ECE 3574: Event Driven Programming

Changwoo Min

Administrivia

- **Milestone 1: due Monday**
  - From Sunday, the night builder will compile in strict mode
  - Need *five or more git commits* to prove *good development practice*
  - Don't forget to tag `final`
  - See thegrading policy
- **Extended instructor's office hour**
  - tomorrow: 2 - 5 PM

Review Exercise 11

- See code

Event Driven Programming

- Today we will learn how to design systems that respond to internal and external events in an application
  - Events and Event Handlers
  - Events from Windowing System
  - Timers and other internal events
  - Observer Pattern
  - Callbacks versus Polymorphism for implementing event handlers
  - Qt Event System
  - Exercise

Events are inputs that are not predictable from the program flow

- Examples:
  - Hardware Event: the user presses a key on a keypad
  - Software Event: the user clicks on a button in a windowing system
- The program should be able to respond to these events, i.e handle them, whenever they occur.

Typically events are collected in an event loop using polling

- Round-Robbin

```
while(true){
    // check status of switch
    // handle if changed
    // ... etc.
}
```

Typically events are collected in an event loop using polling

- Queuing, or *posting* the event (like onto a bulletin board)

```
while(true){
    // check status of switch
    // post the event, queue it to be handled
    // ... etc.
    // handle N events from the queue
}
```

The code that is run in response to an event is a handler

- The handler should:
  - do the minimum amount of work possible
  - never block execution
- Otherwise the system lags to input or locks up and does not respond to events.

How much work can be done?

- Each iteration of the event loop should be limited in time.
- How much depends on the application
  - in a user interface around 250ms (?)
  - in a control loop, perhaps as little as a 1ms (?)
- Add up the total number of events and the time to execute each

How does one do more work in a handler?

- Concurrency, let the OS handle it (see lectures 18-27)
- Split work into small chunks, post an event itself
- Implement a coroutine, a function that can be restarted where it left off (not discussed)

Examples of Events from a Windowing System

- show/draw/render the object
- focus the object
- mouse enter/leave
- mouse down, up for left, right, middle, etc
- key K press/release
- resize object
- move object
- gestures

Examples of internal events

- timers
- events posted by other handlers

Event systems are an example of the Observer Pattern

- Observers are objects which observe other objects. Possible implementations:
  - callback functions
  - dynamic polymorphism (inheritance)
- See example code

Exercise

- See website

Further readins

- Observer pattern
- Understanding and Implementing Observer Pattern in C++

Next Actions and Reminders

- Read about Qt Signals and Slots
- Reminder: Milestone 1 is due next Monday.