SPRING 2010

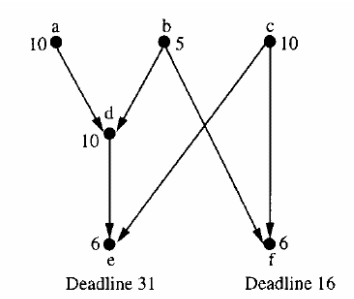
即時控制系統設計
Design of Real-Time Control Systems

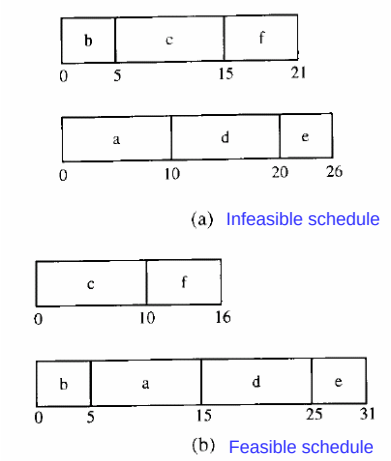
Lecture 13
Task Assignment & Scheduling

Feng-Li Lian
NTU-EE
Feb10 – Jun10

- Introduction
- Characterizing Real-Time Systems & Tasks
- Task Assignment & Scheduling
- Real-Time Programming Languages and Tools
- Real-Time Database
- Real-Time Communications
- Fault-Tolerance Techniques
- Reliability Evaluation Techniques
- Clock Synchronization

- An Example:





- The Question:
  - Will my real-time application really meet its timing constraints or requirements?
- The Problem:
  - Given a set of tasks, precedence constraints, resource requirements, their execution times, release times, and deadlines, and one or more processing systems
  - Assign tasks to different processing systems
  - Design a feasible/optimal allocation/scheduling on the processing system

- Definitions:
  - Tasks:
    - Consume resources (e.g., processor time, memory, input data), and
    - Put out one or more results
  - Precedence Constraints:
    - Specify if any task(s) needs to precede other tasks
    - Represented by the means of a precedence graph
  - Resource Requirements:
    - All tasks require
      - > some execution time on a processor,
      - > a certain amount of memory or
      - > access to a bus (network)
    - Exclusive or non-exclusive

- Definitions:
  - Release Time:
    - The time at which all the data that are required to begin executing the task are available
  - Deadline:
    - The time by which the task must complete its execution
    - Hard or soft, depending on the nature of the corresponding task
  - Relative Deadline:
    - The absolute deadline minus the release time

- Definitions:
  - Periodic:
    - The task is released periodically
    - Only to run exactly once every period; not required for being run exactly one period apart
  - Sporadic:
    - Not periodic, but at irregular intervals
    - Characterized by an upper bound on the rate at which the tasks may be invoked
  - Aperiodic:
    - Same as sporadic, OR
    - For not periodic and w/o upper bound on the invocation time

- Definitions:
  - Feasible:
    - A task assignment/schedule is said to be feasible if all tasks start after their release times and complete before their deadlines
  - A-Feasible:
    - If an assignment/schedule algorithm A results in a feasible schedule
  - Offline or Online Scheduling:
    - Schedule in advance
    - Schedule as the tasks arrive

Definitions:

- Priority:
  - A function of the nature of the tasks themselves and the current state of the controlled process
- Static- & Dynamic-Priority Algorithms:
  - Task priority does not change within a mode
  - Task priority can change with time
- Preemptive & Non-preemptive Schedule:
  - Tasks can be interrupted by other tasks (and then resumed)
    - > Flexibility
  - Task schedule must be run to completion or until it gets blocked over a resource
    - > Causing anomalies

Objective in Scheduling:

- For non-real-time applications
  - Minimize the total time required to execute all the tasks in the application
- For real-time applications
  - Meet the timing constraints of the individual tasks

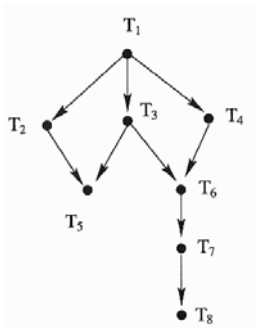
Characteristics in RT Scheduling Algorithms:

