Real-Time Scheduling

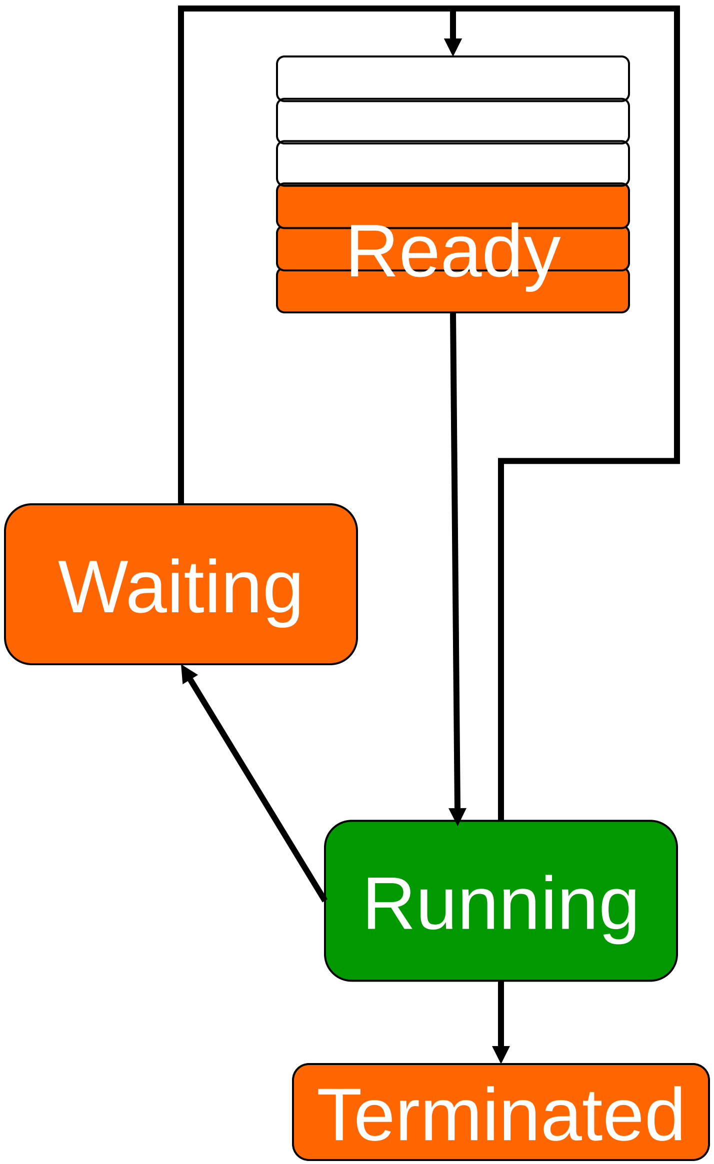
These lecture notes created by Mr. James B. (Jim) Carlson, NCSU

Today

- Operating System task scheduling
  - Traditional (non-real-time) scheduling
  - Real-time scheduling

Scheduling

- Choosing which *ready* thread to run next
- Common criteria CPU Utilization –fraction of time is the CPU busy
  - Throughput – number of tasks are completed per unit time
  - Turnaround time – time delay from task first being submitted to OS to finally completing
  - Waiting time – amount of time a task spends in waiting queue



Common Scheduling Algorithms

- First-Come, First Served (FCFS)
  - All queues operate as strict FIFOs without priority
  - Problems: large average delay, not preemptive
- Round Robin: add time-sharing to FCFS
  - At end of time tick, move currently running task to end of ready queue
  - Problems: Still have a large average delay, choosing time-tick is trade-off of context-switching overhead vs. responsiveness
- Shortest Job First (SJF)
  - Job = process
  - SJF is provably optimal in minimizing average waiting time
  - Problem: How do we determine how long the next job will take?
    - Could predict it based on previous job?

