* Name of the project :

ProcMon(Process Monitoring Tool).

* Technology Used :

WIN 32 SDK

(It is 32-bit application development on windows using C/C++ language)

* User Interface Used :

Command line user interface(CUI)

* Platform Required :

Windows NT Operating System.

* Hardware Required :

32 bit processor with minimum 512 mb RAM.

* Description of the project :

Project supports Windows platform only and is used to retrieve information of all ongoing(running) process and its dependencies

We can Monitorize and analyse all the running process.

With the processes we can also fetch other related information as process name, process id(PID), process size in RAM, the thread count, names of dependent dll, we can maintain a log file of the process

We can also search a specific process as well as terminate that process

* Data Structures used in Project :
* Actual Code of the Project :

//Header files

#include<stdio.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<iostream>

#include<string.h>

#include<windows.h>

#include<tlhelp32.h>

#include<stdio.h>

#include<io.h>

using namespace std;

//typedef the structure

typedef struct LogFile

{

char ProcessName[100];

unsigned int pid;

unsigned int ppid;

unsigned int thread\_cnt;

} LOGFILE;

class ThreadInfo

{

private:

DWORD PID; //Double Word

HANDLE hThreadSnap;

THREADENTRY32 te32;

public:

ThreadInfo(DWORD);

BOOL ThreadsDisplay();

};

// Name of function :ThreadInfo.

// input parameters :DWORD no,DWORD is a object of structure LogFile.

// return value :void.

// Description and Use:creating a snapshot offunction and shows information

ThreadInfo::ThreadInfo(DWORD no)

{

PID=no;

hThreadSnap=CreateToolhelp32Snapshot(TH32CS\_SNAPTHREAD,PID);

if(hThreadSnap== INVALID\_HANDLE\_VALUE)

{

cout<<"unable to create snapshot of current thread pool"<<endl;

return;

}

te32.dwSize=sizeof(THREADENTRY32);

}

// Name of function :ThreadDisplay.

// input parameters :

// return value :BOOL(boolean)

// Description and Use:Displays Threads

BOOL ThreadInfo :: ThreadsDisplay()

{

if(!Thread32First(hThreadSnap,&te32))

{

cout<<"Error:In Getting the First Thread"<<endl;

CloseHandle(hThreadSnap);

return FALSE;

}

cout<<endl<<"THREAD OF THIS PROCESS:"<<endl;

do

{

if(te32.th32OwnerProcessID == PID)

{

cout<<"\t THREAD ID:"<<te32.th32ThreadID<<endl;

}

}while(Thread32Next(hThreadSnap,&te32));

CloseHandle(hThreadSnap);

return TRUE;

}

class DLLInfo

{

private:

DWORD PID;

MODULEENTRY32 me32;

HANDLE hProcessSnap;

public:

DLLInfo(DWORD);

BOOL DependentDLLDisplay();

};

// Name of function :DLLInfo

// input parameters :DWORD no,DWORD is a object of structure LogFile.

// return value :void.

// Description and Use:creating a snapshot of current thread and shows info.

DLLInfo::DLLInfo(DWORD no)

{

PID=no;

hProcessSnap=CreateToolhelp32Snapshot(TH32CS\_SNAPMODULE32,PID);

if(hProcessSnap==INVALID\_HANDLE\_VALUE)

{

cout<<"ERROR : Unable To Create SnapShot Of Current Thread pool"<<endl;

return;

}

me32.dwSize=sizeof(MODULEENTRY32);

}

// Name of function :DependentDLLDisplay

// input parameters :

// return value :BOOl(boolean)

// Description and Use:dependent DLL of current process

BOOL DLLInfo :: DependentDLLDisplay()

{

char Arr[200];

if(!Module32First(hProcessSnap,&me32))

{

cout<<"FAILED To Get DLL Information"<<endl;

CloseHandle(hProcessSnap);

return FALSE;

}

cout<<"DEPENNDET DLL OF THIS PROCESS"<<endl;

do

{

wcstombs\_s(NULL,Arr,200,me32.szModule,200);

cout<<Arr<<endl;

}while(Module32Next(hProcessSnap,&me32));

CloseHandle(hProcessSnap);

return TRUE;

}

class ProcessInfo

{

private:

DWORD PID;

DLLInfo \*pdobj;

ThreadInfo \*ptobj;

HANDLE hProcessSnap;

PROCESSENTRY32 pe32;

public:

ProcessInfo();

BOOL ProcessDisplay(char \*);

BOOL ProcessLog();

BOOL ReadLog(DWORD,DWORD,DWORD,DWORD);

BOOL ProcessSearch(char \*str);

BOOL KillProcess(char \*str);

};

// Name of function :ProcessInfo.

