University of Glasgow | School of Computing Science

Priority-driven Scheduling of Periodic Tasks (1)

Advanced Operating Systems (M)
Lecture 4

Priority-driven Scheduling

- Assign priorities to jobs, based on their deadline or other timing constraint
  - Make scheduling decisions based on the priorities, when events such as releases and job completions occur
  - Jobs are placed in one or more queues; at each event, the ready job with the highest priority is executed
  - The assignment of jobs to priority queues, along with rules such a whether preemption is allowed, completely defines a priority scheduling algorithm
- Priority-driven algorithms make locally optimal decisions about which job to run
  - Locally optimal scheduling decisions are often not globally optimal
  - Priority-driven algorithms never intentionally leave any resource idle; leaving a resource idle is not locally optimal

Advantages and Disadvantages

- Priority-driven scheduling has many advantages over clock-driven scheduling
  - Better suited to applications with varying time and resource requirements, since needs less a priori information
  - Run-time overheads are small
- But, harder to validate for correctness:
  - Scheduling anomalies can occur for multiprocessor systems, if preemption is disallowed, or if there is contention for resources
    - Reducing the execution time of a job in a task can increase the total response time of the task: not sufficient to show correctness with worse-case execution times, must simulate with all possible execution times for all jobs comprising a task
  - Can be proved that anomalies do not occur for independent, jobs with fixed release times, where preemption is allowed, executed using any priority-driven scheduler on a single processor

Priority-driven Scheduling

- Many priority-driven real-time scheduling algorithms exist
  - Earliest deadline first
  - Least slack time
  - Rate monotonic
  - Deadline monotonic

- Each has different characteristics

Fixed- and Dynamic-Priority Algorithms

- A priority-driven scheduler is an on-line scheduler
  - It does not pre-compute a schedule: instead assigns priorities to jobs when released, places them on a run queue in priority order
  - When pre-emption is allowed, a scheduling decision is made whenever a job is released or completed
  - At each scheduling decision time, the scheduler updates the run queues and executes the job at the head of the queue
- The priority of jobs within a task may vary:
  - Jobs in a task may be assigned the same priority (task level fixed-priority) or different priorities (task level dynamic-priority)
  - The priority of each job is usually fixed (job level fixed-priority); but some systems vary the priority of a job after it has started (job level dynamic-priority)

Rate Monotonic Scheduling

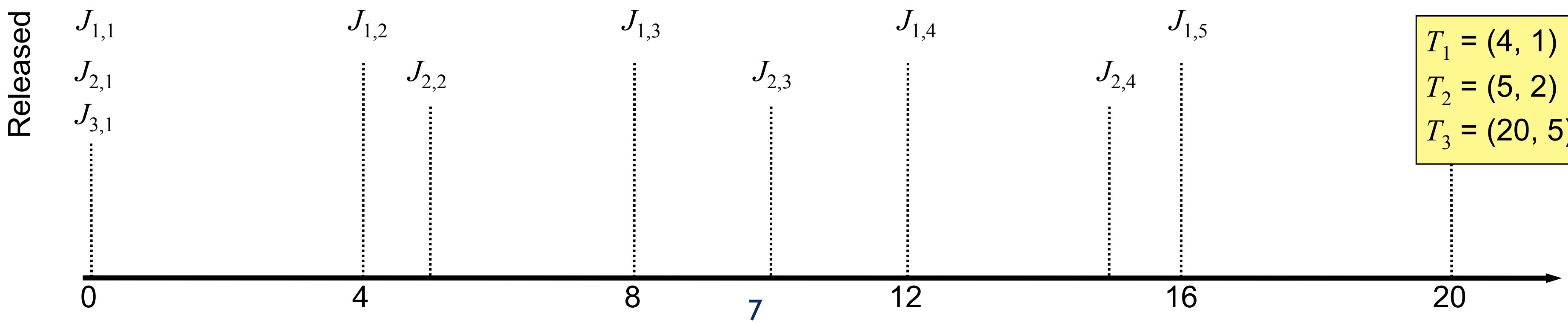
- Well known fixed-priority algorithm
- Assigns priorities to tasks based on their periods
  - The shorter the period, the higher the priority; the rate (of job releases) is the inverse of the period, so jobs with higher rate have higher priority

- For example, consider a system of 3 tasks:
  - T_1 = (4, 1) ⇒ rate = 1/4
    T_2 = (5, 2) ⇒ rate = 1/5
    T_3 = (20, 5) ⇒ rate = 1/20
  - Relative priorities: $T_1 > T_2 > T_3$

