Белгородский Государственный Технологический Университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №4 по теме: «Архитектура памяти Windows»

Выполнил:  
студент группы ПВ-31  
Адаменко И. И.

Проверил:  
старший преподаватель  
Гарибов А. И.

Белгород  
2015

Цель работы: получение практических навыков по использованию Win32 API для исследования памяти Windows.

## Задание

Разработать ПО для приложения, которое:

1. Выдаёт информацию, получаемую при использовании API GetSystemInfo.
2. Выдаёт информацию, получаемую при использовании API GlobalMemoryStatus. При выводе информации использовать диаграммы.
3. Составляет карту виртуальной памяти для любого процесса.

## Отчёт

### Основные понятия

**Типы памяти**

Физическая память — это реальные микросхемы RAM, установленные в компьютере. Каждый байт физической памяти имеет физический адрес, который представляет собой число от нуля до числа на единицу меньшего, чем количество байтов физической памяти. Физическая память (в отличие от файла подкачки и виртуальной памяти) является исполняемой, то есть памятью, из которой можно читать и в которую центральный процессор может посредством системы команд записывать данные.

Виртуальная память — это просто набор чисел, о которых говорят, как о виртуальных адресах. Программист может использовать виртуальные адреса, но Windows не способна по этим адресам непосредственно обращаться к данным, поскольку такой адрес не является адресом реального физического запоминающего устройства, как в случае физических адресов и адресов файла подкачки. Для того чтобы код с виртуальными адресами можно было выполнить, такие адреса должны быть отображены на физические адреса, по которым действительно могут храниться коды и данные. Эту операцию выполняет диспетчер виртуальной памяти (VMM).

Как известно, наименьший адресуемый блок памяти — байт. Однако самым маленьким блоком памяти, которым оперирует Windows VMM, является страница памяти, называемая также страничным блоком памяти. На компьютерах с процессорами Intel объем страничного блока равен 4 Кб.

Страничный файл, который называется также файлом подкачки в Windows находится на жёстком диске. Он используется для хранения данных и программ точно так же, как и физическая память, но его объем обычно превышает объем физической памяти. Windows использует файл подкачки для хранения информации, которая не помещается в RAM, производя, если нужно, обмен страниц между файлом подкачки и RAM.

**Управление памятью в ОС**

Базовые задачи управления памятью:

1. Отслеживание свободной и занятой памяти.
2. Выделение памяти процессам и её освобождение после завершения процесса.
3. Вытеснение кодов и данных процессов из ОП, когда размер ОП недостаточен для размещения в ней всех процессов, и возвращение их при необходимости.
4. Настраивание адресов программ на конкретную область физ. памяти.

Типы адресов

1. Символьные — имена переменных, которые присваивает пользователь при написании программы.
2. Виртуальные — данные адреса вырабатывает транслятор, присваивая их переменным и командам.
3. Физические — данные адреса соответствуют номерам ячеек ОП, где в действительности расположены переменные и команды в ходе выполнения команд препроцессора.

Адресное пространство процесса

Каждый процесс Win32 получает виртуальное адресное пространство, называемое также адресным пространством, или пространством процесса, объем которого равен 4 Гб. Таким образом, код процесса может ссылаться на адреса с &Н00000000 по &HFFFFFFFF (или с 0 по (4 294 967 295 в десятичной системе счисления)). Конечно, так как виртуальные адреса — это просто числа, заявление о том, что каждый процесс получает своё собственное виртуальное адресное пространство, выглядит довольно бессмысленным. Это все равно, что сказать, что каждый человек получает свой собственный диапазон возраста от 0 до 150.

Распределение виртуальной памяти

Каждая страница виртуального адресного пространства может находиться в одном из трёх состояний:

* Reserved (зарезервирована) — страница зарезервирована для использования;
* Committed (передана) — для данной виртуальной страницы выделена физическая память в файле подкачки или в файле, отображаемом в память;
* Free (свободна) — данная страница не зарезервирована и не передана, и поэтому в данный момент она недоступна для процесса.

Виртуальная память может быть зарезервирована или передана с помощью вызова API-функции VirtualAlloc.

Разделение процедур резервирования и передачи памяти имеет некоторые преимущества. Например, резервирование памяти является очень полезной процедурой с точки зрения практичности. Если приложению требуется большой объем памяти, можно зарезервировать всю память, а выделить только ту часть, которая нужна в данный момент, раздвигая, таким образом, временные рамки более трудоёмкой операции выделения физической памяти.

Windows тоже использует этот подход, когда выделяет память под стек каждого вновь создаваемого потока. Система резервирует 1 Мб виртуальной памяти под стек каждого потока, но выделяет первоначально только две страницы (8 Кб).

Защита памяти

Допустим, некоторая страница физической памяти совместно используется двумя процессами. Если она помечена как «только для чтения», то два процесса без проблем могу совместно пользоваться этой страницей. Однако возможны ситуации, когда каждому процессу требуется разрешить запись в эту память, но без воздействия и другой процесс. После установки защиты «запись копированием» при попытке записи в совместно используемую страницу система создаст её копию специально для процесса, которому нужно осуществить запись. Таким образом, данная страница перестаёт быть совместно используемой, а представление её данных в других процессах остаётся неизменным.

