB의 한 항목을 더블클릭하면 이 프로시저가 실행된다.
  현재 탭 페이지가 Heaps이면 HeapViewForm을 띄워서 힙 내용을 표시한다. }
var
  NumRead: DWORD;
  HE: THeapEntry32;
  MemSize: integer;
begin
  inherited;
  if DetailTabs.TabIndex = 2 then
  begin
    HE := PHeapEntry32(DetailLB.Items.Objects[DetailLB.ItemIndex])^;
    MemSize := HE.dwBlockSize;         // 힙 크기를 얻는다.
    { 힙의 최대 크기를 ProcMemMaxSize로 제한한다. }
    if MemSize > ProcMemMaxSize then MemSize := ProcMemMaxSize;
    ProcMem := AllocMem(MemSize);     // 임시 버퍼에 메모리를 할당
    Screen.Cursor := crHourGlass;
    try
```

```
      { 힙 내용을 임시 버퍼에 복사한다. }
      if Toolhelp32ReadProcessMemory(FCurProc.th32ProcessID,
        Pointer(HE.dwAddress), ProcMem^, MemSize, NumRead) then
        { HeapViewForm을 띄워서 임시 버퍼의 내용을 표시한다. }
        ShowHeapView(ProcMem, MemSize)
      else
        MessageDlg(SHeapReadErr, mtInformation, [mbOk], 0);
    finally

      Screen.Cursor := crDefault;
      FreeMem(ProcMem, MemSize);
    end;
  end;
end;
```

이 메소드는 우선 현재 탭 페이지가 힙 목록 페이지인지 확인한다. 힙 목록 페이지이면 임시 버퍼를 할당한 후 그 버퍼를 Toolhelp32ReadProcessMemory()에 넘겨서 힙 블록의 내용으로 채운다. 버퍼가 채워지면 TddgMemView 컴포넌트인 HeapView가 있는 HeapViewForm을 모달로 띄워서 버퍼의 내용을 표시한다. 모달 폼이 닫히면 버퍼를 해제한다. 그림 14.10은 힙 내용이 표시된 모습이다.

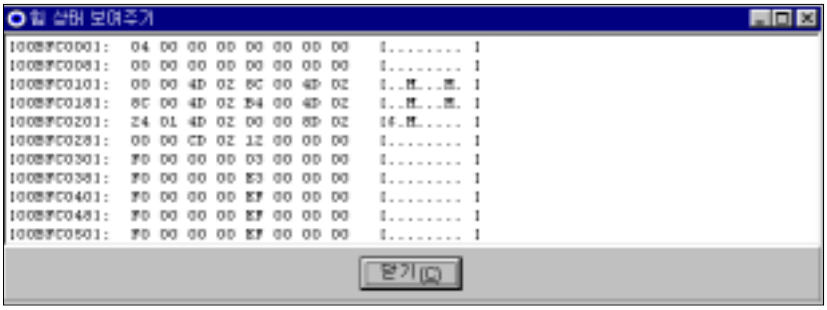


그림 14.10
윈도우즈 95의 한 프로세스에 속한 힙의 내용이 표시된 모습

소스 코드

목록 14.2는 W95Info.pas의 소스 코드이고, 목록 14.3은 Detail95.pas의 소스 코드이다.



Ch14\SysInfor\W95Info.pas

```
unit W95Info;

interface

uses Windows, InfoInt, Classes, TlHelp32, Controls, ComCtrls;

type
  TWin95Info = class(TInterfacedObject, IWin32Info)
  private
    FProcList: TList;
    FWinIcon: HICON;
    FSnap: THandle;
    procedure Refresh;
  public
    constructor Create;
    destructor Destroy; override;
    procedure FillProcessInfoList(ListView: TListView; ImageList:
TImageList);
    procedure ShowProcessProperties(Cookie: Pointer);
  end;

implementation

uses ShellAPI, CommCtrl, SysUtils, Detail95;

const
  ProcessInfoCaptions: array[0..3] of string = (
    'ProcessName', 'Threads', 'ID', 'ParentID');

{ TProcList }

type
  TProcList = class(TList)
```

```
    procedure Clear; override;
  end;

procedure TProcList.Clear;
var
  I: Integer;
begin
  for I := 0 to Count - 1 do Dispose(PProcessEntry32(Items[I]));
  inherited Clear;
end;

{ TWin95Info }

constructor TWin95Info.Create;
begin
  FProcList := TProcList.Create;
  FWinIcon := LoadImage(0, IDI_WINLOGO, IMAGE_ICON, LR_DEFAULTSIZE,
    LR_DEFAULTSIZE, LR_DEFAULTSIZE or LR_DEFAULTCOLOR or LR_SHARED);
end;

destructor TWin95Info.Destroy;
begin
  DestroyIcon(FWinIcon);
  if FSnap > 0 then CloseHandle(FSnap);
  FProcList.Free;
  inherited Destroy;
end;

procedure TWin95Info.FillProcessInfoList(ListView: TListView;
  ImageList: TImageList);
var
  I: Integer;
  ExeFile: string;
  PE: TProcessEntry32;
  HAppIcon: HIcon;
begin
  Refresh;
  ListView.Columns.Clear;
  ListView.Items.Clear;
```