Example: Rate Monotonic Scheduling

Time	Ready to run	Running
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		

Time	Ready to run	Running
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		



Deadline Monotonic Scheduling

- The deadline monotonic algorithm assigns task priority according to relative deadlines – the shorter the relative deadline, the higher the priority
- When relative deadline of every task matches its period, then rate monotonic and deadline monotonic give identical results
- When the relative deadlines are arbitrary:
    - Deadline monotonic can sometimes produce a feasible schedule in cases where rate monotonic cannot; rate monotonic always fails when deadline monotonic fails
    - Hence deadline monotonic preferred if deadline ≠ period

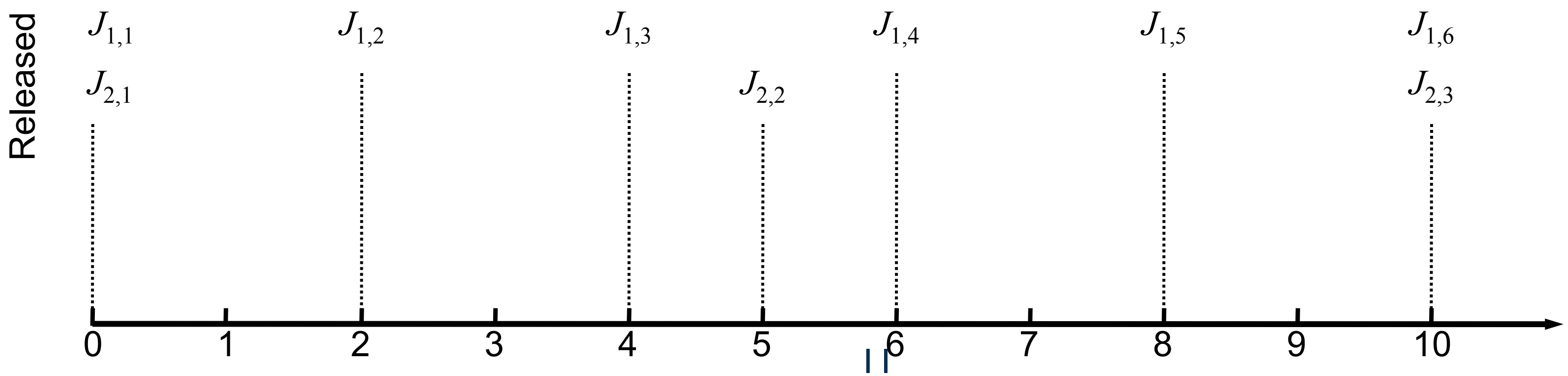
The EDF and LST Scheduling Algorithms

- Two popular dynamic priority algorithms
- Earliest deadline first (EDF)
  - Assign priority to jobs based on deadline: earlier deadline = higher priority
  - Simple, just requires knowledge of deadlines
- Least Slack Time first (LST)
  - A job J_i has deadline d_i, execution time e_i, and was released at time r_i
  - At time $t < d_i$: remaining execution time $t_{\text{rem}} = e_i - (t - r_i)$
  - Assign priority based on least slack time, $t_{\text{slack}} = d_i - t - t_{\text{rem}}$
  - Strict LST: scheduling decision made whenever a queued job's slack time becomes smaller than the executing job's slack time – high overhead, not used; Non-strict LST: scheduling decisions made only when jobs release or complete
  - More complex, requires knowledge of execution times and deadlines

Example: Earliest Deadline First

Time	Ready to run	Running

Time	Ready to run	Running



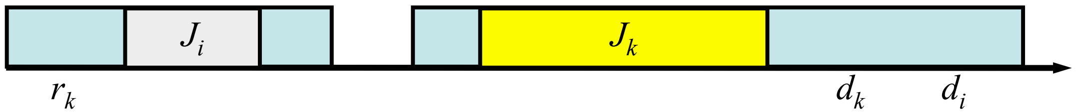
T_1 = (2, 1)
T_2 = (5, 2.5)

Optimality of EDF and LST

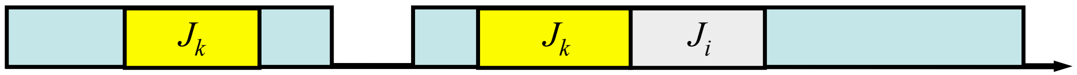
- The EDF and LST algorithms are optimal
  - On a single processor, as long as preemption is allowed and jobs do not contend for resources; can fail to schedule a feasible set of jobs if there are multiple processors, or if preemption is allowed

