SYSC 5701
Operating System Methods for Real-Time Applications

Priority-Driven Scheduling for Periodic Tasks

Winter 2014

Assumptions (Liu Ch. 6)

1. no aperiodic or sporadic tasks
2. tasks are independent
3. uniprocessor

- will relax assumptions 1 & 2 later
  - aperiodic & sporadic → Liu Ch. 7
  - interdependency → Liu Ch. 8
    - Already seen “Access Control”!

Uniprocessor

- why not relax this assumption?
- multiprocessor typically managed by allocating a set of tasks to each processor
  - static: once allocated, task handled only by that processor
  - tasks do not migrate among processors

→ have a fixed task set for each processor

Priority-Driven Scheduling Algorithms

- **Static-(or Fixed-)Priority** – assigns the same priority to all jobs in a task.
- **Dynamic-Priority** – may assign different priorities to individual jobs within each task
  - e.g., earliest-deadline-first (EDF) algorithm

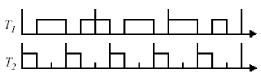
Static-Priority *vs.* Dynamic Priority

- **Static-Priority**: All jobs in task have same priority.
- example:

Rate-Monotonic: “The shorter the period, the higher the priority.”

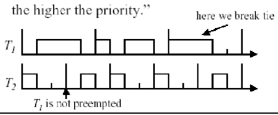
$T_1 = (5, 3, 5)$

$T_2 = (3, 1, 3)$



- **Dynamic-Priority**: May assign different priorities to individual jobs.
- example:

Earliest-Deadline-First: “The nearer the absolute deadline, the higher the priority.”



Processor Utilization

- recall that for a periodic task T_i , the ratio

  $u_i = e_i / p_i$ → **utilization of task T_i**
- the **total utilization U** of all tasks in a system is the sum of the utilizations of all individual tasks:

$$\boxed{U = \sum_{i=1}^{n} \frac{e_i}{p_i}}$$

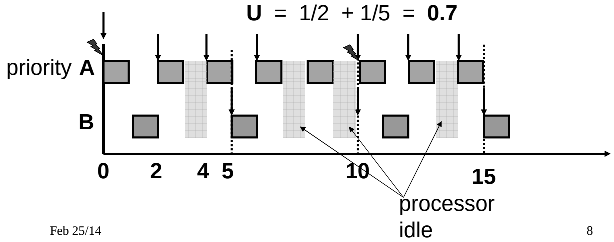
Fixed-Priority Scheduling of Periodic Tasks

1. consider some examples
2. consider some methods that can be used to determine the schedulability of a task set:
   - Utilization-based test
   - Response-time (or time-based) test

Example #1

Task T_i	Period p_i	Deadline D_i	Run-Time e_i
A (High Priority)	2	2	1
B (Low Priority)	5	5	1

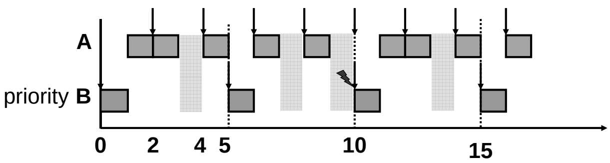
U = 1/2 + 1/5 = **0.7**



Example #2

Task T_i	Period p_i	Deadline D_i	Run-Time e_i
A (Low Priority)	2	2	1
B (High Priority)	5	5	1

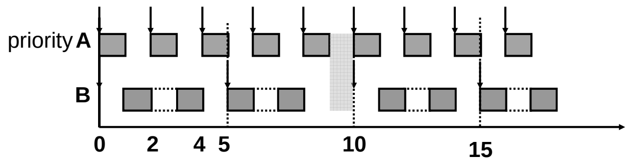
U = 1/2 + 1/5 = **0.7**



Example #3

Task T_i	Period p_i	Deadline D_i	Run-Time e_i
A (High Priority)	2	2	1
B (Low Priority)	5	5	2

