

Thực hành hệ điều hành 1

Prerequisites: Cơ bản về Window OS, C Programming.

Phần thực hành:

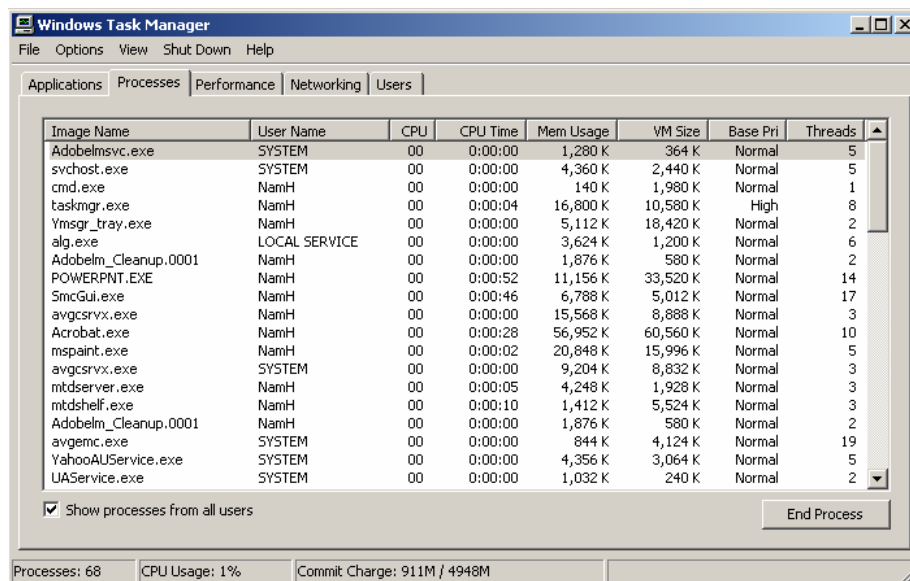
Bài 1:

Tìm hiểu hệ điều hành Window thông qua các công cụ hệ thống của Microsoft: QuickSlice, Process Explorer, CPU Stress, Task Manager ...

Mục đích: Giúp sinh viên nắm bắt các hiểu biết cơ bản khi một ứng dụng được thực thi trong môi trường window. Các thông tin mà sinh viên cần quan sát khi 1 ứng dụng thực thi như: *Process, Thread, CPU Usage, Memory Usage, Privileged & User Time*.

a./ Chạy chương trình Task Manager và quan sát các thông tin trên tab Applications & Processes

Task Manager là chương trình quản lý hệ thống của window, cung cấp các thông tin về chương trình & tiến trình đang thực thi cũng như thông tin đo lường hiệu suất của hiện hành của hệ thống.



- Khởi tạo bằng 1 trong các cách:
 - > Nhấn tổ hợp phím *Ctrl-Alt-Del*.
 - > Right click mouse trên thanh *Taskbar*-> *Task Manager*
 - > *Start* -> *Run* -> *taskmgr.exe*
- *Tab Applications*: sẽ liệt kê các chương trình ứng dụng window đang chạy có cửa sổ window tương tác với người sử dụng. Trạng thái running nghĩa là nó đang đợi một thông điệp window. Right click vào 1 chương trình trên *tab Applications* -> "*Go to process*" để đi đến *tab Processes* cho chương trình này.

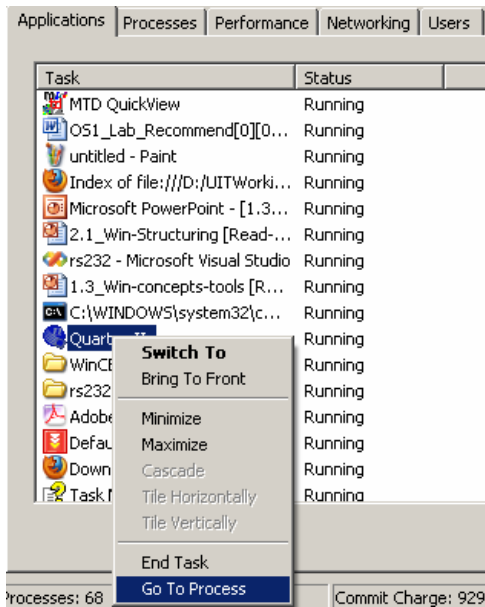
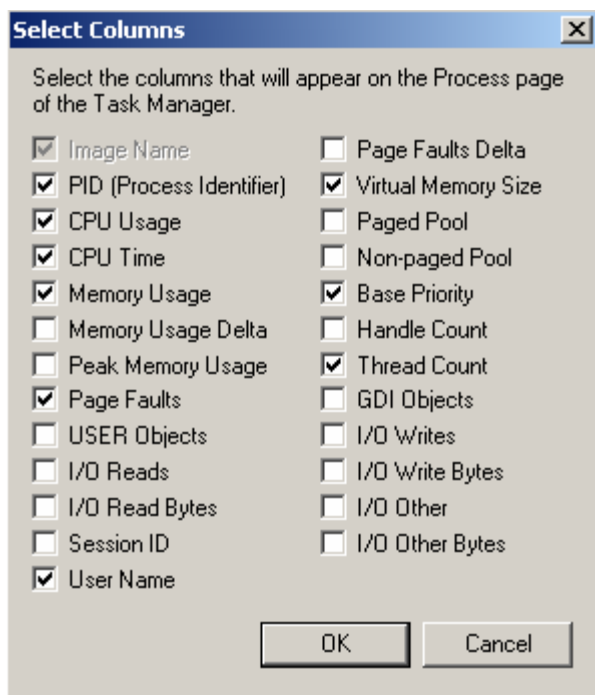
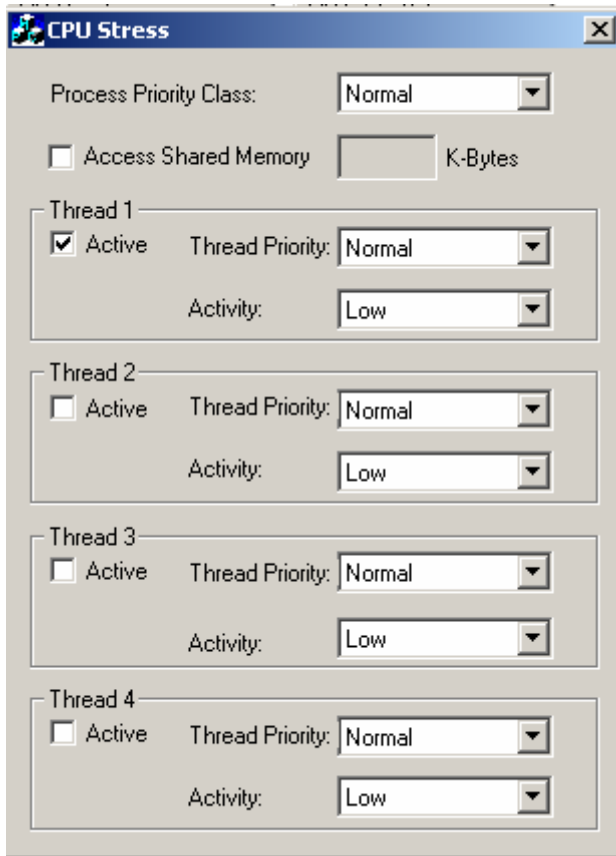


Image Name	User Name	CPU	Private Bytes
utility.exe	NamH	00	0:0
EnergyCut.exe	NamH	00	0:0
jqs.exe	SYSTEM	00	0:0
svchost.exe	LOCAL SERVICE	00	0:0
svchost.exe	NETWORK SE...	00	0:0
Smc.exe	SYSTEM	00	0:0
jtagserver.exe	SYSTEM	00	0:0
avgwdsvc.exe	SYSTEM	00	0:0
firefox.exe	NamH	00	0:0
ApntEx.exe	NamH	00	0:0
explorer.exe	NamH	00	0:0
S24EvMon.exe	SYSTEM	00	0:0
flashget.exe	NamH	00	0:0
EvtEng.exe	SYSTEM	00	0:0
svchost.exe	SYSTEM	00	0:0
quartus.exe	NamH	00	0:0
svchost.exe	NETWORK SE...	00	0:0
svchost.exe	LOCAL SERVICE	00	0:0
UniKeyNT.exe	NamH	00	0:0

- *Tab Processes*: Hiển thị thông tin các tiến trình đang chạy. Để hiển thị các thông tin khác cho các tiến trình, chọn *View-> Select Columns*.

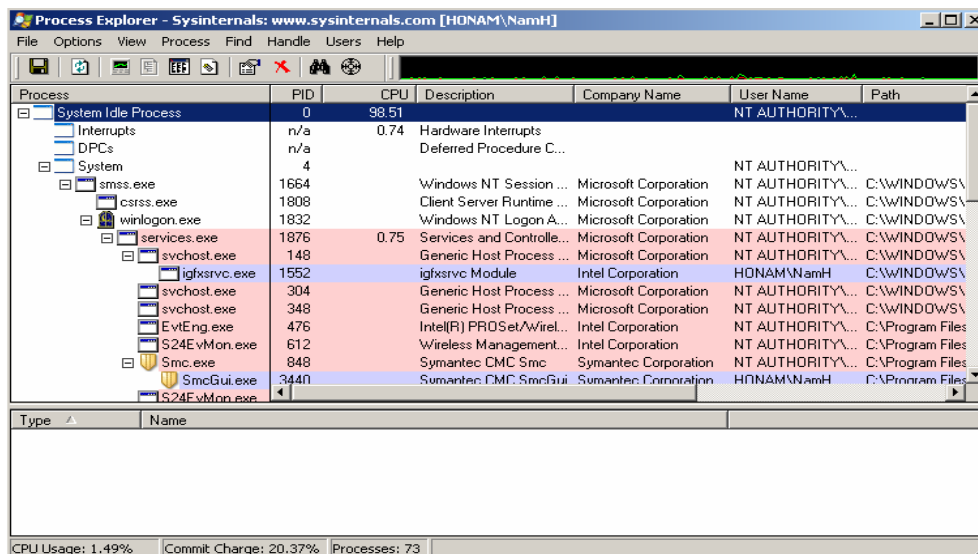


- *Thực hành*: Sinh viên chọn *File->New Task(Run)* rồi trở đến thư mục chứa chương trình *CPUSTRES.exe*. Thay đổi các thông tin trên chương trình *CPUSTRES* rồi quan sát các thông số *PID*, *CPU*, *CPU Time*, *Mem Usage*, *Page Fault*, *Base Pri*, *Threads* trên tab *Processes*. So sánh nó với tiến trình *System Idle Process*. Sau đó kill tiến trình *CPUSTRES* này.

