Threads, Events

Control systems and Computer Networks

Dr Alun Moon

Lecture 6.2

Threads

MBED Threads look a lot like the POSIX threads you've seen.

```
Thread worker;
worker.start( flash_red );

void flash_red(void) { while(1){  } }
```

- each thread has it's own loop
- `while{1}` means the loop and thread keep going forever.
- functions like `join` exist

Events and Dispatch

- MBED Events are handled by `EventQueue`.
- Events can be generated by libraries for devices, or programmatically.
- Events are *dispatched* to their handlers
- The `EventQueue` can dispatch its events for a given length of time, or continuously
- The *dispatch* functions return when finished
  - For continuous operation the `EventQueue` needs to be in its own thread.

```
Thread worker;
EventQueue queue ;

worker.start(callback(&queue,
            &EventQueue::dispatch_forever ));
```

Periodic events

Remember the problem of working out the timing of loops using `wait`:

- If I want a loop at a particular period
- I have to use a `wait` time that takes into account the execution time of the code (which might vary considerably)

We can register events to be triggered at a periodic rate

```
void blink(void){
    green = !green;
}

queue.call_every(300, blink);
```

Note: the event function does not need a `while(1)` loop, it is called *once* at each period.

Events and Interrupts

Recall *Interrupt Service Routines* (*ISR*) cannot perform complex or lengthy operations, such as serial or networks communications.

- An ISR can trigger an event
- The event is handled in the context of the *event-loop* outside of the ISR.

```
void blink(void){
    pc.printf("This is not in an ISR so I can do long (time
}

Thread worker;
EventQueue queue ;

InterruptIn sw(SW2);
sw.fall(queue.event(blink));
```