Необходимо отметить, что атрибуты защиты страницы могут быть изменены с помощью API-функции VirtualProtect.

Совместно используемые страницы

Ситуация с совместно используемой физической памятью является значительно более сложной. VMM использует концепцию, называемую прототипированием элементов таблицы страниц. Идея заключается в том, что обычные элементы таблицы каждого из совместно использующих память процессов указывают не на физическую память, а на общий прототип элемента таблицы страниц. А тот, в свою очередь, может ссылаться на совместно используемую физическую память.

Отображения виртуальной памяти

Функция Win32 API VirtualQuery может использоваться для получения информации о состоянии адресов виртуальной памяти. Используется также функция VirtualQueryEx, расширенная версия VirtualQuery, которая позволяет получать информацию о внешних виртуальных адресных пространствах.

### Код программы

|  |  |
| --- | --- |
| 001 | static class win32API |
| 002 | { |
| 003 | [DllImport("kernel32", SetLastError = true)] |
| 004 | public static extern void GetSystemInfo(out SystemInfoStruct lpSystemInfo); |
| 005 |  |
| 006 | public enum ProcessorArchitecture |
| 007 | { |
| 008 | X86 = 0, |
| 009 | X64 = 9, |
| 010 | Arm = -1, |
| 011 | Itanium = 6, |
| 012 | Unknown = 0xFFFF, |
| 013 | } |
| 014 |  |
| 015 | [StructLayout(LayoutKind.Sequential)] |
| 016 | public struct SystemInfoStruct |
| 017 | { |
| 018 | public ProcessorArchitecture ProcessorArchitecture; |
| 019 | public uint PageSize; |
| 020 | public IntPtr MinimumApplicationAddress; |
| 021 | public IntPtr MaximumApplicationAddress; |
| 022 | public IntPtr ActiveProcessorMask; |
| 023 | public uint NumberOfProcessors; |
| 024 | public uint ProcessorType; |
| 025 | public uint AllocationGranularity; |
| 026 | public ushort ProcessorLevel; |
| 027 | public ushort ProcessorRevision; |
| 028 | } |
| 029 |  |
| 030 | [DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)] |
| 031 | [return: MarshalAs(UnmanagedType.Bool)] |
| 032 | public static extern bool GlobalMemoryStatus(ref MEMORYSTATUS lpBuffer); |
| 033 |  |
| 034 | [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)] |
| 035 | public struct MEMORYSTATUS |
| 036 | { |
| 037 | public uint dwLength; |
| 038 | public uint dwMemoryLoad; |
| 039 | public UIntPtr dwTotalPhys; |
| 040 | public UIntPtr dwAvailPhys; |
| 041 | public UIntPtr dwTotalPageFile; |
| 042 | public UIntPtr dwAvailPageFile; |
| 043 | public UIntPtr dwTotalVirtual; |
| 044 | public UIntPtr dwAvailVirtual; |
| 045 | } |
| 046 |  |
| 047 | [DllImport("kernel32.dll")] |
| 048 | public static extern int VirtualQueryEx( |
| 049 | IntPtr handle, |
| 050 | IntPtr adress, |
| 051 | out ProcessQueryInformation processQuery, |
| 052 | uint length |
| 053 | ); |
| 054 |  |
| 055 | public enum ProtectionConstant |
| 056 | { |
| 057 | Execute = 0x10, |
| 058 | ExecuteAndRead = 0x20, |
| 059 | ExecuteAndReadAndWrite = 0x40, |
| 060 | ExecuteWithWriteCopy = 0x80, |
| 061 | NoAccess = 0x01, |
| 062 | ReadOnly = 0x02, |
| 063 | ReadAndWrite = 0x04, |
| 064 | WriteCopy = 0x08, |
| 065 | Guard = 0x100, |
| 066 | NoCache = 0x200, |
| 067 | WriteCombine = 0x400, |
| 068 | } |
| 069 |  |
| 070 | public enum QueryState |
| 071 | { |
| 072 | Commit = 0x1000, |
| 073 | Free = 0x10000, |
| 074 | Reserved = 0x2000, |
| 075 | } |
| 076 |  |
| 077 | public enum MemoryType |
| 078 | { |
| 079 | Image = 0x1000000, |
| 080 | Mapped = 0x40000, |
| 081 | Private = 0x20000, |
| 082 | } |
| 083 |  |
| 084 | [StructLayout(LayoutKind.Sequential)] |
| 085 | public struct ProcessQueryInformation |
| 086 | { |
| 087 | public IntPtr BaseAdress; |