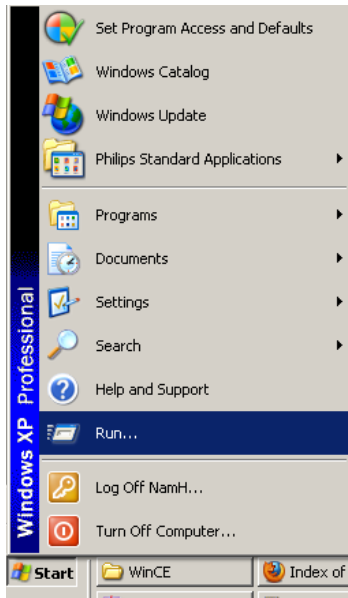


b/ Chạy chương trình Process Explorer và quan sát các thông tin hiển thị. So sánh với Task Manager.

Process Explorer là chương trình quản lý hệ thống nâng cao, cung cấp các thông tin về tiến trình tương tự như Task Managers. Có ưu điểm hơn khi nó cung cấp chi tiết các thông tin về tiến trình, chẳng hạn nó hiển thị cho thấy được mối quan hệ cha con của các tiến trình



- Khởi tạo: Vào thư mục chứa file procexp.exe, chạy chương trình procexp.exe.
- Chọn *View->Select Columns* và chọn các thông số tương tự như Task Manager. So sánh cách tổ chức thông tin các tiến trình với Task Manager.
- Chạy chương trình command line của window bằng cách *Start->Run*. Gõ lệnh *cmd*.



- Chạy chương trình *CPUSTRES.EXE* từ thư mục chứa nó trong cửa sổ command line này.

```

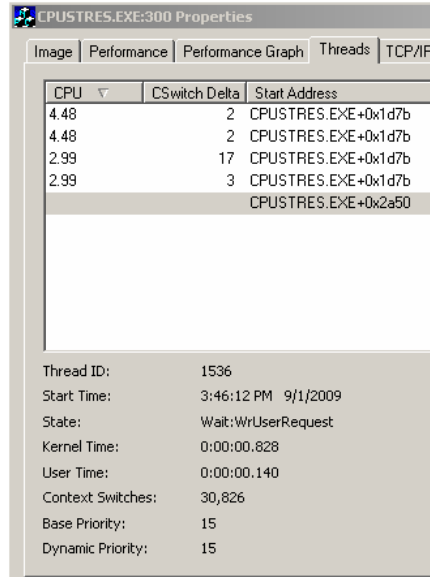
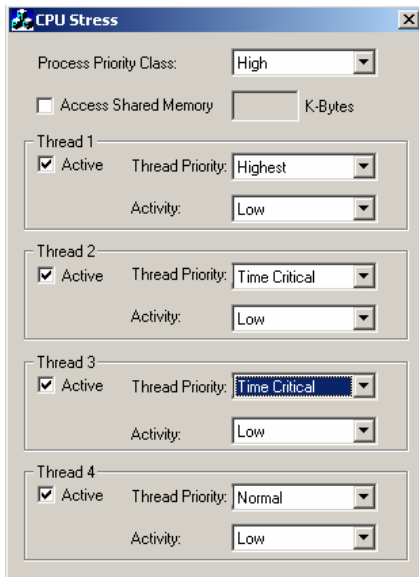
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\NamH>D:
D:\>cd D:\UITWorking\OSLab_reskit
D:\UITWorking\OSLab_reskit>ls
CPUSTRES.EXE  INSTSRU.EXE  SRUANY.EXE  depends.cnt  pvview.exe
EXETYPE.EXE  LEAKYAPP.EXE  SRUINSTW.EXE  depends.dll  pvviewer.exe
EXETYPE.INI  MLTITHRD.EXE  UADUMP.EXE  depends.exe
IMAGECFG.EXE  QSLICE.EXE  depends.GID  depends.hlp

D:\UITWorking\OSLab_reskit>CPUSTRES.EXE

```

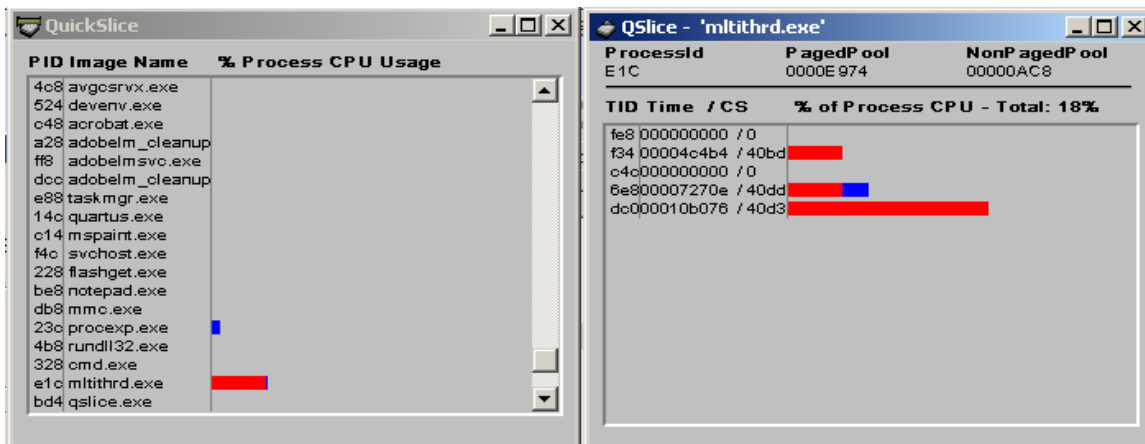
- Quan sát trên cửa sổ *Process Explorer* sẽ thấy tiến trình *CPUSTRES.EXE* xuất hiện là tiến trình con của tiến trình cha *cmd.exe*.
- Double click vào *CPUSTRES.EXE* trên cửa sổ *Process* của *Process Explorer*, quan sát các thông tin mà nó hiển thị cho tiến trình này như: *Performance*, *Thread* khi cho phép các thread (chọn active) chạy cũng như thay đổi thông số priority cho các thread này trên cửa sổ của *CPUSTRES*. Quan sát các thông tin trên tab *Performance* trong trường hợp kill, suspend thread.



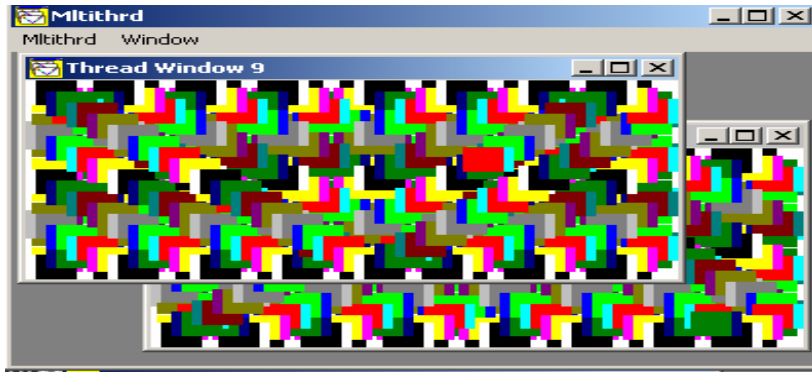
- *Thực hành:* Sinh viên lập lại các thao tác trên với chương trình *MLTITHRD.EXE* trong thư mục. Sau đó hãy thử suspend tiến trình này rồi quay lại xem chương trình *MLTITHRD.EXE* có còn chạy nữa không.

c./ Quan sát các tiến trình đang chạy với công cụ QuickSlice

- QuickSlice là chương trình hiển thị thông tin sử dụng tài nguyên CPU của các tiến trình.
- Chạy bằng cách double click vào *QSLICE.EXE* từ thư mục.
- Chạy chương trình *MLTITHRD.EXE*, *QuickSlice* sẽ hiển thị thông tin tiến trình này trên cửa sổ của nó. Double click vào nó trên *QuickSlice* để quan sát các thông tin sử dụng tài nguyên của các thread.
- Thử các thao tác suspend, kill thread trên *Process Explorer* và quan sát trên *QuickSlice*.



(Màu đỏ: % CPU Usage khi tiến trình thực thi trong kernel mode, màu xanh: % CPU Usage cho user mode)



Tài liệu tham khảo thêm:

- Nguồn các tool tại: www.sysinternals.com
- Mark E. Russinovich and David A. Solomon, *Microsoft Windows Internals*, 4th Edition, Microsoft Press.

Bài 2:

Tìm hiểu lập trình C trên môi trường window bằng Visual Studio.

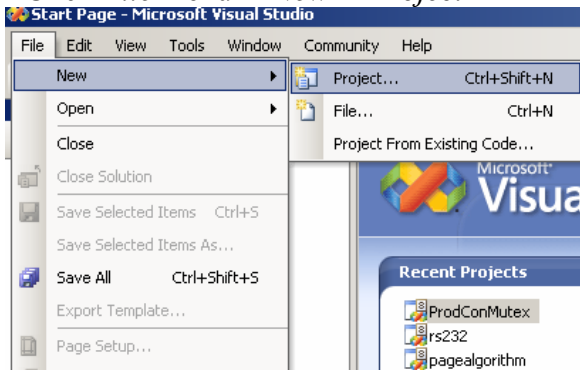
Mục đích: Giúp sinh viên làm quen với môi trường lập trình C bằng Visual Studio.