// input parameters :

// return value :void.

// Description and Use:creating a snapshot of running process

ProcessInfo::ProcessInfo()

{

ptobj=NULL;

pdobj=NULL;

hProcessSnap=CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS,0);

if(hProcessSnap==INVALID\_HANDLE\_VALUE)

{

cout<<"Unable To Create the SnapShot Of Running Process"<<endl;

return;

}

pe32.dwSize=sizeof(PROCESSENTRY32);

}

// Name of function :ProcessLog

// input parameters :

// return value :BOOL(boolean)

// Description and Use:creating a log file of process,

BOOL ProcessInfo::ProcessLog()

{

char \*month[]={"JAN","FEB","MAR","APR","MAY","JUN","JUL","AUG","SEP","OCT","NOV","DEC"};

char FileName[50],Arr[512];

int ret=0,fd=0,count=0;

SYSTEMTIME it;

LOGFILE fobj;

FILE \*fp;

GetLocalTime(&it);

sprintf\_s(FileName,"C://MarvellousLog %02d\_%02d\_%02d%s.txt",it.wHour,it.wMinute,it.wDay,month[it.wMonth-1]);

fp=fopen(FileName,"wb");

if(fp==NULL)

{

cout<<"Unable to Create Log File"<<endl;

return FALSE;

}

else

{

cout<<"Log File SuccesFully Created as :"<<FileName<<endl;

cout<<"Time Of Log File Creation is :"<<it.wHour<<":"<<it.wMinute<<":"<<it.wDay<<":"<<month[it.wMonth-1]<<endl;

}

if(!Process32First(hProcessSnap,&pe32))

{

cout<<"ERROR : In Finding First Process"<<endl;

CloseHandle(hProcessSnap);

return FALSE;

}

do

{

wcstombs\_s(NULL,Arr,200,pe32.szExeFile,200);

strcpy\_s(fobj.ProcessName,Arr);

fobj.pid=pe32.th32ProcessID;

fobj.ppid=pe32.th32ParentProcessID;

fobj.thread\_cnt=pe32.cntThreads;

fwrite(&fobj,sizeof(fobj),1,fp);

}while(Process32Next(hProcessSnap,&pe32));

CloseHandle(hProcessSnap);

fclose(fp);

return TRUE;

}

// Name of function :ProcessDisplay.

// input parameters :character pointer which name is option

// return value :BOOL(boolean)

// Description and Use:find the process and display it

BOOL ProcessInfo:: ProcessDisplay(char \*option)

{

char Arr[200];

if(!Process32First(hProcessSnap,&pe32))

{

cout<<"ERROR : in Finding First Process"<<endl;

CloseHandle(hProcessSnap);

return FALSE;

}

do

{

cout<<"-------------------------------------------------------------------------------";

wcstombs\_s(NULL,Arr,200,pe32.szExeFile,200);

cout<<endl<<"PROCESS NAME:"<<Arr;

cout<<endl<<"PID :"<<pe32.th32ProcessID;

cout<<endl<<" PARENT PID:"<<pe32.th32ParentProcessID;

cout<<endl<<"No Of Threads:"<<pe32.cntThreads;

if((\_stricmp(option,"-a")==0)||(\_stricmp(option,"-d")==0)||(\_stricmp(option,"-t")==0))

{

if((\_stricmp(option,"-t")==0)||(\_stricmp(option,"-a")==0))

{

ptobj= new ThreadInfo(pe32.th32ProcessID);

ptobj->ThreadsDisplay();

delete ptobj;

}

if((\_stricmp(option,"-d")==0)||(\_stricmp(option,"-a")==0))

{

pdobj= new DLLInfo(pe32.th32ProcessID);

pdobj->DependentDLLDisplay();

//pdobj->DependentDLLDisplay;

delete pdobj;

}

}

cout<<endl<<"------------------------------------------------------------";

}while(Process32Next(hProcessSnap,&pe32));

CloseHandle(hProcessSnap);

return TRUE;

}

// Name of function :ReadLog.

