Windows Programming

Lecture 09

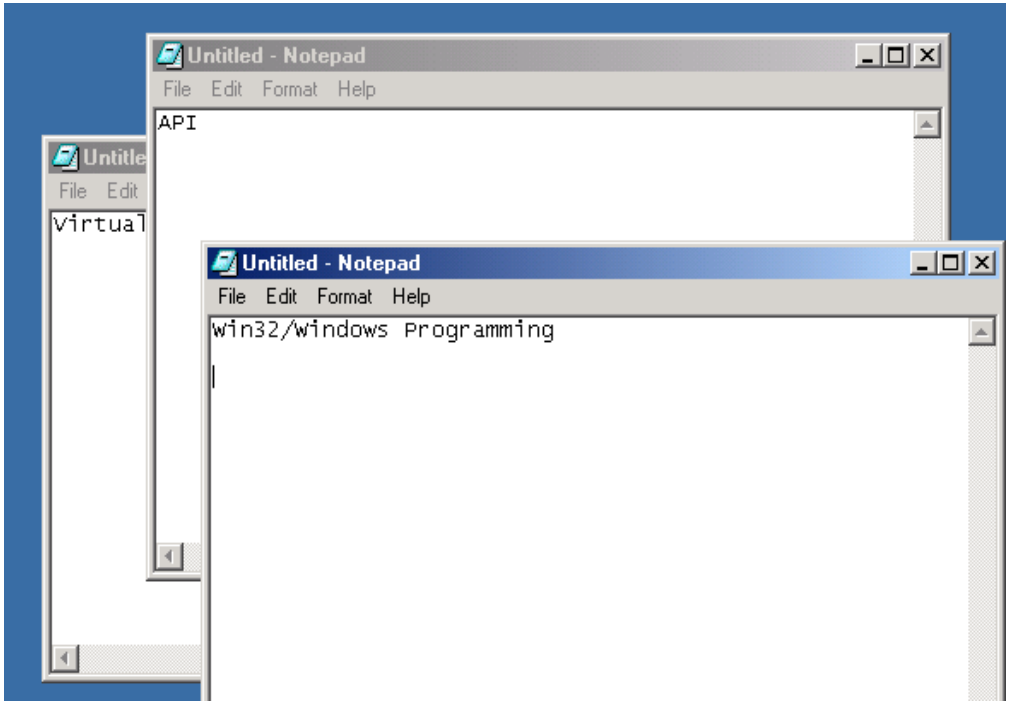
Processes in Windows

- In Windows, every running application is called a process.
- Every process has a process ID that uniquely identifies it. You can pass a process ID to the OpenProcess API function to get a process handle. Most API functions that provide process information require a handle rather than an ID.

Instances

- The term instance handle comes from 16-bit Windows, where an instance handle was tied to the data of a program. Each instance of the program had its own unique data but shared its code with other instances. In 32-bit Windows, each instance of a program has its own separate copy of both the code and the data, and the addresses within that copy are unintelligible to other instances.

Multiple instances of Notepad have the same look, shape etc.



Window Class

- A window class is a set of attributes that the application uses as a template to create a window. Every window belongs to a window class. All window classes are process specific.

Contd.

- Every application creates its own window
- Each window belongs to a specific window class
- Window class tells the OS about the **characteristics** and **physical layout** of its windows.

Window Class

Each window class has an associated window procedure shared by all windows of the same class. The window procedure processes messages for all windows of that class and therefore controls their behavior and appearance.

Registering a Window Class

A process must register a window class before it can create a window of that class. Registering a window class associates a window procedure, class styles, and other class attributes with a class name.

Registered Window Classes

- List of registered windows' classes is maintained by Windows
- You can register your own Window class using **`RegisterClass()`** API function

`RegisterClass()`

`RegisterClass()` registers a new window class for use.

RegisterClass()

Win32 API function

```
ATOM RegisterClass
(
  CONST WNDCLASS *lpWndClass
            //Pointer to WNDCLASS variable
);
```

WNDCLASS Structure

- ```
  typedef struct _WNDCLASS {
  UINT            style;
  WNDPROC         lpfnWndProc;
  int             cbClsExtra;
  int             cbWndExtra;
  HINSTANCE       hInstance;
  HICON           hIcon;
  HCURSOR         hCursor;
  HBRUSH          hbrBackground;
  LPCTSTR         lpszMenuName;
  LPCTSTR         lpszClassName;
  } WNDCLASS, *PWNDCLASS;
  ```

