Window programming

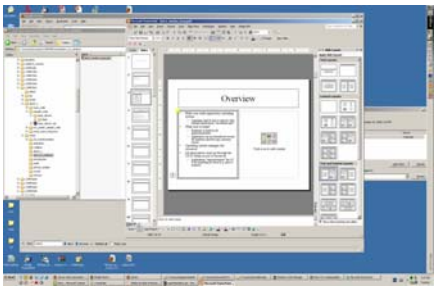
Objectives

- Understand the mechanism of window programming
- Understand the concept and usage of of callback functions
- Create a simple application

- Overview
- Windows system
- “Hello world!”

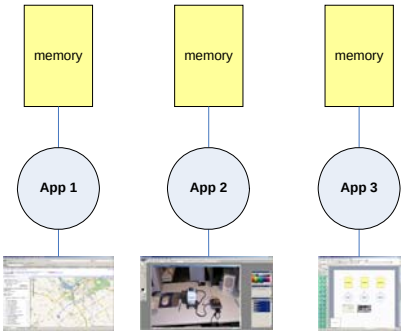
Overview

- Multi-user multi-application operating system
  - Operating must be able to interact with multiple applications “simultaneously”
- What does it mean?
  - Hardware is shared by all applications/users
  - Application cannot (should not) access the hardware directly (cpu, memory, screen)



Overview

- Operating system manages the resources
- All applications must interact with the OS to obtain access to resources
  - Applications "instruct/request" the OS to do something for them (e.g. show a window)

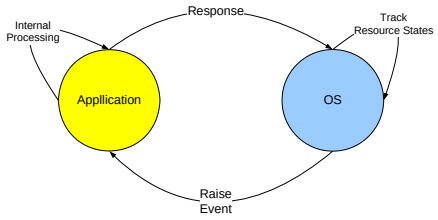


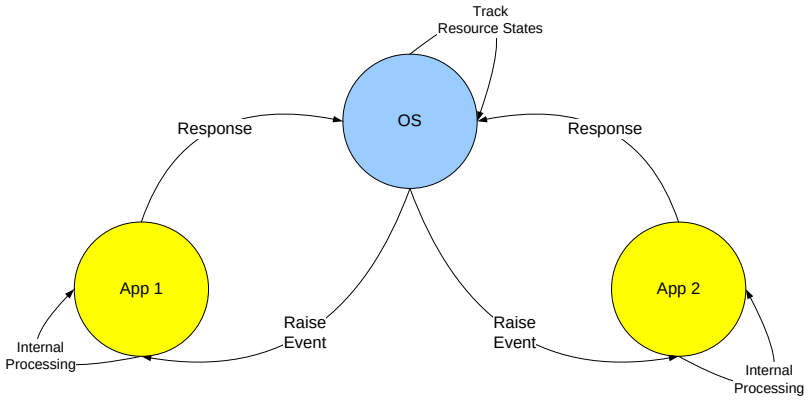
OS Resource Management

- How does an application know that a user has pressed the left mouth button?
- How does the application know that the user has scrolled the mouth over one of its windows?
- OS controls (manages) the resources
  - OS tracks the different resources and their state
  - OS determines whether a change in state has the potential to affect an application, and
  - if so informs the application

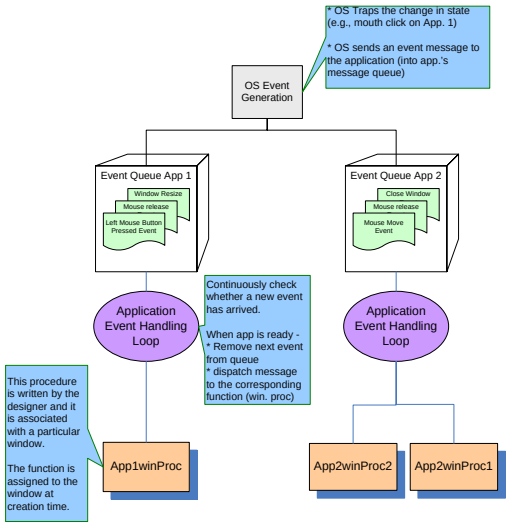
Event Driven Programming

- Application
  - waits until an event (related to the application) occurred
  - Reacts to the event (if it knows about the event)
  - Events types – window is visible, window was resized
- OS
  - Tracks the state change
  - Determine if it is related to an application
  - Alerts the application by sending it an event