```
  for I := Low(ProcessInfoCaptions) to High(ProcessInfoCaptions) do
    with ListView.Columns.Add do
    begin
      if I = 0 then Width := 285
      else Width := 75;
      Caption := ProcessInfoCaptions[I];
    end;
  for I := 0 to FProcList.Count - 1 do
  begin
    PE := PProcessEntry32(FProcList.Items[I])^;
    HAppIcon := ExtractIcon(HInstance, PE.szExeFile, 0);
    try
      if HAppIcon = 0 then HAppIcon := FWinIcon;
      ExeFile := PE.szExeFile;
      if ListView.ViewStyle = vsList then
        ExeFile := ExtractFileName(ExeFile);
      // 새 항목을 추가하고 제목을 설정한 후 부 항목들을 추가한다.
      with ListView.Items.Add, SubItems do
      begin
        Caption := ExeFile;
        Data := FProcList.Items[I];
        Add(IntToStr(PE.cntThreads));
        Add(IntToHex(PE.th32ProcessID, 8));
        Add(IntToHex(PE.th32ParentProcessID, 8));
        if ImageList <> nil then
          ImageIndex := ImageList_AddIcon(ImageList.Handle, HAppIcon);
      end;
    finally
      if HAppIcon <> FWinIcon then DestroyIcon(HAppIcon);
    end;
  end;
end;

procedure TWin95Info.Refresh;
var
  PE: TProcessEntry32;
  PPE: PProcessEntry32;
begin
  FProcList.Clear;
```

```
  if FSnap > 0 then CloseHandle(FSnap);
  FSnap := CreateToolHelp32Snapshot(TH32CS_SNAPPROCESS, 0);
  if FSnap = INVALID_HANDLE_VALUE then
    raise Exception.Create('CreateToolHelp32Snapshot failed');
  PE.dwSize := SizeOf(PE);
  if Process32First(FSnap, PE) then   // 첫 번째 프로세스 정보를 얻는다.
    repeat
      New(PPE);                        // 새 PPE를 생성하고
      PPE^ := PE;                      // 채운 다음
      FProcList.Add(PPE);             // 목록에 추가한다.
    until not Process32Next(FSnap, PE);  // 다음 프로세스 정보를 얻는다.
end;

procedure TWin95Info.ShowProcessProperties(Cookie: Pointer);
begin
  ShowProcessDetails(PProcessEntry32(Cookie));
end;

 end.
```

목록 14.3 Ch14\SysInfor\Detail95.pas

```
unit Detail95;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ComCtrls, HeadList, TlHelp32, Menus, SysMain, DetBase;

type
  TListType = (ltThread, ltModule, ltHeap);

  TWin95DetailForm = class(TBaseDetailForm)
    procedure DetailTabsChange(Sender: TObject);
```

```
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure DetailLBDblClick(Sender: TObject);
  private
    FCurSnap: THandle;
    FCurProc: TProcessEntry32;
    DetailLists: array[TListType] of TStringList;
    ProcMem: PByte;
    HeapListAlloc: Boolean;
    procedure FreeHeapList;
    procedure ShowList(ListType: TListType);
    procedure WalkThreads;
    procedure WalkHeaps;
    procedure WalkModules;
  public
    procedure NewProcess(P: PProcessEntry32);
  end;

procedure ShowProcessDetails(P: PProcessEntry32);

implementation

{$R *.DFM}

uses ProcMem;

const
  { 각 목록의 컬럼 제목에 표시할 문자열들 }
  HeaderStrs: array[TListType] of TDetailStrings = (
      ('Thread ID', 'Base Priority', 'Delta Priority', 'Usage Count'),
      ('Module', 'Base Addr', 'Size', 'Usage Count'),
      ('Heap ID', 'Base Addr', 'Size', 'Flags'));

  { 각 목록의 꼬리말에 표시할 문자열들 }
  ACountStrs: array[TListType] of string[31] = (
      'Total Threads: %d', 'Total Modules: %d', 'Total Heaps: %d');

  TabStrs: array[TListType] of string[7] = ('Threads', 'Modules', 'Heaps');
```

```
  SCaptionStr  = 'Details for %s';       // 폼의 제목
  SThreadStr   = '%x'#1'%s'#1'%s'#1'%d'; // ID, 기본 우선순위, 우선순위
                                         // 변화치, 참조 횟수
  SModuleStr   = '%s'#1'$%p'#1'%d bytes'#1'%d'; // 이름, 주소, 크기, 참조 횟수
  SHeapStr     = '%x'#1'$%p'#1'%d bytes'#1'%s'; // ID, 주소, 크기, 플래그
  SHeapReadErr = 'This heap is not accessible for read access.';

  ProcMemMaxSize = $7FFE;                // 힙 블록의 최대 크기

procedure ShowProcessDetails(P: PProcessEntry32);
var
  I: TListType;
begin
  with TWin95DetailForm.Create(Application) do
    try
      for I := Low(TabStrs) to High(TabStrs) do
        DetailTabs.Tabs.Add(TabStrs[I]);
      NewProcess(P);
      Font := MainForm.Font;
      ShowModal;
    finally
      Free;
    end;
end;

function GetThreadPriorityString(Priority: Integer): string;
{ 스레드의 우선순위를 뜻하는 문자열을 돌려주는 함수 }
begin
  case Priority of
    THREAD_PRIORITY_IDLE:          Result := '%d (Idle)';
    THREAD_PRIORITY_LOWEST:        Result := '%d (Lowest)';
    THREAD_PRIORITY_BELOW_NORMAL:  Result := '%d (Below Normal)';
    THREAD_PRIORITY_NORMAL:        Result := '%d (Normal)';
    THREAD_PRIORITY_ABOVE_NORMAL:  Result := '%d (Above Normal)';
    THREAD_PRIORITY_HIGHEST:       Result := '%d (Highest)';
    THREAD_PRIORITY_TIME_CRITICAL: Result := '%d (Time critical)';
  else
    Result := '%d (unknown)';
  end;
```