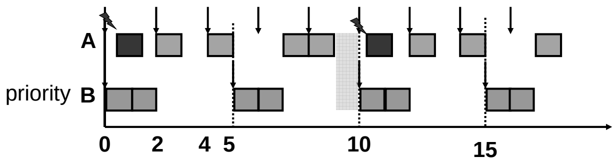
U = 1/2 + **2**/5 = **0.9**



Example #4

Task T_i	Period p_i	Deadline D_i	Run-Time e_i
A (Low Priority)	2	2	1
B (High Priority)	5	5	2

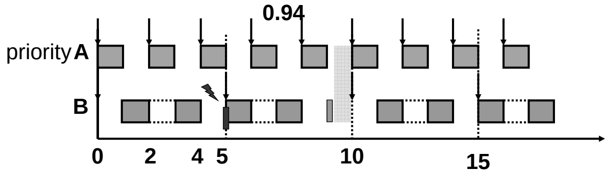
U = 1/2 + 2/5 = **0.9**



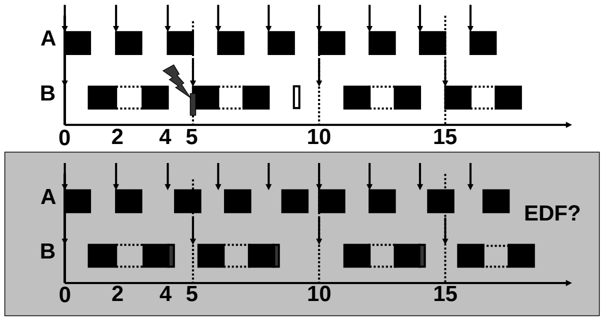
Example #5

Task T_i	Period p_i	Deadline D_i	Run-Time e_i
A (High Priority)	2	2	1
B (Low Priority)	5	5	2.2

U = 1/2 + 2.2/5 = **0.94**



Is There a Feasible Schedule for Example 5?



Analysis of Examples

- Changing the static priorities assigned to each task can impact the task set's feasibility
  - e.g., examples 3 and 4.
- Even if the total task utilization is less than 1.0, the task set may not have a feasible (<u>static</u>) priority assignment
  - e.g., example 5

Is there an upper bound on processor utilization that ensures schedulability?

Issues in Fixed Priority Assignment

- How to assign priorities?
- How to determine which assignment is the best; e.g., how to evaluate a priority assignment algorithm (method)?
- How to compare different priority assignment algorithms?

Fixed Priority Assignment Methods

- According to **execution times** (e_i)
  - smallest/largest execution time first
- According to **periods** (p_i)
  - smallest/largest period first
- According to **task utilization** (e_i / p_i)
  - smallest/ largest task utilization first
- Other? **Deadlines (DMA)**, etc.

Rate-Monotonic Algorithm (RM)

- **rate** (frequency) of task is inverse of its period

  $f_i = 1 / p_i$
- **higher rate (shorter period)** = **higher priority**

classic paper → posted! **read it!**

C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", JACM, Vol. 20, No. 1, pages 46-61, 1973.

Deadline-Monotonic Algorithm (DM)

- tasks with **shorter relative deadlines** are assigned **higher priorities**.
- when relative deadlines (D_i) equal to their periods (p_i), the rate-monotonic algorithm is the same as the deadline-monotonic algorithm.

Rate-Monotonic Assumptions

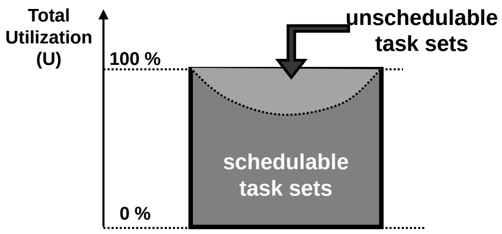
- tasks may be preempted
- tasks are periodic
- tasks execution times (e_i) are constant

Optimal Priority Assignment

- a given priority assignment algorithm is **optimal** <u>if</u> whenever a task set can be scheduled by some fixed priority assignment, <u>then</u> it can also be scheduled by the given algorithm
- Liu and Layland show that:
  – rate-monotonic algorithm is optimal

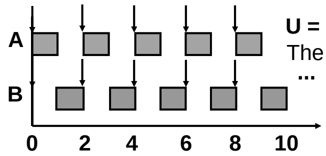
Maximum Achievable Utilization

A task set is **fully utilized** if any increase in run-time of any task would result in a missed deadline.



