Homework 5 Solution

Problem 1: *Design a scheduling algorithm which ensures that over a relatively long interval CPU time is divided evenly between all tasks. For example, consider a system running this algorithm with two tasks, one I/O bound, the other compute bound. The run time of the tasks is long compared to the interval. The I/O-bound task and the CPU-intensive task would each get 50% of the CPU time (unless the time to complete I/O requests is very large).*

Key Points:

- When a task goes into the **Wait** state it normally forfeits the remainder of its quantum.
- The scheduling algorithm should allow the task to "catch up."

Solution:

Use two priority levels:

1: Normal tasks.

2: Tasks which have fallen behind.

Within each priority level, use FCFS scheduling.

OS is task-preemptive.

The following actions are taken when a priority-1 task must wait:

- The remaining time in its quantum is stored, use symbol t_{rq} for this time.
- The time at which it went into the wait state is stored, t_{w}.
- Its priority is set to 2 and it is placed in the wait list.

Priority 1 tasks used a fixed quantum and are scheduled normally.

Tasks at priority 2 will be allowed to run for a total of t_{rq}, after which they will move back to priority 1.

When a task is moved back to the priority-1 ready list, its arrival time is set to t_{w}, *not the current time*.

By setting the arrival time to this value the task does not loose its place in line.

Priority 2 task X is run with a quantum of $t_{\mathrm{rq}}(X)$, for all X at priority 2.

If it uses up its quantum it is moved back to priority 1.

Otherwise $t_{\mathrm{rq}}(X)$ is replaced with $t_{\mathrm{rq}}(X) - t_x$, where t_x is the amount of time it ran.

Problem 2: *Find timing constraints for the code in the self-balancing washing machine example. Briefly justify each constraint. Show how the code might be scheduled, including interrupt handlers and tasks. Show a situation in which there might be timing difficulties and explain how they might be resolved.*

Solution steps:

- Estimate run times.
- Estimate timing constraints.
- Find scheduling.

Run Times

`CDT`, $t_h = 1\,\mu\mathrm{s}$. Time is small because it only has to increment a variable and, sometimes, read a wobble.

`SPEED`, $t_h = 5\,\mu\mathrm{s}$. Must compute speed and update a table. Checking for table-length overflow and other conditions results in the longer run time.

`SPRAY`, $t_h = 50\,\mu\mathrm{s}$. In addition to adjusting actuators must set next timer interrupt. It might have to do some computation to determine when the next timer interrupt should be.

`wobble`, run time 500 ms.

Timing Constraints

Suppose maximum spin speed is 1200 RPM.

Suppose CDT has 1024 marks.

Then minimum period for CDT is $49\,\mu$s.

Reasonable assumption: response must occur before next event.

Constraint:

Normal:Mark $\longrightarrow$ CtrInc $\quad < 49\,\mu$s,

where event *Mark* occurs when a mark passes under mark readers in the CDT,

and response *CtrInc* occurs when `CDT` finishes.

Similarly, for speed:

Normal:Speed $\longrightarrow$ Cntr $\quad < 49\,\mu$s,

where event *Speed* occurs at the timer interrupt and,

response *Cntr* occurs after `SPEED` has run.

Suppose buckets had a target that was 30° wide and jets could turn on or off no more than 10° late.

Then response time constrain is the time for the tub to turn 10° at maximum speed.

Normal:JetOnI $\longrightarrow$ JetOn $\quad < 1.39\,$ms,

Normal:JetOffI $\longrightarrow$ JetOff $\quad < 1.39\,$ms,

where event *JetOnI* and *JetOffI* are the respective timer interrupts,

and response *JetOn* and *JetOff* occur when the `SPRAY` handler finishes.

Scheduling

The SPRAY handler's run time could result in CDT or SPEED missing their deadlines so:

Strong priority 2: CDT and SPEED.

Strong priority 1: SPRAY.

The wobble task will be scheduled as a dæmon task since it is not associated with any hard deadlines.

Problem 3: *Find the actual run time, latency, and response time for the event types described in the table below.*

Event Name	Strong Pri.	Weak Pri.	Handler Run Time	Occurrence
A	3	3	$10\,\mu$s	Periodic, $t_b(A) = 54\,\mu$s.
B	3	2	$4\,\mu$s	Twice, any time between occurrences of A.
C	3	1	$12\,\mu$s	Once, any time between occurrences of A.
D	2	1	$50\,\mu$s	Periodic, $t_b(D) = 23\,$ms.
E	1	1	$10\,\mu$s	One shot.

Event B will not recur until after a previous occurrence of event B has been responded to. That is, there cannot be an occurrence of B after an occurrence of B and before the response for this occurrence of B.

Events A and C occur the same number of times, B occurs twice as many times as A and C.

Solution:

Strong Priority Level 3:

Latency of A: $t_l(A) = t_h(C) = 12\,\mu$s.

Latency of B: $t_l(B) = t_h(A) + t_h(C) = 22\,\mu$s.

Event B can occur twice just before and just after A so:

Latency of C: $t_l(C) = t_h(A) + 4t_h(B) = 26\,\mu$s.

Actual run times: $t_a(A) = t_h(A)$, $t_a(B) = t_h(B)$, and $t_a(C) = t_h(C)$.

Since there is no preemption at the highest strong priority level response times are sum of latency and actual run times:

$t_r(A) = t_l(A) + t_a(A) = 22\,\mu$s, $t_r(B) = t_l(B) + t_a(B) = 26\,\mu$s, and $t_r(C) = t_l(C) + t_a(C) = 38\,\mu$s.

Strong Priority Level 2:

Unlucky situation for D: C, two Bs, an A, 2 Bs, and a C.

Latency of D: $t_l(D) = t_h(A) + 4t_h(B) + 2t_h(C) = 50\,\mu$s.

A similar unlucky situation:

Actual duration $t_a(D) = 3t_h(A) + 8t_h(B) + 4t_h(C) + t_h(D) = 160\,\mu$s.

In this case, response time $t_r(D) = t_a(D)$.

Strong Priority Level 1:

Latency of E is response time of D, $t_l(E) = 160\,\mu$s.

Actual duration is $t_a(E) = t_r(D) + t_h(E) = 170\,\mu$s.

Response time is $t_r(E) = t_a(E) = 170\,\mu$s.