G52OSC
OPERATING SYSTEMS AND CONCURRENCY

Creating Windows Programs
Message/Event Loops

Dr Jason Atkin

Last lecture

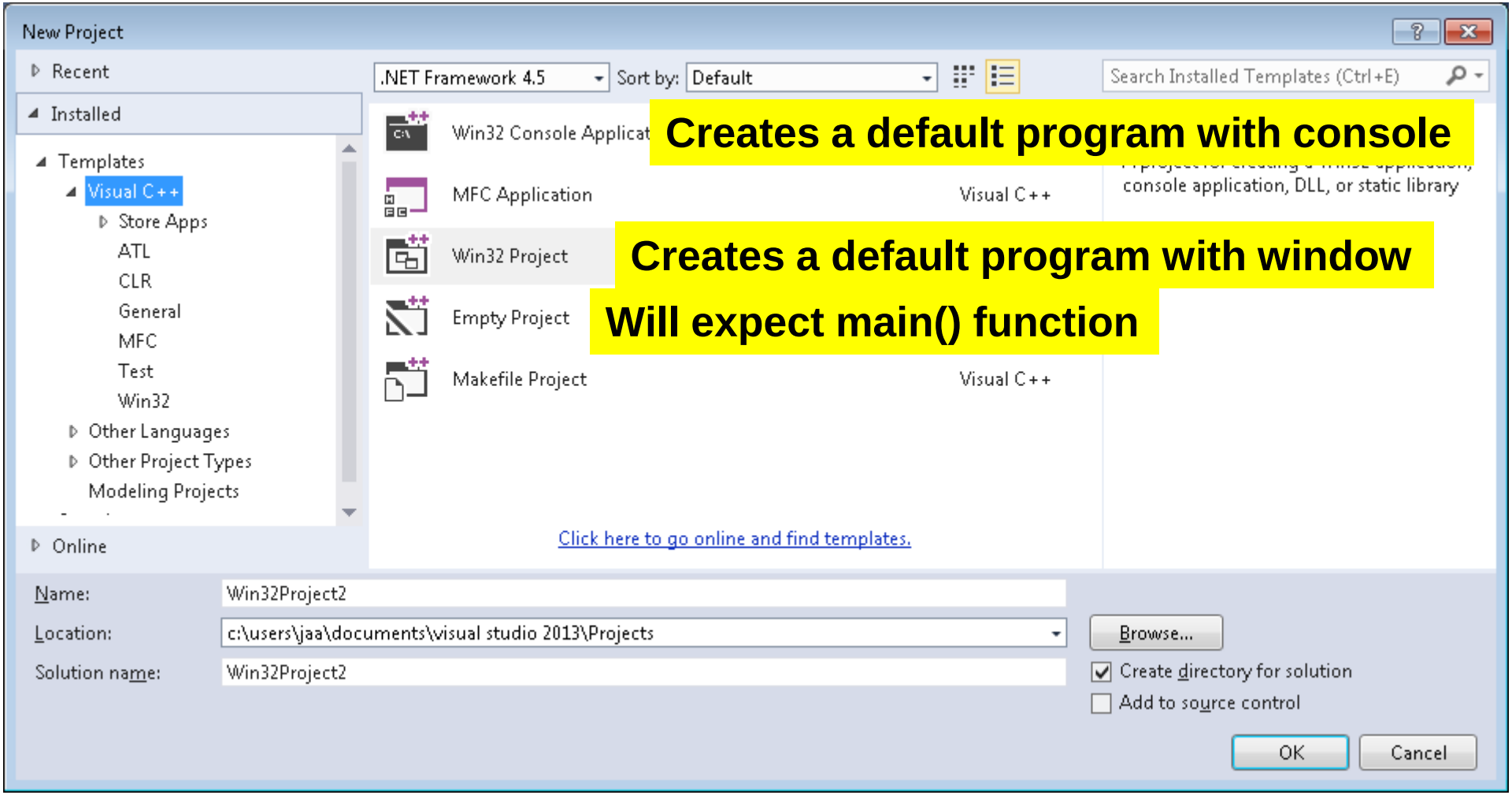
- Processes vs threads
  - Separate address space/resources/data vs Shared address space/resources/data
- Linux:
  - fork()
  - pthread_create(…, function, params)
- Windows
  - CreateThread(…, function, params, …)
  - CreateProcess : later this lecture