```
  Result := Format(Result, [Priority]);
end;

function GetClassPriorityString(Priority: DWORD): String;
{ 프로세스의 우선순위 등급을 뜻하는 문자열을 돌려주는 함수}
begin
  case Priority of
    4:   Result := '%d (Idle)';
    8:   Result := '%d (Normal)';
    13:  Result := '%d (High)';
    24:  Result := '%d (Real time)';
  else
    Result := '%d (non-standard)';
  end;
  Result := Format(Result, [Priority]);
end;

function GetHeapFlagString(Flag: DWORD): String;
{ 힙 종류를 뜻하는 문자열을 돌려주는 함수 }
begin
  case Flag of
    LF32_FIXED:    Result := 'Fixed';
    LF32_FREE:     Result := 'Free';
    LF32_MOVEABLE: Result := 'Moveable';
  end;
end;

procedure TWin95DetailForm.ShowList(ListType: TListType);
{ 인자에 따라 스레드나 힙, 또는 모듈 목록을 DetailLB에 표시한다. }
var
  i: Integer;
begin
  Screen.Cursor := crHourGlass;
  try
    with DetailLB do
    begin
      for i := 0 to 3 do
        Sections[i].Text := HeaderStrs[ListType, i];
      Items.Clear;
```

```
      Items.Assign(DetailLists[ListType]);
    end;
     DetailSB.Panels[0].Text := Format(ACountStrs[ListType],
       [DetailLists[ListType].Count]);
     if ListType = ltHeap then
       DetailSB.Panels[1].Text := 'Double-click to view heap'
     else
       DetailSB.Panels[1].Text := '';
  finally
    Screen.Cursor := crDefault;
  end;
end;

procedure TWin95DetailForm.WalkThreads;
{ ToolHelp32 함수들을 이용해서 스레드들을 나열한다. }
var
  T: TThreadEntry32;
begin
  DetailLists[ltThread].Clear;
  T.dwSize := SizeOf(T);
  if Thread32First(FCurSnap, T) then
    repeat
      { 현재 프로세스의 스레드인지 확인한다. }
      if T.th32OwnerProcessID = FCurProc.th32ProcessID then
        DetailLists[ltThread].Add(Format(SThreadStr, [T.th32ThreadID,
          GetClassPriorityString(T.tpBasePri),
          GetThreadPriorityString(T.tpDeltaPri), T.cntUsage]));
    until not Thread32Next(FCurSnap, T);
end;

procedure TWin95DetailForm.WalkModules;
{ ToolHelp32 함수들을 이용해서 모듈들을 나열한다. }
var
  M: TModuleEntry32;
begin
  DetailLists[ltModule].Clear;
  M.dwSize := SizeOf(M);
  if Module32First(FCurSnap, M) then
    repeat
```

```
      DetailLists[ltModule].Add(Format(SModuleStr, [M.szModule,
              M.ModBaseAddr, M.ModBaseSize, M.ProcCntUsage]));
    until not Module32Next(FCurSnap, M);
end;

procedure TWin95DetailForm.WalkHeaps;
{ ToolHelp32의 함수들을 이용해서 힙 목록을 나열한다. }
var
  HL: THeapList32;
  HE: THeapEntry32;
  PHE: PHeapEntry32;
begin
  DetailLists[ltHeap].Clear;
  HL.dwSize := SizeOf(HL);
  HE.dwSize := SizeOf(HE);
  if Heap32ListFirst(FCurSnap, HL) then
    repeat
      if Heap32First(HE, HL.th32ProcessID, HL.th32HeapID) then
        repeat
          New(PHE);      // 개별 힙 블록에 대한 정보를 보여주기 위해
          PHE^ := HE;    // THeapList32 레코드를 DetailLists 배열에 추가한다.
          DetailLists[ltHeap].AddObject(Format(SHeapStr, [HL.th32HeapID,
            Pointer(HE.dwAddress), HE.dwBlockSize,
            GetHeapFlagString(HE.dwFlags)]), TObject(PHE));
        until not Heap32Next(HE);
    until not Heap32ListNext(FCurSnap, HL);
  HeapListAlloc := True;
end;

procedure TWin95DetailForm.FreeHeapList;
{ DetailLists 배열에 추가된 PHeapList32 객체들을 모두 해제한다. }
var
  i: integer;
begin
  for i := 0 to DetailLists[ltHeap].Count - 1 do
    Dispose(PHeapEntry32(DetailLists[ltHeap].Objects[i]));
end;

procedure TWin95DetailForm.NewProcess(P: PProcessEntry32);
```

```
{ 프로그램의 메인 폼이 이 폼을 띄울 때 실행되는 프로시저이다. }
begin
  { 현재 프로세스에 대한 스냅샷을 생성한다. }
  FCurSnap := CreateToolhelp32Snapshot(TH32CS_SNAPALL, P^.th32ProcessID);
  if FCurSnap = INVALID_HANDLE_VALUE then
    raise Exception.Create('CreateToolHelp32Snapshot failed');
  HeapListAlloc := False;
  Screen.Cursor := crHourGlass;
  try
    FCurProc := P^;
    { Include module name in detail form caption }
    Caption := Format(SCaptionStr, [ExtractFileName(FCurProc.szExeFile)]);
    WalkThreads;                        // ToolHelp32의 목록들을 나열
    WalkModules;
    WalkHeaps;
    DetailTabs.TabIndex := 0;           // 0은 Threads 탭임.
    ShowList(ltThread);                 // Threads 페이지를 표시한다.
  finally
    Screen.Cursor := crDefault;
    if HeapListAlloc then FreeHeapList;
    CloseHandle(FCurSnap);              // 스냅샷 핸들을 종료한다.
  end;
end;

procedure TWin95DetailForm.DetailTabsChange(Sender: TObject);
{ 탭셋의 OnChange 이벤트 처리 프로시저. 선택된 탭에 해당하는 목록을 표시한다. }
begin
  inherited;
  ShowList(TListType(DetailTabs.TabIndex));
end;

procedure TWin95DetailForm.FormCreate(Sender: TObject);
var
  LT: TListType;
begin
  inherited;
  { 목록들을 생성한다. }
  for LT := Low(TListType) to High(TListType) do
    DetailLists[LT] := TStringList.Create;
```

