Written Assignment 1

CPEN 432: Real-time System Design

Due: Feb 10, 2021, by 11:59 p.m., on Gradescope

Notation: $[n] = \{1, \ldots, n\}$ for integer $n \geq 1$.

1. **[10 points]** (Predictability) In general, a real-time (scheduling) system is said to be predictable if it does not have scheduling anomalies. We saw one such anomaly in class. Here we will attempt to develop one plausible notion of predictability and make it formal. Suppose that we want to schedule n **jobs** $J_1, \ldots, J_n$. For every job J_i, instead of a single execution time estimate, we are given *two* execution time estimates c_i^- and c_i^+, where c_i^- is the minimum execution time and c_i^+ is the maximum execution time of job J_i. Fix a scheduling policy. We call the schedule generated by the given scheduling policy when the execution time of every job has its minimum value a **minimal** schedule. A **maximal** schedule is defined analogously. Note that the *actual* (exact) schedule under the given policy is unknown because, at run time, job J_i might demand anything in $[c_i^-, c_i^+]$. Let s_i^- and f_i^- denote the start and finish time of job J_i, respectively, in the minimal schedule produced by the fixed scheduling policy. The quantities s_i^+ and f_i^+ are the counterparts corresponding to the maximal policy.

   Let s_i and f_i denote job J_i's actual start and finish time under the given policy. We say that J_i is **predictable** under the given policy if both $s_i^- \leqslant s_i \leqslant s_i^+$ and $f_i^- \leqslant f_i \leqslant f_i^+$. The execution of the entire job set under a given policy is predictable if every J_i is predictable.

   Show that the execution of every job in a set of independent jobs with fixed arrival times that are scheduled preemptively on a single processor using any priority-based scheme is predictable.

2. **[20 points]** (Coincidence, anomaly, there's a difference?) Consider a job system consisting of nine non-preemptable jobs $J_1, \ldots, J_9$. Their execution times are $3, 2, 2, 2, 4, 4, 4, 4, 9$, they are all ready to run at time 0, and they have equal deadlines of 12. Job J_i is said to be a **predecessor** of job J_j if job J_j cannot start execution until job J_i has finished execution entirely. Job J_i is said to be an **immediate predecessor** of J_j if J_i is a predecessor of J_j and there is no J_k with $k \neq i \neq j$ such that both (1) J_k is a predecessor of J_j and (2) J_i is a predecessor of J_k. In our job system, J_1 is the immediate predecessor of J_9, and J_4 is the immediate predecessor of J_5, J_6, J_7, and J_8. There is no other precedence constraints. A job with a lower index has a higher priority; that is, J_j has higher priority than J_k iff $j < k$.

   (a) **[3 points]** A **precedence graph** is a directed graph where there is a vertex for each job, and a directed edge emanates from vertex i towards vertex j if job J_i is the immediate predecessor of job J_j. *Draw the precedence graph corresponding to our job system.*

   (b) **[5 points]** Suppose that each node in the precedence graph has a weight, which is the execution time of the corresponding job. Given an arbitrary weighted precedence graph G with n jobs (i.e., n vertices), node weights $c_1, \ldots, c_n$, and a job $i \in [n]$, and assuming that all n jobs arrive at time 0, *devise an algorithm to compute the **earliest time** at which job i can start execution.* Your algorithm should run in time that is at most $O(n^2)$.

(c) **[3 points]** Can the jobs in our 9-job set meet their deadlines if scheduled on 3 processors? All 3 processors are identical (job i will require c_i on any processor). Distinct jobs can run in parallel on any of the 3 processors. Since scheduling in this part is non-preemptive, a job that starts on one processor must finish on the same processor; the job cannot be interrupt by other jobs on the same processor and cannot resume execution on another processor (no migration allowed). Justify your answer. *In this and all the subsequent parts, the best way to demonstrate your answer is to draw a **timing diagram** of schedules with horizontal parallel axes, each corresponding to a processor.*

(d) **[3 points]** Can our jobs meet their deadlines on the 3 processors if we allow preemption and schedule preemptively? Justify your answer. In the case of preemption, the execution of one job cannot overlap on two or more processors: You cannot run two portions of the same job in parallel concomitantly; however, you may **migrate** a job across processors by interrupting its execution on the processor upon which it is running and resuming its execution on another. A job can also be preempted and resume on the same processor.

(e) **[3 points]** Can our jobs meet their deadlines if scheduled on 4 processors non-preemptively? Justify your answer.

(f) **[3 points]** Suppose that all three processors were upgraded to faster processors, and this upgrade resulted in a reduction of the execution time of each of our 9 jobs by 1 time unit. *Can the jobs meet their deadlines when scheduled non-preemptively on the 3 faster processors?* Justify your answer.

3. **[10 points]** (It's just a silly phase I'm going through.) In all our analyses in class, we assumed that task phases are zero. Now suppose that tasks are allowed to have non-zero phases.

   (a) **[5 points]** Give an example of a taskset where there is no point in time at which the arrivals of all tasks are aligned.

   (b) **[5 points]** Does RM remain optimal in the case of arbitrary task phases? If true, give a proof, and if not, give a counterexample.

   (c) **[Bonus: +10 points]** Devise an algorithm to determine whether or not two tasks τ_1 and τ_2 with phases Φ_1 and Φ_2 exhibit a time instant where their releases are aligned. Analyze the running time of your algorithm. You may assume that the periods are positive integers. More credit will be given for more efficient solutions.

4. **[10 points]** Complete Case 2 for the proof of optimality of RM for two tasks with zero phases. We discussed Case 1 in class. See lecture 5 slides.