- Tạo một project cho console application
- Lập trình các chương trình bằng C.
- Hiểu các cách build/debug một chương trình.

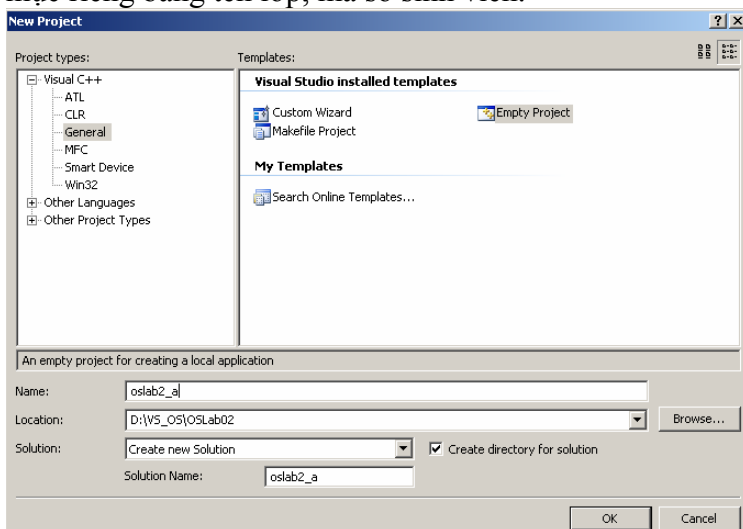
Visual Studio là bộ công cụ IDE (Integrated Development Environment) của Microsoft hỗ trợ viết, biên tập, kiểm soát mã chương trình cho những người phát triển ứng dụng trên windows. Trong bài thực hành này sinh viên chủ yếu tập trung vào cách tạo một project cho lập trình ngôn ngữ C đối với ứng dụng console, và làm quen với cách debug lỗi.

a./ Tạo một project cho một chương trình C trên Visual Studio

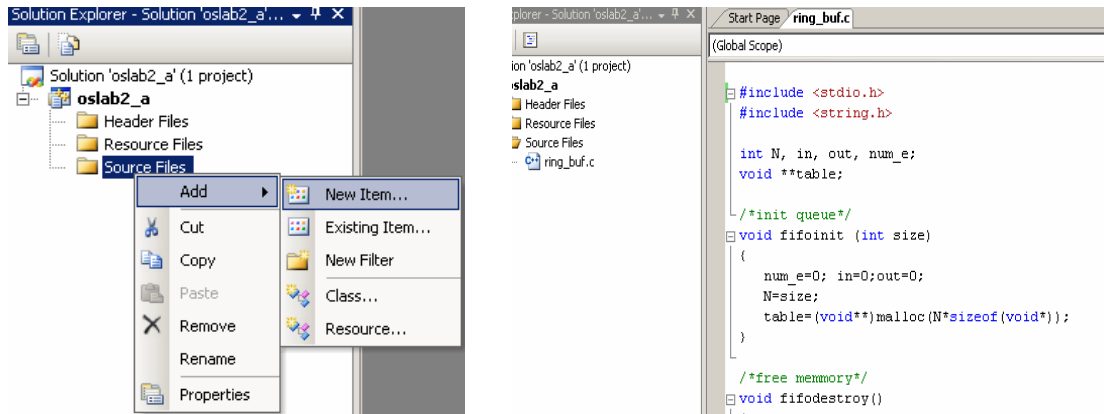
- Click *File* menu-> *New*-> *Project*



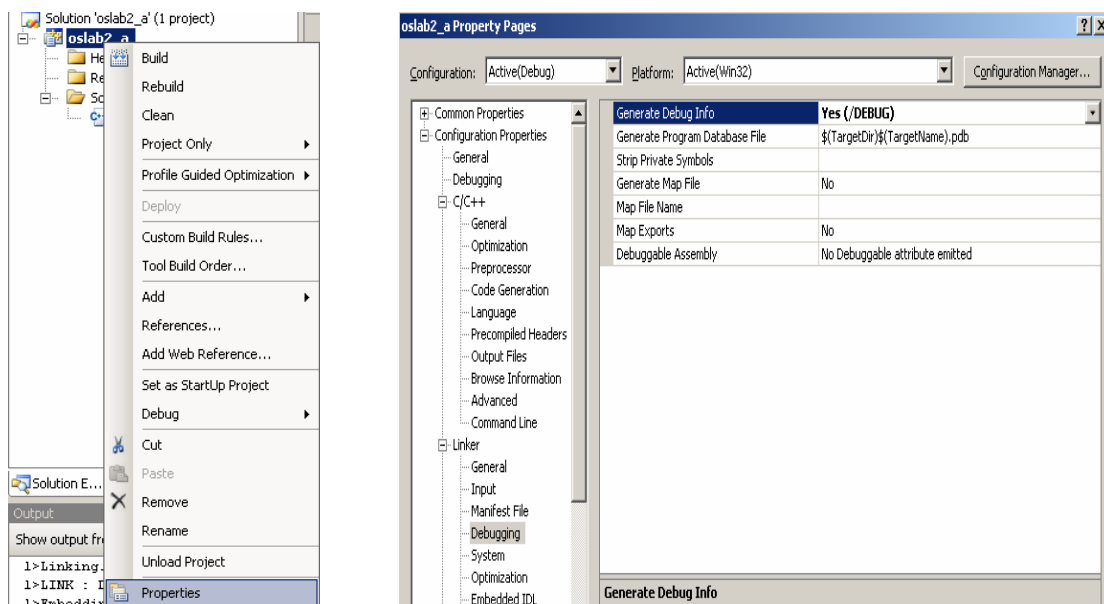
- Chọn Project types là *General*. *Templates* là *Empty Project*. Sau đó đặt tên cho bài lab này(ex: oslab2_a) và chứa trong thư mục của sinh viên. Mỗi sinh viên nên có một thư mục riêng bằng tên lớp, mã số sinh viên.



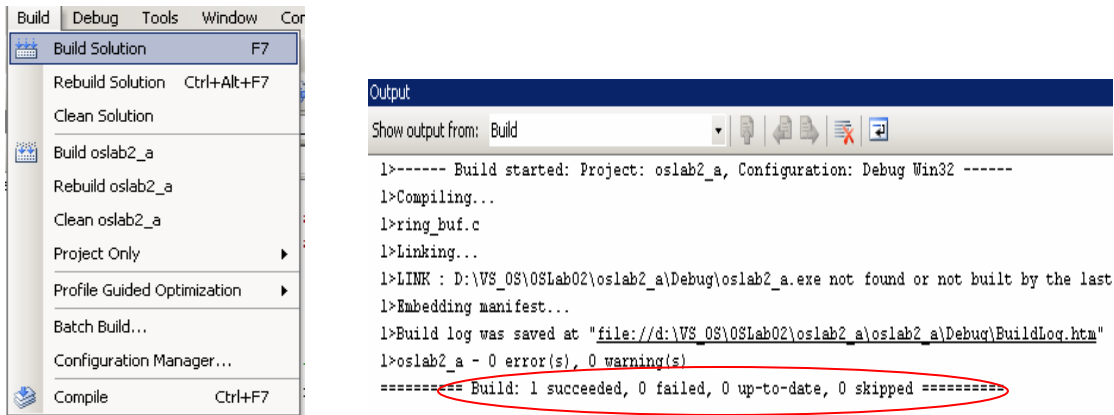
- Tạo một file nguồn C bằng cách: Trên *Solution Explorer*, right click *Source Files* -> *Add* -> *New Item* để tạo file mới hoặc *Add*-> *Existing Item* để lấy 1 file có sẵn đưa vào. Chép đoạn mã nguồn hiện thực một ring buffer từ file *ring_buf.c* save lại với tên file *ring_buf.c* với thao tác *Add* -> *New Item* hoặc thêm vào project trực tiếp file này với thao tác *Add*-> *Existing Item*.



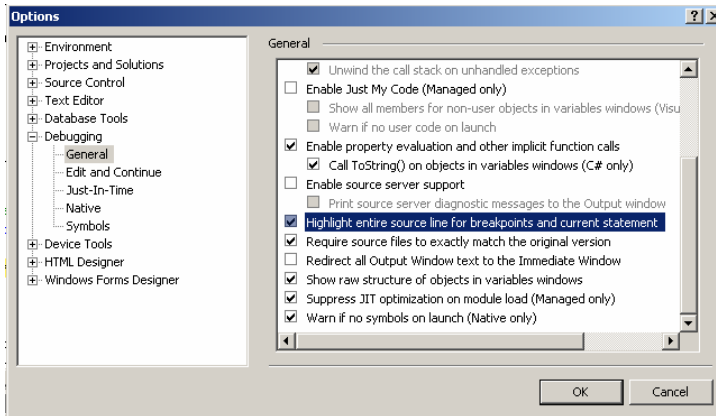
- Cấu hình chế độ gỡ rối (debug mode): Right click vào **Project** -> **Properties**.
 - o Trong tab C/C++ chọn “General”, phần “Debug Information Format” chọn là “C7 Compatible (/Z7)”.
 - o Trong tab C/C++ chọn “Advanced”, phần “Compile As” chọn là “Compile as C Code (/TC)”
 - o Trong tab “Linker”: chọn Debugging, phần “Generate Debug Info” chọn là “Yes (/DEBUG)”.
 - o Trong tab C/C++ chọn “Optimization”, phần “Optimization” chọn là “Disabled(/Od)”.