This Lecture

- Working with the Windows Operating system : see what it needs/uses
- Windows Create Process
- Creating a windows program
  - Register window class
  - Create window
  - Message loops
  - Windows Procedure
- Message boxes
- Send/Post message

Creating projects in Visual Studio

Creating a project



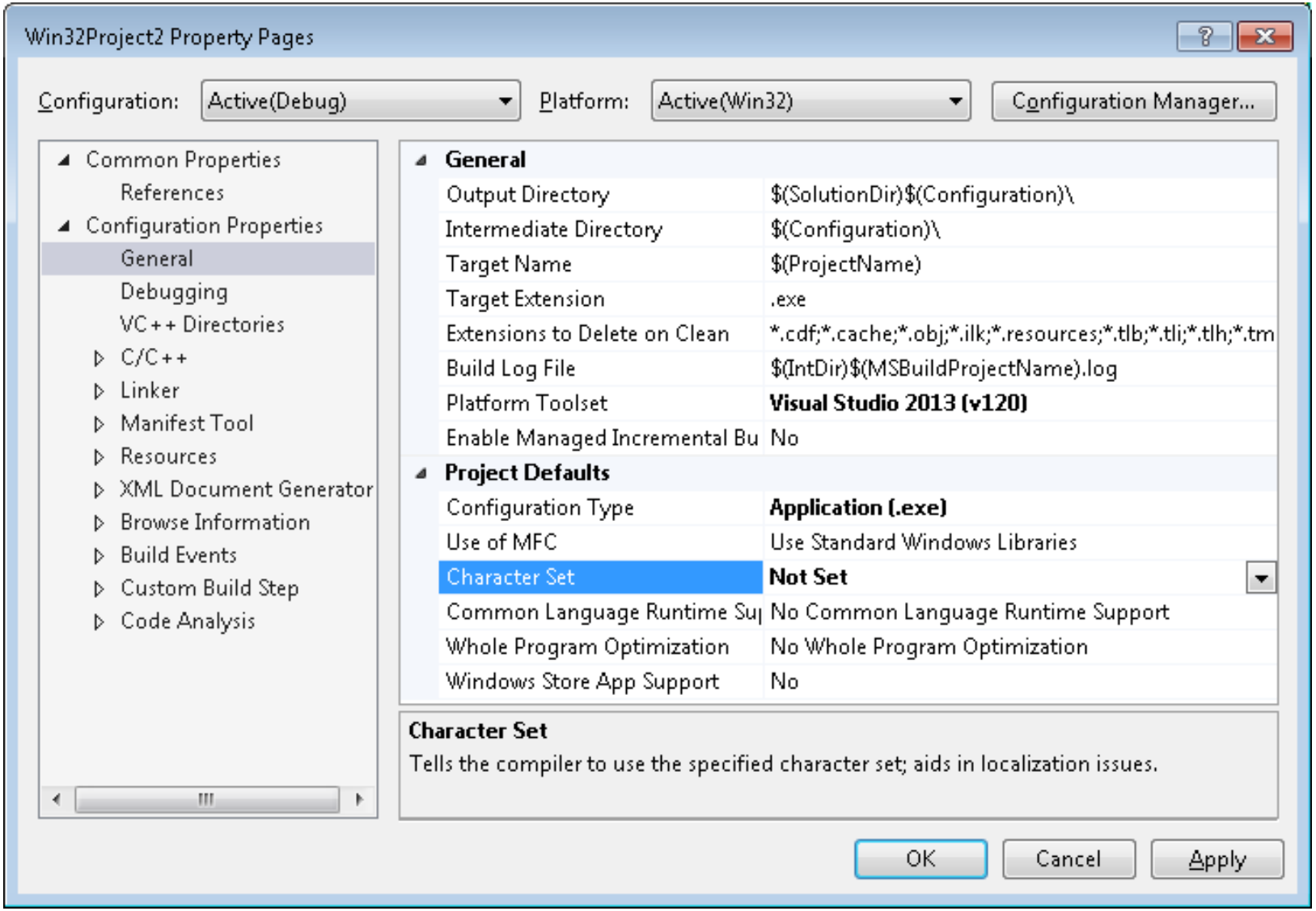
Creates a default program with console

Creates a default program with window

Will expect main() function

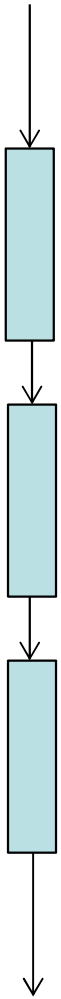
Sometimes a Win32 project with 'empty project' selected is also useful

