# 实验一 进程的创建和撤销

# 专业 软件工程 姓名 毛济洲 学号 20182344050

## 一．实验目的

(l)了解Windows内存管理机制，理解页式存储管理技术。

(2)熟悉Windows内存管理基本数据结构。

(3)掌握Windows内存管理基本API的使用。

## 二．实验内容

使用Windows系统提供的函数和数据结构显示系统存储空间的使用情况，当内存和虚拟存储空间变化时，观察系统显示变化情况

**三、实验准备知识**

1.相关系统数据结构说明

系统结构MEMORYSTATUS中包含当前物理内存和虚拟内存信息，使用函数GlobalMemoryStatus()可以将这些信息存储在结构MEMORYATATUS中。

结构原型

Typedef struct\_MEMORYSTATUS{

DWORD dwLength;

DWORD dwMwmoryLoad;

DWORD dwTotalPhys；

DWORD dwTotalPageFile;

DWORD dwAvailVirtual;

DWORD dwTotalVirtual;

DWORD dwAvailVirtual;

} MEMORYSTATUS , \*LPMEMORYSTATUS;

参数说明

(1) dwLength: MEMORYSTATUS数据结构的大小,单位为字节。

(2)dwMemoryLoad:当前内存利用率,取值范围为0~100%,0表示内存没有被使用，

100%表示内存全部被使用。

(3) dwTotalPhys:物理内存的总字节数。

(4) dwAvailPhys:可用物理内存的字节数。

(5)dwTotalPageFile:页文件的总字节数。页文件是虚拟内存系统占用的磁盘空间。

(6) dwAvailPageFile:页文件中可用字节数。

(7)dwTotalVirtual:用户模式下调用进程可以访问的虚拟地址空间总字节数。

(8)dwAvailVirtual:用户模式下调用进程虚拟地址空间中未提交和未保留的内存总

字节数,即可用虚拟地址空间大小。

2.相关API函数介绍

(1)获取系统物理内存和虚拟内存使用信息。

原型 VOID GlobalMemoryStatus(

LPMEMORYSTATUS lpBuffer

);

参数说明:

lpBuffer:指向MEMORYSTATUS数据结构的指针，函数GlobalMemoryStatus( )将

内存的当前信息存储在该结构中。

返回值:

该参数没有返回值。

(2)保留或提交某--段虚拟地址空间。函数VirtualAlloe()可以在调用进程的虚拟地

址空间中保留或提交若干页面。保留意味着这段虚拟地址不能被使用,当提交时,这段虚拟

地址才真正被分配给该进程。

原型:

LPVOID VirtualAlloc (

VOID lpAddress,

DWORD dwSize,

DWORD flAllocat ionType,

DWORD flProtect

);

参数说明:

①lpAddress:待分配空间的起始位置。若该值为NULL,系统将为其分配一个合适的

起始地址，否则用户要指定一个准确的起始地址。

②dwSize:待分配空间的大小。如果参数lpAddress不为NULL,则待分配空间范围

为lpAddress~ IpAddress + dwSize。

③flAllocationType: 分配类型,可以为表3-1所示标志的任意组合。

④flProtect: 指定存取保护的类型。若虚拟地址空间已经被提交,则在指定下面任何

-个属性时要随同PAGE\_ GUARD(页保护)和PAGE\_ NOCACHE(页无缓存)这两个属

性。存取保护位的类型如表3-2所示。

返回值:

如果丽数调用成功,则返回值为已分配虚拟地址空间的起始地址。如果丽数调用失败，

则返回值为NULL。若要得到更多的错误信息,可调用GetLastError()函数。

(3)释放或注销某--段虚拟地址空间。丽数VirtualFree()用于释放或注销某一段虚拟

地址空间。

原型:

BOOL VirtualFree (

LPVOID lpAddress,

DWORD dwSize,

DWORD dwE reeType

);

参数说明:

1. lpAddress:待释放空间的起始位置。如果dwFreeType值为MEM\_ RELEASE, 该

参数必须使用VirtualAlloc()丽数返回的地址。

②dwSize:待释放空间的大小。如果dwFreeType值为MEM\_ RELEASE, 该参数必

须为O,否则待释放空间的范围为lpAddress~lpAddress+dwSize。

③dwFreeType:释放类型。它可以为表3-3所示标志的任意组合。

返回值:

如果丽数调用成功,则返回值为非零。

如果函数调用失败.则返回值为零。若要得到更多的错误信息,可调用GetI astError()函数。

(4)分配内存空间。

原型:

void \* malloc(size\_ t size);

参数说明:

size:要分配内存大小,单位为B(字节)。

返回值:

该函数返回分配内存空间void类型的指针。如果丽数返回NULL,说明没有有效的内

存空间可供分配。

(5)释放内存空间。

原型:

void free(void \* memblock);

参数说明:

\*memblock:要释放的内存地址。

返回值:

