Scheduling Basics

KF6010 – Distributed Real-time Systems

Dr Alun Moon
alun.moon@northumbria.ac.uk

Lecture 3.1

The Super-loop architecture

```
init()       /* prepare to start */

    while(1) {  /* repeat forever          */
        f();    /* execute the operations */
}
```

- Pros
  - Simple – easy to understand
  - Uses almost no resources
- Cons
  - Difficult to ensure that `f()` is called at precise instants of time.

Many Embedded systems require precise timing

- Periodic tasks
- One-shot tasks

Can we fix the Super-loop

```
init();

    while(1) {    /* repeat forever          */
        f();      /* execute the operations */
            delay(n); /* delay for some time    */
    }
```

- This *might* work – repeat `f()` every $m + n$, where m is the execution time of `f()`
- But...
  - to choose n we need to know m precisely
  - execution time of `f()` must be the same each time round the loop

- These are unrealistic assumptions

Fix again

```
init();

    while(1) {
        start = gettime();
            f();
            delay( start + p-gettime() );
    }
```

- This is better
  - repeat `f()` every p
  - time for `f()` to execute can vary on each iteration, but period remains constant.
- But ...
  - Need to allow for time taken to get the time and configure the delay
  - Difficult to break the controller down into multiple functions that can execute at different rates.

Interrupts – Towards a better solution

- Use timer based interrupts to ensure that functions are called at precise instants of time.
- For example. . .

```
void f(void) { /* Interrupt handler -- control functions */
        toggle(LED);

        clearInterruptFlag(Timer0); /* clear interrupt flag */
}

int main() {
    initTimer(Timer0, f, p); /* set timer to interrupt every p

        while(1) {
        }
}
```

Executing multiple tasks at different time intervals

- Embedded systems may consist of multiple tasks that need to execute at (widely) different time intervals.
  - Read input from an ADC every millisecond
  - Read one or more switches every 200 milliseconds
  - Update LCD display every 3 milliseconds

How to solve this problem?

- Use multiple timers? – doesn't scale well
- Use a time-triggered scheduler? – yes

What can't we use multiple timers?

1. May not have enough timers.
   100 tasks and only 4 timers!
2. Code becomes hard to maintain.
   Adding another task may not be possible
3. Need to handle simultaneous interrupts
   - Difficult to manage, hard to predict behaviour
   - System design and analysis is much simpler if there is only a single interrupt source.

What is a time-triggered scheduler?

- extraordinarily simple operating system that allows tasks to be called on a periodic and one-shot basis.
- Uses a single timer ISR shared by many tasks, so...
  - only one timer needs to be initialised.
  - changes of timing source require only local changes
  - uses the same scheduler, no matter how many tasks there are.
- A time-triggered scheduler relies on a *static* schedule for its correct operation.

Static and Dynamic Scheduling

Static scheduling

- In the static scheduling approach, all decisions about which task should run at any time are made *off-line*, before run-time.
- The job of the scheduler at run-time is very simple; it consults a scheduling table to see which task should run next.
- Typically execution is *non-pre-emptive*, the tasks run to completion.

Dynamic Scheduling

- In dynamic scheduling, decisions about which task should execute are made *on-line*, at run-time.
- The job of the scheduler is to determine which task should execute next, according to some criteria.
- Typically execution is *pre-emptive*, tasks run in their own "super-loop" and do not complete.

Periodic Task model

The scheduling problem can be solved using precise mathematical models ahead of time:

- We assume that the scheduler is to run a set of periodic and one-shot tasks at pre-determined times.
- The Periodic tasks are characterised by their:

  Period T

  Phase ϕ also known as offset

  WCET C Worst case execution time

  Deadline D
- We assume a system of N tasks comprises an indexed set of periodic tasks J

$$J = \{J_i : i \in 1 \ldots N\}$$

- Each periodic task can be regarded as generating *instances* of itself for execution, with instances numbered from 0
- The arrival-time of the jth instance of task i is

$$\alpha(J_{i,j}) = j \times T_i + \phi_i$$

Terminology

Harmonic periods the periods of a task set are *harmonic* iff every period in the task set is an integer multiple of all smaller periods in the set

Hyper-period the *hyper-period* of a task set is the greatest time that elapses until the pattern of task arrivals is repeated.

- The hyper-period is equal to the least-common-multiple (LCM) of the periods in the task set.
- For a task set with harmonic periods, the hyper-period is equal to the greatest period of the tasks in the set.

Utilisation the *utilisation*, U, of the task set is given by

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i}$$

Structure of a time-triggered scheduler

- A time-triggered scheduler can be implemented simply by using a *periodic* timer interrupt.
- The period of the timer interrupt defines a frame-length
- Several tasks may be scheduled sequentially within a frame
- Let Z be the frame-length and H be the hyper-period. Then the table that drives the scheduler has $F = \frac{H}{Z}$ entries
- Each entry lists jobs to be executed in that frame
- The scheduler
  - is called by the timer interrupt
  - determines which frame should be scheduled
  - executes all jobs sequentially in the current frame
- The schedule repeats itself every hyper-period

Requirements for a schedule

1. Hyper-period H is least common multiple of periods in the task set
2. Frame size Z should be an integer divisor of H
   - as least as big as $\max\{C_i\}$
   - no bigger than $\min\{T_i\}$
   - Usually $\gcd\{T_i\}$ is a good choice
3. Every job instance should be scheduled in exactly one frame
4. No job instance should be scheduled before its release time
5. The sum of the worst case execution times of the job instances scheduled in any frame should be no bigger than the frame-size
6. The deadline for any job instance should be no earlier than the start of the next frame following the one in which its scheduled.

These requirements can be expressed formally as an *Integer Linear Programming* problem, and a schedule can be produced by a solver for this class of problem