Change character set to ASCII

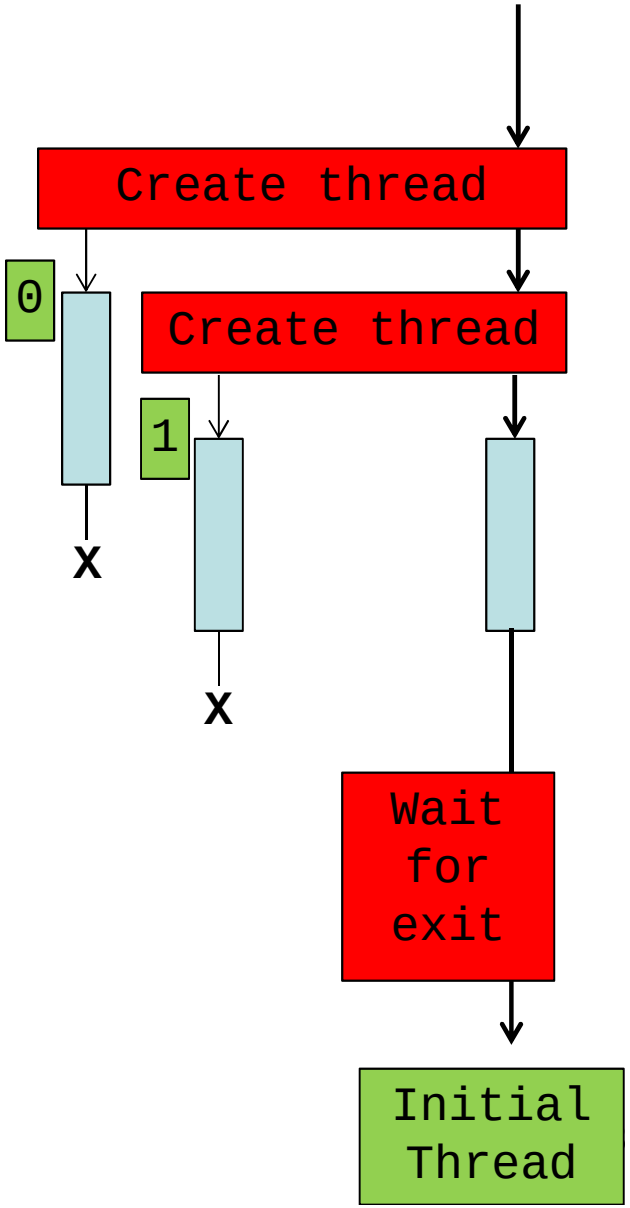


Threading code
from last lecture

Parallelising the code (3 times)



- We could make these simultaneous
- By creating multiple threads
  - Create them
  - Run one function call in each
  - Wait for them to exit



Pre-amble

```
#define WIN32_LEAN_AND_MEAN
#include <Windows.h>
#include <stdio.h>
#include <stdlib.h>

#define NUM_THREADS 3

volatile DWORD dwTotal = 0;
```

Purpose:
Each thread will increase the value of this variable by one million.