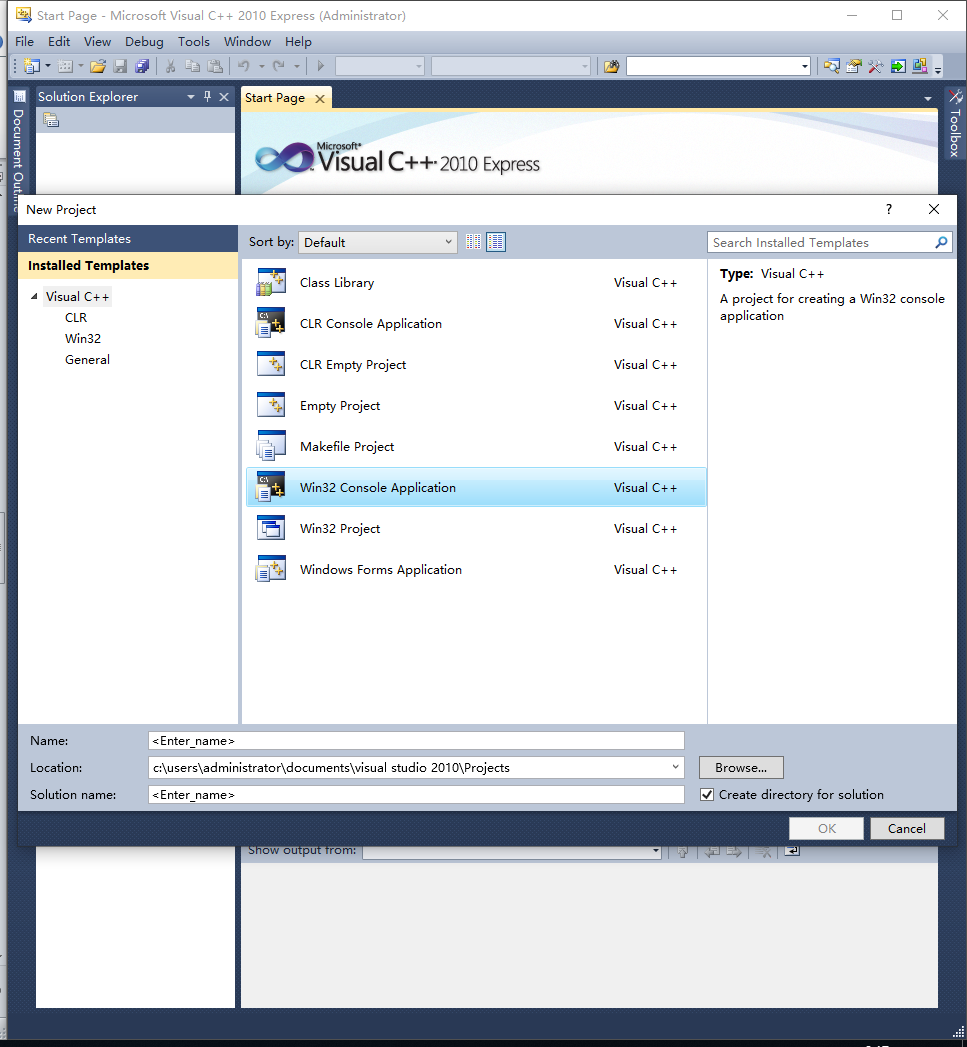
无。

**四、实验要求**

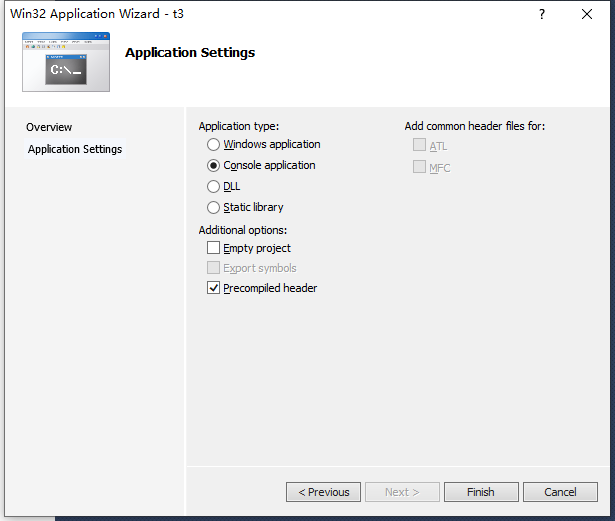
能正确使用系统丽数GlobalMemoryStatus()和数据结构MEMORYSTATUS了解系统内存和虚拟存储空间使用情况，会使用VirtualAlloc() 丽数和VirtualFree()丽 数分配和释放虚拟存储空间。

