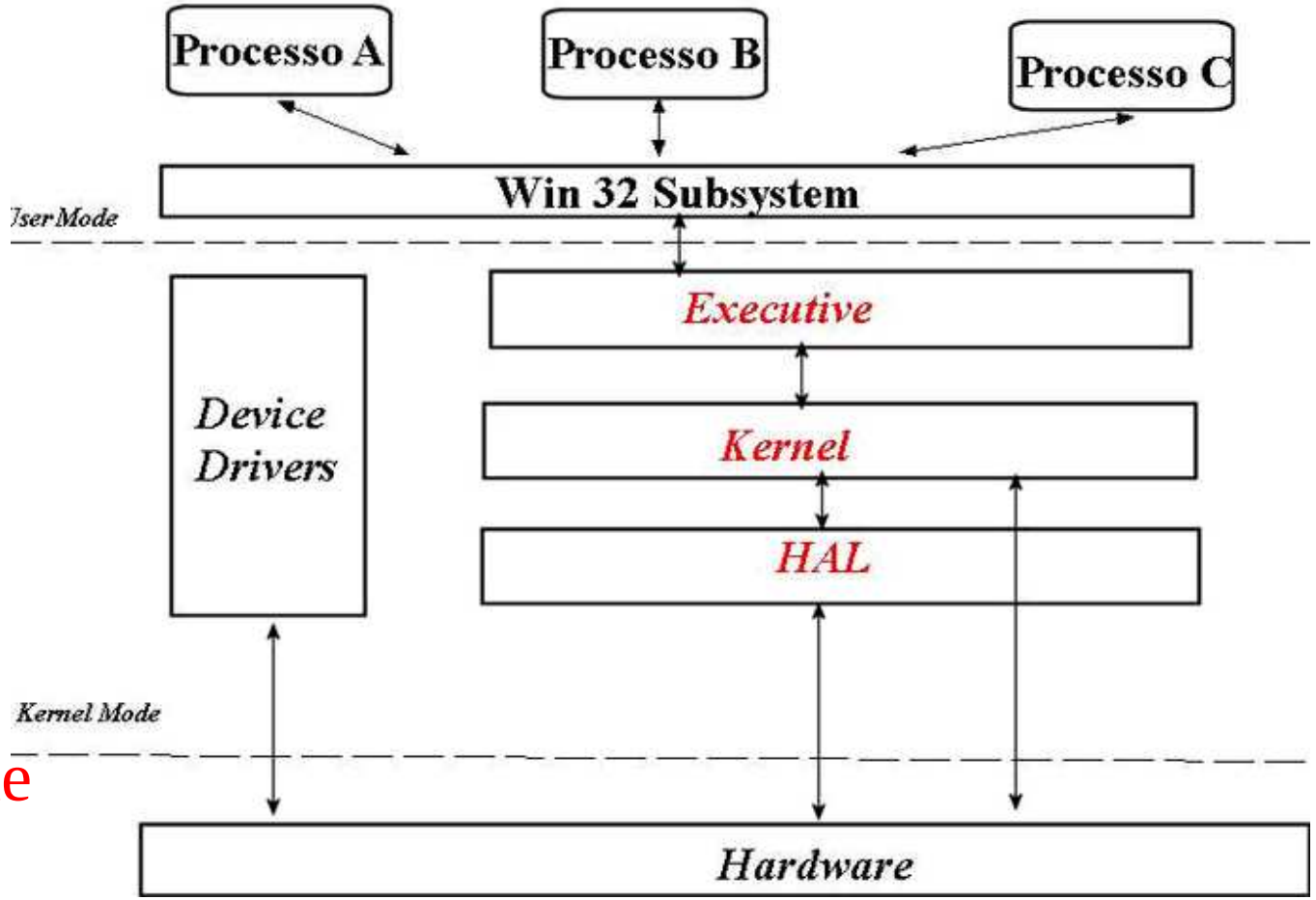
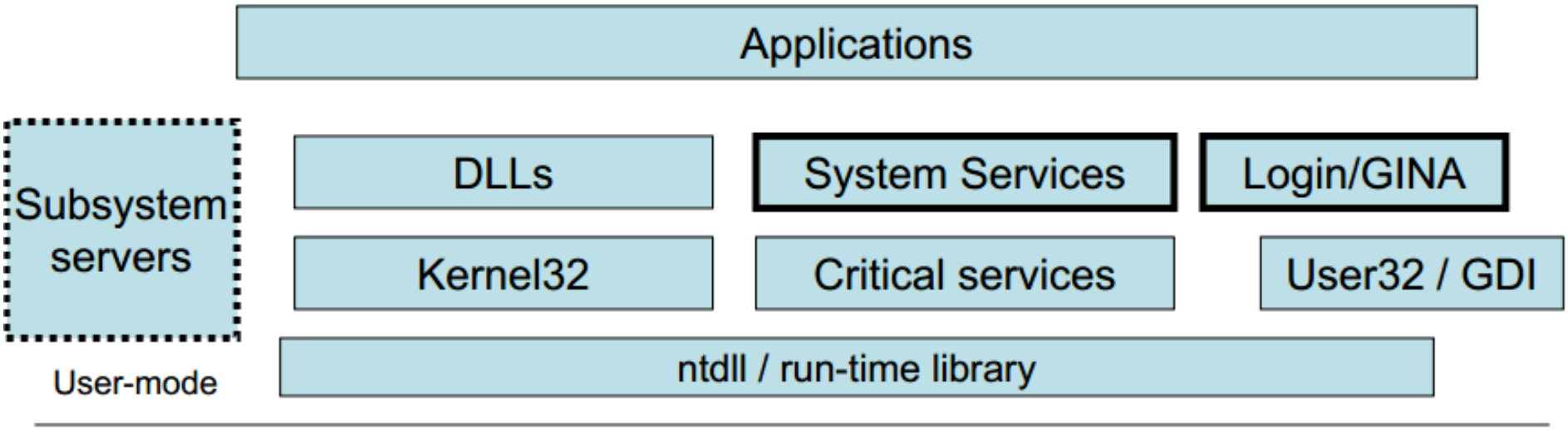
Sistema Operativo Windows NT