Example #6

Task T_i		Period p_i	Deadline D_i	Run-Time e_i
A	(High Priority)	2	2	1
B	(Low Priority)	2	2	1



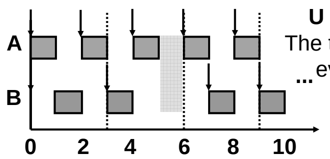
U = 1/2 + 1/2 = 1.0 = 100 %

The task set is fully utilized**.**

...

Example #7

Task T_i		Period p_i	Deadline D_i	Run-Time e_i
A	(High Priority)	2	2	1
B	(Low Priority)	3	3	1



U = 1/2 + 1/3 = 0.8333

The task set is fully utilized,

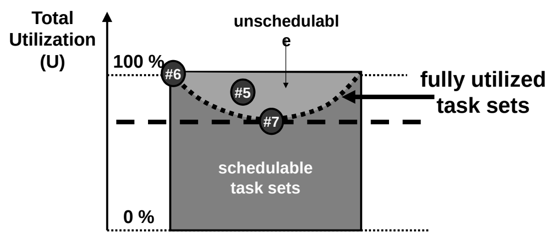
...even though U < 1.0**.**

increase e_A?

increase e_B?

Fully utilized task sets

different algorithms have different "fully utilized" curves



Utilization-Based Test

- **A sufficient, but not necessary,** test for schedulability of a task set that is assigned priorities using the rate-monotonic algorithm.
- Compute total task utilization **U(n) = U**.

Liu and Layland's Results

Theorem 2: If a feasible fixed priority assignment exists for some task set, then the rate-monotonic priority assignment is feasible for that task set.

Theorem 4: For a set of n tasks with fixed priority assignment, the least upper bound to the processor utilization factor is

$$\mathbf{U_{RM}(n) = n\ (\ 2^{1/n} - 1\ )}$$

Values for $U_{RM}(n)$

- U(1) = 1.0
- U(2) = 0.828
- U(3) = 0.779
- U(4) = 0.756
- U(∞) = 0.69 (ln 2)

RM Utilization Test

Utilization vs worst-case utilization bound

- also called **schedulable utilization**

  $\mathbf{U_{RM}(n) \longleftrightarrow U}$

  - If **U > 1** , then the task set **is not** schedulable
  - If $\mathbf{U} \le \mathbf{U_{RM}}(n)$, then the task set **is** schedulable
  - Otherwise: $\mathbf{U_{RM}}(n)$ **< U ≤ 1**
    - → **no conclusion** can be made
    - → try more detailed analysis

Response Time Tests

- for use when $\mathbf{U_{RM}}(n)$ **< U ≤ 1**
- analyze tasks to determine the worst case response time for jobs
- if worst case response of a job exceeds its deadline, then no feasible schedule
- for independent tasks, only delays are due to preemption by higher priority tasks

Worst-Case Simulation

- assume a critical instant for all tasks
- construct schedule according to the scheduling algorithm
- only need to consider largest task period
- if all tasks meet their deadlines
  - then tasks are feasibly schedulable

Time-Demand Analysis

- tasks place incremental demands on processor time
  - let $\omega_i(t)$ be demand from task i and <u>all higher priority tasks</u>
- processor delivers (processing) linearly
- check each task i, to be feasible:

  $\omega_i(t) = t$ for some $t \le p_i$
- how is this different from worst-case simulation?

schedule vs. calculation !