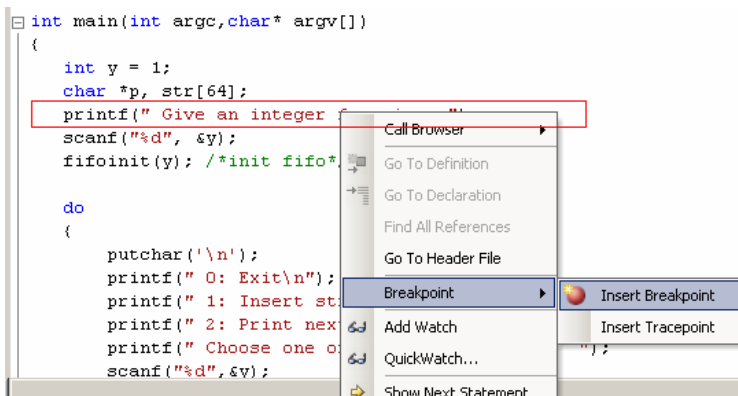
- Build Project bằng cách: *Build*->*Build Solution*. Quan sát thông tin trên cửa sổ *output*.



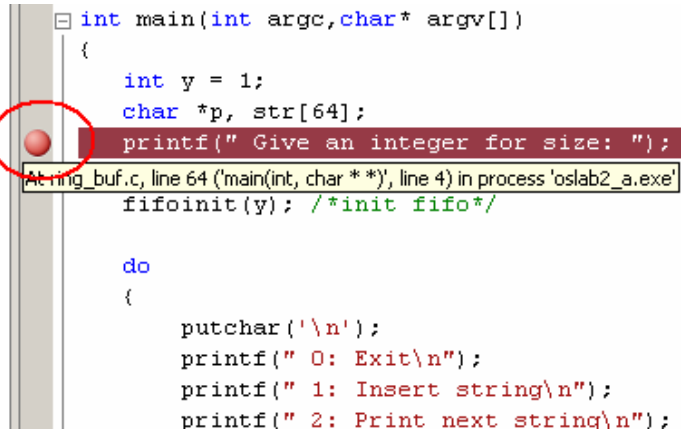
- Chọn *Tools -> Options*. Sau đó trong *Debugging* chọn *General*, chọn phần “*Highlight entire source line for breakpoints and current statement*”.



- Bắt đầu chạy chương trình: *Debug-> Start Debugging*. Quan sát chương trình được thực thi thế nào. Chạy chương trình *Process Explorer* để quan sát chương trình này.
- Bật *Breakpoint* để chạy từng bước: Có 2 cách:
 - o Chọn 1 dòng nào đó trong đoạn code trong hàm **main()**, *right click -> Breakpoint-> Insert Breakpoint*.



- Hoặc là click chuột trái tại điểm đầu dòng bên trái của đoạn code.



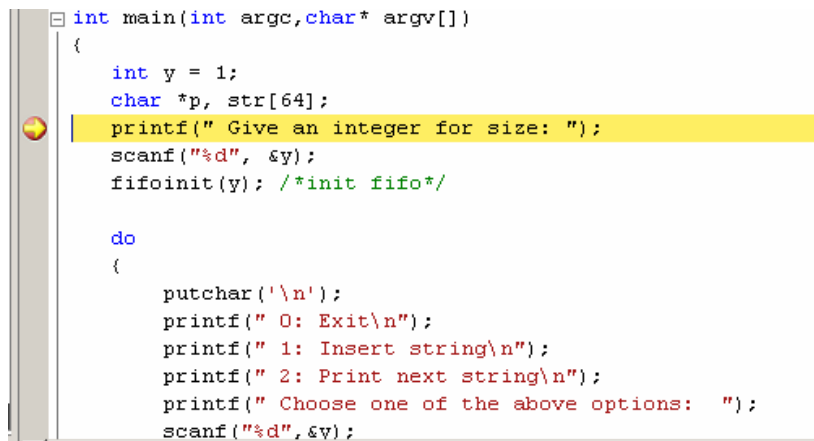
```

int main(int argc, char* argv[])
{
    int y = 1;
    char *p, str[64];
    printf(" Give an integer for size: ");
    scanf("%d", &y);
    fifoinit(y); /*init fifo*/

    do
    {
        putchar('\n');
        printf(" 0: Exit\n");
        printf(" 1: Insert string\n");
        printf(" 2: Print next string\n");
    }
}

```

- Sau đó thử thử chạy debug chương trình. Visual Studio sẽ dừng ngay tại điểm vừa đặt breakpoint.



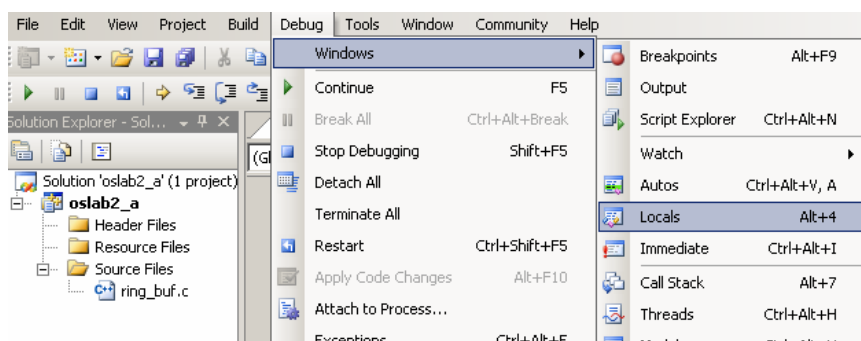
```

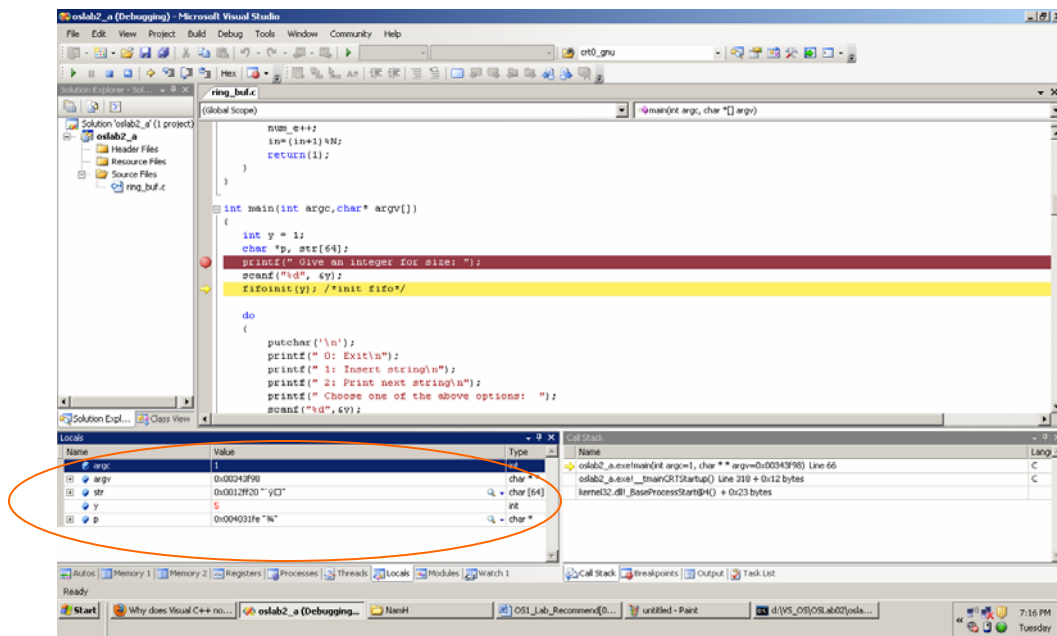
int main(int argc, char* argv[])
{
    int y = 1;
    char *p, str[64];
    printf(" Give an integer for size: ");
    scanf("%d", &y);
    fifoinit(y); /*init fifo*/

    do
    {
        putchar('\n');
        printf(" 0: Exit\n");
        printf(" 1: Insert string\n");
        printf(" 2: Print next string\n");
        printf(" Choose one of the above options: ");
        scanf("%d", &y);
    }
}

```

- Trong quá trình debug từng bước, sinh viên quan sát các biến cục bộ được thay đổi như thế nào khi chọn cửa sổ *Debug -> Windows -> Locals*.

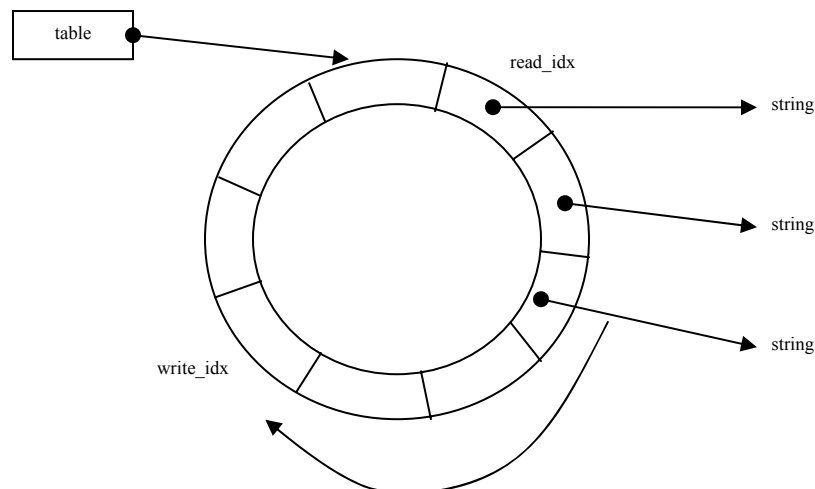




- Sinh viên có thể chọn thêm *Debug -> Windows -> Watch* để xem thêm các biến toàn cục bằng cách gõ các tên biến này vào cửa sổ Watch.
- Hãy đặt breakpoint tại các dòng code trong các hàm *fifoput*, *fifoget* để quan sát các biến thay đổi như thế nào trong quá trình chạy từng bước để hiểu rõ cấu trúc dữ liệu của ring buffer này.