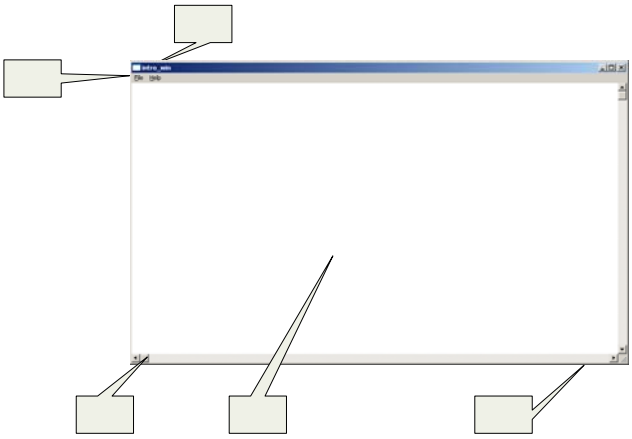




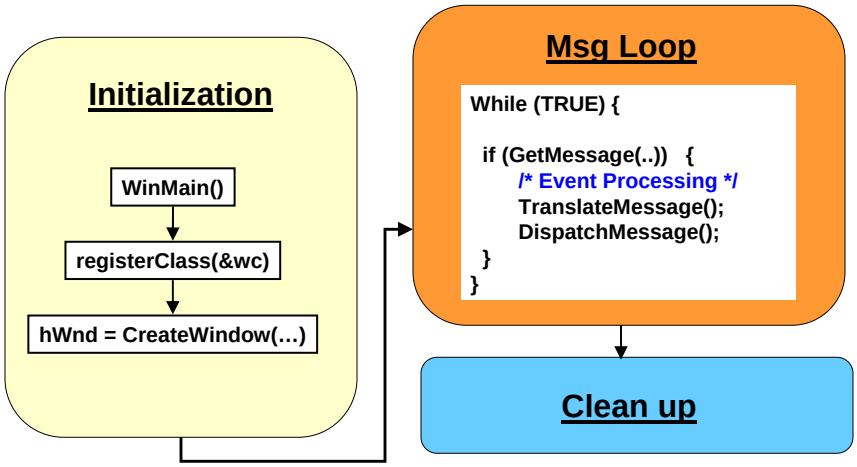
Event Based Programming



Window Components



Window Program Overview



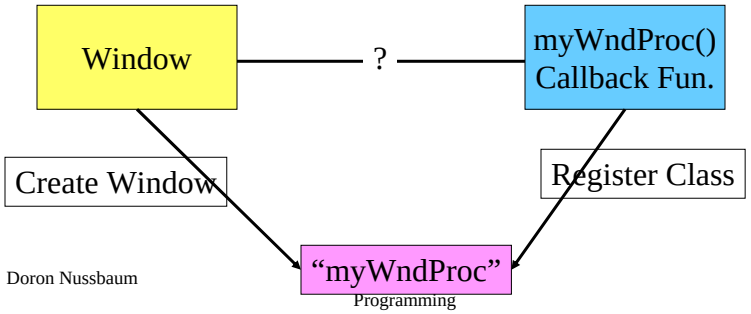
The main() function in Win32

```
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow);
```

- "WinMain()" replaces the "main()" function in C/C++
- Windows entry point for initializing the program WinMain()
  - hInstance: a handle for this application
  - hPrevInstance: for pre-32 bit apps that use the same address space for the same app
  - lpCmdLine: command line arguments
  - nCmdShow: directive for controlling the window appearance (can be ignored in our case

Event Driven Programming (Win32)

- OS controls the activities (multi tasking)
- As needed the OS invokes the application
  - Requires an entry point ➔ a callback function



Create A Window

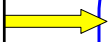
```
wc.lpfnWndProc   = myWndProc;
wc.winClassName = “myGame”;
wc.style = CS_HREDRAW | CS_VREDRAW;
...
registerClass(&wc);

hWnd = CreateWindow(NULL, “myGame”,
                    "First D3D Program”,
                    WS_OVERLAPPEDWINDOW,
                    0, 0, 300, 400,
                    NULL, NULL,
                    hInstance, NULL );
```

The entry point into program (callback function)

- Create the Window Class by registering `wc` class
- Create a Window using `CreateWindow()`
- Return a handle for this newly created window