The thread function

```
DWORD WINAPI thread_function( LPVOID lpParam )
{
   for ( int j = 0; j < 20 ; j++ )
   {
      printf( "Thread %d running %d... total = %d\n",
              (int)lpParam,       j,     dwTotal );
      for ( int i = 0; i < 1000000/20; i++ )
         dwTotal++;
   }
   printf( "Total when thread %d ended was %d\n",
         (int)lpParam,      dwTotal );
   return 0;
}
```

We use this as a thread number

Outer loop

Inner loop

Increment the variable

Purpose:
Thread function will increase the value of this variable by one million.
Does it in 20 parts so we can see what it does (via printf).

Create the threads

```
int main()
{
   HANDLE arrdwThreadHandles[NUM_THREADS];

   for ( int iTN = 0; iTN < NUM_THREADS - 1; ++iTN )
      arrdwThreadHandles[iTN] = CreateThread(
            NULL, /* No security change */
            0, /* Default stack size */
            thread_function, /* Name of function to call */
            (LPVOID)iTN, /* parameter you can give */
            0, /* Extra flags */
            NULL /* You can get the thread id if you wish */
      );

   /* Do the last one in the current thread */
   thread_function( (LPVOID)(NUM_THREADS - 1) );
```

Wait and then exit

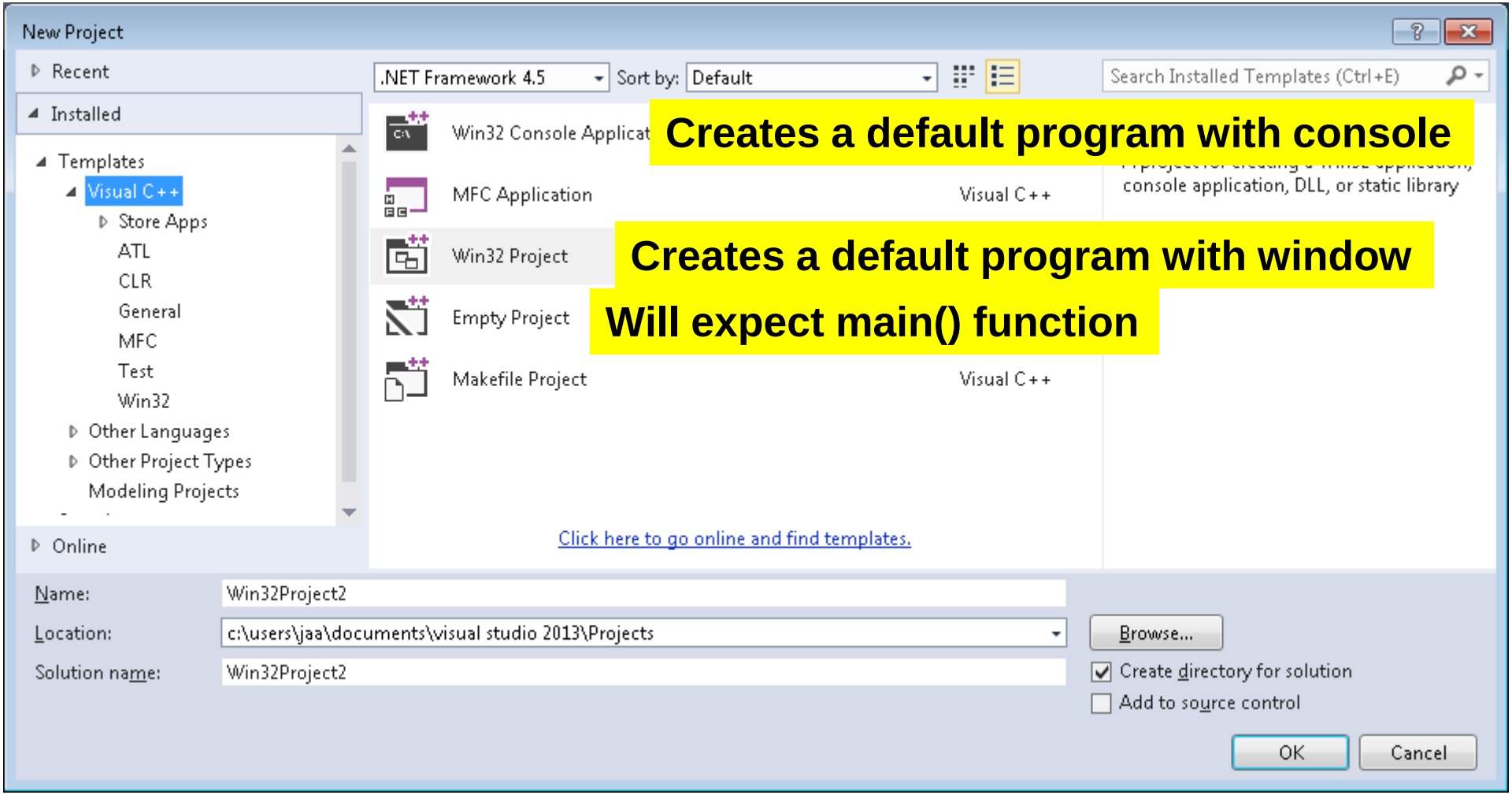
```
    WaitForMultipleObjects(
          NUM_THREADS - 1,   /* Number elements in array */
          arrdwThreadHandles, /* Array of handles */
          TRUE, /* Wait for all rather than just one */
          10000 ); /* Wait for up to 10 secs */

    /* Code to keep the window open until you press ENTER */
    printf( "Press RETURN" );
    while ( getchar() != '\n' )
          ;
    return 0;
}
```