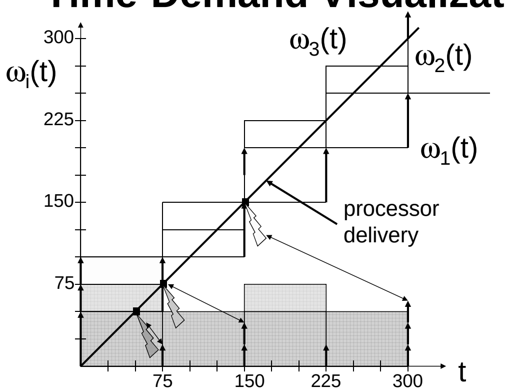
Example #8

T_1 :	$e_1 = 50$	$p_1 = 75$	$u_1 = 0.666$
T_2 :	$e_2 = 25$	$p_2 = 150$	$u_2 = 0.167$
T_3 :	$e_3 = 25$	$p_3 = 300$	$u_3 = 0.083$

$U = \sum u_i = \mathbf{0.916} > \mathbf{0.779}$ ← U(3)

$\therefore$ does not meet utilization bound!

Time-Demand Visualization



How to Solve ?

For each task:

- consider demand at <u>each</u> scheduling point
  - <u>before</u> next demand
- if task's demand ≤ delivery before deadline,
  - then feasible !

Example 9

T_1 :	$C_1 = 30$	$p_1 = 70$	$u_1 = 0.429$
T_2 :	$C_2 = 60$	$p_2 = 200$	$u_2 = 0.3$
T_3 :	$C_3 = 78$	$p_3 = 375$	$u_3 = 0.208$

$U = \sum u_i = \mathbf{0.937} > \mathbf{0.779}$ (U(3))

$\therefore$ does not meet utilization bound! ☹

Consider Each Task

- Task 1: highest priority → meets deadline

  30 (execution_1) ≤ 70 (period_1)
- Will need to know <u>Scheduling Points</u>:
  - periods: 70, 200, 375
  - scheduling points: (when new demand is released)

    0, 70, 140, 200, 210, 280, 350, 375

    all released

Continue (Task 2)

- Task 2: can only be delayed by Task 1
  - first scheduling point: t = **70** (T_1)
    - demand = 30 + 60 = **90** ☹
  - second scheduling point: t = **140** (T_1)
    - demand = 2*30 + 60 = **120** ☺

      <u>before</u> next release! i.e. <u>before</u> T_1 scheduled

Continue (Task 3)

Now for Task 3:

- first scheduling point: t = **70** (T_1)
  - demand = 30 + 60 + 78 = **168** ☹
- second scheduling point: t = **140** (T_1)
  - demand = 2*30 + 60 + 78 = **198** ☹
- third scheduling point: t = **200** (T_2)
  - demand = 3*30 + 60 + 78 = **228** ☹

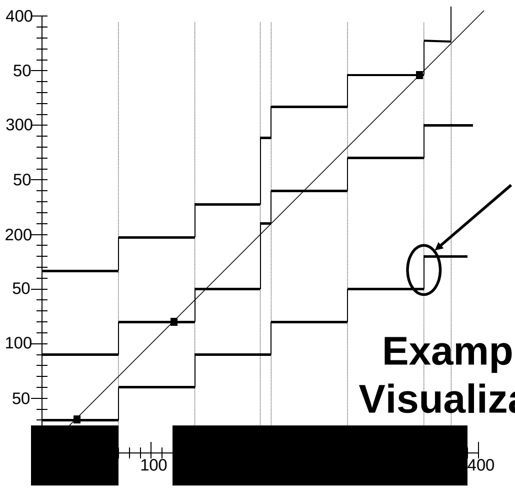
Continue (Task 3 con't)

- fourth scheduling point: t = **210** (T_1)
  - demand = 3*30 + 2*60 + 78 = **288** ☹
- fifth scheduling point: t = **280** (T_1)
  - demand = 4*30 + 2*60 + 78 = **318** ☹
- sixth scheduling point: t = **350** (T_1)
  - demand = 5* 30 + 2*60 + 78 = **348** ☺
- whew! all tasks feasible

Easier way?