- Thực hành:

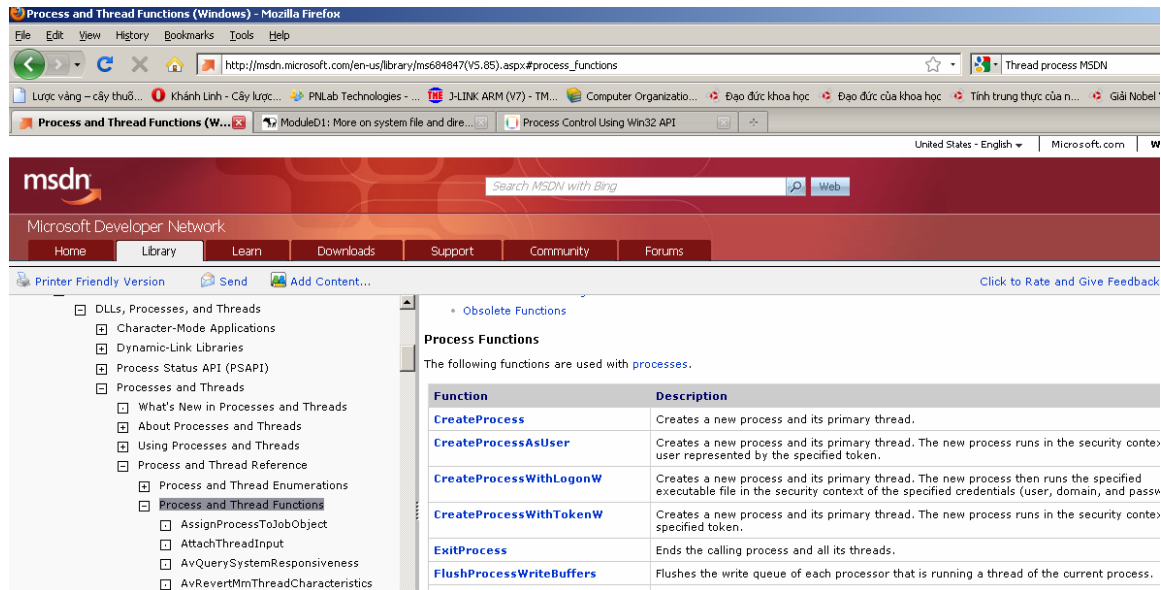
Cấu trúc dữ liệu ring buffer như sau: table là 1 bảng chứa các con trỏ chuỗi, số phần tử sẽ tăng dần khi có 1 thao tác put chuỗi vào table, ngược lại khi có 1 thao tác get, số phần tử sẽ giảm dần:



- Đoạn code trên hiện giờ chỉ hiện thực được thao tác insert một phần tử (hàm *fifoput(void *next)*) vào table của ring buffer. Sinh viên hãy hiện thực thao tác get (hàm *fifoget()*) trả về một chuỗi string là một phần tử trong table.

b./ Lập trình với process trên window

Trong phần này sinh viên tìm hiểu thử lập trình tạo một process bằng cách sử dụng API của window. Tài liệu các hàm API quản lý process/ thread tham khảo ở thư viện MSDN của Microsoft.



Bản thân chương trình bắt đầu hàm main() khi được biên dịch đã là một tiến trình. Một số hàm cơ bản về quản lý tiến trình của windows như sau:

```
DWORD GetCurrentProcessId(void);
```

Trả về định danh hiện hành của tiến trình.

```
BOOL CreateProcess(  
    LPCTSTR lpApplicationName,  
    LPCTSTR lpCommandLine,  
    LPSECURITY_ATTRIBUTES lpProcessAttributes,  
    LPSECURITY_ATTRIBUTES lpThreadAttributes,  
    BOOL bInheritHandles,  
    DWORD dwCreationFlags,  
    LPVOID lpEnvironment,  
    LPCTSTR lpCurrentDirectory,  
    LPSTARTUPINFO lpStartupInfo,  
    LPPROCESS_INFORMATION lpProcessInformation  
);
```

Tạo một tiến trình (tiến trình con) từ tiến trình hiện hành tại đường dẫn của thông số lpApplicationName.

```

BOOL TerminateProcess(
    HANDLE hProcess,
    UINT uExitCode
);

```

Kết thúc một tiến trình con và tất cả các thread của nó.

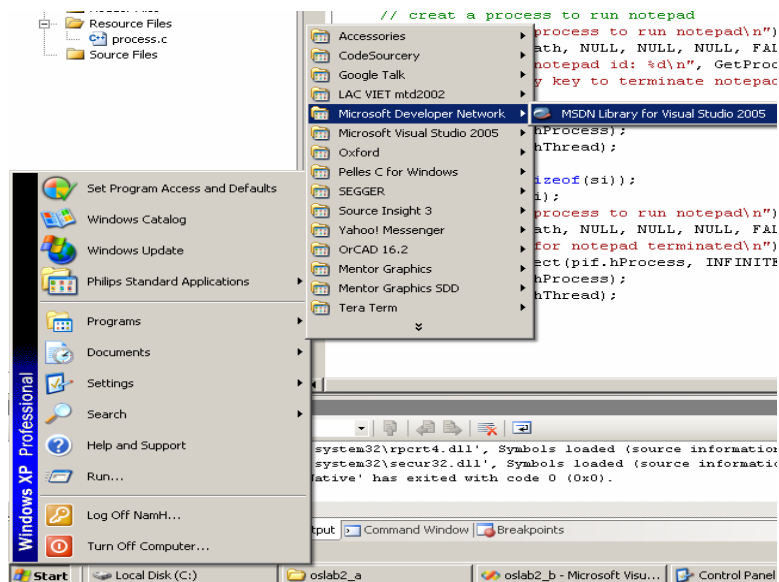
```

DWORD WaitForSingleObject(
    HANDLE hHandle,
    DWORD dwMilliseconds
);

```

Hàm này chờ một tín hiệu từ một đối tượng được chỉ rõ trong hHandle. Có thể dùng hàm này để chờ một tiến trình con kết thúc từ tiến trình cha.

Thực hành: Sinh viên tạo 1 project cho đoạn source code sau. Trong quá trình debug, chạy chương trình *Process Explorer* để quan sát quá trình khởi tạo các tiến trình. Trong quá trình lập trình, sử dụng thư viện MSDN, được cài sẵn trong máy để xem mô tả các hàm hệ thống của window.



```

#include <stdio.h>
#include <windows.h>

char path[] = "C:\\WINDOWS\\system32\\notepad.exe" ;

int main()
{
    PROCESS_INFORMATION pif;
    STARTUPINFO si;

```

```

printf("Current Process ID = %d\n", GetCurrentProcessId());

ZeroMemory(&si, sizeof(si));
si.cb = sizeof(si);

// creat a process to run notepad
printf("Creat a process to run notepad\n");
CreateProcess( path, NULL, NULL, NULL, FALSE, 0, NULL, NULL, &si,
&pif);
printf("Press any key to terminate notepad ...\n");
getch();
TerminateProcess(pif.hProcess, 0);
CloseHandle(pif.hProcess);
CloseHandle(pif.hThread);

ZeroMemory(&si, sizeof(si));
si.cb = sizeof(si);
printf("Creat a process to run notepad\n");
CreateProcess( path, NULL, NULL, NULL, FALSE, 0, NULL, NULL, &si,
&pif);
printf("Waiting for notepad terminated\n");
WaitForSingleObject(pif.hProcess, INFINITE);
CloseHandle(pif.hProcess);
CloseHandle(pif.hThread);
return 0;
}

```

- Sinh viên thêm vào các đoạn code để lấy thông tin về *Process ID*, *Priority Class* của tiến trình notepad bằng cách sử dụng các hàm sau. So sánh thông tin này với thông tin có được từ *Process Explorer* hoặc *Task Manager* khi quan sát tiến trình notepad. Sau đó hãy cài đặt (setting) Priority cho tiến trình notepad này là `HIGH_PRIORITY_CLASS`. Hãy quan sát sự thay đổi này trên Process Explorer và Task Manager.

```

DWORD GetPriorityClass(
    HANDLE hProcess
);

DWORD GetProcessId(
    HANDLE Process
);

BOOL SetPriorityClass(
    HANDLE hProcess,
    DWORD dwPriorityClass
);

```

- Để hiểu rõ priority trong window như thế nào, tham khảo sách *Microsoft Windows Internals trong thư mục bài thực hành buổi 1 tại trang 327*.

c./ Lập trình với thread trong window

Một số hàm cơ bản về thread của windows:

```
HANDLE CreateThread(  
    LPSECURITY_ATTRIBUTES lpThreadAttributes,  
    SIZE_T dwStackSize,  
    LPTHREAD_START_ROUTINE lpStartAddress,  
    LPVOID lpParameter,  
    DWORD dwCreationFlags,  
    LPDWORD lpThreadId  
);
```

Tạo một thread từ không gian địa chỉ ảo của tiến trình hiện hành.

```
DWORD GetThreadId(  
    HANDLE Thread  
);
```

Trả về định danh cho thread.

```
int GetThreadPriority(  
    HANDLE hThread  
);
```

Trả về độ ưu tiên định thời cho thread.

```
DWORD SuspendThread(  
    HANDLE hThread  
);
```

Suspend một thread

```
DWORD ResumeThread(  
    HANDLE hThread  
);
```

Resume một thread

```
BOOL TerminateThread(  
    HANDLE hThread,  
    DWORD dwExitCode  
);
```

Kết thúc một thread

Thực hành: Sinh viên tạo một project trên Visual Studio cho đoạn source code sau. Debug và quan sát trên *Process Explorer*.

```
#include <stdio.h>
#include <windows.h>

DWORD WINAPI PrintThreads (LPVOID);

int main ()
{
    HANDLE hThread;
    DWORD dwThreadID;
    int i;
    for (i=0; i<5; i++)
    {
        hThread=CreateThread(
            NULL, //default security attributes
            0, //default stack size
            PrintThreads, //function name
            (LPVOID)i, // parameter
            0, // start the thread immediately
            &dwThreadID
        );
        printf("ThreadID = %d\n", dwThreadID);
        if (hThread)
        {
            printf ("Thread launched successfully, hThread
= %d\n", hThread);
            //CloseHandle (hThread);
        }
        Sleep (1000);
        getch();
        return (0);
    }
    //function PrintThreads
    DWORD WINAPI PrintThreads (LPVOID num)
    {
        while (1)
        {
            printf ("Thread Number is %d%d%d\n", num,num,num);
            printf("Thread Id = %d is running\n",GetCurrentThreadId());
            Sleep(1000);
        }
        return 0;
    }
}
```

- Sinh viên hãy thêm vào các đoạn code làm các công việc sau:

- Đối với mỗi thread đang chạy, hãy in ra thread priority của nó.
- Thread chính trong hàm main sẽ đợi người dùng nhập 1 thread number vào, và cho phép người dùng suspend, resume, terminal, hoặc thay đổi priority của 1 thread nào đó.
- Quan sát chương trình của mình trên Process Explorer hoặc Task Manager.