Creating windows

Using the code which gets created for us as an example

Creating a project



Creates a default program with console

Creates a default program with window

Will expect main() function

Sometimes a Win32 project with 'empty project' selected is also useful

WinMain

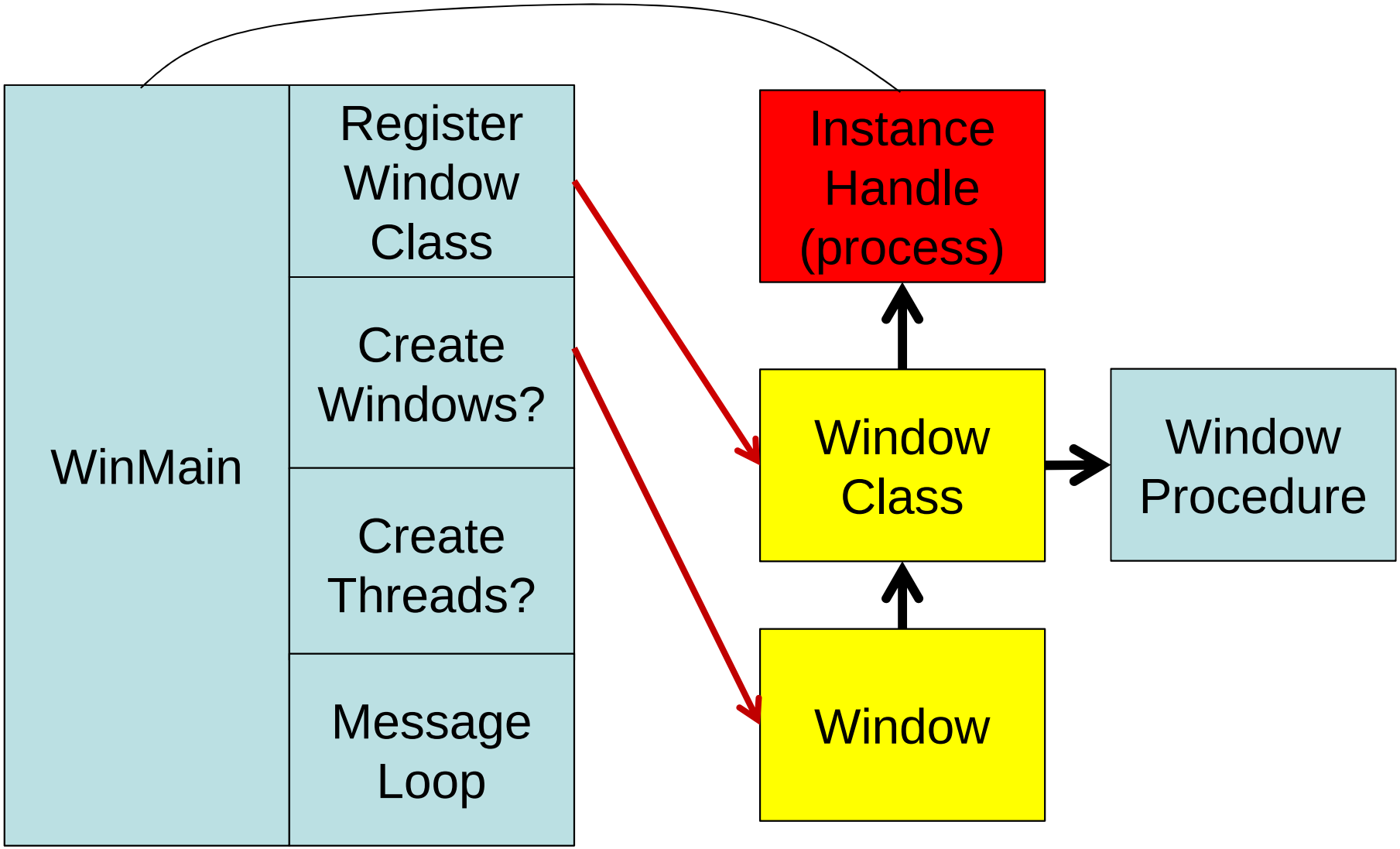
```
int APIENTRY WinMain(
    HINSTANCE   hInstance,        // Own handle
    HINSTANCE   hPrevInstance, // Parent
    LPTSTR      lpCmdLine, // Command line
    int         nCmdShow) // Show or hide
```

- Replaces the main() function for windows programs
- Useful to have hInstance

HINSTANCE

- Handles refer to operating system resources
- We can refer to the current instance/process
- HINSTANCE for current instance
  - Passed in to the WinMain function
  - Or use GetModuleHandle(NULL)
- Windows each have handles
- As do threads, events, and the things we use for locking (e.g. mutexes)

Overview


WinMain
Register Window Class
Create Windows?
Create Threads?
Message Loop
Instance Handle (process)
Window Class
Window Procedure
Window

A window class (extended)

```
WNDCLASSEX wcex;
   wcex.cbSize = sizeof(WNDCLASSEX);
   wcex.style= CS_HREDRAW | CS_VREDRAW;
   wcex.lpfnWndProc= WndProc; // Callback function
   wcex.cbClsExtra= 0;
   wcex.cbWndExtra= 0;
   wcex.hInstance= hInstance; // Handle to our process
   wcex.hIcon= LoadIcon(hInstance,
            MAKEINTRESOURCE(IDI_WIN32PROJECT2));
   wcex.hCursor= LoadCursor(NULL, IDC_ARROW);
   wcex.hbrBackground= (HBRUSH)(COLOR_WINDOW+1);
   wcex.lpszMenuName= MAKEINTRESOURCE(IDC_WIN32PROJECT2);
   wcex.lpszClassName= “Your name for the class”;
   wcex.hIconSm= LoadIcon(wcex.hInstance,
            MAKEINTRESOURCE(IDI_SMALL) );
RegisterClassEx(&wcex);
```