- why not just check demand at end of p_3?
  - if T_3 meets deadline,
    then should have slack then?
- scheduling point: t = **375** (T_3)
  - demand = 6* 30 + 2*60 + 78 = **378** ☹

HUH?



Example 9 Visualization

Alternative ?

- let I_i be the delay in task i's response time due to higher priority tasks
- response time $\mathbf{R_i} = e_i + I_i$ (Equ. 1)
- worst case response time: task i and all higher priority tasks release a job at the same instant
  - I_i = sum of all higher priority jobs execution times

of Jobs Over an Interval

- suppose: periodic task j:
- number of jobs in [0, R) =

$$\left\lceil \frac{R}{p_j} \right\rceil$$

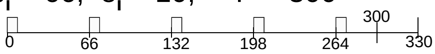
"ceiling" function: integer round-up

- delay to lower priority tasks due to these jobs is:

$$\left\lceil \frac{R}{p_j} \right\rceil e_j$$

Consider Task with period p_i Over Time Interval P: p_i < P

- p_i = 66, e_i = 10, P = 300



- work associated with task is "requested" at the beginning of each period
- there are $\lceil \mathbf{P/p_i} \rceil$ (max.) requests in interval
  e.g. 300 / 66 = 4.54, rounds up to 5
- to meet deadline P, must perform work $\lceil P/p_i \rceil$ times during P

of Higher Priority Jobs Over an Interval

- suppose: i periodic tasks, all with phase 0
- rate monotonic priority assignment
- total delay to task i due to higher priority tasks is:

$$I_i = \sum_{j=0}^{i-1} \left\lceil \frac{R_i}{p_j} \right\rceil e_j \qquad \text{(Equ. 2)}$$

Response Time Equ.

- Substitute Equ 2 into Equ 1:

$$R_i = e_i + \sum_{j=0}^{i-1} \left\lceil \frac{R_i}{p_j} \right\rceil e_j \qquad \text{(Equ. 3)}$$

- can solve using a recurrence relation:
  - initially, estimate $\mathbf{R_i^0 = e_i}$ (no delay)
  - use estimate to calculate better estimate, recurse
  - stop when solution found

Recurrence Relation

$$R_i^0 = e_i$$

substitute R_i^0
solve for R_i^1

$$R_i^{n+1} = e_i + \sum_{j=0}^{i-1} \left\lceil \frac{R_i^n}{p_j} \right\rceil e_j \qquad \ddots$$

substitute R_i^n
solve for R_i^{n+1}

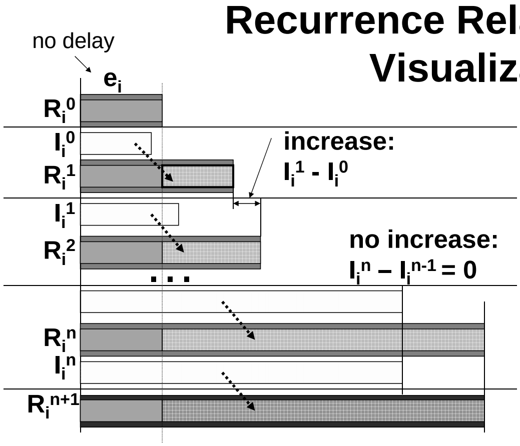
stop when $R_i^{n+1} = R_i^n$

if $R_i^n \le D_i$ then task i meets deadline!

Recurrence Relation: Conceptually

- initial estimate = execution time of task i
  - during this time, there will be delay from higher priority tasks
    - how much?
    - add this delay to estimate
    - results in "larger" time estimate
- stop when estimate does not increase

Recurrence Relation Visualization



Scheduling Visualization

- what does each estimate mean in terms of delay vs. the amount of task i execution ?
- what does each increase between estimates represent (in these terms) ?
- when the recursion stops, what does the absence of increase represent (in these terms) ?
- convince yourself that you understand this ☺

Response Time Analysis

- For each task, T_i , compute worst-case response time (R_i).
- If ($R_i \leq D_i$) for each task T_i **,** then the task set is feasible (schedulable).
- Response Time Analysis is both **<u>necessary</u>** and **<u>sufficient</u>**.
- How does this relate to Time-Demand Analysis?