Il Sistema Operativo Windows è strutturato in maniera modulare e stratificata e si possono distinguere due moduli

1) User mode
2) Kernel mode



L’ User mode gestisce le applicazioni utente e quindi offre l’insieme di tutti i servizi utili per l’esecuzione di una applicazione (DDL,Login ect.)



Il Kernel mode gestisce i servizi base come

1) Process management
   - Creazione dei processi/thread
2) Security reference monitor
   - Access checks, token management
3) Memory manager
   - Pagefaults, virtual address, physical frame, pagefile management, sharing, copy-on-write, mapped files, GC support, large apps
4) Lightweight Procedure Call (LPC)
   - Il Native transport del Remote Procedure Call (RPC) che consente di scrivere applicazioni distribuite con architettura Client/Server
   - L'user-mode system services.

5) I/O manager (& plug-and-play & power)

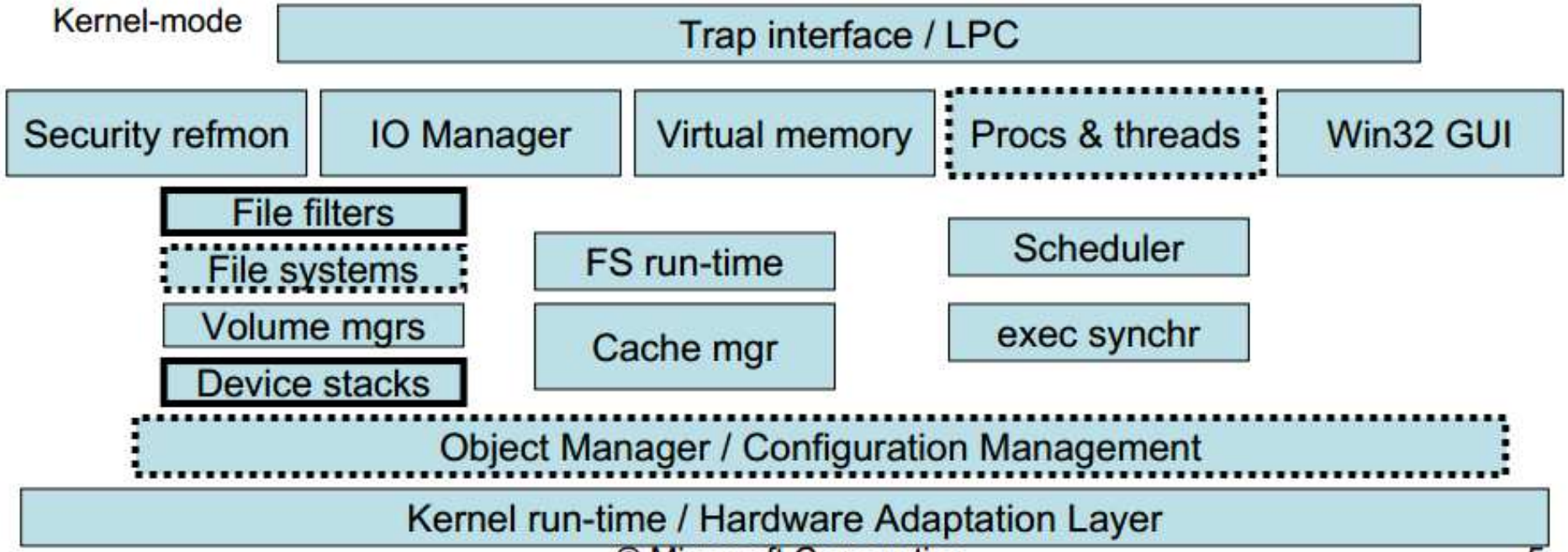
- Maps user requests into IRP requests, configures/manages I/O devices, implements services for drivers