Priority Scheduling

- Run the ready task with highest priority
- Define priority
  - Internal: Time limits, memory requirements
  - External: Importance to application, fees paid, department submitting task
- Problem: indefinite blocking (starvation)
  - Low level processes may never get to run in heavily loaded system
  - Two outcomes
    - Processes run during winter break
    - Processes disappear when computer eventually crashes

From OS to RTOS

- Traditional (non-real-time) Operating System
  - Hard to predict response time…
  - Hard to guarantee that a task will always run before its deadline
- Real-Time Operating System
  - Easy to determine that a task will always run before its ***deadline***
  - Designed for ***periodic*** tasks
- What does Real-Time mean?

Real-Time means right now.

Late answers are wrong answers!

Scheduling – Selecting a *Ready* task to run

- Goals
  - Meet all task and ISR deadlines
  - Maximize processor *utilization* (U)
    - U = Fraction of time CPU performs useful work
    - Limit scheduling overhead (choosing what to run next)
    - Limit context switching overhead
- Assigning priority based *only* on importance doesn't work – why not?
- How do we assign priorities to task?
  - Statically – priority based on period (doesn't change)
  - Dynamically – priority based on time left (changes)

Definitions for Task i

- Task execution time = T_i
- Task execution period = τ_i: time between arrivals
- Utilization = fraction of time which CPU is used
  - For a task i

$$U_i = \frac{T_i}{\tau_i}$$

  - Overall, for all n tasks in the system

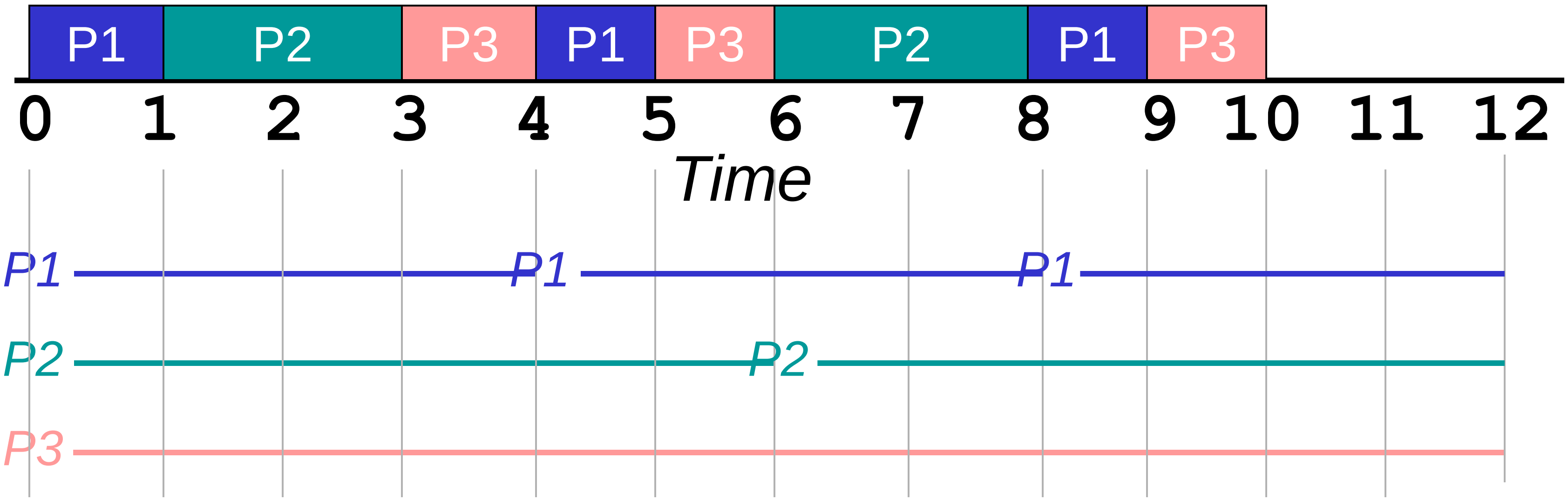
$$U = \sum_{i=1}^{n} \frac{T_i}{\tau_i}$$

