Understanding the Message Loop

Studying the messages I've used so far I can see the HEX values

```
#define WM_LBUTTONDOWN          0x0201
#define WM_CLOSE                0x0010
#define WM_DESTROY              0x0002
```

Meaning that

```
case 0x0201: // is the same as  case WM_LBUTTONDOWN:
{
...
}
```

💡 The prefix WM stand for General Message category, but there are many others. See **System-Defined Messages**

💡 I can create my own messages if I stay within the 0x0400 to 0x7FFF range

Messages can be manually sent with functions like

`PostMessage(hwnd, WM_CLOSE, 0, 0);`

In this example we'll have the same effect as clicking[X] on the window.
(`SendMessage()` can also be used)

A little more in depth

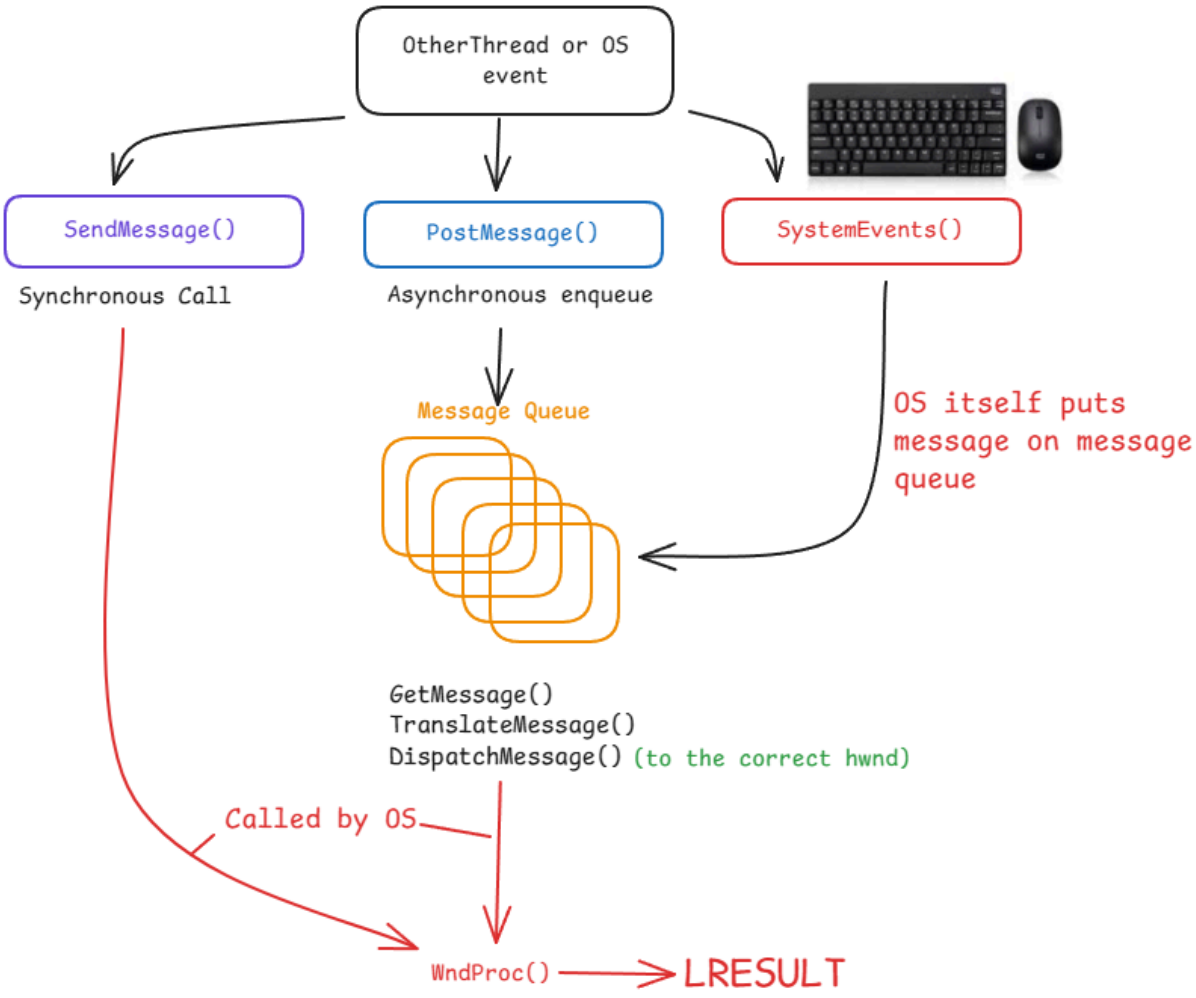
SendMessage() (Synchronous)

Calls the window procedure and does not return until the window has processed the message.

PostMessage() (Asynchronous)

Places (posts) a message in the message queue associated with the thread that created the specified window and returns without waiting for the thread to process the message. returns without waiting for the thread to process the message

The flow so far


OtherThread or OS event
SendMessage()
PostMessage()
SystemEvents()
Synchronous Call
Asynchronous enqueue
Message Queue
OS itself puts message on message queue
GetMessage()
TranslateMessage()
DispatchMessage() (to the correct hwnd)
Called by OS
WndProc()
LRESULT