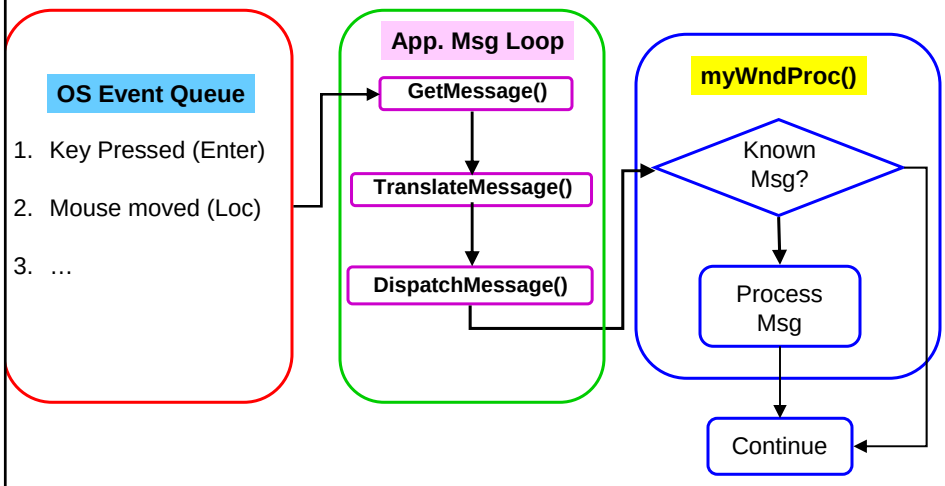
```
WNDCLASS wc;  // the windows class structure

wc.style = CS_HREDRAW | CS_VREDRAW;  // redraw behaviour attribtes
wc.cbClsExtra     = 0;
wc.cbWndExtra = 0;
wc.hInstance      = hInstance;                  // application instance
wc.hCursor = LoadCursor(NULL, IDC_ARROW);       // the cursor type
wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1); // bg colour of client rea
wc.lpszMenuName = NULL;           // no menu
wc.lpfnWndProc = myWndProc;    // the callback win.  proc.
wc.lpszClassName= "myGame";    // class name - links window and callback
wc.hIcon = NULL; // an icon of the class (if NULL➔ system uses the default)


RegisterClass(&wc) ;
```

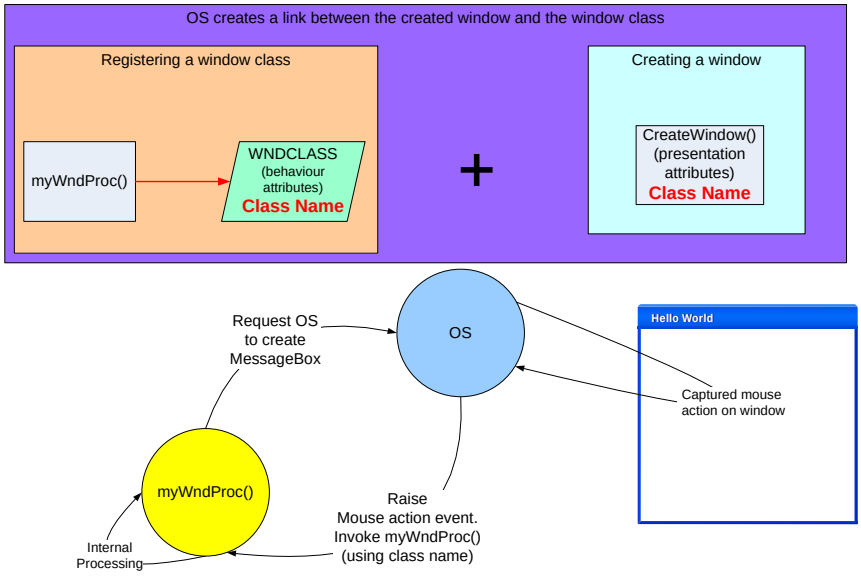

Create Window Function

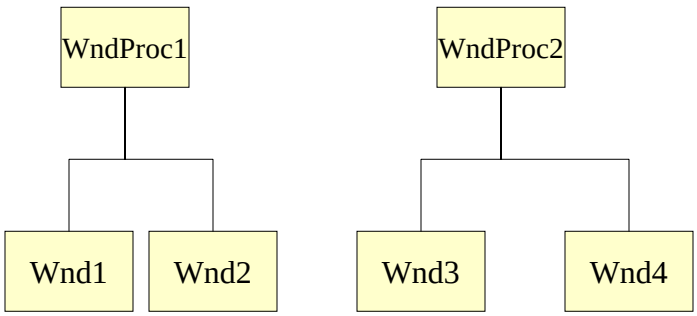


```
HWND CreateWindow(
ClassName,      // window class name (link to win proc)
WindowName,  // the window name (title)
Style,          // the window style
x, y,           // the window location on the screen
Width,Height,   // the window size
WndParent,      // the parent window
Menu,           // the menu associated with the window
Instance,       // the application instance
Param           // not used
);
```