- Uniprocessor or multiprocessor
  - For multi-processors, shared memory or message-passing system
- Periodic or aperiodic
- Preemptible or non-preemptible
- Criticality
- Independence
- Resource
- Placement constraints
- Strictness of deadlines

Terminologies:

- Feasibility
- Optimality
- Lateness
- Absolute/relative/effective deadlines
- Absolute/effective release times
- Periodic, sporadic, aperiodic

- Components of Task Model:
  - Precedence relation: $\prec (T)$
    - Set of tasks that must be completed before task T can begin its execution
  - Resource requirements:
    - Processor, memory, bus, disk, etc.
    - Exclusive
    - Shared (read-only, read-write)
  - Schedule S:
    - { set of processors } X { time } → { set of tasks }
    - Off-line or online
    - Static or dynamic priority algorithm
    - Preemptive or non-preemptive
    - Uniprocessor or multiprocessor



- Commonly Used RT Scheduling Approaches:
  - Time-driven:
    - Determines when to execute which job
    - All parameters of hard RT jobs are fixed and known
    - A schedule is computed off-line and stored for use at runtime
  - Weighted round-robin:
    - For high-speed networks, where length of a round = sum of all weights
  - Priority-driven:
    - Assigns priorities to jobs and executes jobs in priority order
    - Static priority assignment:
      - > Rate or Deadline Monotonic (RM or DM)
    - Dynamic priority assignment:
      - > Earliest Deadline First (EDF), Minimum Laxity First (MLF)

- Four Paradigms of Scheduling Approaches:
  - Static table-driven scheduling:
  - Static priority preemptive scheduling:
  - Dynamic planning-based scheduling:
  - Dynamic best effort scheduling:
  - Impact of:
    - Quality-timeliness tradeoffs
    - Fault-tolerance constraints
    - Resource reclaiming on scheduling

- RTOS should have:
  - CPU scheduling
  - Resource allocation
  - Predictability, requiring bounded OS primitives
- RT Scheduling involves the allocation of resources and time to tasks

Analyzing Scheduling Algorithms:

- Performance metrics
- Scheduling paradigms
- Scheduling algorithms
- Other important scheduling issues

Performance metrics

- Static non-real-time systems
  - Minimize schedule length
- Dynamic non-real-time systems
  - Minimize response time
  - Increase throughput
- Both static & dynamic real-time systems
  - Achieve timeliness

Performance metrics

- Task characteristics:
  - Computation times
  - Resource requirements
  - Importance levels (or priorities, criticalness)
  - Precedence relationships
  - Communication requirements
  - Timing constraints

Performance metrics

- In static scheduling:
  - Since schedule off-line
  - So, meet all deadlines
  - If exists,
    - > Maximize average earliness
  - If not,
    - > Minimize average tardiness
- In dynamic scheduling:
  - Since information is not known a priori
  - So, maximize number of arrivals meeting deadlines

- Performance metrics
  - Levels of predictability:
    - Using a particular approach
      how well can we predict that
      the tasks will meet their deadlines?
  - Schedulability analysis or feasibility checking
    - Statically or dynamically

- Four Paradigms of Scheduling Approaches:
  - Static table-driven scheduling:
    - Static schedulability analysis
    - Resulting schedule (or table) used at run time
  - Static priority-based preemptive scheduling:
    - Static schedulability analysis
    - No explicit schedule
    - Highest priority task first
  - Dynamic planning-based scheduling:
    - Feasibility checked at run time
      - > Dynamically accept arriving task if feasible schedule found
  - Dynamic best effort scheduling
    - No feasibility check
    - Try its best to meet deadlines & may be aborted

- Static table-driven scheduling:
  - For periodic tasks
  - Given task characteristics,
    - Table is constructed by using , e.g., search heuristics
    - With Identifying start & completion times
    - Tasks dispatched according to table
  - Highly predictable, but highly inflexible