Optimality of EDF and LST: Proof

- Any feasible schedule can be transformed into an EDF schedule
  - If J_i is scheduled to run before J_k, but J_i's deadline is later than J_k's either:
    - The release time of Jk is after the Ji completes ⇒ they're already in EDF order
    - The release time of Jk is before the end of the interval in which Ji executes:



    - Swap J_i and J_k (this is always possible, since J_i's deadline is later than J_k's)



    - Move any jobs following idle periods forward into the idle period

    - The result is an EDF schedule
  - So, if EDF fails to produce a feasible schedule, no such schedule exists
    - If a feasible schedule existed it could be transformed into an EDF schedule, contradicting the statement that EDF failed to produce a feasible schedule [proof for LST is similar]

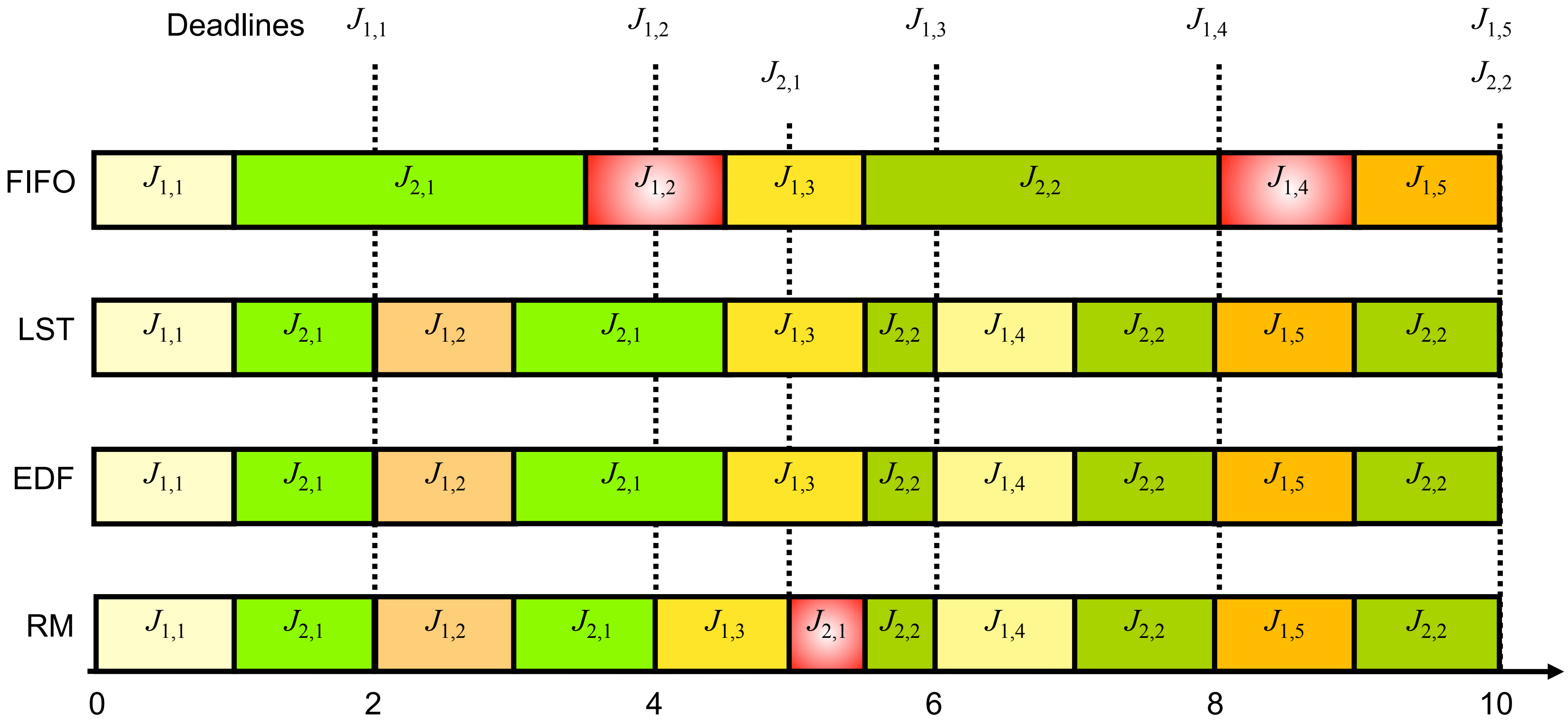
Relative Merits

- Fixed- and dynamic-priority scheduling algorithms have different properties; neither appropriate for all scenarios
- The EDF algorithm gives higher priority to jobs that have missed their deadlines than to jobs whose deadline is still in the future
  - Not necessarily suited to systems where occasional overload unavoidable
- Dynamic algorithms like EDF can produce feasible schedules in cases where RM and DM cannot
  - But fixed priority algorithms often more predictable, lower overhead

Example: Comparing Different Algorithms

- Compare performance of RM, EDF, LST and FIFO scheduling
- Assume a single processor system with 2 tasks:
  - T_1 = (2, 1)
  - T_2 = (5, 2.5) H = 10
- The total utilisation is 1.0; there is no slack time
  - Expect some of these algorithms to lead to missed deadlines!
  - This is one of the cases where EDF works better than RM/DM

Example: RM, EDF, LST and FIFO



- Demonstrate by exhaustive simulation that LST and EDF meet deadlines, but FIFO and RM don't

Schedulability Tests

- Simulating schedules is both tedious and error-prone… can we demonstrate correctness without working through the schedule?

- Yes, in some cases. This is a schedulability test