- Completion Time = time at which task finishes. This is also called turn-around time (for non-RT applications) or response time (for RT applications).
- Critical Instant = time at which task's completion time is maximized. All tasks arrive simultaneously.
- Schedulable = a schedule exists which allows all tasks to meet their deadlines, even for the critical instant

Rate Monotonic Scheduling

- Assumptions
  - Tasks are periodic with period τ_i
  - Single CPU
  - $T_{ContextSwitch} = T_{scheduler} = 0$
  - No data dependencies between tasks
  - Constant process execution time T_i
  - Deadline = end of period = τ_i
- Assign priority based on period (rate)
  - Shorter period means higher priority

Processor Behavior – Graphical Analysis



Task	Exec. Time T	Period τ	Priority
P1	1	4	High
P2	2	6	Medium
P3	3	12	Low

Exact Schedulability Test for Task *i*

- Account for all processing at critical instant
- Consider possible additional task arrivals
- a_n = nth estimate of time when task i completes
- Loop
  - Estimate higher priority job arrivals, compute completion time
  - Recompute based on any new arrivals
- Iterate until
  - $a_n > \tau_i$: not schedulable
  - $a_{n+1} = a_n <= \tau_i$: schedulable

$$a_0 = \sum_{j=0}^{i} T_j$$

$$a_{n+1} = T_i + \sum_{j=0}^{i-1} \left\lceil \frac{a_n}{\tau_j} \right\rceil T_j$$

Exact Schedulability Test for Example

$$a_0 = \sum_{j=0}^{i} T_j = 1+2+3 = 6$$

$$a_1 = 3 + \sum_{j=0}^{i-1} \left\lceil \frac{6}{\tau_j} \right\rceil T_j = 3 + \left\lceil \frac{6}{4} \right\rceil * 1 + \left\lceil \frac{6}{6} \right\rceil * 2 = 3+2+2 = 7$$

$$a_2 = 3 + \sum_{j=0}^{i-1} \left\lceil \frac{7}{\tau_j} \right\rceil T_j = 3 + \left\lceil \frac{7}{4} \right\rceil * 1 + \left\lceil \frac{7}{6} \right\rceil * 2 = 3+2+4 = 9$$

$$a_3 = 3 + \sum_{j=0}^{i-1} \left\lceil \frac{9}{\tau_j} \right\rceil T_j = 3 + \left\lceil \frac{9}{4} \right\rceil * 1 + \left\lceil \frac{9}{6} \right\rceil * 2 = 3+3+4 = 10$$

$$a_4 = 3 + \sum_{j=0}^{i-1} \left\lceil \frac{10}{\tau_j} \right\rceil T_j = 3 + \left\lceil \frac{10}{4} \right\rceil * 1 + \left\lceil \frac{10}{6} \right\rceil * 2 = 3+3+4 = 10$$

Iterate until $a_{n-1} = a_n$

$a_3 = a_4 < 12$, *so system is schedulable*

Utilization Bound Test for RMS

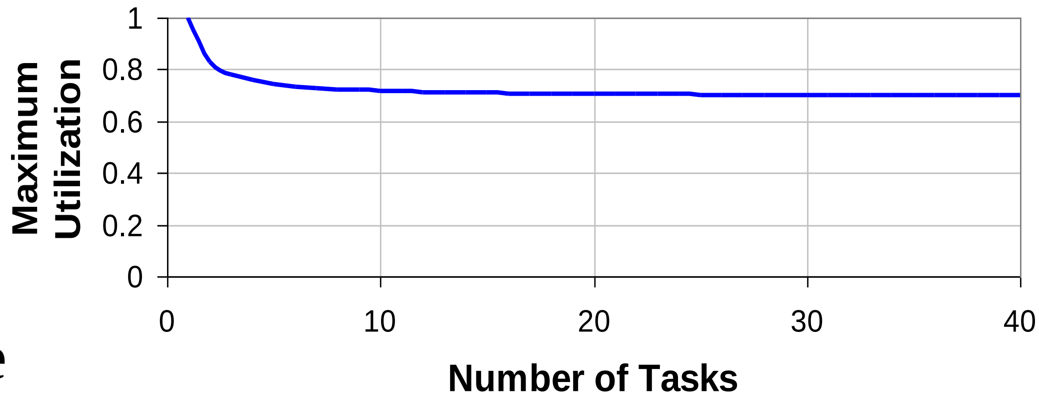
- Utilization U for n tasks
  - Fraction of time spent on tasks

