ECE 3574: Signals and Slots

Changwoo Min

Project milestone

Milestone	Duration	Points	Instructor's effort	Average student's effort
Milestone 0	3 weeks	20	1 hour	2 days (?)
Milestone 1	**3 weeks**	**48**	**8 hours**	**2 weeks (?)**
Milestone 2	4 weeks	70	1 week (?)	4 weeks (?)
Milestone 3	?	?	?	?
Milestone 4	?	?	?	?

Milestone 2

- **The most challenging milestone in this semester**
    - Develop the simulator for our system and a text-mode interface
    - Due: 3/26
    - Specification
- **Start today**

How to debug in Linux: use `gdb`

- `gdb` : text mode debugger in Linux

```
$> mkdir build
$> cd build
$> cmake -DCMAKE_BUILD_TYPE=Debug ..  # debug build for debugging
$> make
$> gdb --args ./unit_tests "[parser]" # run gdb for a command
gdb> b parser.cpp:100                 # set a breakpoint at Line 100 in parser.cpp
gdb> run                              # actually run ./unit_tests
gdb> # When the break point his, press Ctrl-x Ctrl-a to see the source code
gdb> n                                # next
gdb> s                                # step into
gdb> p VAR                            # print VAR
gdb> p *ADDR                          # print the contents at ADDR
gdb> quit                             # quit
```

- See gdb cheatsheet

Signals and Slots

- Today we will learn about a variation of the Observer design pattern that is used prominently within Qt, called signals and slots.
    - Observer and Publish/Subscribe Pattern
    - Observers as callback functions
    - Observers using signals
    - Qt signals
    - Examples
    - Exercise

Observer design pattern

- Also known as *publish/subscribe design pattern*
- A way to communicate among objects without them knowing much about one another.
- Recall the notion of an event handler.
    - To call the event handler we need a pointer or reference to the object handling the event
    - This is an example of a callback function
- A callback is simply a pointer to a function.

Example 1: a simple callback function

- See `callbacks.cpp`
- See `std::function`

Example 2: using a member function as a callback

- See `callbacks_methods.cpp`
- See Bind function and placeholders in C++
    - See `std::bind` and `std::placeholders`

There are drawbacks to callbacks as illustrated in Example 1 and 2

- They represent a one-to-one communication
- The communication is always-on
- Fixing this requires a good deal of effort to manage the callback connections
    - make the callback a list of callbacks
    - call each callback in the list
- Factoring this code out into a library results in managed callbacks, or *signals and slots.*

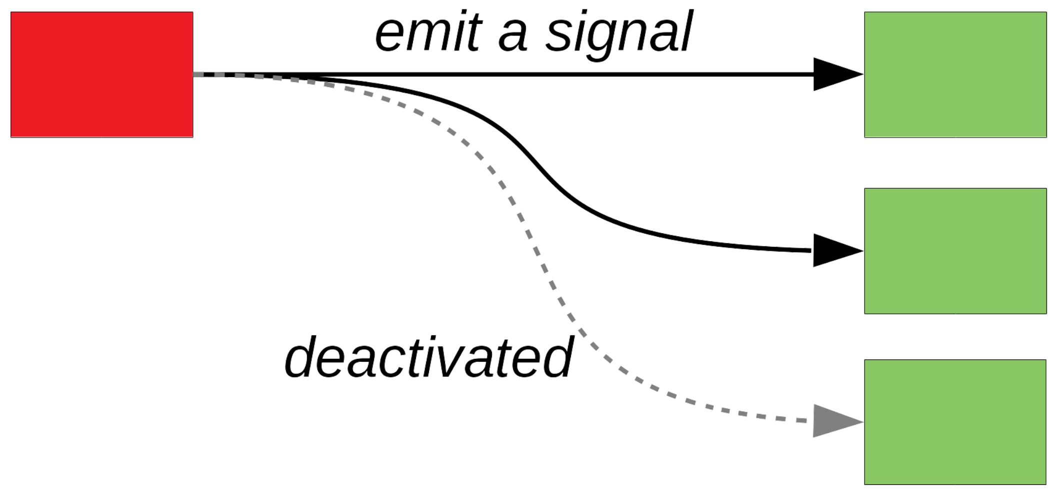
Signals and Slots

- *Signals (publishers)* are callbacks with multiple targets or *slots (receivers or subscribers)*.
- Signals are *connected* to slots
- Signals are *emitted*
- Slots connected to a signal are called when the signal is emitted
- This raises an important issue, how are return values from slots used?
    - Some systems do not use them (`Qt`)
    - Other systems provide a way to aggregate them (`boost::signals`)

Signals and Slots

Signal
: event, publisher

Slots
: event handler, subscriber



C++ libraries that provide a signal/slot mechanism

- `Boost` is a very popular collection of C++ library that provides `boost::signal`.
- `POCO` is another popular collection that provides an event system that works like signals/slots.
- Qt has a signals and slots mechanism implemented as an extension of C++.

Qt signals and slots extend the syntax of C++

- Every class that wants to communicate via signals and slots must derive from `QObject` directly or indirectly (derive from a subclass of `QObject`)
- The class should have the macro `Q_OBJECT` in its private section.
- slots are defined in a private, protected, or public section called slots and implemented
- signals are defined in a section called signals, but **not** implemented

Qt signals and slots extend the syntax of C++

- signals are emitted using the keyword `emit`
- connections are made using the `QObject::connect` function.
- The connections between signals and slots can be synchronous or queued.

An Example: a settings widget

- See `qtmain.cpp`, `receiver_object.*`, `settings_widget.*`, and `settings.h`.

Exercise

- See website
- See QRadioButton
- See QTimer

Next Actions and Reminders

- Read about integration tesing with QtTest
- **Start Milestone 2 today**