| 088 | public IntPtr AllocationBase; |
| 089 | public ProtectionConstant AllocationProtect; |
| 090 | public uint RegionSize; |
| 091 | public QueryState State; |
| 092 | public ProtectionConstant Protect; |
| 093 | public MemoryType Type; |
| 094 | } |
| 095 |  |
| 096 | [DllImport("kernel32.dll", SetLastError = true)] |
| 097 | public static extern UIntPtr VirtualAlloc(UIntPtr lpAddress, UIntPtr dwSize, |
| 098 | AllocationType flAllocationType, MemoryProtection flProtect); |
| 099 |  |
| 100 | [DllImport("kernel32.dll", SetLastError = true)] |
| 101 | public static extern bool VirtualFree(UIntPtr lpAddress, UIntPtr dwSize, |
| 102 | FreeType dwFreeType); |
| 103 |  |
| 104 | [Flags()] |
| 105 | public enum AllocationType : uint |
| 106 | { |
| 107 | COMMIT = 0x1000, |
| 108 | RESERVE = 0x2000, |
| 109 | RESET = 0x80000, |
| 110 | LARGE\_PAGES = 0x20000000, |
| 111 | PHYSICAL = 0x400000, |
| 112 | TOP\_DOWN = 0x100000, |
| 113 | WRITE\_WATCH = 0x200000 |
| 114 | } |
| 115 |  |
| 116 | [Flags()] |
| 117 | public enum FreeType : uint |
| 118 | { |
| 119 | DECOMMIT = 0x4000, |
| 120 | RELEASE = 0x8000 |
| 121 | } |
| 122 |  |
| 123 | [Flags()] |
| 124 | public enum MemoryProtection : uint |
| 125 | { |
| 126 | EXECUTE = 0x10, |
| 127 | EXECUTE\_READ = 0x20, |
| 128 | EXECUTE\_READWRITE = 0x40, |
| 129 | EXECUTE\_WRITECOPY = 0x80, |
| 130 | NOACCESS = 0x01, |
| 131 | READONLY = 0x02, |
| 132 | READWRITE = 0x04, |
| 133 | WRITECOPY = 0x08, |
| 134 | GUARD\_Modifierflag = 0x100, |
| 135 | NOCACHE\_Modifierflag = 0x200, |
| 136 | WRITECOMBINE\_Modifierflag = 0x400 |
| 137 | } |
| 138 | } |
| 139 |  |
| 140 | class MemoryInfo |
| 141 | { |
| 142 | public uint MemoryLoad { get; private set; } |
| 143 | public UIntPtr TotalPhys { get; private set; } |
| 144 | public UIntPtr AvailPhys { get; private set; } |
| 145 | public UIntPtr TotalPageFile { get; private set; } |
| 146 | public UIntPtr AvailPageFile { get; private set; } |
| 147 | public UIntPtr TotalVirtual { get; private set; } |
| 148 | public UIntPtr AvailVirtual { get; private set; } |
| 149 |  |
| 150 | public uint pageSize { get; set; } |
| 151 | public uint pageLimit { get; set; } |
| 152 | private List<UIntPtr> baseAddresses = new List<UIntPtr>(); |
| 153 |  |
| 154 | public MemoryInfo() { } |
| 155 |  |
| 156 | public void update() |
| 157 | { |
| 158 | var tmp = new win32API.MEMORYSTATUS(); |
| 159 | win32API.GlobalMemoryStatus(ref tmp); |
| 160 |  |
| 161 | MemoryLoad = tmp.dwMemoryLoad; |
| 162 | TotalPhys = tmp.dwTotalPhys; |
| 163 | AvailPhys = tmp.dwAvailPhys; |
| 164 | TotalPageFile = tmp.dwTotalPageFile; |
| 165 | AvailPageFile = tmp.dwAvailPageFile; |
| 166 | TotalVirtual = tmp.dwTotalVirtual; |
| 167 | AvailVirtual = tmp.dwAvailVirtual; |
| 168 | } |
| 169 |  |
| 170 | public void VirtualAlloc() |
| 171 | { |
| 172 | if (pageSize > 0 && pageLimit > 0) |
| 173 | { |
| 174 | UIntPtr address = win32API.VirtualAlloc( |
| 175 | UIntPtr.Zero, |
| 176 | new UIntPtr(pageSize \* pageLimit), |
| 177 | win32API.AllocationType.RESERVE | win32API.AllocationType.COMMIT, |
| 178 | win32API.MemoryProtection.NOACCESS |
| 179 | ); |
| 180 |  |
| 181 | if (address != UIntPtr.Zero) |
| 182 | { |
| 183 | baseAddresses.Add(address); |
| 184 | } |
| 185 | } |
| 186 | } |
| 187 |  |
| 188 | public void VirtualFree() |
| 189 | { |
| 190 | if (baseAddresses.Count > 0) |
| 191 | { |
| 192 | UIntPtr adress = baseAddresses[0]; |
| 193 | baseAddresses.RemoveAt(0); |
| 194 |  |
| 195 | win32API.VirtualFree( |
| 196 | adress, |
| 197 | UIntPtr.Zero, |
| 198 | win32API.FreeType.RELEASE |
| 199 | ); |
| 200 | } |
| 201 | } |
| 202 | } |

### Скриншоты приложения

### 

### 