// input parameters :DWORD no,DWORD min,DWORD date,DWORD month

// return value :BOOL

// Description and Use:read the logs and shows information

BOOL ProcessInfo :: ReadLog(DWORD hr, DWORD min,DWORD date,DWORD month)

{

char FileName[50];

char \*Montharr[]={"JAN","FEB","MAR","APR","MAY","JUN","JUL","AUG","SEP","OCT","NOV","DEC"};

int ret=0,count=0;

LOGFILE fobj;

FILE \*fp;

sprintf\_s(FileName,"C://MarvellousLog %02d\_%02d\_%02d%s.txt",hr,min,date,Montharr[month-1]);

fp=fopen(FileName,"rb");

if(fp==NULL)

{

cout<<"Unable to Open Log File Named as"<<FileName<<endl;

return FALSE;

}

while ((ret=fread(&fobj,1,sizeof(fobj),fp))!=0)

{

cout<<"----------------------------------------------------------------"<<endl;

cout<<endl<<"PROCESS NAME:"<<fobj.ProcessName<<endl;

cout<<endl<<"PID :"<<fobj.pid<<endl;

cout<<endl<<" PARENT PID:"<<fobj.ppid<<endl;

cout<<endl<<"No Of Threads:"<<fobj.thread\_cnt<<endl;

}

return TRUE;

}

// Name of function :ProcessSearch.

// input parameters :char \*name.

// return value :BOOL

// Description and Use:search the process

BOOL ProcessInfo:: ProcessSearch(char \*name)

{

char Arr[200];

BOOL flag=FALSE;

if(!Process32First(hProcessSnap,&pe32))

{

CloseHandle(hProcessSnap);

return FALSE;

}

do

{

wcstombs\_s(NULL,Arr,200,pe32.szExeFile,200);

if(\_strcmpi(Arr,name)==0)

{

cout<<endl<<"PROCESS NAME:"<<Arr;

cout<<endl<<"PID :"<<pe32.th32ProcessID;

cout<<endl<<" PARENT PID:"<<pe32.th32ParentProcessID;

cout<<endl<<"No Of Threads:"<<pe32.cntThreads;

flag=TRUE;

break;

}

}while(Process32Next(hProcessSnap,&pe32));

CloseHandle(hProcessSnap);

return flag;

}

// Name of function :KillProcess

// input parameters :char name

// return value :BOOL

// Description and Use:to kill the process

BOOL ProcessInfo:: KillProcess(char \*name)

{

char Arr[200];

int pid= -1;

BOOL bret;

HANDLE hProcess;

if(!Process32First(hProcessSnap,&pe32))

{

CloseHandle(hProcessSnap);

return FALSE;

}

do

{

wcstombs\_s(NULL,Arr,200,pe32.szExeFile,200);

if(\_stricmp(Arr,name)==0)

{

pid=pe32.th32ProcessID;

break;

}

}while(Process32Next(hProcessSnap,&pe32));

if(pid==-1)

{

cout<<"ERROR : There is No Such Process"<<endl;

return FALSE;

}

hProcess=OpenProcess(PROCESS\_TERMINATE,FALSE,pid);

if(hProcess==NULL)

{

cout<<"ERROR :There is No Acess To Terminate"<<endl;

return FALSE;

}

bret=TerminateProcess(hProcess,0);

if(bret==FALSE)

{

cout<<"ERROR: Unable to Terminate Process"<<endl;;

return FALSE;

}

}

// Name of function :HardwareInfo.

// input parameters :

// return value :BOOL.

// Description and Use:provides hardware information

BOOL HardwareInfo()

{

SYSTEM\_INFO siSysInfo;

GetSystemInfo(&siSysInfo);

cout<<"Hardware Information of current system is"<<endl;

cout<<"OEM ID:"<<siSysInfo.dwOemId<<endl;

cout<<"Number Of Processors:"<<siSysInfo.dwNumberOfProcessors<<endl;

cout<<"Page Size:"<<siSysInfo.dwPageSize<<endl;

cout<<"Processor Type"<<siSysInfo.dwProcessorType<<endl;

cout<<"Minumum Application Address"<<siSysInfo.lpMinimumApplicationAddress<<endl;

cout<<"Maximum Application Address"<<siSysInfo.lpMaximumApplicationAddress<<endl;

cout<<"Active Process Mask"<<siSysInfo.dwActiveProcessorMask<<endl;

return TRUE;

}

// Name of function :Help

// input parameters :

// return value :void.