- Static priority-based preemptive scheduling:
  - Traditionally used for non-real-time systems
  - Tasks have priorities
    - Assigned maybe statically or dynamically or at any time
    - Execute highest-priority task
    - Preemption:
      - > Arrival of higher-priority tasks preempt
        the execution of low-priority task
    - If priorities are assigned systematically in such a way that
      timing constraints can be taken into account,
      then the resulting scheduler can also be used for real-time systems

- Dynamic planning-based scheduling:
  - With flexibility and predictability
  - For new arrival,
    - Try to create a schedule containing previously guaranteed tasks as well as the new arrival
    - If fail, take other actions

- Dynamic best effort scheduling
  - A priority-driven preemptive approach
    - > e.g., use deadlines as priorities & without any planning
  - Priority is computed based on task's characteristics
  - Schedule based on priority
  - Confidence via extensive simulations
  - Lack of predictability and sub-optimality
  - Try its best to meet deadlines
  - But, do NOT know whether a timing constraint will be met

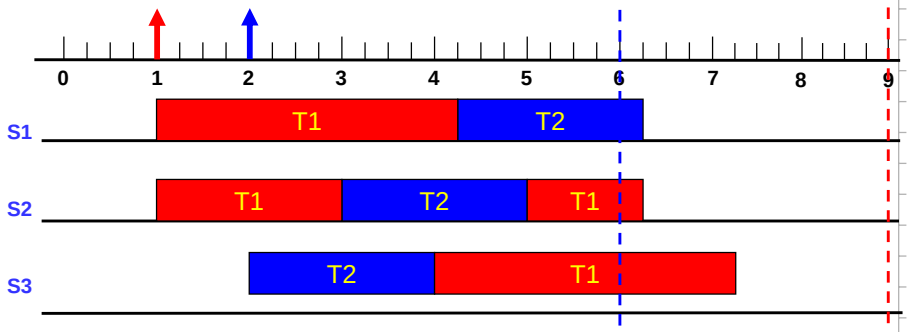
- Uniprocessor Scheduling Algorithms:
  - When to execute (scheduling)
- Multiprocessor Scheduling Algorithms:
  - Where to execute (assignment), and
  - When to execute (scheduling)
  - They are NP-hard; so, need heuristics to find suboptimal solutions

- Notations: $T_i = (I_i, P_i, e_i, d_i)$
  - n: Number of tasks in the task set
  - e_i: Execution time of task T_i
  - P_i: Period of task T_i, if it is periodic
  - I_i: kth period of (periodic) task T_i begins at time $I_i + (k-1)P_i$, where I_i is call the phasing of task T_i
  - d_i: Relative deadline of task T_i
  - D_i: Absolute deadline of task T_i
  - r_i: Release time of task T_i
  - $h_T(t)$: Sum of the execution times of task iterations in task set T that have their absolute deadlines <= t

- Assumptions:
  - A1: Fully preemptible with negligible costs
    - Can preempt any task at any time and resume it later without penalty
  - A2: CPU is the only resource to deal with
    - i.e., don't care with memory, I/O, etc.
  - A3: Independent task
    - i.e., no precedence constraints between tasks
  - A4: All periodic tasks
  - A5: Relative deadline = period

- Example: (a two-task system)

Time\Task	T1	T2
Release Time	1	2
Deadline	9	6
Execution Time	3.25	2



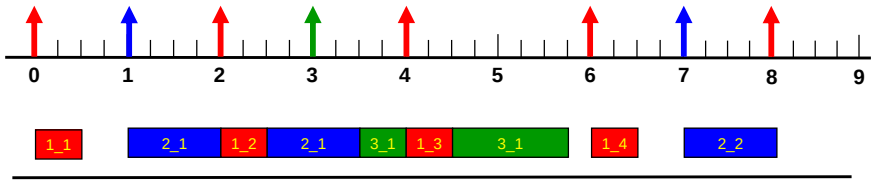
- Rate Monotonic (RM) Algorithm:
  - Assign higher priorities to tasks with lower periods (or higher rates)
    - The priority of a task is inversely related to its period
    - Higher-priority tasks can preempt lower-priority tasks
  - Optimal fixed priority scheduling algorithm
  - Sufficient schedulability condition:

$$U = \sum_{i=1}^{n} \frac{e_i}{P_i} \leq n(2^{1/n} - 1) \rightarrow 0.69 \quad \text{as } n \rightarrow \infty$$