$$U = \sum_{i=1}^{n} \frac{T_i}{\tau_i}$$

- Maximum utilization U_{Max} for m tasks
  - Max. value of U for which we can guarantee RMS works

$$U_{Max} = m\left(2^{1/m} - 1\right)$$

- Utilization bound test
  - $U < U_{Max}$: always schedulable with RMS
  - $U_{Max} < U < 1.0$: inconclusive
  - $U > 1.0$: Not schedulable



- Why is U_{Max} so small? (approaches ln(2))
  Conservative

Example of Scheduling with RMS and UB

Task	Exec. Time T	Period τ	Priority
P1	1	4	High
P2	2	6	Medium
P3	3	12	Low

$$U = \frac{T_1}{\tau_1} + \frac{T_2}{\tau_2} + \frac{T_3}{\tau_3} = \frac{1}{4} + \frac{2}{6} + \frac{3}{12} = 0.833$$

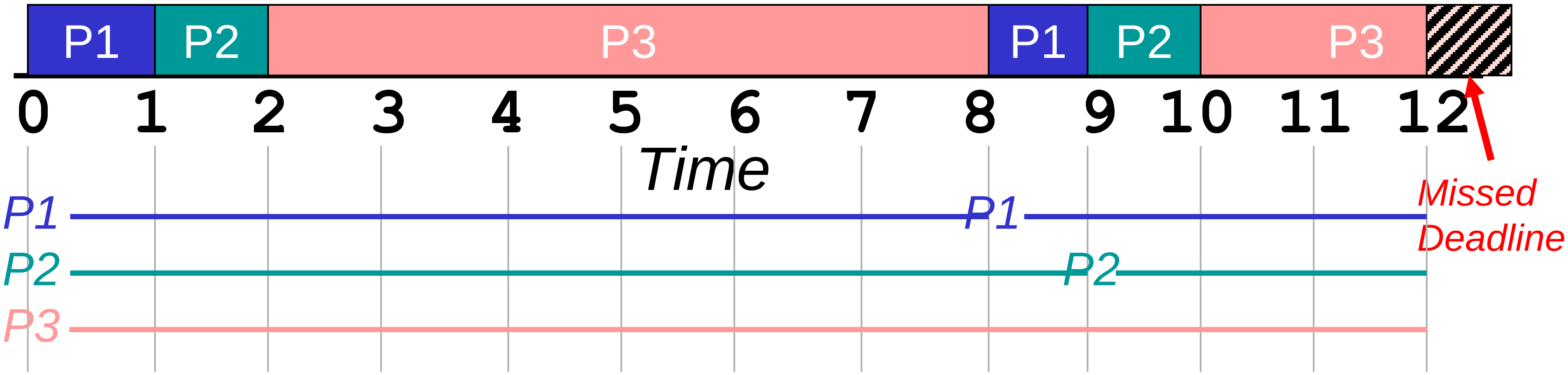
$$U_{Max} = m(2^{1/m} - 1) = 3(2^{1/3} - 1) = 0.780$$

0.833 > 0.780, so Utilization Bound Test is inconclusive

RMS Sometimes Fails Under 100% Utilization

- For some workloads with utilization below 100%, RMS priority allocation can fail
- Tasks P1, P2 have later deadlines than P3 yet preempt it due to their shorter periods