6) Cache manager

- Per il buffer dei file e dell'I/O, che lavora a supporto del memory manager

7) Scheduler (aka 'kernel')

- Schedula le esecuzioni dei sui processi



Kernel-mode è organizzato in

1) NTOS (kernel-mode services)
   - Run-time Library, Scheduling, Executive services, object manager, services for I/O, memory, processes,
   - Cuore del sistema che fornisce i 4 servizi fondamentali:
     1) lo sheduling dei thread
     2) la gestione degli interrupt
     3) la sincronizzazione a basso livello
     4) gestione del consumo energetico

2) Hal (hardware-adaptation layer)
   - Strato software per la gestione diretta dell'hardware del computer. La HAL opera a un livello tra i componenti hardware e i servizi esecutivi di Windows, le applicazioni

e i driver di periferica non hanno la necessità di essere a conoscenza di eventuali informazioni specifiche dell'hardware.

- Nella HAL sono disponibili routine che consentono ad un driver di supportare una periferica in piattaforme hardware diverse, semplificando notevolmente il loro sviluppo.
- In particolare la HAL consente di nascondere dettagli di dipendenti di hardware come;
    1. Le interfacce di I/O
    2. Il controller di interrupt
    3. I meccanismi di comunicazione
    4. Il cambio di contesto
    5. Clocks
    6. Spinlocks