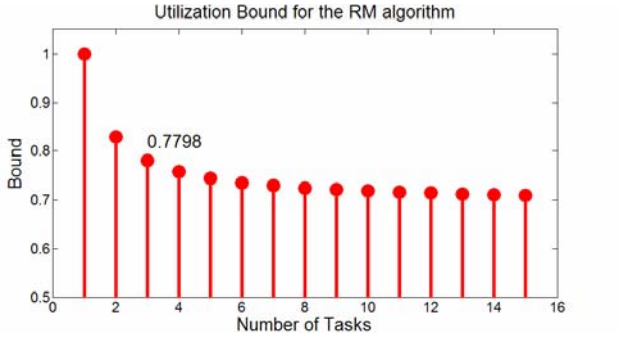
- Example: (a 3-task system)

Time\Task	T1	T2	T3
I	0	1	3
P	2	6	10
e	0.5	2	1.75

- Since P1 < P2 < P3, priority: T1 > T2 > T3

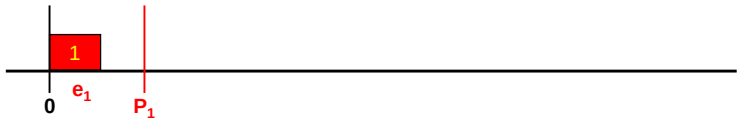


- Example: (a 3-task system)
  - Check sufficient schedulability condition



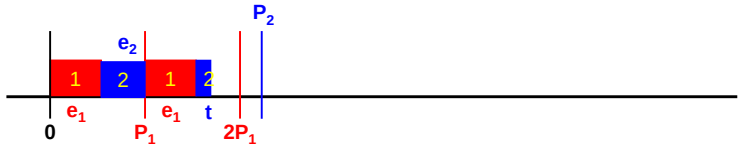
$$U = \frac{0.5}{2} + \frac{2}{6} + \frac{1.75}{10} = 0.7583$$

- Necessary (& Sufficient) Schedulability Conditions
  - T_1: feasibly scheduled



$$\Longleftrightarrow \quad e_1 \le P_1$$

- Necessary (& Sufficient) Schedulability Conditions
  - T_2: feasibly scheduled

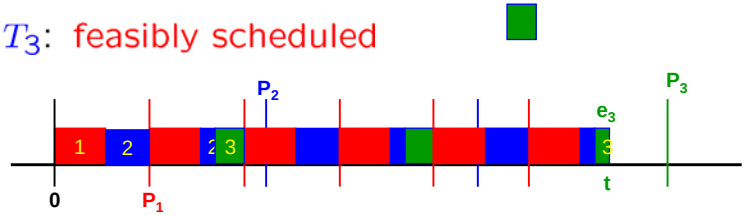


$$\Longleftrightarrow \quad t = \lceil \frac{t}{P_1} \rceil e_1 + e_2 \quad \& \quad t \in [0, P_2]$$

$$\Longleftrightarrow \quad t \ge \lceil \frac{t}{P_1} \rceil e_1 + e_2 \quad \& \quad t \le P_2$$

Check only t at multiples of P_1

- Necessary (& Sufficient) Schedulability Conditions
  - T_3: feasibly scheduled



$$\Longleftrightarrow \quad t = \lceil \frac{t}{P_1} \rceil e_1 + \lceil \frac{t}{P_2} \rceil e_2 + e_3 \quad \& \quad t \in [0, P_3]$$

$$\Longleftrightarrow \quad t \ge \lceil \frac{t}{P_1} \rceil e_1 + \lceil \frac{t}{P_2} \rceil e_2 + e_3 \quad \& \quad t \le P_3$$