Structure Name

typedef of _WNDCLASS
  ```

WNDCLASS Structure

- The WNDCLASS structure contains the window class attributes.

- The WNDCLASS structure holds most of the information while registering a window class. This information is used by any windows which belong to the class. The only item which this structure does not hold is a handle to the class's small icon. (The more advanced WNDCLASSEX structure does.)

WNDCLASS Structure

Members

- **style**

  Specifies the class style(s).

- **lpfnWndProc**

  Pointer to the window procedure.

- **cbClsExtra**

  Specifies the number of extra bytes to allocate following the window-class structure. The system initializes the bytes to zero.

- **cbWndExtra**

  Specifies the number of extra bytes to allocate following the window instance.

WNDCLASS Structure

- **hInstance**

  Handle to the instance that contains the window procedure for the class.

- **hIcon**

  Handle to the class icon. This member must be a handle to an icon resource. If this member is NULL, the system provides a default icon.

- **hCursor**

  Handle to the class cursor. This member must be a handle to a cursor resource. If this member is NULL, an application must explicitly set the cursor shape whenever the mouse moves into the application's window.

WNDCLASS Structure

- **hbrBackground**

  Handle to the class background brush. This member can be a handle to the physical brush to be used for painting the background, or it can be a color value.

- **lpszMenuName**

  Pointer to a null-terminated character string that specifies the resource name of the class menu, as the name appears in the resource file. If you use an integer to identify the menu, use the MAKEINTRESOURCE macro.

- **lpszClassName**

  Pointer to a null-terminated string or is an atom. If this parameter is an atom, it must be a class atom created by a previous call to the **RegisterClass** or **RegisterClassEx** function.

`ATOM` datatype

- Unique Identifier of the registered window class returned by **`RegisterClass()`** API function call.

`UnregisterClass()`

The **UnregisterClass** function unregisters a window class, freeing the memory required for the class.

UnregisterClass()

```
BOOL UnregisterClass(
LPCTSTR lpClassName,
        // class name
HINSTANCE hInstance
  // handle to application
instance );
```

`CreateWindow()`

Win32 API function

- The **`CreateWindow()`** function creates an overlapped, pop-up, or child window. It specifies the window class, window title, window style, and (optionally) the initial position and size of the window. The function also specifies the window's parent or owner, if any, and the window's menu.

CreateWindow()

```
HWND CreateWindow(
   LPCTSTR lpClassName,
                                  // registered class name
   LPCTSTR lpWindowName,
                                  // window name
   DWORD dwStyle,
                                  // window style
   int x,
                                  // horizontal position of window
   int y,
                                  // vertical position of window
```

CreateWindow()