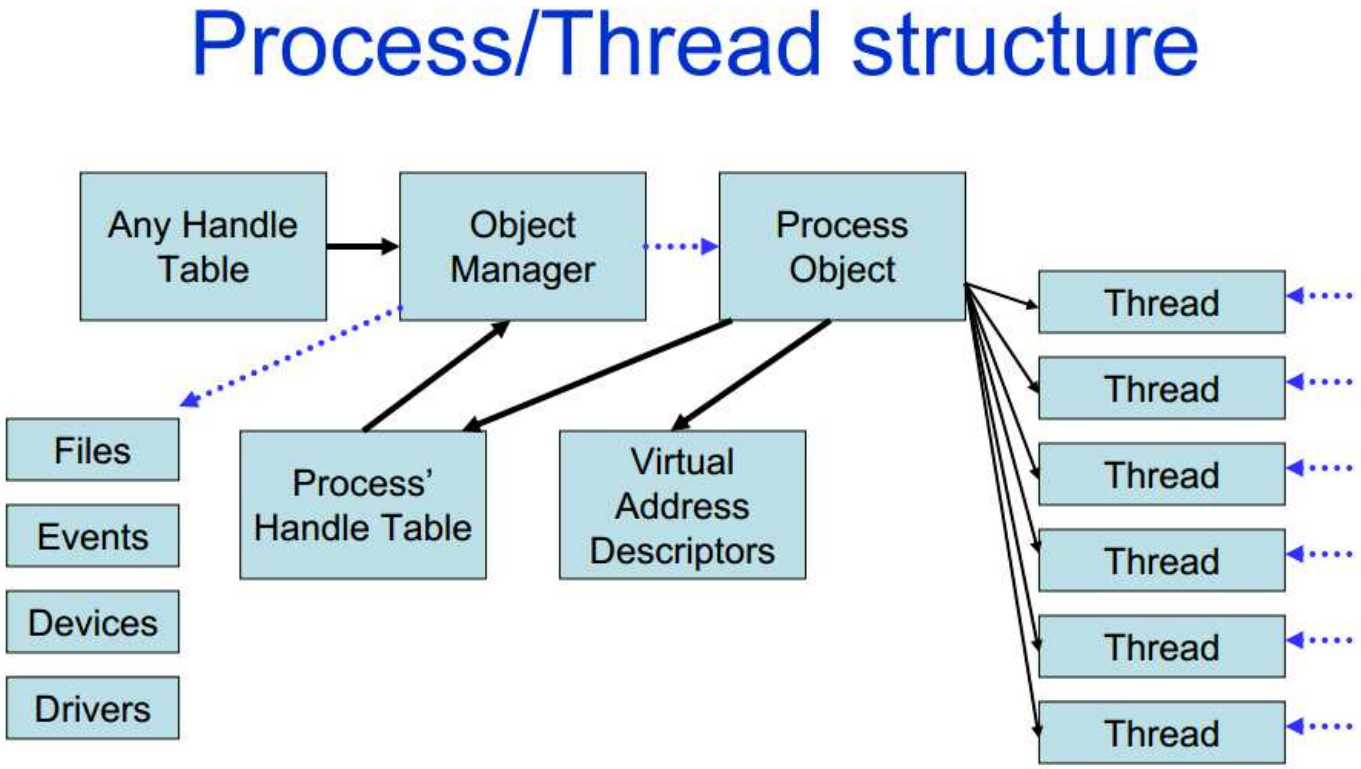
3) Drivers o Executive

- fornisce un insieme di servizi in modalità utente come

1) la gestione degli oggetti di sistema
2) la gestione della memoria virtuale
3) il file system e L'I/O
4) la creazione dei processi e dei thread.

Processi in WINDOW

Un processo in Windows NT è un “oggetto di nucleo” (kernel object) che definisce uno spazio di indirizzamento.



Un processo è caratterizzato da:

- Un identificatore
- Uno spazio di indirizzamento privato
- Uno o più thread di esecuzione
- Una directory corrente
- Varie tabelle contenenti le risorse del processo

Un thread è un “oggetto di nucleo” che definisce una entità concorrente e schedulabile. È caratterizzato da:

- Un identificatore
- Una funzione da eseguire
- Un contesto (insieme di registri del processore)
- Un puntatore allo stack

La politica di scheduling dei thread in Windows è intermedia tra la politica prioritaria e quella round-robin”.

La priorità di un thread viene calcolata come somma di due componenti:

- Una classe di priorità associata al processo a cui il thread appartiene
- Una priorità relativa del thread

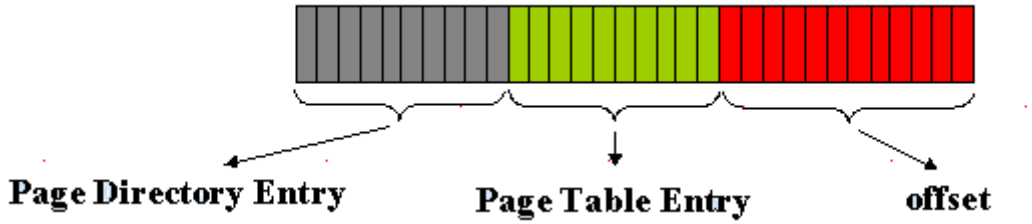
La gestione della memoria Virtuale in Windows NT prevede:

- Dimensione della memoria virtuale di 4 gigabyte (indirizzo virtuale di 32 bit)
- Memoria virtuale paginata (paginazione a domanda) con pagine di dimensioni fisse (le dimensioni della pagina dipendono dalla particolare macchina fisica)
- Spazio virtuale suddiviso in due sottospazi di 2 gigabyte ciascuno:
  - il sottospazio inferiore è privato di ogni processo
  - il sottospazio superiore è condiviso tra tutti i processi e contiene il sistema operativo.

Il Meccanismo di paginazione è a due livelli:

1) la tabella di primo livello (Page Directory) contiene 1024 elementi (Page Directory Entry)
2) Ciascuno dei quali punta ad una tabella di secondo livello (Page Table) anch'essa di 1024 elementi (Page Table Entry).