```
end;

procedure TWin95DetailForm.FormDestroy(Sender: TObject);
var
  LT: TListType;
begin
  inherited;
  { 목록들을 해제한다. }
  for LT := Low(TListType) to High(TListType) do
    DetailLists[LT].Free;
end;

procedure TWin95DetailForm.DetailLBDblClick(Sender: TObject);
{ 사용자가 DetailLB의 한 항목을 더블클릭하면 이 프로시저가 실행된다.
 현재 탭 페이지가 Heaps이면 HeapViewForm을 띄워서 힙 내용을 표시한다. }
var
  NumRead: DWORD;
  HE: THeapEntry32;
  MemSize: integer;
begin
  inherited;
  if DetailTabs.TabIndex = 2 then
  begin
    HE := PHeapEntry32(DetailLB.Items.Objects[DetailLB.ItemIndex])^;
    MemSize := HE.dwBlockSize;         // 힙 크기를 얻는다.
    { 힙의 최대 크기를 ProcMemMaxSize로 제한한다. }
    if MemSize > ProcMemMaxSize then MemSize := ProcMemMaxSize;
    ProcMem := AllocMem(MemSize);     // 임시 버퍼에 메모리를 할당
    Screen.Cursor := crHourGlass;
    try
      { 힙 내용을 임시 버퍼에 복사한다. }
      if Toolhelp32ReadProcessMemory(FCurProc.th32ProcessID,
        Pointer(HE.dwAddress), ProcMem^, MemSize, NumRead) then
        { HeapViewForm을 띄워서 임시 버퍼의 내용을 표시한다. }
        ShowHeapView(ProcMem, MemSize)
      else
        MessageDlg(SHeapReadErr, mtInformation, [mbOk], 0);
    finally
      Screen.Cursor := crDefault;
```

```
        FreeMem(ProcMem, MemSize);
      end;
    end;
  end;

  end.
```

윈도우즈 NT의 PSAPI 사용법

앞에서도 말했지만, 윈도우즈 NT에서는 ToolHelp32 API를 사용할 수 없다. 대신, 윈도우즈 NT의 SDK에는 PASPI.DLL이라는 DLL이 있는데, 이것을 이용하면 다음과 같은 것들에 대한 정보들을 얻을 수 있다.

- 실행중인 프로세스들
- 프로세스가 로드한 모듈들
- 적재된 장치 드라이버들
- 프로세스의 메모리
- 프로세스에 맵핑된 메모리 맵 파일

이후 버전의 윈도우즈 NT에는 PSAPI.DLL이 정식으로 포함되겠지만, 윈도우즈 NT SDK에 대한 라이센스를 가지고 있으면 PSAPI.DLL도 애플리케이션에 포함시켜서 배포할 수 있다. 델파이의 PSAPI.pas는 이 DLL에 대한 인터페이스를 정의한 유닛이다. PSAPI.pas를 사용하는 델파이 애플리케이션은 PSAPI.DLL이 있든 없든 실행은 되지만, PSAPI.DLL이 없으면 제대로 작동하지는 않는다.

현재 델파이 5에는 Delphi5\Source\RTL\Win\에 PSAPI.pas가 들어 있다.

PSAPI로 프로세스 정보를 얻는 첫 번째 단계는 EnumProcesses() 함수를 호출하는 것인데, 이 함수의 선언은 다음과 같다.

```
function EnumProcesses(lpidProcess: LPDWORD; cb: DWORD;
  var cbNeeded: DWORD): BOOL;
```

- lpidProcess는 DWORD 배열에 대한 포인터로, 함수를 호출하면 DWORD 배열에 프로세스 ID들이 채워진다.
- cb는 lpidProcess가 가리키는 배열의 원소, 즉 DWORD들의 개수이다.
- cbNeeded는 lpidProcess가 가리키는 배열의 메모리 크기이다. 따라서, cbNeeded div SizeOf(DWORD)를 하면 배열의 원소 개수, 다시 말해 실행중인 프로세스 개수가 나온다.

이 함수를 호출하면 lpidProcess가 가리키는 배열에 프로세스 ID들이 채워진다. 프로세스 ID를 API 함수 OpenProcess()에 넣어서 호출하면 프로세스 핸들을 얻을 수 있다. 프로세스 핸들을 얻은 다음에는 PSAPI 함수들이나 다른 Win32 API 함수들을 이용해서 프로세스에 대한 정보를 얻으면 된다.

PSAPI는 장치 드라이버들에 대한 정보를 얻는 함수도 제공하는데, 이름도 거의 비슷한 형태인 EnumDeviceDrivers()가 바로 그것이다. 이 함수의 선언은 다음과 같다. 인자들도 거의 비슷하다.

```
function EnumDeviceDrivers(lpImageBase: PPointer; cb: DWORD;
  var lpcbNeeded: DWORD): BOOL;
```

- lpImageBAse는 각 장치 드라이버의 기준 주소들이 들어갈 Pointer 배열이다.
- cb는 lpImageBase가 가리키는 포인터 배열의 원소 개수이다.
- lpcbNeeded는 ImageBase에 복사된 바이트 수이다.

SysInfo 프로젝트에서 IWin32Info 인터페이스를 구현하는 것은 TWinNTInfo 클래스이다. 다음 코드는 TWinNTInfo 클래스의 private 메소드인 Refresh()인데, 이 메소드는 프로세스와 장치 드라이버 정보를 얻어서 목록에 추가한다.