Handling Windows Events and Messages








```
int executeMsgLoop(void)
{

    int rc = 0;
    MSG msg;



    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return ((int) msg.wParam);
}
```

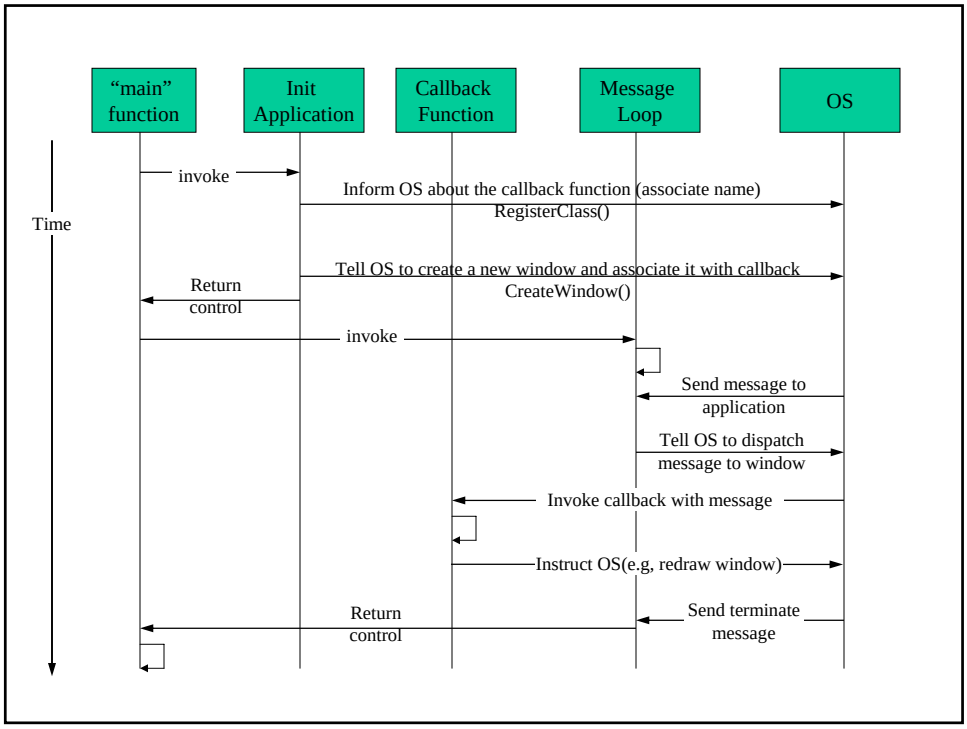
Listen to messages

Ask OS to forward message to window

```
LRESULT CALLBACK myWndProc(
                        HWND hWnd, // handle to the window
                        UINT msg,  // the message that was sent
                        WPARAM wParam,
                        LPARAM lParam)
{
    switch (msg) {

    case WM_DESTROY:
          PostQuitMessage(0);
          break;

    default:
          return DefWindowProc(hWnd, msg, wParam, lParam);
    }
    return 0;
}
```



```
switch (msg) {
case WM_LBUTTONDOWN:
        ::MessageBox(NULL, "Hello World!", "My Hello App", MB_OK);
        break;

case WM_RBUTTONDOWN:
        char s[128], char title[64];
        int xPos, yPos;
        xPos = GET_X_LPARAM(lParam);
        yPos = GET_Y_LPARAM(lParam);
        sprintf(title,"Hello Cursor!");
        sprintf(s,"Hello World! \n The cursor position is (%d, %d)",xPos, yPos);
        ::MessageBox(hWnd, s, title, MB_OK); // show a message box
        break;

case WM_DESTROY:
        PostQuitMessage(0);
        break;

default:
        return DefWindowProc(hWnd, msg, wParam, lParam);
```


"main" function
Init Application
Callback Function
Message Loop
OS
Time
invoke
Inform OS about the callback function (associate name)
RegisterClass()
Tell OS to create a new window and associate it with callback
CreateWindow()
Return control
invoke
Send message to application
Tell OS to dispatch message to window
Invoke callback with message
Instruct OS(e.g, redraw window)
Send terminate message
Return control