Indirizzo virtuale suddiviso in tre parti:



Il file system è di tipo transazionale con capacità di recupero dopo una caduta del sistema (crash), e fornisce il supporto al concetto di partizioni logica (volume), dove ogni volume è suddiviso in cluster (insieme di settori fisici del disco) che rappresentano le unità elementari di allocazione dei dati.

Creazione di Processi di WINDOW in C

Esistono vari meccanismi per la creazione di un nuovo processo all'interno di un programma C, alcuni esclusivamente di window altri condivisi con altri SO come linux

I Meccanismi comuni a Windows e Unix/Linux sono quelli visti nella lezione 5 ossia

- system
- spawn
- exec

Meccanismi di creazione di processi di Windows

In window esistono varie funzioni che generano nuovi processi, i più utilizzati sono WinExec, LoadModule e CreateProcess

- WinExec

La sintassi di WinExec è la seguente:

```
#include <Windows.h>
UINT WINAPI WinExec(LPCSTR lpCmdLine,UINT uCmdShow);
```

dove lpCmdLine di tipo LPCSTR Long Pointer Costant STR puntatore a 32 bit a stinga, è la stringa in cui si trova la riga di comando, e CmdShow un intero Unsigned INT che specifica come mostrare la finestra del programma.

I possibili valori di CmdShow sono i seguenti:

SW_MINIMIZE	Finestra iconizata
SW_HIDE	Finestra è nascosta e non è attiva.
SW_MAXIMIZE	Finestra a pieno schermo
SW_RESTORE **SW_SHOW**	Finestra attiva e mostrata con le proprie dimensioni
SW_SHOWDEFAULT	Finestra attiva e mostrata secondo i valori specificati all'atto della creazione della finestra
SW_SHOWMAXIMIZED	Finestra attiva e a pieno schermo
SW_SHOWMINIMIZED	Finestra attiva e iconizata
SW_SHOWNORMAL	Finestra attiva e mostrata nella posizione e grandezza orginale

Nel file header `<Windows.h>` si trovano le definizione di tutti questi tipi non standard voluti dagli sviluppatori di windows oltre che delle innumerevoli funzioni.

La funzione ritorna un valore >=31 se ha successo, viceversa ritorna uno dei seguenti valori:

0	The system is out of memory or resources.
ERROR_BAD_FORMAT	The .exe file is invalid.
ERROR_FILE_NOT_FOUND	The specified file was not found.
ERROR_PATH_NOT_FOUND	The specified path was not found.

La funzione WinExec lancia il programma eseguibile ma non c'è modo di controllarne l'esito, e questa funzione viene utilizzata solo per compatibilità con applicazioni windows a 16bit. Le applicazioni devono utilizzare la funzione CreateProcess.

Vediamo un esempio di utilizzo in cui viene lanciato il programma notepad.exe

```
// File di intestazione di Windows:
#include <windows.h>
#include <stdio.h>

void main( VOID )
{int a=WinExec("C:\\windows\\system32\\notepad.exe", SW_SHOWNORMAL);
 printf("WinExec a restituito %d",a);
 getchar();
}
```

- LoadModule

La sintassi di LoadModule è la seguente:

```
#include <Windows.h>
DWORD LoadModule(LPCTSTR lpName,LPVOID lpParameterBlock);
```

Dove **lpName** è una Stringa che contiene il filename dell'applicazione, e **lpParameterBlock** è un puntatore ad una

struttura di tipo LOADPARMS32 che definisce i parametri della nuova applicazione.

La struttura LOADPARMS32 è la seguente:

```
typedef struct tagLOADPARMS32 {
LPSTR lpEnvAddress; // Puntatore all'ambiente
strings LPSTR lpCmdLine; // stringa del comando
LPSTR lpCmdShow; // punatore ad un cmdshow
DWORD dwReserved; // deve essere zero
} LOADPARMS32;
```

La funzione LoadModule ritorna un valore >=31 se ha successo, viceversa ritorna uno dei valori già visti per la funzione WinExec.

- CreateProcess

La funzione CreateProcess crea un nuovo processo insieme al suo thread primario. Il nuovo processo esegue un file eseguibile.

