Introduction to Real-Time Systems and Multitasking
Real-time systems

• Real-time system: A system that must respond to signals within explicit and bounded time requirements

• Categories
  – *Soft real-time system*: A system whose performance is degraded by failure to meet time constraints but continues to function. The usefulness of a result degrades after its deadline.
  – *Firm real-time system*: A system with deadlines where a low occurrence of missing a deadline can be tolerated. The usefulness of a result is zero after its deadline.
  – *Hard real-time system*: A system where failure to meet constraints leads to system failure.

Examples?
Real-time systems

• If your program only performs a single task, then it is not too hard to estimate the response time
• However, most embedded systems need to perform more than one task at a time
• Since actual execution is sequential, the time to complete execution of a process will affect other tasks
• We need to be sure that we do not over-load the CPU with too many tasks, and that each task runs in a specified time
CPU loads

• Virtually all programs for real time systems are implemented as endless loops, that run periodically.
• For a system with a single task, the CPU load is
  \[ L = \frac{T_T}{T_p} \]
• where
  \[ T_T = \text{the task’s execution time} \]
  \[ T_p = \text{the task’s execution period (i.e., how often the task needs to run)} \]
• Obviously \( L \) must be less than or equal to 1
• For a system with \( n \) tasks,
  \[ L = \frac{T_{T_1}}{T_{p_1}} + \frac{T_{T_2}}{T_{p_2}} + \ldots + \frac{T_{T_n}}{T_{p_n}} \]
Event-driven systems

- A system that must detect and respond to events
- Can be implemented using event loops, or interrupts
CPU Load – event loops

- Each event has a period (average and minimum)
- The time to process an event includes both detection time and service time

\[ L_{peak} = \frac{\left( T_{det} + T_{serv} \right)}{T_{epmin}} \]
Interrupt-driven system

- Two ways to use interrupts:
  - Event-generated interrupts
    - Caused by the event
    - An ISR services the event directly
  - Periodic interrupts
    - Generated by a periodic timer event
    - Is used for timed event loops, that will run at precise times

Note - there is some overhead in using interrupts – the “context switch” time
Timed event loops

- The main event loop is synchronized to a periodic timer event
- The loop time must be greater than the worst case event detection and service time
Basic Multitasking

• A “kernel” (or executive) is a program that schedules and runs tasks

• The simplest is a cyclic scheduler, which is a loop that sequentially executes each task

• Each task can run only once and then return to the scheduler
Time slice cyclic scheduler

- The main loop is run every timer event
- The period of the timer is called the “time slice”
- The time slice period is less than or equal to the greatest common divisor of all the task periods
Example

- A system has four tasks that need to run concurrently

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution time</th>
<th>Task period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task1</td>
<td>40 µs</td>
<td>1 ms</td>
</tr>
<tr>
<td>Task2</td>
<td>200 µs</td>
<td>2 ms</td>
</tr>
<tr>
<td>Task3</td>
<td>400 µs</td>
<td>2 ms</td>
</tr>
<tr>
<td>Task4</td>
<td>500 µs</td>
<td>10 ms</td>
</tr>
</tbody>
</table>

- Time slice

  \( T_{slice} \) must be less than or equal to the greatest common divisor of the task polling periods \( \rightarrow 1 \text{ ms} \)

- CPU load

  \[ L_{peak} = \frac{T_{T1}}{T_{p1}} + \frac{T_{T2}}{T_{p2}} + \frac{T_{T3}}{T_{p3}} + \frac{T_{T4}}{T_{p4}} \]

  \[ = 40\text{us}/1\text{ms} + 200\text{us}/2\text{ms} + 400\text{us}/2\text{ms} + 500\text{us}/10\text{ms} = 0.39 \]
Task schedule

• But all four tasks can’t run in the same time slice

\[ T_{T1} + T_{T2} + T_{T3} + T_{T4} = 40\text{us} + 200\text{us} + 400\text{us} + 500\text{us} = 1.140\text{ms} \] (which is greater than the 1ms time slice)

• So we need to schedule tasks:
  – Task1 runs every slice
  – Task2, Task3 run every other slice
  – Task4 runs every 10\text{th} slice

• Also, make sure that some tasks are never executed in the same slice; i.e., Task3 and Task4
  – So, make Task3 run every odd time slice
  – and make Task4 run every tenth (even) time slice

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution time</th>
<th>Task period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task1</td>
<td>40 (\mu)s</td>
<td>1 ms</td>
</tr>
<tr>
<td>Task2</td>
<td>200 (\mu)s</td>
<td>2 ms</td>
</tr>
<tr>
<td>Task3</td>
<td>400 (\mu)s</td>
<td>2 ms</td>
</tr>
<tr>
<td>Task4</td>
<td>500 (\mu)s</td>
<td>10 ms</td>
</tr>
</tbody>
</table>
More capable kernels

- A “preemptive” kernel can preempt a task, to execute a higher priority task
- That way each task can run as if it has the CPU all to itself

- Many commercial real-time operating systems are available
- Some free ones include FreeRTOS and uCOS II (they have ports for the 9S12 Freescale Processors)
Summary / Questions

• A “real time system” is a system that must guarantee response within strict time constraints.

• Is a computer running Microsoft Windows a real time system? Why or why not?

• A “real time operating system” must schedule and swap multiple processes, or tasks, such that each can meet its processing deadline.