Bài 3:

Tìm hiểu lập trình Process, Thread trong môi trường window.

Mục đích: Hiểu rõ các giải pháp giải quyết tranh chấp, đồng bộ, giao tiếp giữa các process/thread, các phương pháp định thời. Tìm hiểu các hàm API hỗ trợ của window.

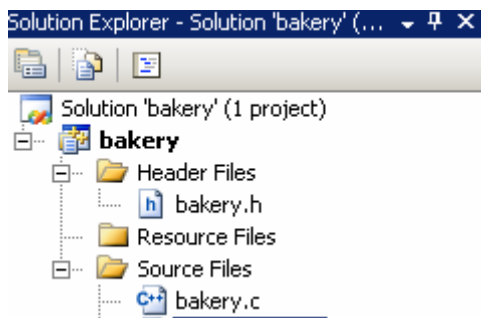
- Xem xét các phương pháp giải quyết vấn đề tranh chấp-đồng bộ (Bakery Algorithm).
- Hiểu các hàm đồng bộ do window cung cấp như: semaphore, mutex thông qua bài toán Producer & Consumer.
- Hiểu các phương pháp giao tiếp giữa các process. Tìm hiểu cơ chế Pipe thông qua chương trình Talk Application.
- Tìm hiểu giải thuật định thời thông qua chương trình mô phỏng schedsim

a./ Giải thuật tranh chấp – đồng bộ

Thực hành: Giải thuật Bakery được cung cấp với source code *bakery.c*, *bakery.h*. Sinh viên tạo một project **hiện thực** sử dụng giải thuật này trong việc giải quyết tranh chấp giữa các thread. Thread thứ nhất sẽ tăng một biến count, 2 thread còn lại sẽ giảm biến count này. So sánh nó với trường hợp không sử dụng giải thuật tranh chấp này.

```
Thread function
{
    Do {
        // Xin vào vùng tranh chấp
        EnterCriticalSection();
        If thread id là 1 then
            Tăng biến count
        Else
            Giảm biến count
        // Trả lại vùng tranh chấp
        LeaveCriticalSection();
    While (1)
}
```

Source code được thêm vào như sau:



b./ Bài toán Producer & Consumer.

Window cung cấp một số hàm hỗ trợ đồng bộ giữa các thread như mutex, semaphore:

```
HANDLE CreateMutex(
```

```

LPSECURITY_ATTRIBUTES lpMutexAttributes,

BOOL bInitialOwner,

LPCTSTR lpName

);

```

Tạo mutex.

```

BOOL ReleaseMutex(

HANDLE hMutex

);

```

Nhả mutex

```

HANDLE CreateSemaphore(

LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,

LONG lInitialCount,

LONG lMaximumCount,

LPCTSTR lpName

);

```

Tạo một semaphore

```

BOOL ReleaseSemaphore(

HANDLE hSemaphore,

LONG lReleaseCount,

LPLONG lpPreviousCount

);

```

Nhả semaphore

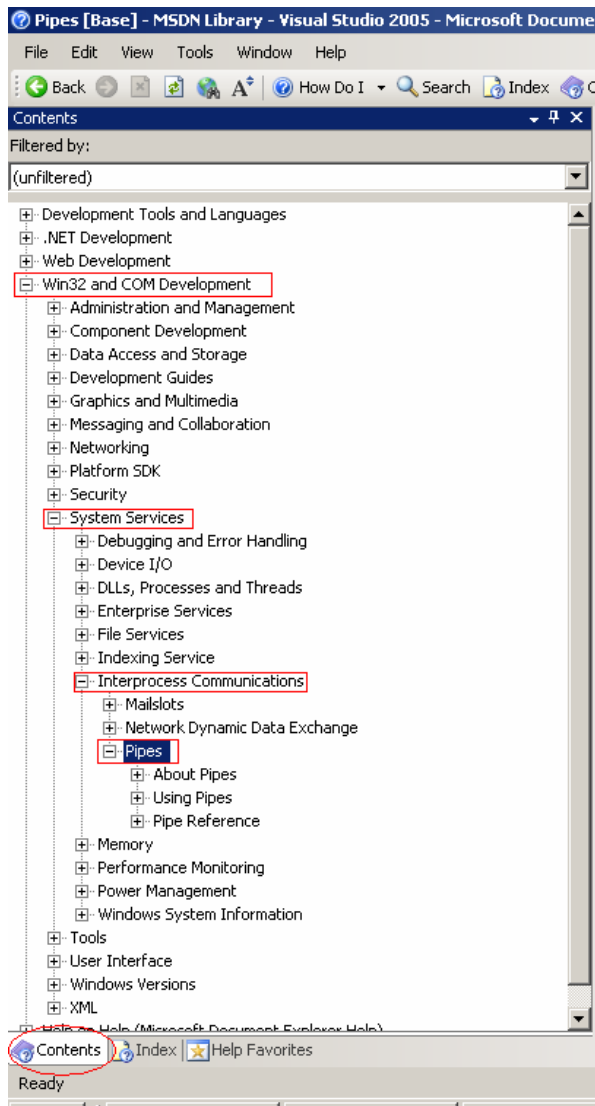
Thực hành:

- Sinh viên tạo một project cho bài toán *producer & consumer* với source code được cung cấp trong file *prodcon_mutex.c*. Sau đó hãy thử dùng **semaphore** thay vì mutex.
- Trong source code này thay vì chỉ việc tăng biến count, hãy áp dụng bài toán này trong **trường hợp sử dụng tranh chấp ring buffer** ở bài thực hành 2.

c./ Bài toán giao tiếp giữa các process.

Windows cung cấp cơ chế pipe hỗ trợ giao tiếp giữa các process. Sinh viên tìm hiểu cơ chế này trong tài liệu MSDN và chương trình *Talk Application* bằng cách build **2 project riêng lẻ trên Visual Studio** với source code được cung cấp trong thư mục *TalkClient* và *TalkServer*. 1 project cho TalkServer và 1 project cho TalkClient.

Sinh viên tham khảo mô tả pipe trong MSDN.



- Một hàm main nếu được khai báo như sau:

```
main (int argc, char * argv[])
```

Nó sẽ cho phép lấy các thông số từ command line. argc cho biết có bao nhiêu thông số truyền vào, argv[] là 1 mảng chứa các thông số nào. Nếu 1 chương trình khi được dịch ra với tên file là abc.exe, thì nếu từ command line chúng ta gõ:

D:\UITFolder\abc.exe a1 a2 a3

Các thông số sẽ được truyền vào hàm main là

argc = 4

argv[0] = abc.exe

argv[1] = a1

argv[2] = a2

argv[3] = a3

- Trong ví dụ Talk Application, chương trình TalkServer sẽ cần 1 thông số là <tên pipe> mà người sử dụng sẽ nhập vào dùng để tạo 1 kênh giao tiếp với chương trình TalkClient.

- Trong chương trình TalkClient, người dùng cần nhập vào 2 thông số <tên pipe>, <địa chỉ ip của host mà chương trình TalkServer đang chạy>.

- Sau khi dịch xong, 2 chương trình TalkServer, TalkClient sẽ nằm trong thư mục project tại thư mục \debug với tên là TalkServer.exe, TalkClient.exe. Từ command line, để chạy nó ta làm như sau:

Chạy chương trình TalkServer:

```
D:\>cd US_OS\OSLab03\TalkApp\TalkServer\debug\
D:\US_OS\OSLab03\TalkApp\TalkServer\debug>dir
Volume in drive D has no label.
Volume Serial Number is 98D4-546B

Directory of D:\US_OS\OSLab03\TalkApp\TalkServer\debug

08/11/2009  11:10 AM    <DIR>          .
08/11/2009  11:10 AM    <DIR>          ..
08/11/2009  11:10 AM                155,648 TalkServer.exe
08/11/2009  11:10 AM                655,636 TalkServer.ilc
08/11/2009  11:10 AM                945,152 TalkServer.pdb
               3 File(s)                1,756,436 bytes
               2 Dir(s)  45,137,018,880 bytes free

D:\US_OS\OSLab03\TalkApp\TalkServer\debug>TalkServer.exe server_pipe
Server is awaiting next request.
```