La sintassi di CreateProcess è la seguente:

```
#include <Windows.h>
CreateProcess
          (LPCTSTR lpApplicationName,
           LPTSTR lpCommandLine ,
           LPSECURITY_ATTRIBUTES  lpProcessAttributes ,
           LPSECURITY_ATTRIBUTES lpThreadAttributes ,
           BOOL bInheritHandles ,
           DWORD dwCreationFlags ,
           LPVOID lpEnvironment ,
           LPCTSTR lpCurrentDirectory,
           LPSTARTUPINFO lpStartupInfo,
           LPPROCESS_INFORMATION lpProcessInformation);
```

Con

lpApplicationName stringa contenente il modulo da eseguire. Questo parametro è NULL nel caso che il parametro lpCommandLine non sia nullo. Il modulo specifica una applicazione Win32-based (anche MS-DOS o OS/2)

lpCmomandLine stringa contenente la linea di commando da eseguire. lpCommandLine può essere NULL, ma in questo caso deve essere non nullo il parametro lpApplicationName

lpProcessAttributes puntatore alla strutturaSECURITY_ATTRIBUTES che determina se il puntatore handle ritornato può essere utilizzato da un processo figlio.

bInheritHandles indica se il nuovo processo ha ereditato l'handle dal processo chiamante. Se TRUE, ogni handle aperto nel processo chiamante può essere richiamato dal nuovo processo.

dwCreationFlags Specifica i flags di controllo, ossia come creare il processo e la sua priorita, i possibili valori sono:

- CREATE_DEFAULT_ERROR_MODE (Il processo non eredita la modalità di errore, ma adotta la default error mode.
- CREATE_NEW_CONSOLE (creato con una nuova console)

- CREATE_NEW_PROCESS_GROUP (Il processo sarà un processo radice di un nuovo gruppo di processi. I gruppi di processi sono usati dalla funzione GenerateConsoleCtrlEvent per abilitare il segnale di CTRL+BREAK a tutti i processi del gruppo
- CREATE_SUSPENDED(Si deve richiamare la funzione ResumeThread per attivarlo).

I possibili valori di priorità sono:

HIGH_PRIORITY_CLASS
IDLE_PRIORITY_CLASS
NORMAL_PRIORITY_CLASS
REALTIME_PRIORITY_CLASS

lpEnvironment Puntatore ad un environment block. Se NULL, il nuovo processo usa lo stesso environment del processo chiamante. Un environment block sono delle stringhe del tipo name=value

lpCurrentDirectory Stringa che specificala directory del processso

lpStartupInfo Puntatore ad una struttura STARTUPINFO che specifica come la finestra principale dovrà apparire, per maggiori informazioni andata su http://msdn.microsoft.com/en-us/library/ms686331(VS.85).aspx

lpProcessInformation Puntatore ad una struttura PROCESS_INFORMATION che riceve le informazione del nuovo processo.

```
typedef struct _PROCESS_INFORMATION {
HANDLE hProcess; // puntatore al nuovo processo
HANDLE hThread;   // puntatore al primary thread del processo
DWORD  dwProcessId; //ID del processo
DWORD  dwThreadId;  //TID del thread
} PROCESS_INFORMATION, *LPPROCESS_INFORMATION;
```

La funzione CreateProcess ritorna un valore non zero se la funzione ha successo, viceversa ritorna zero, e per avere informazioni sull'errore bisogna richiamare la funzione GetLastError .

Nel seguente esempio vediamo come si crea un processo esterno e come si aspetta la sua terminazione.

```
// File di intestazione di Windows:
#include <windows.h>
#include <stdio.h>

void main( VOID )
{
STARTUPINFO si;
PROCESS_INFORMATION pi;
ZeroMemory( &si, sizeof(si) );
//riempe di 0 un blocco di memoria
si.cb = sizeof(si);
// Start the child process.
if( !CreateProcess( NULL, // No module use command line).
  "c://WINDOWS//system32//notepad.exe", // Command line.
  NULL, // Process handle not inheritable.
  NULL, // Thread handle not inheritable.
  FALSE, // Set handle inheritance to FALSE.
  0, // No creation flags.
  NULL, // Use parent's environment block.
  NULL, // Use parent's starting directory.
  &si, // Pointer to STARTUPINFO structure.
  &pi ) // Pointer to PROCESS_INFORMATION structure.
)
{ printf("CreateProcess failed." ); }
// Wait until child process exits.
WaitForSingleObject( pi.hProcess, INFINITE );
// Close process and thread handles.
CloseHandle( pi.hProcess );
```

```
CloseHandle( pi.hThread );
printf("fine funzione ");
getchar();
}
```

Da notare la funzione `WaitForSingleObject( pi.hProcess, INFINITE );` che implementa una attesa non attiva sull'Headle del processo.