```
procedure TWinNTInfo.Refresh;
var
  Count: DWORD;
  BigArray: array[0..$3FFF - 1] of DWORD;
begin
  // 프로세스 ID들의 배열을 얻는다.
  if not EnumProcesses(@BigArray, SizeOf(BigArray), Count) then
    raise Exception.Create(SFailMessage);
  SetLength(FProcList, Count div SizeOf(DWORD));
  Move(BigArray, FProcList[0], Count);
  // 드라이버 주소의 배열을 얻는다.
  if not EnumDeviceDrivers(@BigArray, SizeOf(BigArray), Count) then

    raise Exception.Create(SFailMessage);
  SetLength(FDrvList, Count div SizeOf(DWORD));
  Move(BigArray, FDrvList[0], Count);
end;
```

이 메소드는 우선 EnumProcess()를 호출해서 BigArray라는 지역 배열 변수를 채운 후 BigArray의 내용을 FProcList라는 동적 배열에 넣는다.

마찬가지 방법으로, EnumDeviceDrivers()를 호출해서 채운 BigArray는 FDrvList에 넣는다. 임시 배열을 사용하는 이유는 EnumProcess()와 EnumDeviceDrivers()를 호출한 다음에야 배열의 크기를 판단할 수 있기 때문이다. 그래서 미리 만들어 놓은 충분히 큰 배열을 이용해서 정보를 얻은 다음, 필요한 만큼만 동적 배열에 옮기는 것이다.

TWinNTInfo의 FillProcessInfoList() 메소드는 FillProcesses()와 FillDrivers()라는 프로시저를 호출해서 메인 폼의 TListView에 프로세스와 드라이버 정보를 채운다. FillProcess()의 코드는 다음과 같다.

```
procedure TWinNTInfo.FillProcesses(ListView: TListView;
  ImageList: TImageList);
var
  I: Integer;
  Count: DWORD;
  ProcHand: THandle;
  ModHand: HMODULE;
  HAppIcon: HICON;
  ModName: array[0..MAX_PATH] of char;
begin
  for I := Low(FProcList) to High(FProcList) do
  begin
    ProcHand := OpenProcess(PROCESS_QUERY_INFORMATION or PROCESS_VM_READ,
          False, FProcList[I]);
    if ProcHand > 0 then
      try
        EnumProcessModules(Prochand, @ModHand, 1, Count);
        if GetModuleFileNameEx(Prochand, ModHand, ModName, SizeOf(ModName))
             > 0 then
        begin
          HAppIcon := ExtractIcon(HInstance, ModName, 0);
          try
            if HAppIcon = 0 then HAppIcon := FWinIcon;
            with ListView.Items.Add, SubItems do
            begin
              Caption := ModName;                    // 파일 이름
              Data := Pointer(FProcList[I]);          // ID를 보존한다.
              Add(SProcName);                        // "process"를 추가
              Add(IntToStr(FProcList[I]));            // 프로세스 ID를 추가
             Add('$' + IntToHex(ProcHand, 8));      // 프로세스 핸들을 추가
              // 우선순위 등급을 추가
              Add(GetPriorityClassString(GetPriorityClass(ProcHand)));
              // 아이콘 처리
              if ImageList <> nil then
             ImageIndex := ImageList_AddIcon(ImageList.Handle, HAppIcon);
            end;
          finally
            if HAppIcon <> FWinIcon then DestroyIcon(HAppIcon);
          end;
        end;
      finally
        CloseHandle(ProcHand);
      end;
  end;
end;
```

이 메소드는 프로세스 ID를 OpenProcess() API 함수에 넣어서 그 프로세스의 핸들을 얻는다. OpenProcess() 함수의 첫 번째 인자에는 여러 플래그들을 조합해서 넣을 수 있는데, 이 메소드의 목적은 PSAPI를 통해서 프로세스 정보를 얻는 것이므로 PROCESS_QUERY_INFORMATION과 PROCESS_VM_READ만 넣어주면 된다. 프로세스 핸들을 얻은 후에는 EnumProcessModules()에 프로세스 핸들을 넣어서 프로세스의 파일 이름을 얻는다. EnumProcessModules()의 선언은 다음과 같다.

```
function EnumProcessModules(hProcess: THandle; lphModule: LPDWORD;
  cb: DWORD; var lpcbNeeded: DWORD): BOOL;
```

이 함수는 다른 PSAPI 함수와 거의 비슷한 방식으로 작동한다. hProcess는 프로세스 핸들이고, lphModule은 모듈 핸들 배열을 가리키는 포인터이며, cb는 배열의 원소 개수이다. 그리고 마지막 인자 lpcbNeeded는 lphModule에 복사된 바이트 수이다.

일단 지금은 프로세스의 기본 모듈만 얻으면 되므로 원소가 하나인 배열을 넘겨주면 된다. EnumProcessModules()가 채워주는 핸들 배열의 첫 번째 것이 프로세스의 기본 모듈 핸들이다. FillProcesses()는 얻어낸 모듈 정보를 TListView에 추가하는데, TListView에 정보를 추가하는 과정은 TWin95Info에서의 것과 다를 것이 없으므로 설명하지 않겠다.

FillDriver() 함수의 동작 과정도 FillProcess()와 거의 동일하다. 단, FillDriver()는 GetDeviceDriverFileName() 함수를 이용해서 드라이버에 대한 정보를 얻는다.

GetDeviceDriverFileName()의 선언은 다음과 같다.

```
function GetDeviceDriverFileName(ImageBase: Pointer; lpFileName: PChar;
  nSize: DWORD): DWORD;
```

이 메소드의 첫 번째 인자는 장치 드라이버 이미지의 기준 주소이고, 두 번째 인자는 파일 이름을 뜻하는 문자열 버퍼의 포인터이며, 마지막 인자는 그 버퍼의 크기이다. 이 함수가 성공적으로 수행되면 lpFileName에는 장치 드라이버의 파일 이름이 들어간다. 이 함수를 사용하는 FillDrivers() 메소드는 다음과 같다.