// Description and Use:gives commands for help

void DisplayHelp()

{

cout<<"Developed By Marvellous student"<<endl;

cout<<"ps : Display All Information About Process"<<endl;

cout<<"ps-t : Display All Information About Thread"<<endl;

cout<<"ps-d : Display All Information About DLL"<<endl;

cout<<"cls : Clear the Contens of Console"<<endl;

cout<<"log : Create Log of Current Process on C Drive"<<endl;

cout<<"readlog : Display the Information From Specified Log"<<endl;

cout<<"sysinfo : Display current hardware Configuration"<<endl;

cout<<"search : Search And Display Information Of Specific Running Process"<<endl;

cout<<"exit : Terminate Marvellous ProcMon"<<endl;

}

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

{

BOOL bret;

char \*ptr=NULL;

ProcessInfo \*ppobj=NULL;

char command[4][80],str[80];

int count,min,date,month,hr;

while (1)

{

fflush(stdin);

strcpy\_s(str,"");

cout<<endl<<"Marvellous ProcMon: >";

fgets(str,80,stdin);

count=sscanf(str,"%s %s %s %s",command[0],command[1],command[2],command[3]);

if(count==1)

{

if(\_stricmp(command[0],"ps")==0)

{

ppobj=new ProcessInfo();

bret=ppobj->ProcessDisplay("-a");

if(bret==FALSE)

{

cout<<"Error:unable to Display Process"<<endl;

}

delete ppobj;

}

else if(\_stricmp(command[0],"log")==0)

{

ppobj=new ProcessInfo();

bret=ppobj->ProcessLog();

if(bret==FALSE)

{

cout<<"Error:unable to Create Log File"<<endl;

}

delete ppobj;

}

else if(\_stricmp(command[0],"sysinfo")==0)

{

bret=HardwareInfo();

if(bret==FALSE)

{

cout<<"Error:unable to get hardware information"<<endl;

}

}

else if(\_stricmp(command[0],"readlog")==0)

{

ProcessInfo \*ppobj;

ppobj=new ProcessInfo();

cout<<"Enter Log File Details As:"<<endl;

cout<<"Hour:";

cin>>hr;

cout<<"Minute:";

cin>>min;

cout<<"Date:";

cin>>date;

cout<<"Month:";

cin>>month;

bret= ppobj->ReadLog(hr,min,date,month);

if(bret==FALSE)

{

cout<<"ERROR: unabale to Read Specified File"<<endl;

delete ppobj;

}

}

else if(\_stricmp(command[0],"clear")==0)

{

system("cls");

continue;

}

else if(\_stricmp(command[0],"help")==0)

{

DisplayHelp();

continue;

}

else if(\_stricmp(command[0],"exit")==0)

{

cout<<endl<<"Terminating the Marvellous ProcMon"<<endl;

break;

}

else

{

cout<<endl<<"ERROR: command Not Found"<<endl;

continue;

}

}

else if(count==2)

{

if(\_stricmp(command[0],"ps")==0)

{

ppobj=new ProcessInfo;

bret=ppobj->ProcessDisplay(command[1]);

if(bret==FALSE)

{

cout<<"ERROR : Unable To Display Process Information"<<endl;

}

delete ppobj;

}

else if(\_stricmp(command[0],"search")==0)

{

ppobj=new ProcessInfo;

bret=ppobj->ProcessSearch(command[1]);

if(bret==FALSE)

{

cout<<"ERROR : There Is No Such Process"<<endl;

}

delete ppobj;

}

else if(\_stricmp(command[0],"kill")==0)

{

ppobj=new ProcessInfo;

bret=ppobj->KillProcess(command[1]);

if(bret==FALSE)

{

cout<<"ERROR : There Is No Such Process"<<endl;

}

else

{

cout<<command[1]<<"Terminater SuccesFully"<<endl;

}

delete ppobj;

continue;

}

}

else

{

cout<<endl<<"ERROR: Command Not Found!!"<<endl;

continue;

}

}

return 0;

}