University of Newcastle
School of Electrical Engineering and Computer Science

COMP2240 - Operating Systems
Workshop 4
Topics: Real-time and Multiprocessor Scheduling

1. Consider a set of five aperiodic tasks with the execution profiles of Table 1. Develop scheduling diagrams for Earliest Deadline, Earliest Deadline with unforced idle time and FCFS algorithms for this set of tasks.

Process	Arrival Time	Execution Time	Starting Deadline
A	10	20	100
B	20	20	20
C	40	20	60
D	50	20	80
E	60	20	70

Table 1: Execution profile for Problem 1

2. **Least laxity first (LLF)** is a real-time scheduling algorithm for periodic tasks. Slack time, or laxity, is the amount of time between when a task would complete if it started now and its next deadline. This is the size of the available scheduling window. Laxity can be expressed as

Laxity = (deadline time) – (current time) – (processor time needed)

LLF selects the task with the minimum laxity to execute next. If two or more tasks have the same minimum laxity value, they are serviced on a FCFS basis.

a) Suppose a task currently has a laxity of *t*. By how long may the scheduler delay starting this task and still meet its deadline?
b) Suppose a task currently has a laxity of *0*. What does this mean?
c) What does it mean if a task has negative laxity?
d) Consider a set of three periodic tasks with the execution profiles of Table . Develop scheduling diagrams for this set of tasks that compare rate monotonic, earliest-deadline first, and LLF. Assume preemption may occur at 5-ms intervals. Comment on the results.

Task	Period	Execution Time
A	6	2
B	8	2
C	12	3

Table 2: Light load execution profile

3. This problem demonstrates that although Equation for rate monotonic scheduling is a sufficient condition for successful scheduling, it is not a necessary condition (i.e., sometimes successful scheduling is possible even if Equation is not satisfied).

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \leq n\left(2^{1/n} - 1\right)$$

a) Consider a task set with the following independent periodic tasks:
Task P1:*C1 = 20*; *T1 = 100*
Task P2:*C2 = 30*; *T2 = 145*
Can these tasks be successfully scheduled using rate monotonic scheduling?

b) Now add the following task to the set:
Task P3:*C3 = 68; T3 = 150*
Is Equation (10.2) satisfied?

c) Suppose that the first instance of the preceding three tasks arrives at time. Assume that the first deadline for each task is the following:

D1 = 100; D2 = 145; D3 = 150

Using rate monotonic scheduling, will all three deadlines be met? What about deadlines for future repetitions of each task?

4. Consider two processes, *P1* and *P2*, where *T1 = 50*, *C1 = 25*, *T2 = 75*, and *C2 = 30*.

a) Can these two processes be scheduled using rate-monotonic scheduling? Illustrate your answer using a Gantt chart.

b) Illustrate the scheduling of these two processes using earliest deadline- first (EDF) scheduling.

5. Suppose that an application has three threads *T1, T2* and *T3* having decreasing priority. Give a scenario that may cause unbounded priority inversion.

Supplementary problems:

S1. What happens if three CPUs in a multiprocessor attempt to access exactly the same word of memory at exactly the same instant?

S2. Explain why interrupt and dispatch latency times must be bounded in a hard real-time system.

S3. The Linux scheduler implements “soft” real-time scheduling. What features necessary for certain real-time programming tasks are missing? How might they be added to the kernel? What are the costs (downsides) of such features?

S4. Some descriptions of UNIX would indicate that it is unsuitable for real-time applications because a process executing in kernel mode may not be preempted. Elaborate. Then consider and discuss if it is possible that UNIX may be modified to accommodate real-time applications at all?

S5. Discuss ways in which the priority inversion problem could be addressed in a real-time system. Also discuss whether the solutions could be implemented within the context of a proportional share scheduler.