## 五．实验步骤

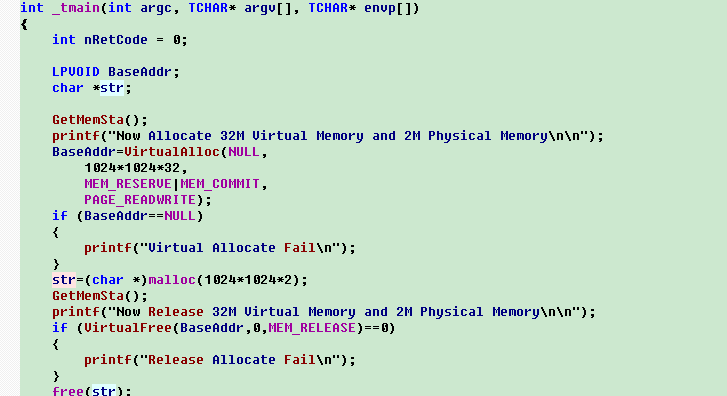
1. 启动安装好的Visual C++ 6.0。
2. 选择File->New，新建Win32 Console Application程序, 由于内存分配、释放及系统存储空间使用情况均是Microsoft Windows操作系统的系统调用，因此选择An application that support MFC。单击确定按钮，完成本次创建。

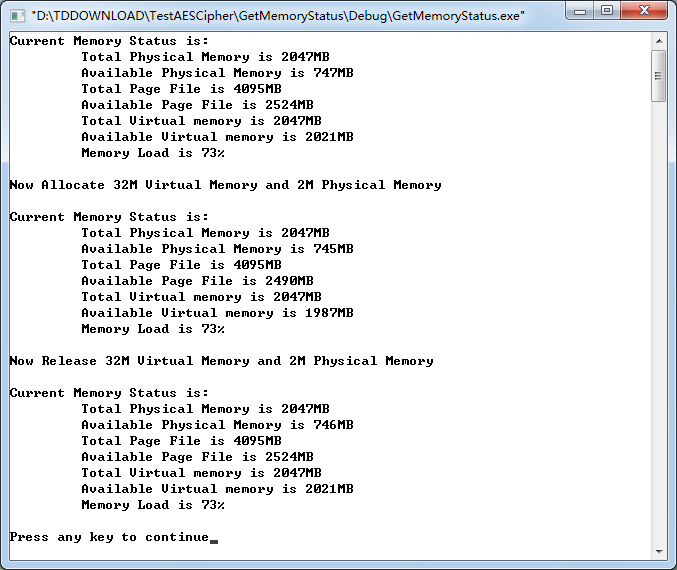


1. 创建一个支持MFC的工程，单击完成。



1. 打开编辑环境后，编辑程序，并且编译、链接并运行该程序。





**六、分析与结论**

该程序完成了内存空间的显示、申请及释放。开始时，可用物理内存为747M，可用页文件大小为2524M，可用虚拟内存为2021M。使用函数VirtualAlloca()和malloc()分配了32M虚拟内存和2M物理内存后，系统显示可用物理内存为745M，少了2M，可用页文件大小为2490M，少了34M，可用虚拟内存为1987M，少了32M，当使用了VirtualAlloc()和free()函数后，只有可用物理内存没有达到预定的747M，只为746M，少了1M，其他恢复申请前状况。当多次运行程序后发现，可用物理内存有时候有可以恢复，所以得出与系统有关系，不是程序的关系。

**七、源码**

// GetMemoryStatus.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include "GetMemoryStatus.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#undef THIS\_FILE

static char THIS\_FILE[] = \_\_FILE\_\_;

#endif

/////////////////////////////////////////////////////////////////////////////

// The one and only application object

CWinApp theApp;

using namespace std;

void GetMemSta(void);

int \_tmain(int argc, TCHAR\* argv[], TCHAR\* envp[])

{

int nRetCode = 0;

LPVOID BaseAddr;

char \*str;

GetMemSta();

printf("Now Allocate 32M Virtual Memory and 2M Physical Memory\n\n");

BaseAddr=VirtualAlloc(NULL,

1024\*1024\*32,

MEM\_RESERVE|MEM\_COMMIT,

PAGE\_READWRITE);

if (BaseAddr==NULL)

{

printf("Virtual Allocate Fail\n");

}

str=(char \*)malloc(1024\*1024\*2);

GetMemSta();

printf("Now Release 32M Virtual Memory and 2M Physical Memory\n\n");

if (VirtualFree(BaseAddr,0,MEM\_RELEASE)==0)

{

printf("Release Allocate Fail\n");

}

free(str);

GetMemSta();

return nRetCode;

}

void GetMemSta()

{

MEMORYSTATUS MemInfo;

GlobalMemoryStatus(&MemInfo);

printf("Current Memory Status is:\n");

printf("\t Total Physical Memory is %dMB\n",MemInfo.dwTotalPhys/(1024\*1024));

printf("\t Available Physical Memory is %dMB\n",MemInfo.dwAvailPhys/(1024\*1024));

printf("\t Total Page File is %dMB\n",MemInfo.dwTotalPageFile/(1024\*1024));

printf("\t Available Page File is %dMB\n",MemInfo.dwAvailPageFile/(1024\*1024));

printf("\t Total Virtual memory is %dMB\n",MemInfo.dwTotalVirtual/(1024\*1024));

printf("\t Available Virtual memory is %dMB\n",MemInfo.dwAvailVirtual/(1024\*1024));

printf("\t Memory Load is %d% % \n\n",MemInfo.dwMemoryLoad);

}