Recall Example #8

T_1 :	$e_1 = 50$	$p_1 = 75$	$u_1 = 0.666$
T_2 :	$e_2 = 25$	$p_2 = 150$	$u_2 = 0.167$
T_3 :	$e_3 = 25$	$p_3 = 300$	$u_3 = 0.083$

U = $\sum u_i$ = **0.916** > **0.779** ← U(3)

∴ does not meet utilization bound!

let's work the recursive response time analysis on the board !

What about Assumptions?

1. deadline = period
   → now!
2. strictly periodic tasks (Liu Ch. 7)
   → next (Aperiodic)
2. tasks are independent (Liu Ch. 8)
   → next next (Access Control)

Arbitrary Response Times

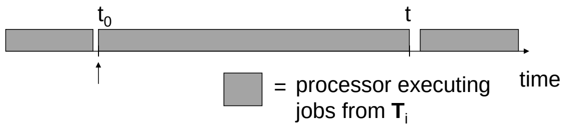
- $D_i \neq p_i$
- if $D_i < p_i$ → tighter deadline
- if $D_i > p_i$ → may have more than one released & ready job for task i
  - in these jobs assume FIFO scheduling of task i
- will use concept of level-π_i busy interval

Level-π_i Busy Interval $(t_0, t]$

- task subset $\mathbf{T}_i$ – all tasks with priority π_i or higher
- starts at $\mathbf{t_0}$, when:
  - all jobs in $\mathbf{T}_i$ released before t_0 have completed
  - a job in $\mathbf{T}_i$ is released
- ends at $\mathbf{t}$:
  - first instant after t_0 when all jobs in $\mathbf{T}_i$ released since $\mathbf{t_0}$ have completed

Conceptually

- no pending work from $\mathbf{T}_i$ when interval starts
- during interval, no slack time & processor always executing jobs with priority π_i or higher
- no pending work from $\mathbf{T}_i$ when interval ends



Critical Instant?

- Worst case load when all tasks in T_i release a job at t_0
  - → then critical instant!

Task i Schedulability Test

Assume:

- critical instant for $\mathbf{T}_i$ at t_0
- $\mathbf{T}_i$ contains all tasks with priority π_i or higher
- all tasks in $\mathbf{T}_i$ other than task i meet deadlines

1. If first job of each task (including $\mathbf{T}_i$)completes before end of its **period**, and $J_{i,1}$ meets deadline → schedulable!
   - if $J_{i,1}$ misses deadline → not schedulable

$\mathbf{T_i}$ Schedulability Test (con't)

2. If first job of some task does not complete before end of its period :
   a) compute length of level-π_i busy interval
   - solve recurrence relation:

$$\mathbf{R_i^{n+1}} = \sum_{j=0}^{i} \left\lceil \frac{\mathbf{R_i^n}}{\mathbf{p_j}} \right\rceil \mathbf{e_j}$$

$\mathbf{T_i}$ Schedulability Test (step 2 con't)

b) compute response time for each task i job in level-π_i busy interval – for response time of j^{th} job, solve:

$$\mathbf{R_i^{n+1}} = \mathbf{je_i} + \sum_{k=0}^{i-1} \left\lceil \frac{\mathbf{R_i^n}}{\mathbf{p_k}} \right\rceil \mathbf{e_k}$$

– if all task i jobs meet deadlines
  → schedulable

Is Test Finite?

- YES! **U ≤ 1** (has to be!) **AND** no slack time in level-π_i busy interval (by definition of interval)
  - → level-π_i busy interval is finite
  - → can find length of interval
  - → can find job response times