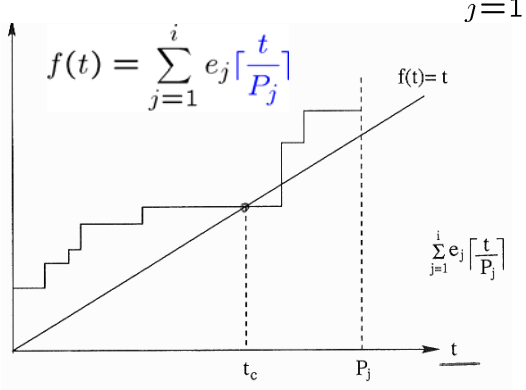
Check only t at multiples of P_1 & P_2

Necessary (& Sufficient) Schedulability Conditions

- $T = \{T_1, T_2, \cdots, T_n\}$ where $T_i = (P_i, e_i)$
- WLOG, assume $P_1 \le P_2 \le \cdots \le P_n$
- T_i released at $t = 0$
- T_i's completion time: t_c
- Within time t_c, T_i is preempted by each higher priority task T_j exactly $\lceil \frac{t_c}{P_j} \rceil$ times $\Rightarrow t_c = \sum_{j=1}^{i-1} e_j \lceil \frac{t_c}{P_j} \rceil + e_i$

Necessary (& Sufficient) Schedulability Conditions

- For schedulability, we must have $t_c \le P_i$
- $\forall i, \exists t_c \in [0, P_i]$ such that $t_c = \sum_{j=1}^{i} e_j \lceil \frac{t_c}{P_j} \rceil$



Theorem:

Given n periodic tasks with $P_1 \le P_2 \le \cdots \le P_n$, &
$W_i(t) = \sum_{j=1}^{i} e_j \lceil \frac{t}{P_j} \rceil$

THEN, task T_i: feasibly scheduled using RM
IFF $L_i = \min_{0<t\le P_i} \frac{W_i(t)}{t} \le 1$

$\Rightarrow$ In fact, only need to compute $W_i(t)$ at

$$\tau_i = \left\{ kP_j \;\middle|\; j = 1, 2, \cdots, i;\ k = 1, \cdots, \lfloor \frac{P_i}{P_j} \rfloor \right\}$$

Example: (a 4-task system)

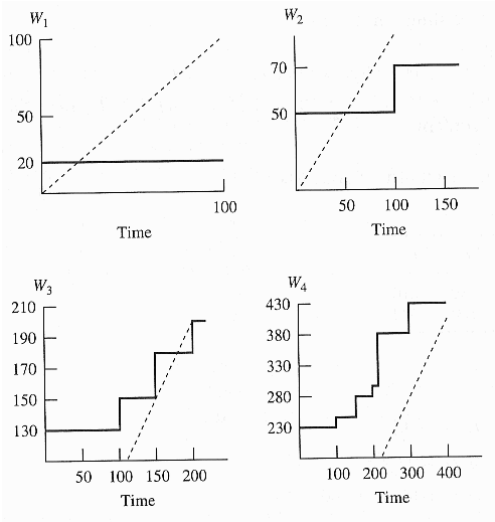
Time\Task	T1	T2	T3	T4
P	100	150	210	400
e	20	30	80	100

- Set of points of interest:

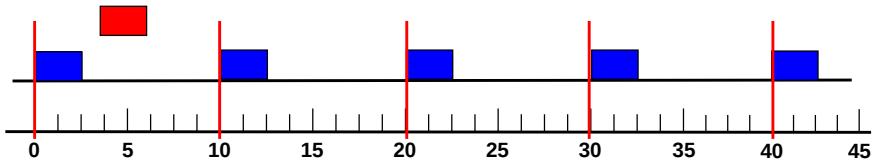
$$\begin{aligned} \tau_1 &= \{100\} \\ \tau_2 &= \{100, 150\} \\ \tau_3 &= \{100, 150, 200, 210\} \\ \tau_4 &= \{100, 150, 200, 210, 300, 400\} \end{aligned}$$

