**ECE155: Engineering Design with Embedded Systems** Winter 2013

Lecture 5 — January 15, 2013

*Patrick Lam* *version 1*

## Miscellaneous Comments about Events

Here are a couple notes about events that you might want to know.

**Priorities.** Some problems are more important to deal with right away than others, e.g.:

- your mom calls;
- supper is burning; and
- the laundry is done.

Which would you do first?

What about events with similar priorities? How would you choose? Do you always need to choose?

Last time, I mentioned that we could put events into dispatcher queues. Using the *priority queue* data structure, the dispatcher can efficiently pick out the highest-priority event and handle it first.

**Events and finite-state machines.** One way of implementing functionality in event-driven software is via finite-state machines (FSMs), containing an infinite loop between states. You'll see FSMs in ECE124 this term. FSMs consume events and update system state. You may choose to implement your signal processing code using an FSM.

## Polling versus Interrupts

For most of this course, you're just going to assume that events come at you from nowhere. We'll briefly look at sources of events to get a better understanding of event-driven programming.

One source of events is through polling the sensors every so often; sensor readings then generate events. Or, important events may interrupt the processor to let you know about their existence. It's also possible to use both polling and interrupts at the same time.

### Polling

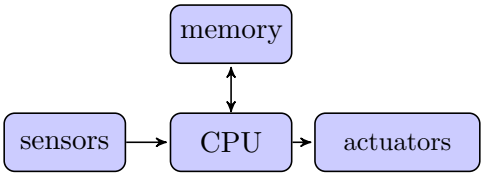
Polling means that the processor requests readings from the device at its convenience. (e.g. what is the current light level?) A polled device does not impose any schedule on the processor, so we can call this *passive synchronization*.

When should a processor check device readings?

- whenever convenient (occasional polling);
- at fixed time intervals (periodic polling);
- constantly (tight polling).

Any of these strategies may be appropriate (with different constants), depending on the importance of the event and the frequencies at which events occur.

**Digression: Input/Output.** Recall this picture from lecture 1:




How does the CPU actually talk to the sensors and actuators? Two methods: 1) memory-mapped I/O and 2) port-mapped I/O, or special instructions. In memory-mapped I/O, the CPU executes what it thinks are reads and writes to memory. In particular, it sends out the appropriate requests on the system bus. Devices listen on the bus and manufacture the appropriate responses. For special instructions, as seen on Intel ia32 processors, the CPU instead executes special `in` and `out` instructions, which may transmit data on a special bus, or set a specific signal on the bus.

**Pseudocode for Tight Polling Loop.** Here is pseudocode for a memory-mapped I/O system.

```
while( statusRegister == 0x0000 ) {
    // Do nothing until statusRegister changes value
}
//  Read data that has changed from a dataRegister and store in memory
incomingData = dataRegister;
```

We'd expect this loop to terminate based on some hardware specification promising `statusRegister` eventually becoming non-zero due to an external event. Data exchange occurs once the device indicates that it is ready to emit data (by setting `statusRegister`); we can call this *polling synchronization*.

## Interrupts

*"In Soviet Russia, event polls you."*

Another way that a processor can find out about an event is via an *interrupt*. Interrupts *actively synchronize* a device and a processor. An interrupt tells the processor one bit of information: that something worth knowing about (high-priority) is occuring. When the processor gets an interrupt, it stops what it's currently doing, saves its state, and starts executing a pre-defined *interrupt handler*. The interrupt handler will typically read the event information (how?) and store it somewhere accessible. After the handler returns, the processor restores its state and resumes what it was doing before.