COP 5615: Distributed Operating Systems Principles

Fall 2016

Professor Sumi Helal

Lab 3: Event Processing in Xinu

Due: 1:00pm EST Tuesday October 25, 2016

Background

Modern operating systems often have a mechanism for sending *events* between processes. Events allow a process to send information to other processes in a system asynchronously. While events sound very similar to message passing, the two differ in how they are received. While a process can check for messages and read them explicitly once received, an event triggers a *handler*, or callback function, that had been previously specified by the target process. Events allow a process to receive a notification on a change without explicitly checking for that change.

Event systems have proven especially useful in the field of the Internet of Things. One such system that has seen widespread use is MQTT. MQTT defines a "publisher/subscriber" model, where one or more *subscribers,* processes wanting to receive an event, can listen to a *topic* (a specific event). A *publisher* can then send data under a topic to be received by all subscribers.

In this lab, you will implement an event system in the spirit of MQTT, allowing XINU processes to subscribe to and publish events.

Part One

The three main components of the XINU event system will be **subscribe, unsubscribe,** and **publish.** A process can subscribe to a topic by providing a topic ID and a callback function. Once subscribed, the specified callback will be triggered when another process publishes data to that topic. If a process unsubscribes, its callback will no longer be triggered when data is received for the specified topic. When a process publishes data to a topic, all processes that have subscribed to that topic will receive the data through their callback function.

The above functions will have the following signatures:

- `syscall subscribe(topic16 topic, void (*handler)(topic16, uint32));`
- `syscall unsubscribe(topic16 topic);`
- `syscall publish(topic16 topic, uint32 data);`

Requirements

The following typedef must be defined (in *kernel.h*):

- `typedef uint16 topic16;`

In addition, the following assumptions will be made:

- There are only 256 possible topics. (0 to 255)
- A maximum of 8 processes can subscribe to a single topic.
- When a process terminates, it will unsubscribe from all of its topics.
- Data published to a topic with no subscribers will be discarded.

Implementation Details

The system will require some data structure holding each topic and the handlers registered for it. Consider creating a table of structures for each of the 256 topics, containing pairs of PIDs and function pointers for the subscribers.

Note that when a data is published to a topic, the subscribers likely are not running on the CPU at that moment. How can the callback functions be triggered if this is the case? One way is to have an additional process, a *broker*, handle distribution of the published data. The publish system call would pass off the data to the broker, which would call each handler registered under the topic during its time slice. This is possible in XINU because all processes share the same address space.

Consider the following pseudo code:

```
pid32 broker_pid = create(broker, /* broker arguments */);
resume(broker_pid);

process broker():
      while (true):
            (topic, data) = get_next_pending_publish();
            for each (handler_ptr in get_handlers(topic)):
                  handler_ptr(topic, data);
```

An Example

A complete event system should behave as follows:

```
Process A subscribes to topic 10 with handler foo()
Process B publishes 77 to topic 10
- Function foo() is called with arguments 10 and 77
Process C subscribes to topic 10 with handler bar()
Process B publishes 42 to topic 10
- Function foo() is called with arguments 10 and 42
- Function bar() is called with arguments 10 and 42
Process D publishes 17 to topic 7
- No functions are called
```

Part Two

To allow more fine-grained control over published data, MQTT implements *namespaces* above topics. These namespaces look like file paths, where the directory part is the namespace, and the file part is the actual topic. For example, *house/temperature* and *office/temperature* both contain the same topic name, *temperature*, but are logically separated by the namespace. So, data published to *house/temperature* would not be received by subscribers to *office/temperature*.

Another common feature of namespaces is the ability to use *wildcards*. A wildcard allows a publisher to specify a wider range of subscribers to send data to. Using the above example, if data was published to *+/temperature,* (where *+* is the wildcard symbol) it would be sent to both *house/temperature* and *office/temperature*.

You will implement a simplified version of namespaces in your XINU event system. When subscribing, a process can specify a specific **group** (namespace) in addition to the topic; publishers should then be able to target an individual group when sending data. Additionally, one group will be reserved as a wildcard, sending to it will send to all groups.

Notice that the *topic16* type is 16 bits wide, but the actual topic can only range from 0-255. (8 bits) The upper 8 bits of *topic16* will specify the group of the topic. No new functions are needed to support topic groups.

Requirements

No new functions need to be declared, and the existing syscalls are left unchanged.

The following assumptions will be made:

- There are 256 available groups. (0 to 255)
- Group 0 is the wildcard group.
- Data published to group 0 will be sent to subscribers under groups 1-255.
- Data published to an empty group will be ignored as in part one.

An Example

The event system with groups should behave as follows:

```
Process A subscribes with a topic16 value of 0x013F and handler foo()
Process B subscribes with a topic16 value of 0x023F and handler bar()
Process C publishes data 0xFF to topic16 0x013F
- Function foo() is called with topic16 0x013F and data 0xFF
Process D publishes data 0x7E to topic16 0x003F
- Function foo() is called with topic16 0x003F and data 0x7E
- Function bar() is called with topic16 0x003F and data 0x7E
Process C publishes data 0x00 to topic16 0x113F
- No functions are called
```

Submitting

You should submit your main file, syscall implementations, and any other modified XINU headers/sources. The TAs will use their own *main()* function to compile and test the functionality of your event system, in addition to inspecting your code.