Lecture notes: GUI Event Driven Programming (continued)

User interaction with a GUI program is asynchronous and not predictable. If the program is intended to do work on a regular basis while still being responsive to GUI driven events, then it needs to use a method that allows for multitasking. An example of this is a video player, where the program needs to continuously display images to the screen while staying responsive to potential user GUI commands to pause, rewind, change the volume, etc.

A **timer** is a construct that can be used to generate an event on a regular interval. The event is handled similar to GUI events, except that it is generated on a regular basis. It can be envisioned as though the program user were regularly clicking a button on the GUI over and over to drive some action. Timers can be provided by hardware, such as in a processor, but are also commonly provided in software in GUI programming environments.

There are three steps to using a timer.

1. First, it must be created. During creation it is given a name (or ID) and interval. Upon creation it starts counting from zero up to the given interval. When it reaches the interval it generates a timer event. It then starts over counting from zero again.
2. Second, the timer-generated event must be handled. This is where the work that needs to be done regularly is accomplished, such as displaying a frame of video.
3. Third, the timer must be halted when its function is no longer needed.

The “plus” program at the course website demonstrates a timer using Win32 programming. The keypress “1” creates the timer, the keypress “2” halts the timer, and the WM_EVENT block of code shows the event handler. Note that this event handler is very simple. It could check for the ID of the timer generating the event, and take different actions for different timers.

Another method for multitasking in a GUI program is to use **multiple threads**. The main thread is typically used for the event loop, while one or more additional threads are created to perform other work. For example, a video playing program could use a child thread to display images on a regular basis in the screen, while the main thread runs the event loop to remain responsive to GUI events.

The plus program demonstrates the use of a child thread for multitasking. The keypress “3” creates a child thread, and the keypress “4” ends the child thread.

Important: In order to use multithreading, the project settings must be configured under C/C++ -> code generation -> use run-time library. The default is single-threaded; this must be changed to multi-threaded. This has already been configured for the plus program.

Both these methods for multitasking can be used simultaneously. This can be demonstrated in the plus program by pressing “1” and “3” and having the timer and child thread run at the same time. While both these methods are working, the GUI is still responsive. This can be further demonstrated by turning on the display of pixel coordinates, so that three pieces of work are being done simultaneously.

An event loop can also be coded in a child thread for use by a sub-function of the program. This is commonly done for a dialog box or form. GUI functions that create dialog boxes will commonly spawn the child thread, manage it, and then end it.

The last topic for GUI programming is widgets. Widgets are elements of a GUI that are used to obtain user input, such as a dialog box. In class, I will demonstrate the creation of widgets using MS Visual Studio, and the format of them in a “resource file”.