WNDCLASSEX structure

- See https://msdn.microsoft.com/en-us/library/windows/desktop/ms633577%28v=vs.85%29.aspx

- There is also a WNDCLASS structure, just choose one

- The **WNDCLASSEX** structure is similar to the **WNDCLASS** structure.
- There are two differences:
  - **WNDCLASSEX** includes the **cbSize** member, which specifies the size of the structure
  - and the **hIconSm** member, which contains a handle to a small icon associated with the window class.

WNDCLASSEX struct

```
typedef struct tagWNDCLASSEX {
   UINT       cbSize;        // sizeof(WNDCLASSEX)
   UINT       style;         // e.g. when to redraw
   WNDPROC    lpfnWndProc;   // Your WndProc function
   int        cbClsExtra;    // Extra bytes per class
   int        cbWndExtra;    // Extra bytes per window
   HINSTANCE  hInstance;     // Instance with the WndProc
   HICON      hIcon;         // Choose an icon for it
   HCURSOR    hCursor;       // Cursor when over window
   HBRUSH     hbrBkground;   // Background fill?
   LPCTSTR    lpszMenuName;  // Name of menu resource
   LPCTSTR    lpszClassName; // Unique name for this class
   HICON      hIconSm;       // Small version of icon
} WNDCLASSEX;
```

Creating a window

```
HWND hWnd = CreateWindow(
      "Name of my Window Class",
      "Title for the window",
      WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT,
      0, NULL, NULL,
      hInstance,
      NULL );
if (!hWnd)
      return FALSE;
ShowWindow(hWnd, nCmdShow);
UpdateWindow(hWnd);
```

Matches the name of the class that you registered

CreateWindow

```
HWND WINAPI CreateWindow(
    LPCTSTR lpClassName,  // Name of class
    LPCTSTR lpWindowName, // Title bar text
    DWORD dwStyle,        // WS_OVERLAPPEDWINDOW
    int x,                // Initial position
    int y,
    int nWidth,           // Initial size
    int nHeight,
    HWND hWndParent,      // Parent window
    HMENU hMenu,          // Menu to use
    HINSTANCE hInstance,  // Current process
    LPVOID lpParam        // For WM_CREATE
);
```

A message loop

```
MSG msg; // Structure to receive message

// Loop getting the next message
// Params allow filtering to only get some
while (GetMessage(&msg, NULL, 0, 0) > 0)
{
     // Convert virtual keystrokes
     TranslateMessage(&msg);
     // Send message to window procedure
     DispatchMessage(&msg);
}
```

Window Procedures

```
LRESULT CALLBACK WndProc(
        HWND hWnd,          // Window message is for
        UINT message,       // The id/type of message
        WPARAM wParam, LPARAM lParam) // Parameters
{
        switch (message)
        {
        case WM_PAINT: // Draw the window contents
                break;
        case WM_DESTROY: // Window destroyed
                PostQuitMessage(0);
                break;
        }
        return 0;
}
```

Example: Painting the window

```
PAINTSTRUCT ps;    // Details of what to paint
HDC hdc;           // Handle to a device context

case WM_PAINT:
      hdc = BeginPaint(hWnd, &ps); // Get started
      // Draw some shapes
      Rectangle( hdc,100,100,200,300 );
      Ellipse( hdc,100,100,200,300 );
      // Simpler in ASCII, draw some text
      TCHAR szMessage[] = _T("Test message");
      UINT  nLen = _tcslen( szMessage );
      TextOut( hdc,100,325,szMessage,nLen );
      // Finished so release the HDC and tell system
      EndPaint(hWnd, &ps); // MANDATORY!!!
      break;
```