Chạy chương trình Client:

```
D:\US_OS\OSLab03\TalkApp\TalkClient\debug>dir
Volume in drive D has no label.
Volume Serial Number is 98D4-546B

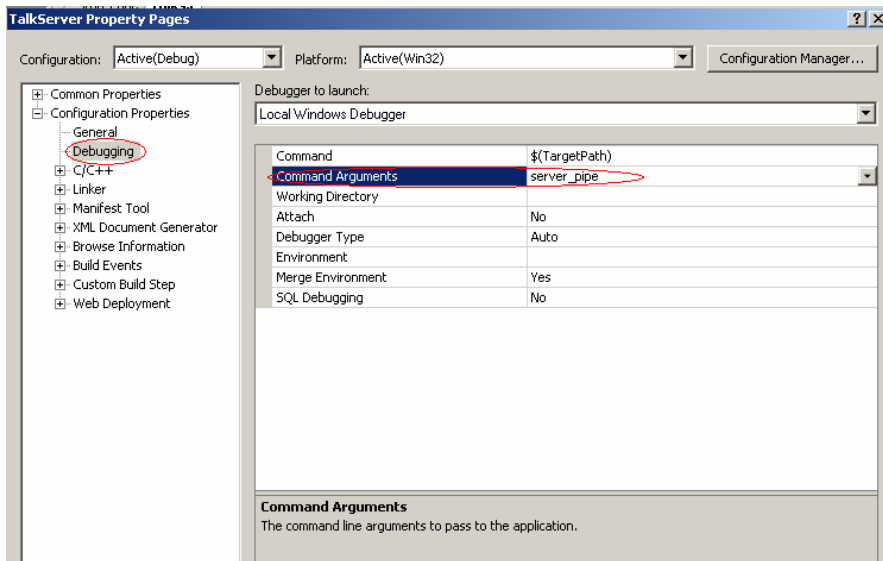
Directory of D:\US_OS\OSLab03\TalkApp\TalkClient\debug

08/11/2009  11:02 AM    <DIR>          .
08/11/2009  11:02 AM    <DIR>          ..
08/11/2009  11:02 AM                155,648 TalkClient.exe
08/11/2009  11:02 AM                650,492 TalkClient.ilc
08/11/2009  11:02 AM                945,152 TalkClient.pdb
               3 File(s)                1,751,292 bytes
               2 Dir(s)  45,137,018,880 bytes free

D:\US_OS\OSLab03\TalkApp\TalkClient\debug>TalkClient.exe server_pipe 192.168.1.3
```

Chú ý: tên pipe của server và client phải trùng nhau.

Trong ví dụ mẫu này, để truyền được các thông số argv[] trong hàm main trong quá trình debug, sinh viên cần cấu hình thêm trong phần Project Properties cho cả TalkServer và TalkClient.



Set breakpoint, debug để hiểu rõ chương trình trên.

d/. Giải thuật định thời

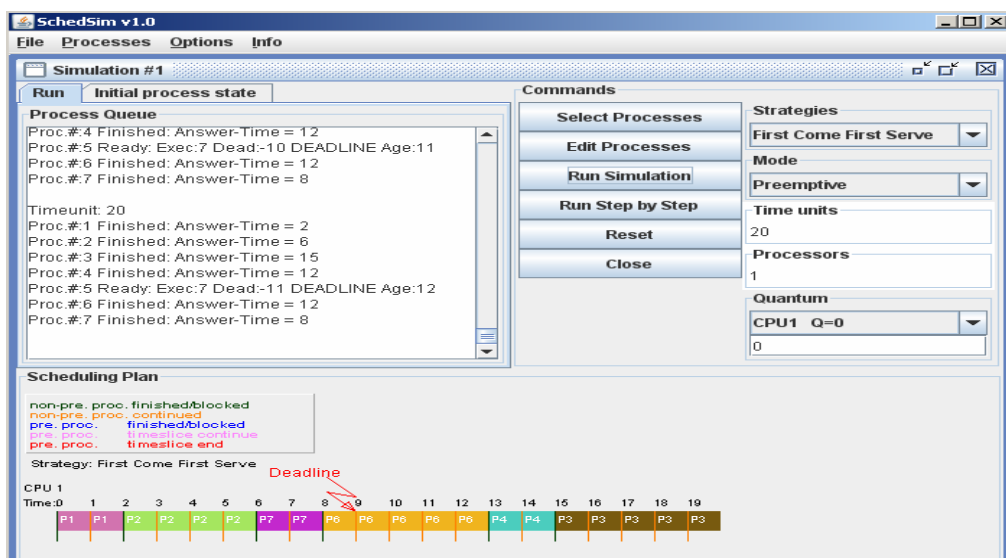
Sinh viên thực hành tìm hiểu các giải thuật định thời thông qua chương trình mô phỏng *schedsim* được cung cấp.

Cài đặt:

- Cài java runtime environment. Chạy file jre-6u16-windows-i586-s.exe

Thực hành :

- Load 1 file chứa cấu hình cho các tiến trình trong thư mục chứa tool schedsim *schedsim_example_processes.txt* bằng cách chọn Select Processes->Load Processes-> *schedsim_example_processes.txt*.
- Chạy từng bước đối với từng giải thuật và so sánh kết quả.



Bài 4:

Tìm hiểu cơ chế quản lý bộ nhớ trong hệ điều hành.

Mục đích: Chạy các chương trình mô phỏng, quan sát để hiểu rõ các giải thuật thay thế trang *FIFO*, *LRU*, *OPT* cũng như cách ánh xạ từ trang nhớ ảo sang khung nhớ vật lý.

- Sinh viên chạy chương trình mô phỏng, xem xét các giải thuật thay thế trang diễn ra như thế nào, các page fault, TLB hit, TLB miss đối với từng giải thuật.
- Quan sát cách ánh xạ từ trang ảo của tiến trình sang khung nhớ vật lý.
- Lập trình với các thư viện quản lý bộ nhớ trong windows

a./ Các giải thuật thay thế trang

Sinh viên thực hành trực tiếp bằng cách chạy file *RepPolicies/default.html* trong thư mục thực hành. Chương trình mô phỏng sẽ giả lập các giải thuật thay thế trang cơ bản của hệ điều hành.

[Help](#)

Check ALL the Policies to Simulate

Random ☐ FIFO ☒

Ideal LRU ☒ Ideal LFU ☐

Optimal ☒ ARC ☐

CAR ☐ LRU-K ☐

Enter Value for K

Number of Page Frames

Enter Page Sequences (space between each)

Choose Display Option

	3	2	1	0	3	
FIFO	0	3	3	3	3	3
	1		2	2	2	2
	2			1	1	1
	3				0	0
Hit/Miss	Cold Start ...	Cold Start ...	Cold Start ...	Cold Start ...	Hit	Hi
ILRU	0	3	3	3	3	3
	1		2	2	2	2
	2			1	1	1
	3				0	0
Hit/Miss	Cold Start ...	Cold Start ...	Cold Start ...	Cold Start ...	Hit	Hi
OPT	0	3	3	3	3	3
	1		2	2	2	2
	2			1	1	1
	3				0	0
Hit/Miss	Cold Start ...	Cold Start ...	Cold Start ...	Cold Start ...	Hit	Hi

FIFO
Number of Page Frames=4
Number of References=18
Number of Hits=3

ILRU
Number of Page Frames=4
Number of References=18
Number of Hits=6

OPT
Number of Page Frames=4
Number of References=18
Number of Hits=11

Thực hành: Sinh viên chọn các giải thuật FIFO, LRU, OPT rồi chạy thử với 1 dãy trang tương ứng.

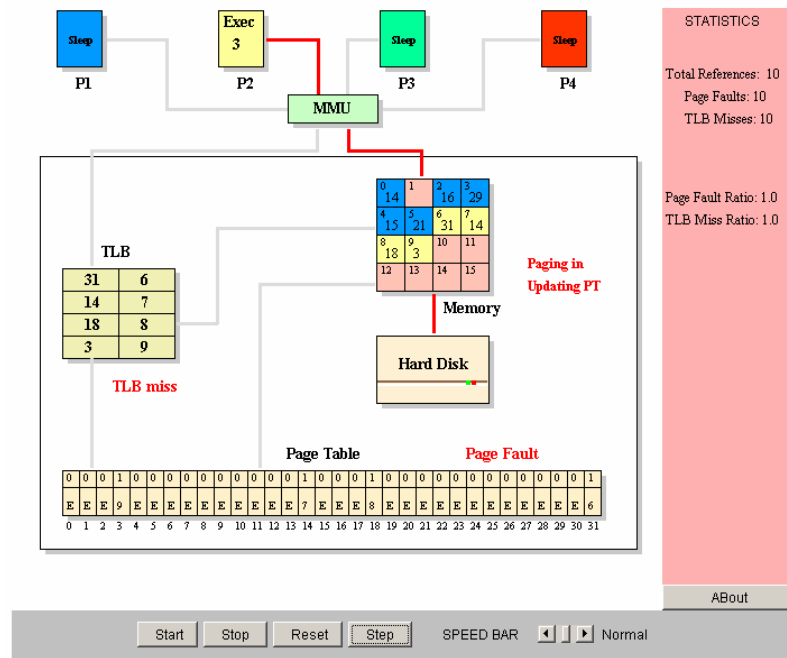
- Giả sử ta có 1 dãy trang tham khảo là: 0 1 2 1 3 0 1 2 0 1 0 2 3. Chọn số page frame tương ứng là 1, 2, 3, 4 rồi chọn với từng giải thuật FIFO, LRU, OPT, chạy từng bước để quan sát bằng cách nhấn vào nút **"Step"**. Sau đó chạy toàn bộ bằng cách nhấn **"Final"**. Hãy cho biết số page fault đối với mỗi trường hợp ? Có nhận xét gì khi số page frame tăng lên? Giải thuật nào là tốt nhất ? Có vấn đề gì không khi hiện thực trong thực tế đối với các giải thuật này?

- Giả sử ta có 1 dãy trang tham khảo là : 3 2 1 0 3 2 4 3 2 1 0 4 2 3 2 1 0 4. Trong trường hợp page frame là 3 hãy cho biết page fault đối từng giải thuật FIFO, LRU, OPT.