Thread	Exec. Time T	Period τ	Priority
P1	1	8	High
P2	1	9	Medium
P3	9	12	Low



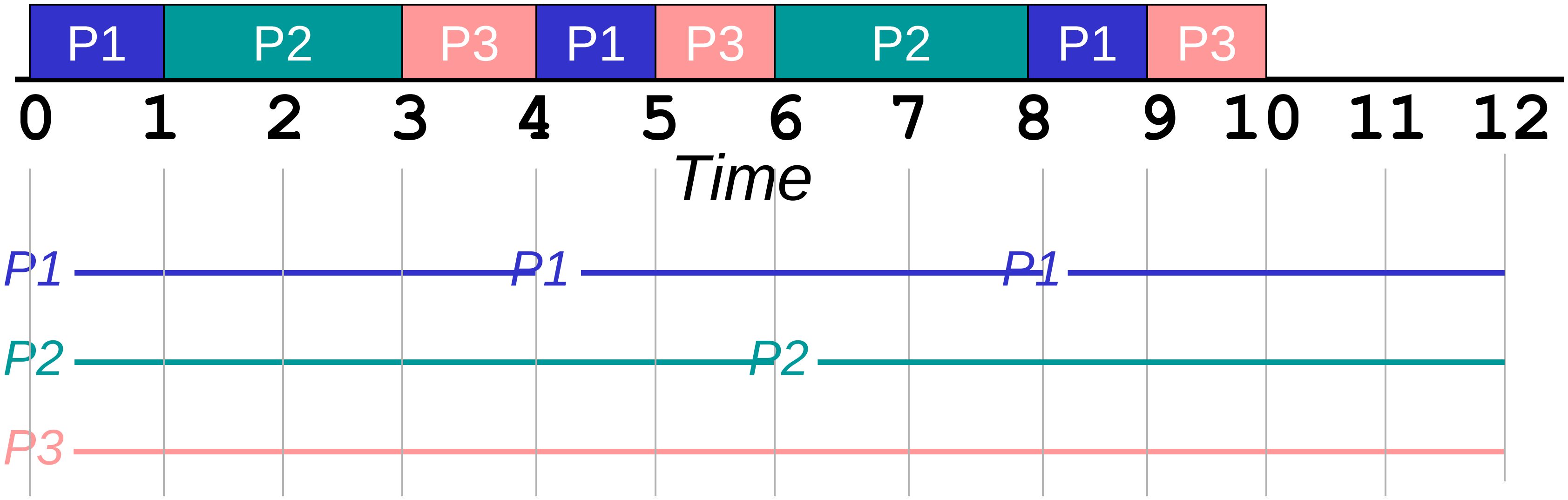
Counter-example provided by C. Palenchar

Earliest Deadline First

- Can guarantee schedulability at up to 100% utilization
- Can't use Exact Schedulability Test for EDF
  - Sum up all possible higher priority tasks, but priority depends on how close deadlines are!
  - Can we modify the test to deal with this?
- How does the kernel keep track of upcoming deadlines?
  - Can determine priority when inserting task into ready queue
    - Need to search through queue to find correct location (based on deadline)
  - Can determine which task to select from ready queue
    - Need to search through queue to find earliest deadline
  - Both are up to O(n) search time
    - Can also do binary search tree

Earliest Deadline First Example

Thread	Execution Time T	Period τ
P1	1	4
P2	2	6
P3	3	12



System Performance During Transient Overload

- RMS – Each task has fixed priority. *So?*
  - This priority determines that tasks will be scheduled consistently
    - Task A will always preempt task B if needed
    - Task B will be forced to miss its deadline to help task A meet its deadline

- EDF – Each task has varying priority. *So?*
  - This priority depends upon when the task's deadline is, and hence when the task becomes ready to run (*arrival time*)
    - Task B may have higher priority than A depending on arrival times
    - To determine whether task A or B will miss its deadline we need to know their arrival times