```
procedure TWinNTInfo.FillDrivers(ListView: TListView;
  ImageList: TImageList);
var
  I: Integer;
  DrvName: array[0..MAX_PATH] of char;

begin
  for I := Low(FDrvList) to High(FDrvList) do
    if GetDeviceDriverFileName(FDrvList[I], DrvName, SizeOf(DrvName)) > 0
          then
      with ListView.Items.Add do
      begin
        Caption := DrvName;
        SubItems.Add(SDrvName);
        SubItems.Add('$' + IntToHex(Integer(FDrvList[I]), 8));
      end;
end;
```

그림 14.11은 윈도우즈 NT 4.0에서 실행중인 SysInfo의 모습이다.

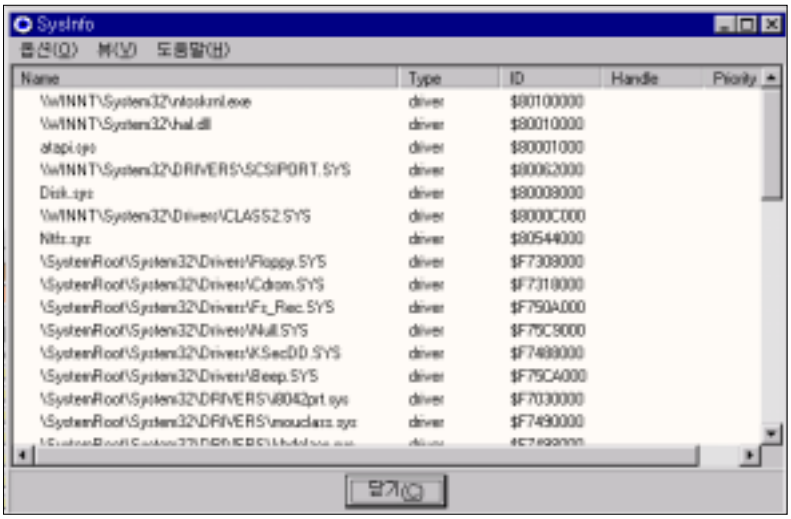


그림 14.11
윈도우즈 NT 환경에서 프로세스와 드라이버 정보가 표시된 모습

TWin95Info에서 구현된 ShowProcessProperties()와 마찬가지로, TWinNTInfo의 ShowProcessProperties() 역시 선택된 프로세스에 대한 자세한 정보를 보여주는 모달 폼을 띄운다. 자세한 정보에는 프로세스 모듈과 메모리 사용 정보가 포함된다. 프로세스의 모듈과 메모리 관련 정보를 얻는 것은 NewProcess() 메소드인데, 이 메소드는 DetailNT.pas에 정의된 TWinNTDetailForm 클래스에 속한 메소드이다. 다음은 NewProcess()의 코드이다.

```
procedure TWinNTDetailForm.NewProcess(ProcessID: DWORD);
const
  AddrMask = DWORD($FFFFF000);
var
  I: Integer;
  Count: DWORD;
  ProcHand: THandle;

  WSPtr: Pointer;
  ModHandles: array[0..$3FFF - 1] of DWORD;
  WorkingSet: array[0..$3FFF - 1] of DWORD;
  ModInfo: TModuleInfo;
  ModName, MapFileName: array[0..MAX_PATH] of char;
begin
  ProcHand := OpenProcess(PROCESS_QUERY_INFORMATION or PROCESS_VM_READ,
      False, ProcessID);
  if ProcHand = 0 then
    raise Exception.Create(
              'No information available for this process/driver');
  try
    EnumProcessModules(ProcHand, @ModHandles, SizeOf(ModHandles), Count);
    for I := 0 to (Count div SizeOf(DWORD)) - 1 do
      if (GetModuleFileNameEx(ProcHand, ModHandles[I], ModName,
        SizeOf(ModName)) > 0) and GetModuleInformation(ProcHand,
        ModHandles[I], @ModInfo, SizeOf(ModInfo)) then
        with ModInfo do
```

```
          DetailLists[ltModules].Add(Format(SModuleStr, [ModName,
                        lpBaseOfDll, SizeOfImage, EntryPoint]));
      if QueryWorkingSet(ProcHand, @WorkingSet, SizeOf(WorkingSet)) then
        for I := 1 to WorkingSet[0] do
        begin
          WSPtr := Pointer(WorkingSet[I] and AddrMask);
          GetMappedFileName(ProcHand, WSPtr, MapFileName,
                     SizeOf(MapFileName));
          DetailLists[ltMemory].Add(Format(SMemoryStr, [WSPtr,
            MemoryTypeToString(WorkingSet[I]), MapFileName]));
        end;
    finally
      CloseHandle(ProcHand);
    end;
  end;
```

이 메소드는 앞에서 배운 OpenProcess()와 EnumProcessModules()를 호출해서 모듈 정보를 얻는다. 그리고 QueryWorkingSet()이라는 또 하나의 PSAPI 함수를 호출해서 프로세스의 메모리 정보를 얻는데, 이 함수의 선언은 다음과 같다.

```
function QueryWorkingSet(hProcess: THandle; pv: Pointer; cb: DWORD): BOOL;
```

hProcess는 프로세스 핸들이고, pv는 DWORD 배열을 가리키는 포인터이며, cb는 배열의 원소 개수이다. 이 함수가 성공적으로 호출되면 pv는 DWORD 배열을 가리키게 된다. 각 DWORD의 상위 20비트는 개별 메모리 페이지의 기준 주소가 들어 있고, 하위 12비트에는 그 페이지의 접근 특성(읽기 가능, 쓰기 가능, 실행 가능 등등)을 뜻하는 플래그가 들어 있다.

목록 14.4 Ch14\SysInfor\WNTInfo.pas