- Example: (a 4-task system)
  - Schedulability Conditions:
    - T_1 is RM-Schedulable iff
      $$e_1 \leq 100$$
    - T_2 is RM-Schedulable iff
      $$e_1 + e_2 \leq 100 \quad \text{OR}$$
      $$2e_1 + e_2 \leq 150$$
    - T_3 is RM-Schedulable iff
      $$e_1 + e_2 + e_3 \leq 100 \quad \text{OR}$$
      $$2e_1 + e_2 + e_3 \leq 150 \quad \text{OR}$$
      $$2e_1 + 2e_2 + e_3 \leq 200 \quad \text{OR}$$
      $$3e_1 + 2e_2 + e_3 \leq 210$$
    - T_4 is RM-Schedulable iff ...

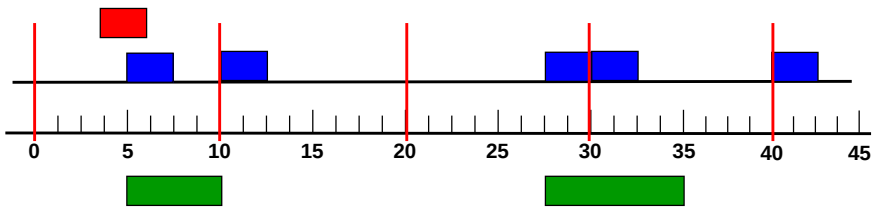
- Example: (a 4-task system)



- With Sporadic Tasks:
  - Define fictitious highest-periodic task & execution time



  - RM with Deferred Server (DS):

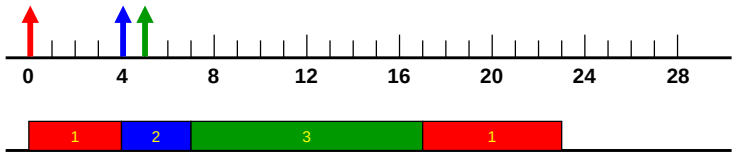


- Earliest Deadline First (EDF) Algorithm:
  - Assign higher priorities to tasks whose absolute deadline is the earliest
  - Optimal dynamic-priority scheduling algorithm
  - Tasks: periodic or aperiodic
  - Schedulable on a uniprocessor by the EDF iff:

$$U = \sum_{i=1}^{n} \frac{e_i}{P_i} \leq 1$$

- Example: (a 3-task system)

Time\Task	T1	T2	T3
Arrival Time	0	4	5
Execution Time	10	3	10
Absolute Deadline	30	10	25



at $t = 4$, $D_2 < D_1$, T_2 preempts T_1

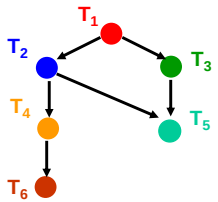
at $t = 5$, $D_2 < D_3$, T_2 continues, T_3 waits

at $t = 7$, $D_3 < D_1$, T_3's term

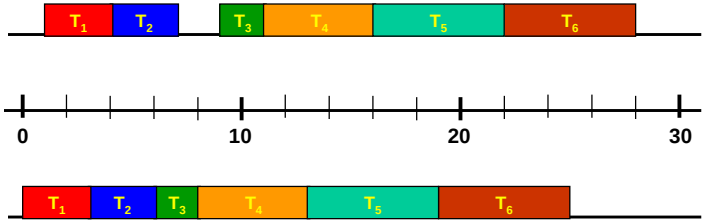
at $t = 17$, T_1 resumes

- Allowing for Precedence & Exclusion Conditions:

Task T_i	1	2	3	4	5	6
e_i	3	3	2	5	6	6
D_i	6	7	20	21	27	28

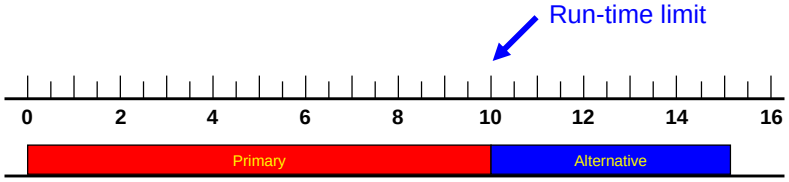


- All released at time 0



- Multiple Task Versions: Primary & Alternative
  - Better-quality service v.s. just-acceptable service