MessageBox

- You can easily create a message box

```
int WINAPI MessageBox(
        HWND hWnd,              // Parent
        LPCTSTR lpText,         // Message
        LPCTSTR lpCaption,      // Caption
        UINT uType );           // e.g. MB_OK
```

- MessageBox(NULL, “Text”, “Title”, MB_OK);
- Try MB_OK, MB_OKCANCEL, MB_YESNO
- Returns IDOK, IDCANCEL, IDYES, IDNO

PostMessage, SendMessage

```
BOOL WINAPI PostMessage(
    HWND hWnd,      // Window to send to
    UINT Msg,       // ID of message to send
    WPARAM wParam,  // Parameters
    LPARAM lParam );
```

- Send a message to a window – puts it in the queue
  - GetMessage will then pick it up eventually
- SendMessage will do the same thing but will wait for the message to be handled
  - In same thread it will call the window proc directly
  - In another thread it will wait for that thread to handle the message

Other functions

- There are a lot of other functions which can be used
- Experiment and search the MSDN
- More information about many things:

  https://msdn.microsoft.com/en-us/library/windows/desktop/ms632586%28v=vs.85%29.aspx

Combine with threads?

- We could consider the message loop system as a cooperative multi-tasking environment
  - Time-sharing between windows
  - Windows/message handlers will not pre-empt / interrupt each other
  - System can interrupt and task-switch though
- Message boxes (and modal dialogs) have their own message loops inside them
- Some **important** things to consider:
  - The thread which created the window will receive the messages for the window
  - You MUST run the message loop within that thread

Create Process

Simple example

```
STARTUPINFO info = { sizeof( info ) }; // Input
PROCESS_INFORMATION processInfo; // Output
if ( CreateProcess(
   _T("C:\\Windows\\System32\\cmd.exe"), // Program
   _T(""),   // Command line
   NULL,NULL,TRUE,0,NULL,NULL,
   &info,
   &processInfo ) )
{
   /* Wait for process to finish – optional */
   ::WaitForSingleObject( processInfo.hProcess,
            INFINITE ); // Wait forever
   CloseHandle( processInfo.hProcess );
   CloseHandle( processInfo.hThread );
}
```

Windows CreateProcess

https://msdn.microsoft.com/en-us/library/windows/desktop/ms682425%28v=vs.85%29.aspx

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

Event loops elsewhere

Equivalent on X (Linux/Unix)

```
done = 0;
while (done==0)
{
        XNextEvent(mydisplay,&myevent);
        switch(myevent.type)
        {
        case Expose: /* Repaint window on expose */
                break;
        case MappingNotify:
                break;
        }
}
```

- **See also:**

  http://www.cs.colorado.edu/~mcbryan/5229.03/mail/35.htm

  http://en.wikipedia.org/wiki/Event_loop

And for many other OSes…

- Similarly for OS-X:
  - https://developer.apple.com/library/mac/documentation/General/Devpedia-CocoaApp-MOSX/MainEventLoop.html

- And for Java:
  - http://en.wikipedia.org/wiki/Event_dispatching_thread
  - http://docs.oracle.com/javase/tutorial/uiswing/concurrency/dispatch.html
- Note: Swing for Java is not thread safe
  - ALL drawing needs to happen in the event dispatch thread – not just any thread
  - Provides invokeLater and invokeAndWait to do this

Summary

- Many operating systems have some kind of event queue and event handling system
  - Pick up events one at a time and handle them
- The windows API allows us to see this explicitly
  - GetMessage(), TranslateMessage(), DispatchMessage()
- We register a callback function (a “window procedure”) for a window class, to receive the messages for windows of that class/type
- The thread which created the window picks up messages one at a time and ‘dispatch’es them to the window procedure

Next Lecture

- Using our multi-threaded programs
- Sharing data
- Coordinating between threads
- Avoiding concurrent updates