```
unit WNTInfo;

interface

uses InfoInt, Windows, Classes, ComCtrls, Controls;
```

```
type
  TWinNTInfo = class(TInterfacedObject, IWin32Info)
  private
    FProcList: array of DWORD;
    FDrvlist: array of Pointer;
    FWinIcon: HICON;
    procedure FillProcesses(ListView: TListView; ImageList: TImageList);
    procedure FillDrivers(ListView: TListView; ImageList: TImageList);
    procedure Refresh;
  public
    constructor Create;
    destructor Destroy; override;
    procedure FillProcessInfoList(ListView: TListView; ImageList:
              TImageList);
    procedure ShowProcessProperties(Cookie: Pointer);
  end;

implementation

uses SysUtils, PSAPI, ShellAPI, CommCtrl, DetailNT;

const
  SFailMessage = 'Failed to enumerate processes or drivers.  Make sure ' +
    'PSAPI.DLL is installed on your system.';
  SDrvName = 'driver';
  SProcname = 'process';
  ProcessInfoCaptions: array[0..4] of string = (
    'Name', 'Type', 'ID', 'Handle', 'Priority');

function GetPriorityClassString(PriorityClass: Integer): string;
begin
  case PriorityClass of
    HIGH_PRIORITY_CLASS: Result := 'High';
    IDLE_PRIORITY_CLASS: Result := 'Idle';
    NORMAL_PRIORITY_CLASS: Result := 'Normal';
    REALTIME_PRIORITY_CLASS: Result := 'Realtime';
  else
    Result := Format('Unknown ($%x)', [PriorityClass]);
  end;
```

```
end;

{ TWinNTInfo }

constructor TWinNTInfo.Create;
begin
  FWinIcon := LoadImage(0, IDI_WINLOGO, IMAGE_ICON, LR_DEFAULTSIZE,
    LR_DEFAULTSIZE, LR_DEFAULTSIZE or LR_DEFAULTCOLOR or LR_SHARED);
end;

destructor TWinNTInfo.Destroy;
begin
  DestroyIcon(FWinIcon);
  inherited Destroy;
end;

procedure TWinNTInfo.FillDrivers(ListView: TListView;
  ImageList: TImageList);
var
  I: Integer;
  DrvName: array[0..MAX_PATH] of char;
begin
  for I := Low(FDrvList) to High(FDrvList) do
    if GetDeviceDriverFileName(FDrvList[I], DrvName, SizeOf(DrvName)) > 0
          then
      with ListView.Items.Add do
      begin
        Caption := DrvName;
        SubItems.Add(SDrvName);
        SubItems.Add('$' + IntToHex(Integer(FDrvList[I]), 8));
      end;
end;

procedure TWinNTInfo.FillProcesses(ListView: TListView;
  ImageList: TImageList);
var
  I: Integer;
  Count: DWORD;
  ProcHand: THandle;
```

```
  ModHand: HMODULE;
  HAppIcon: HICON;
  ModName: array[0..MAX_PATH] of char;
begin
  for I := Low(FProcList) to High(FProcList) do
  begin
    ProcHand := OpenProcess(PROCESS_QUERY_INFORMATION or PROCESS_VM_READ,
          False, FProcList[I]);
    if ProcHand > 0 then
      try
        EnumProcessModules(Prochand, @ModHand, 1, Count);
        if GetModuleFileNameEx(Prochand, ModHand, ModName,
                  SizeOf(ModName)) > 0 then
        begin
          HAppIcon := ExtractIcon(HInstance, ModName, 0);
          try
            if HAppIcon = 0 then HAppIcon := FWinIcon;
            with ListView.Items.Add, SubItems do
            begin
              Caption := ModName;                    // 파일 이름
              Data := Pointer(FProcList[I]);          // ID를 보존한다.
              Add(SProcName);                        // "process"를 추가
              Add(IntToStr(FProcList[I]));            // 프로세스 ID를 추가
             Add('$' + IntToHex(ProcHand, 8));      // 프로세스 핸들을 추가
              // priority class
              Add(GetPriorityClassString(GetPriorityClass(ProcHand)));
              // 아이콘 처리
              if ImageList <> nil then
               ImageIndex := ImageList_AddIcon(ImageList.Handle, HAppIcon);
            end;
          finally
            if HAppIcon <> FWinIcon then DestroyIcon(HAppIcon);
          end;
        end;
      finally
        CloseHandle(ProcHand);
      end;
  end;
end;
```

```
procedure TWinNTInfo.FillProcessInfoList(ListView: TListView;
  ImageList: TImageList);
var
  I: Integer;
begin
  Refresh;
  ListView.Columns.Clear;
  ListView.Items.Clear;
  for I := Low(ProcessInfoCaptions) to High(ProcessInfoCaptions) do
    with ListView.Columns.Add do
    begin
      if I = 0 then Width := 285
      else Width := 75;
      Caption := ProcessInfoCaptions[I];
    end;
  FillProcesses(ListView, ImageList); // 프로세스들을 ListView에 채운다.
  FillDrivers(ListView, ImageList); // 장치 드라이버들을 ListView에 채운다.
end;

procedure TWinNTInfo.Refresh;
var
  Count: DWORD;
  BigArray: array[0..$3FFF - 1] of DWORD;
begin
  // 프로세스 ID의 배열을 얻는다.
  if not EnumProcesses(@BigArray, SizeOf(BigArray), Count) then
    raise Exception.Create(SFailMessage);
  SetLength(FProcList, Count div SizeOf(DWORD));
  Move(BigArray, FProcList[0], Count);
  // 드라이버 주소의 배열을 얻는다.
  if not EnumDeviceDrivers(@BigArray, SizeOf(BigArray), Count) then
    raise Exception.Create(SFailMessage);
  SetLength(FDrvList, Count div SizeOf(DWORD));
  Move(BigArray, FDrvList[0], Count);
end;

procedure TWinNTInfo.ShowProcessProperties(Cookie: Pointer);
```

```
begin
  ShowProcessDetails(DWORD(Cookie));
end;

end.
```



Ch14\SysInfor\DetailNT.pas

```
unit DetailNT;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
 Dialogs, DetBase, ComCtrls, HeadList;

type
  TListType = (ltModules, ltMemory);

