Chapter 6

Processes and Operating Systems
Figure 6.1 Scheduling overhead is paid for at a nonlinear rate.
Figure 6.2 Example definitions of initiation times and deadlines.
Figure 6.3 A sequence of processes with a high initiation rate.
Figure 6.4 Data dependencies among processes.
Figure 6.5 Communication among processes at different rates.
**Figure 6.6** Scheduling states of a process.
Figure 6.7 Sequence diagram for preemptive execution.
Figure 6.8 Sequence diagram for a FreeRTOS.org context switch.
Figure 6.9 An active class in UML.
Figure 6.10 A collaboration diagram with active and normal objects.
Figure 6.11 An example of rate-monotonic scheduling.
/ * processes[] is an array of process activation records, 
  stored in order of priority, with processes[0] being 
  the highest-priority process */
Activation_record processes[NPROCESSES];

void RMA(int current) { /* current = currently executing 
  process */
  int i;
  /* turn off current process (may be turned back on) */
  processes[current].state = READY_STATE;
  /* find process to start executing */
  for (i = 0; i < NPROCESSES; i++)
    if (processes[i].state == READY_STATE) {
      /* make this the running process */
      processes[i].state == EXECUTING_STATE;
      break;
    }
}
Figure 6.13 C code for earliest-deadline-first scheduling.

```c
/* linked list, sorted by deadline */
Activation_record *processes;
/* data structure for sorting processes */
Deadline_tree *deadlines:

void expired_deadline(Activation_record *expired)
    remove(expired); /* remove from the deadline-sorted list */
    add(expired, expired->deadline); /* add at new deadline */
}

void EDF(int current) { /* current = currently executing process */
    int i;
    /* turn off current process (may be turned back on) */
    processes->state = READY_STATE;
    /* find process to start executing */
    for (allink = processes; allink != NULL; allink = allink->next_deadline)
        if (processes->state == READY_STATE) {
            /* make this the running process */
            processes->state == EXECUTING_STATE;
            break;
        }
}
```

Code
Figure 6.14 Shared memory communication implemented on a bus.
Figure 6.15 Message passing communication.
Figure 6.16 Use of a UML signal.
Figure 6.17 Sequence diagram for RTOS interrupt latency.
Figure 6.18 Interrupt latency during a critical section.
Figure 6.19 An L-shaped usage distribution.
Figure 6.20 Architecture of a power-managed system.
Figure 6.21 The Advanced Configuration and Power Interface and its relationship to a complete system.
Figure 6.22 Windows CE layer diagram.
Figure 6.23 OAL architecture in Windows CE.
Figure 6.24 Kernel and user address spaces in Windows CE.
Figure 6.25 User address space in Windows CE.
Figure 6.26 Sequence diagram for an interrupt.
Figure 6.27 The ADPCM coding scheme.
Figure 6.28 An ADPCM compression system.
Figure 6.29 Class diagram for the answering machine.
Figure 6.30 Physical class interfaces for the answering machine.
Figure 6.31 The message classes for the answering machine.
Figure 6.32 Operational classes for the answering machine.
Figure 6.33 State diagram for the Controls activate behavior.
Figure 6.34 State diagrams for the record-msg and playback-msg behaviors.
Figure 6.35 Hardware platform for the answering machine.
Figure 6.36 Engine block diagram.
Figure 6.37 Requirements for the engine controller.

<table>
<thead>
<tr>
<th>Name</th>
<th>ECU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Engine controller for fuel-injected engine</td>
</tr>
<tr>
<td>Inputs</td>
<td>Throttle, RPM, intake air volume, intake manifold pressure</td>
</tr>
<tr>
<td>Outputs</td>
<td>Injector pulse width, spark advance angle</td>
</tr>
<tr>
<td>Functions</td>
<td>Compute injector pulse width and spark advance angle as a function of throttle, RPM, intake air volume, intake manifold pressure</td>
</tr>
<tr>
<td>Performance</td>
<td>Injector pulse updated at 2-ms period, spark advance angle updated at 1-ms period</td>
</tr>
<tr>
<td>Manufacturing cost</td>
<td>Approximately $50</td>
</tr>
<tr>
<td>Power</td>
<td>Powered by engine generator</td>
</tr>
<tr>
<td>Physical size and weight</td>
<td>Approx 4 in × 4 in, less than 1 pound.</td>
</tr>
<tr>
<td>Signal</td>
<td>Variable name</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Throttle</td>
<td>T</td>
</tr>
<tr>
<td>RPM</td>
<td>NE</td>
</tr>
<tr>
<td>Intake air volume</td>
<td>VS</td>
</tr>
<tr>
<td>Injector pulse width</td>
<td>PW</td>
</tr>
<tr>
<td>Spark advance angle</td>
<td>S</td>
</tr>
<tr>
<td>Intake air temperature</td>
<td>THA</td>
</tr>
<tr>
<td>Exhaust oxygen</td>
<td>OX</td>
</tr>
<tr>
<td>Battery voltage</td>
<td>+B</td>
</tr>
</tbody>
</table>

Figure 6.38 Periods for data in the engine controller.
Figure 6.39 Class diagram for the engine controller.
Figure 6.40 State diagram for throttle position sensing.
Figure 6.41 State diagram for injector pulse width.
Figure 6.42 State diagram for spark advance angle.
UN Figure 6.1
UN Figure 6.2
UN Figure 6.3
UN Figure 6.4

Task graph

<table>
<thead>
<tr>
<th>Task</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

Task rates

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>4</td>
</tr>
</tbody>
</table>

Execution times
UN Figure 6.5
UN Figure 6.6
UN Figure 6.8