- Khi page frame là 4 có xảy ra trường hợp bất thường Belady đối với giải thuật FIFO không ? Đối với các giải thuật khác thì sao ?

b./ Bộ nhớ ảo

Sinh viên thực hành trực tiếp bằng cách chạy file *vmsim/vm.html* trong thư mục thực hành. Vmsim là chương trình mô phỏng bộ nhớ ảo của hệ điều hành. Mặc dù nó không thể hiện tường minh các giải thuật thay thế trang, tuy nhiên nó giúp sinh viên thấy được trực quan hình ảnh quá trình ảnh xạ từ một trang bộ nhớ ảo của tiến trình qua khung của bộ nhớ vật lý.



- P1, P2, P3, P4 là 4 tiến trình mà hệ điều hành đang quản lý trong chương trình mô phỏng. Các trạng thái của tiến trình là thực thi (exec), ngủ (sleep).
- MMU là phần cứng của chip vì xử lý hỗ trợ quản lý bộ nhớ.
- Statistics: các thống kê cho thấy các thông số như: tổng số lần tham khảo bộ nhớ, Page Faults, TLB Misses.
- Memory được mô phỏng 16 frame.
- TLB gồm 4 entry.

Thực hành: Chạy từng bước bằng các nhấn vào nút “**Step**”. Sinh viên quan sát và trả lời các câu hỏi sau:

- Hãy quan sát tiến trình nào đang thực thi, tiến trình nào đang ngủ?
- Đối với tiến trình đang được thực thi, quá trình cập nhật trên vùng nhớ vật lý, TLB diễn ra như thế nào?
- Khi nào Page Fault diễn ra ?
- Khi nào TLB Miss/TLB Hit diễn ra ?
- Thông tin cập nhật trên Page Table cho biết điều gì?
- Khi quá trình định thời diễn ra, sẽ xảy ra quá trình chuyển trạng thái từ tiến trình này sang tiến trình khác, lúc này bảng TLB có thay đổi gì không ?

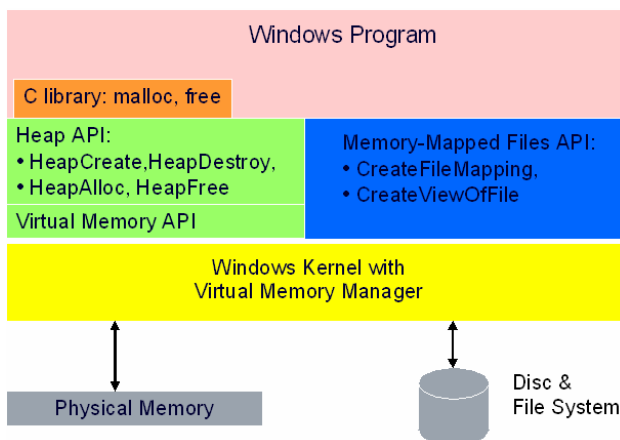
- Lập bảng thông kê khoảng 30 bước chạy, đối với mỗi tiến trình những trang nào được tham khảo và được cấp cho frame nào? Page Fault, TLB Miss/Hit là bao nhiêu?

c./ Quản lý bộ nhớ trong windows

- Windows 32 bits chia không gian địa chỉ 4GB thành 2 vùng. Vùng 2GB từ 0x00000000 – 0x7FFFFFFF dùng chung cho tất cả các tiến trình (user space), vùng còn lại 0x80000000 – 0xFFFFFFFF là của kernel (kernel mode). Mỗi tiến trình có một không gian địa chỉ ảo riêng. Không gian địa chỉ ảo này là 2GB, tất cả các thread của tiến trình đều có thể truy cập vào vùng không gian ảo này. Mỗi không gian địa chỉ ảo của tiến trình đều được kernel quản lý thông qua bảng ánh xạ trang (page map) được lưu trong vùng kernel (page tables).



- Windows hỗ trợ một tập các thư viện lập trình API quản lý bộ nhớ như sau:



- Mỗi tiến trình có một vùng heap mặc định (default heap), vùng này được windows sử dụng cho thư viện C: malloc, free.
- Bên cạnh đó, windows cũng cho phép tạo các vùng heap trong không gian bộ nhớ của tiến trình thông qua Heap API: HeapCreate, HeapDestroy, HeapAlloc, HeapFree.

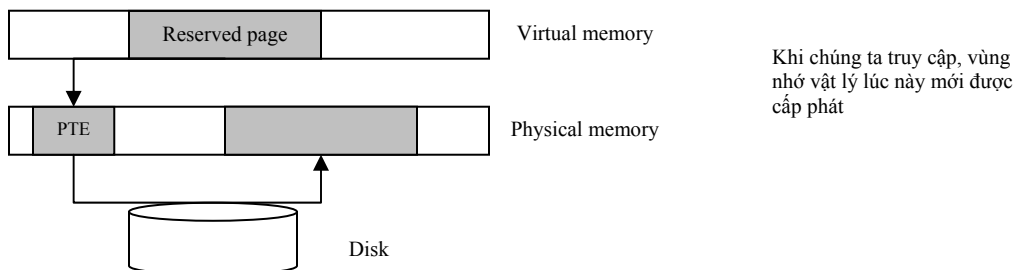
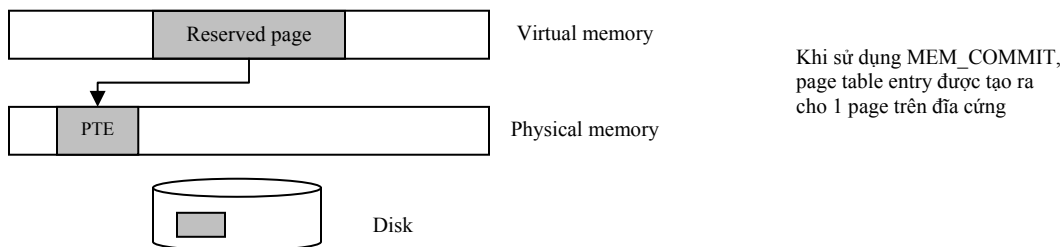
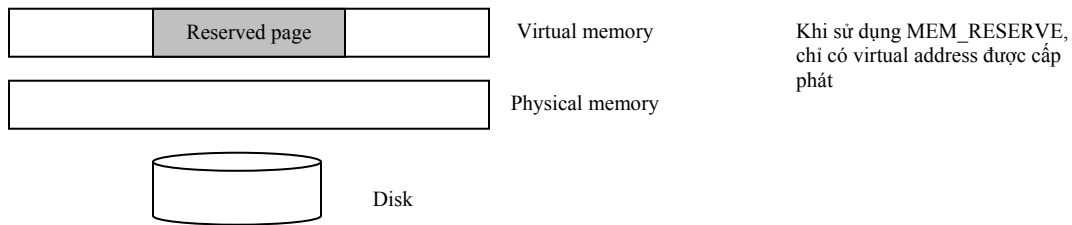
- Virtual Memory API là các hàm hệ thống cho phép tạo các vùng nhớ trực tiếp từ không gian địa chỉ ảo của windows: VirtualAlloc, VirtualFree, ...
 Một số hàm cơ bản của windows:

```
void GetSystemInfo(
    LPSYSTEM_INFO lpSystemInfo
);
```

Trả về thông tin hiện hành của hệ thống như: số lượng CPU, page size, ...

```
LPVOID VirtualAlloc(
    LPVOID lpAddress,
    SIZE_T dwSize,
    DWORD flAllocationType,
    DWORD flProtect
);
```

Cấp phát 1 vùng không gian địa chỉ ảo. Vùng này được dự trữ (nếu thông số *flAllocationType* là MEM_RESERVE), hay là được cấp phát (nếu thông số *flAllocationType* là MEM_COMMIT).



```

BOOL VirtualFree(
    LPVOID lpAddress,
    SIZE_T dwSize,
    DWORD dwFreeType
);

```

Lấy lại vùng nhớ đã được cấp phát.

Thực hành:

- Tạo một project và viết chương trình in ra thông tin của hệ thống thông qua hàm *GetSystemInfo*.

```

SYSTEM_INFO siSysInfo;
GetSystemInfo(&siSysInfo);
// Display the contents of the SYSTEM_INFO structure.
printf("Hardware information: \n");
printf(" OEM ID: %u\n", siSysInfo.dwOemId);
printf(" Number of processors: %u\n",
siSysInfo.dwNumberOfProcessors);
printf(" Page size: %u\n", siSysInfo.dwPageSize);
printf(" Processor type: %u\n", siSysInfo.dwProcessorType);
printf(" Minimum application address: %lx\n",
siSysInfo.lpMinimumApplicationAddress);
printf(" Maximum application address: %lx\n",
siSysInfo.lpMaximumApplicationAddress);
printf(" Active processor mask: %u\n",
siSysInfo.dwActiveProcessorMask);

```

- Lấy thông tin kích thước trang có được từ hàm *GetSystemInfo* ở trên, sử dụng hàm *VirtualAlloc* với tham số *flAllocationType* = *MEM_RESERVE* để cấp phát một vùng nhớ ảo 1GB. Sau đó chạy *Process Explorer* để quan sát tiến trình này với các thông số:

- *Private Bytes*: là tổng kích thước trang nhớ ảo mà đã được cấp phát cho tiến trình hiện giờ đang ở bộ nhớ thứ cấp.
- *Virtual Size*: là tổng kích thước trang nhớ ảo dành cho tiến trình.
- *Working Set*: là tổng kích thước hiện tại trong bộ nhớ vật lý của tiến trình.

- Lấy thông tin kích thước trang ở trên, sử dụng hàm *VirtualAlloc* với tham số *flAllocationType* = *MEM_COMMIT* để cấp phát một vùng nhớ ảo 1GB. Tương tự hãy quan sát các thông số như trên trong *Process Explorer*. So sánh với trường hợp trên.

- Bây giờ hãy truy cập vào vùng nhớ này (bằng cách gán giá trị cho nó), tương tự hãy quan sát thông số trên *Process Explorer*. Có nhận xét gì so với 2 trường hợp trên?