  TWinNTDetailForm = class(TBaseDetailForm)
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure DetailTabsChange(Sender: TObject);
  private
    DetailLists: array[TListType] of TStringList;
    procedure ShowList(ListType: TListType);
  public
    procedure NewProcess(ProcessID: DWORD);
  end;

procedure ShowProcessDetails(ProcessID: DWORD);

implementation

uses PSAPI;
```

```
{$R *.DFM}

const
  TabStrs: array[0..1] of string[7] = ('Modules', 'Memory');

  { 각 목록의 꼬리말에 표시할 문자열들 }
  ACountStrs: array[TListType] of string[31] = (
      'Total Modules: %d', 'Total Pages: %d');

  { 각 목록의 컬럼 제목에 표시할 문자열들 }
  HeaderStrs: array[TListType] of TDetailStrings = (
    ('Module', 'Base Addr', 'Size', 'Entry Point'),
    ('Page Addr', 'Type', 'Mem Map File', ''));

  SCaptionStr  = 'Details for %s';              // 폼의 제목
  SModuleStr   = '%s'#1'$%p'#1'%d bytes'#1'$%p'; // 이름, 주소, 크기, 진입점
  SMemoryStr   = '$%p'#1'%s'#1'%s';       // 주소, 종류, 메모리 맵 파일

procedure ShowProcessDetails(ProcessID: DWORD);
var
  I: Integer;
begin
  with TWinNTDetailForm.Create(Application) do
    try
      for I := Low(TabStrs) to High(TabStrs) do
        DetailTabs.Tabs.Add(TabStrs[I]);
      NewProcess(ProcessID);
      ShowList(ltModules);
      ShowModal;
    finally
      Free;
    end;
end;

function MemoryTypeToString(Value: DWORD): string;
const
  TypeMask = DWORD($0000000F);
begin
  Result := '';
```

```
  case Value and TypeMask of
    1: Result := 'Read-only';
    2: Result := 'Executable';
    4: Result := 'Read/write';
    5: Result := 'Copy on write';
  else
    Result := 'Unknown';
  end;
  if Value and $100 <> 0 then
    Result := Result + ', Shareable';
end;

procedure TWinNTDetailForm.FormCreate(Sender: TObject);
var
  LT: TListType;
begin
  inherited;
  { 목록들을 생성한다. }
  for LT := Low(TListType) to High(TListType) do
    DetailLists[LT] := TStringList.Create;
end;

procedure TWinNTDetailForm.FormDestroy(Sender: TObject);
var
  LT: TListType;
begin
  inherited;
  { 목록들을 해제한다. }
  for LT := Low(TListType) to High(TListType) do
    DetailLists[LT].Free;
end;

procedure TWinNTDetailForm.NewProcess(ProcessID: DWORD);
const
  AddrMask = DWORD($FFFFF000);
var
  I: Integer;
  Count: DWORD;
  ProcHand: THandle;
```

```
  WSPtr: Pointer;
  ModHandles: array[0..$3FFF - 1] of DWORD;
  WorkingSet: array[0..$3FFF - 1] of DWORD;
  ModInfo: TModuleInfo;
  ModName, MapFileName: array[0..MAX_PATH] of char;
begin
  ProcHand := OpenProcess(PROCESS_QUERY_INFORMATION or PROCESS_VM_READ,
     False, ProcessID);
  if ProcHand = 0 then
    raise Exception.Create(
          'No information available for this process/driver');
  try
    EnumProcessModules(ProcHand, @ModHandles, SizeOf(ModHandles), Count);
    for I := 0 to (Count div SizeOf(DWORD)) - 1 do
      if (GetModuleFileNameEx(ProcHand, ModHandles[I], ModName,
        SizeOf(ModName)) > 0) and GetModuleInformation(ProcHand,
         ModHandles[I], @ModInfo, SizeOf(ModInfo)) then
        with ModInfo do
          DetailLists[ltModules].Add(Format(SModuleStr, [ModName,
                      lpBaseOfDll, SizeOfImage, EntryPoint]));
    if QueryWorkingSet(ProcHand, @WorkingSet, SizeOf(WorkingSet)) then
      for I := 1 to WorkingSet[0] do
      begin
        WSPtr := Pointer(WorkingSet[I] and AddrMask);
        GetMappedFileName(ProcHand, WSPtr, MapFileName,
                  SizeOf(MapFileName));
        DetailLists[ltMemory].Add(Format(SMemoryStr, [WSPtr,
          MemoryTypeToString(WorkingSet[I]), MapFileName]));
      end;
  finally
    CloseHandle(ProcHand);
  end;
end;

procedure TWinNTDetailForm.ShowList(ListType: TListType);
var
  I: Integer;
begin
  Screen.Cursor := crHourGlass;
```

```
  try
    with DetailLB do
    begin
      for I := 0 to 3 do
        Sections[I].Text := HeaderStrs[ListType, i];
      Items.Clear;
      Items.Assign(DetailLists[ListType]);
    end;
     DetailSB.Panels[0].Text := Format(ACountStrs[ListType],
       [DetailLists[ListType].Count]);
  finally
    Screen.Cursor := crDefault;
  end;
end;

procedure TWinNTDetailForm.DetailTabsChange(Sender: TObject);
begin
  inherited;
  ShowList(TListType(DetailTabs.TabIndex));
end;

 end.
```

요약

이번 장에서는 델파이로 시스템 정보를 얻는 방법을 설명했다. 윈도우즈 95/98의 ToolHelp32 함수들과 윈도우즈 NT의 PSAPI 함수들을 사용하는 방법을 중점적으로 이야기했으며, 메모리 정보나 환경 변수, 버전 정보 등을 얻을 때 사용하는 Win32 API 함수들도 설명했다. 또한, TListView나 TImageList와 커스텀 컴포넌트인 THeaderListBox, TddgMemView 같은 컴포넌트들을 활용하는 방법도 배울 수 있었을 것이다. 다음 장, "델파이 5로의 이식"은 이전 버전의 델파이 프로젝트를 델파이 5로 옮기는 것에 대해 설명한다.