  - A test to demonstrate that all deadlines are met, when scheduled using a particular algorithm
  - An efficient schedulability test can be used as an on-line acceptance test; clearly exhaustive simulation is too expensive

Schedulable Utilisation

- Recall: a periodic task T_i is defined by the 4-tuple $(\varphi_i, p_i, e_i, D_i)$ with utilisation $u_i = e_i / p_i$
- Total utilisation of system $U = \sum_{i=1}^{n} u_i$ where $0 \leq U \leq 1$
- A scheduling algorithm can feasibly schedule any system of periodic tasks on a processor if U is equal to or less than the maximum schedulable utilisation of the algorithm, U_{ALG}
- This gives a schedulability test, where a system can be validated by showing that $U \leq U_{\mathrm{ALG}}$
  - If $U_{\mathrm{ALG}} = 1$, the algorithm is optimal

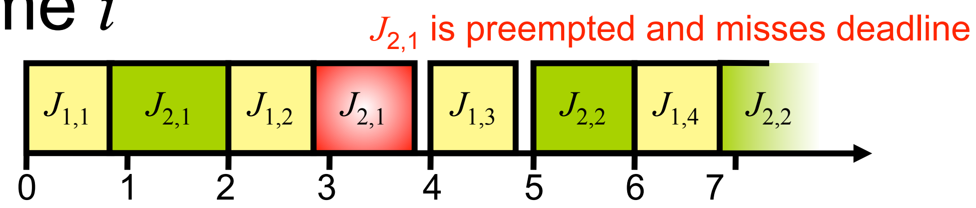
Schedulable Utilisation: EDF

- Theorem: a system of independent preemptable periodic tasks with $D_i = p_i$ can be feasibly scheduled on one processor using EDF if and only if $U \leq 1$
  - $U_{\text{EDF}} = 1$ for independent, preemptable periodic tasks with $D_i = p_i$
  - Corollary: result also holds if deadline longer than period: $U_{\text{EDF}} = 1$ for independent preemptable periodic tasks with $D_i \geq p_i$

- Notes:
  - Result is independent of φ_i
  - Result can also be shown to apply to strict LST

Schedulable Utilisation: EDF

- Test fails if $D_i < p_i$ for some i
  - E.g. T_1 = (2, 0.8), T_2=(5, 2.3, 3)



- However, there is an alternative test:
  - The density of the task, T_i, is $\delta_i = e_i / \min(D_i, p_i)$
  - The density of the system is $\Delta = \delta_1 + \delta_2 + \ldots + \delta_n$
  - Theorem: A system T of independent, preemptable periodic tasks can be feasibly scheduled on one processor using EDT if $\Delta \leq 1$.
- Note:
  - This is a sufficient condition, but not a necessary condition – i.e. a system is guaranteed to be feasible if $\Delta \leq 1$, but might still be feasible if $\Delta > 1$ (would have to run the exhaustive simulation to prove)