```
int nWidth,                    // window width
int nHeight,                   // window height
HWND hWndParent,               // handle to parent or owner
HMENU hMenu,                   // menu handle or child identifier
HINSTANCE hInstance,           // handle to application instance
LPVOID lpParam                 // window-creation data );
```

`CreateWindow()`

- **`CreateWindow()`** returns the handle to the created window if the function succeeds.

- If the function fails, the returned value is NULL.

Summary of window creation process

1. Initialise a **`WNDCLASS`** structure
2. Register a window class by passing as an argument, a **`WNDCLASS`** sturcture's address to **`RegisterClass()`** function.
3. Create the window of registered class with the help of **`CreateWindow()`**, a Win32 API function.

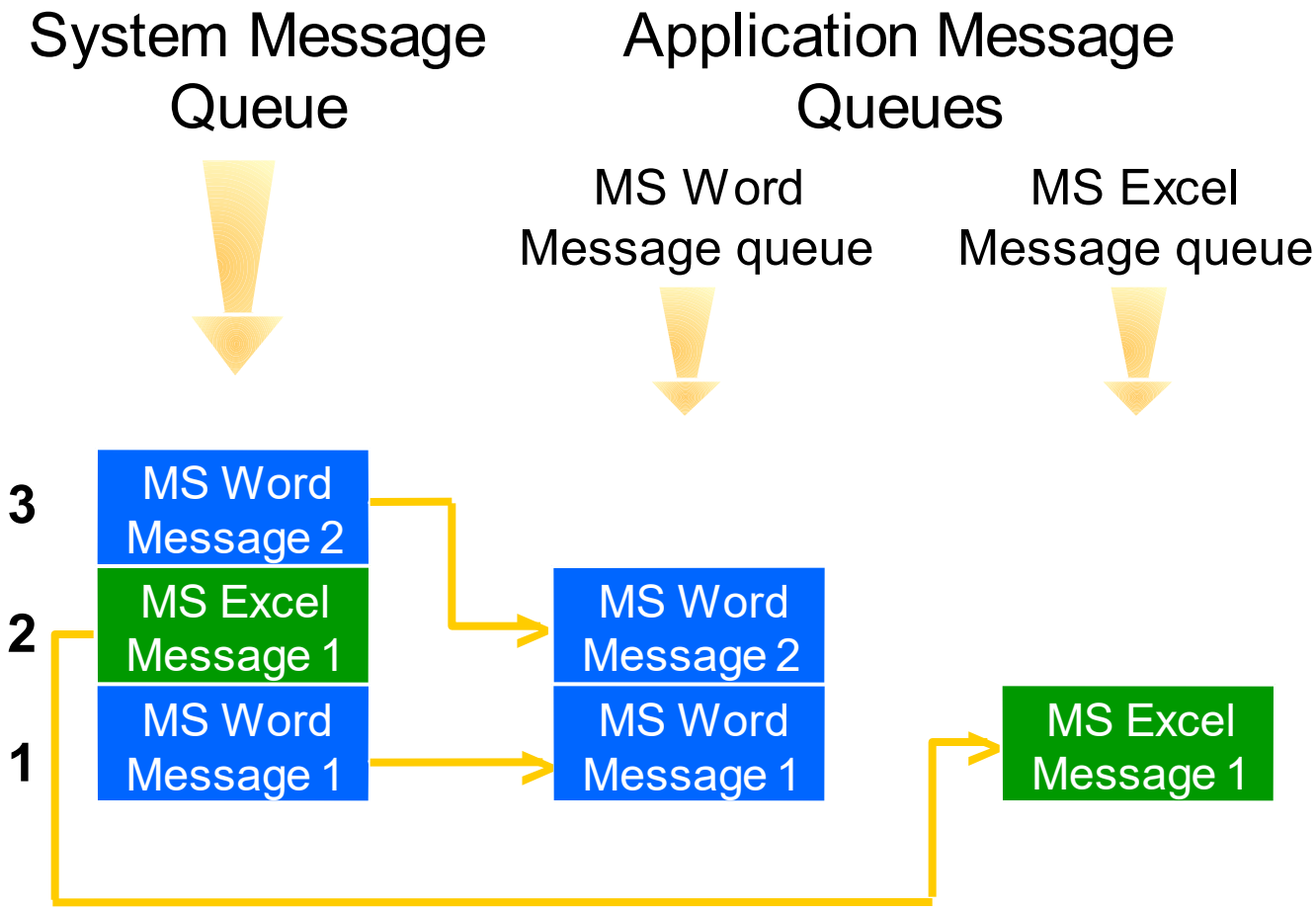
Messages in Windows

- Unlike MS-DOS-based applications, Win32®-based applications are event-driven. They do not make explicit function calls (such as C run-time library calls) to obtain input. Instead, they wait for the system to pass input to them.
- The system passes all input for an application to the various windows in the application. Each window has a function, called a *window procedure*, that the system calls whenever it has input for the window. The window procedure processes the input and returns control to the system. For more information about window procedures,

Message Queuing

- Operating system keeps the generated messages in a queue.
- Every application has its own message queue.

Message Queuing



Message Routing

- The system uses two methods to route messages to a window procedure: posting messages to a first-in, first-out queue called a message queue, a system-defined memory object that temporarily stores messages, and sending messages directly to a window procedure.

- Messages posted to a message queue are called queued messages. They are primarily the result of user input entered through the mouse or keyboard

Posting a Message

An application typically posts a message to notify a specific window to perform a task. **PostMessage** creates an **MSG** structure for the message and copies the message to the message queue. The application's message loop eventually retrieves the message and dispatches it to the appropriate window procedure.

Sending a Message

An application typically sends a message to notify a window procedure to perform a task immediately. The **SendMessage** function sends the message to the window procedure corresponding to the given window. The function waits until the window procedure completes processing and then returns the message result. Parent and child windows often communicate by sending messages to each other.

Window Procedure

Each window is a member of a particular window class. The window class determines the default window procedure that an individual window uses to process its messages. All windows belonging to the same class use the same default window procedure.

Window Procedure

Some important data types used in window procedure

typedef UINT WPARAM;
typedef LONG LPARAM;
typedef LONG LRESULT;

WNDPROC data type

```
typedef LRESULT (CALLBACK* WNDPROC)(HWND,
                    UINT, WPARAM, LPARAM);
```

Prototype of Window Procedure

```
LRESULT CALLBACK WndProc(
HWND hWnd,
UINT msg,
WPARAM wParam,
LPARAM lParam);
```

GetMessage()

The **GetMessage** function retrieves a message from the calling thread's message queue. The function dispatches incoming sent messages until a posted message is available for retrieval.

GetMessage()

```
BOOL GetMessage(
  LPMSG lpMsg,
                  // message information
  HWND hWnd,
                 // handle to window UINT
  wMsgFilterMin,
                 // first message UINT
  wMsgFilterMax
                  // last message );
```

DispatchMessage()

The **DispatchMessage** function dispatches a message to a window procedure. It is typically used to dispatch a message retrieved by the **GetMessage** function.

DispatchMessage()

```
LRESULT DispatchMessage(
  CONST MSG *lpmsg
                        // message information
);
```