Schedulable Utilisation: EDF

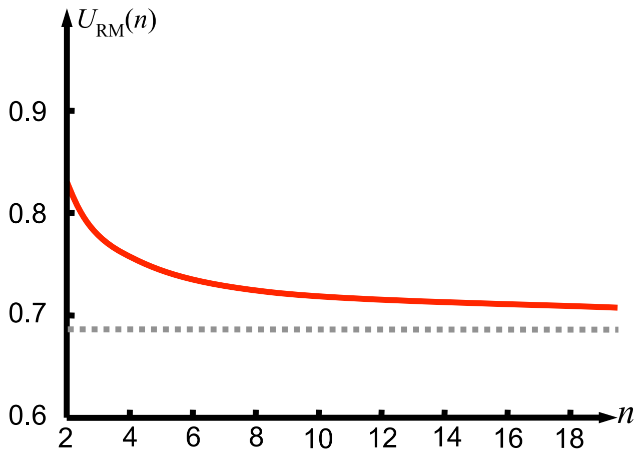
- How can you use this in practice?
  - Assume using EDF to schedule multiple periodic tasks, known execution time for all jobs
  - Choose the periods for the tasks such that the schedulability test is met
- Example: a simple digital controller:
  - Control-law computation task, T_1, takes e_1 = 8 ms, sampling rate is 100 Hz (i.e. p_1 = 10 ms)
    ⇒ u_1 is 0.8

    ⇒ the system is guaranteed to be schedulable
  - Want to add another task, T_2, taking 50ms - will the system still work?

Schedulable Utilisation of RM

- A system of n independent preemptable periodic tasks with $D_i = p_i$ can be feasibly scheduled on one processor using RM if $U \leq n \cdot (2^{1/n - 1})$

  - $U_{RM}(n) = n \cdot (2^{1/n} - 1)$
  - For large $n \rightarrow \ln 2$ (i.e., $n \rightarrow$ 0.69314718056…)



  - $U \leq U_{RM}(n)$ is a sufficient, but not necessary, condition – i.e., a feasible rate monotonic schedule is guaranteed to exist if $U \leq U_{RM}(n)$, but might still be possible if $U > U_{RM}(n)$

Schedulable Utilisation of RM

- What happens if the relative deadlines for tasks are not equal to their respective periods?
- If the deadline is a multiple v of the period: $D_k = v \cdot p_k$
- It can be shown that:

$$U_{RM}(n, v) = \begin{cases} v & \text{for } 0 \leq v \leq 0.5 \\ n((2v)^{\frac{1}{n}} - 1) + 1 - v & \text{for } 0.5 \leq v \leq 1 \\ v(n-1)[(\frac{v+1}{v})^{\frac{1}{n}-1} - 1] & \text{for } v = 2, 3, \ldots \end{cases}$$

Schedulable Utilisation of RM

n	$\upsilon = 4.0$	$\upsilon = 3.0$	$\upsilon = 2.0$	$\upsilon = 1.0$	$\upsilon = 0.9$	$\upsilon = 0.8$	$\upsilon = 0.7$	$\upsilon = 0.6$	$\upsilon = 0.5$
2	0.944	0.928	0.898	0.828	0.783	0.729	0.666	0.590	0.500
3	0.926	0.906	0.868	0.779	0.749	0.708	0.656	0.588	0.500
4	0.917	0.894	0.853	0.756	0.733	0.698	0.651	0.586	0.500
5	0.912	0.888	0.844	0.743	0.723	0.692	0.648	0.585	0.500
6	0.909	0.884	0.838	0.734	0.717	0.688	0.646	0.585	0.500
7	0.906	0.881	0.834	0.728	0.713	0.686	0.644	0.584	0.500
8	0.905	0.878	0.831	0.724	0.709	0.684	0.643	0.584	0.500
9	0.903	0.876	0.829	0.720	0.707	0.682	0.642	0.584	0.500
∞	0.892	0.863	0.810	0.693	0.687	0.670	0.636	0.582	0.500

$D_i > p_i \Rightarrow$ Schedulable utilisation increases

$D_i < p_i \Rightarrow$ Schedulable utilisation decreases

$D_i = p_i$

Summary

- Different priority-driven scheduling algorithms
  - Earliest deadline first, least slack time, rate- and deadline- monotonic
  - Each has different properties, suited for different scenarios
- Scheduling tests, concept of maximum schedulable utilisation
  - Examples for different algorithms
- Next lecture: practical factors, more schedulability tests…