Time\Task	Primary	Alternative
Worst-case run time	20	5
Average run time	7	4
Period	15	15



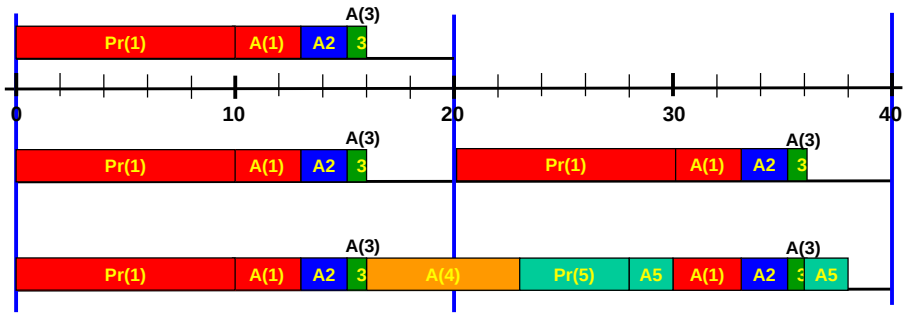
- Multiple Task Versions: 5-task system

Task T_i	1	2	3	4	5
l(i)	10	10	15	10	5
α(i)	3	2	1	7	4
P(i)	20	20	20	40	40

Run-time limit of primary version

Worst-case run-time of alternative version

Period



Increased Reward with Increased Service (IRIS):

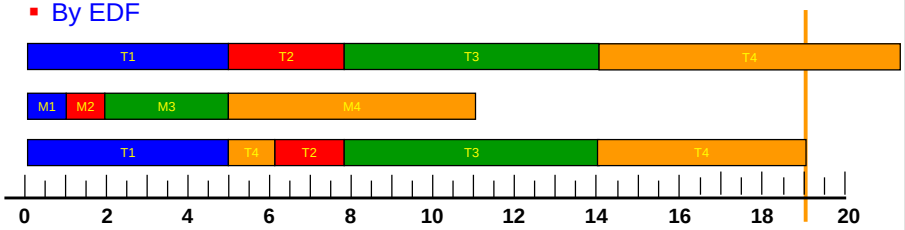
Mandatory **Optional**

Task	m_i	o_i	r_i	D_i
1	1	4	0	10
2	1	2	1	12
3	3	3	1	15
4	6	2	2	19

Identical Linear Reward Function

$$R_i(x) = \begin{cases} 0 & \text{if } x < m_i \\ x - m_i & \text{if } m_i \le x \le o_i + m_i \\ o_i & \text{if } o_i + m_i < x \end{cases}$$

- By EDF



Rate Monotonic (RM, static priority):

- Task set: periodic, preemptible, deadline = period
- Statically assign higher priorities to task with lower periods
- It is schedulable under RM
  if its total processor utilization <= $n(2^{1/n} - 1)$
- RM is an optimal static-priority uniprocessor scheduling algorithm

Rate Monotonic Deferred Server (DS):

- Similar to RM
- Handle both periodic and aperiodic tasks
- Allot some time slots for aperiodic tasks

Earliest Deadline First (EDF, dynamic priority):

- Tasks: preemptible
- The earliest the deadline, the higher the priority
- Optimal if preemption is allowed and
  jobs do not contend for resources
- If a task set is not schedulable on a single processor by EDF,
  no other processor can successfully schedule that task set

Precedence and Exclusion Conditions:

- Take precedence conditions into account
- Algorithm might be with exclusion conditions
  such as some tasks are not allowed to interrupt some,
  irrespective of priority

Multiple Task Versions:

- In some cases,
  the system has primary and alternative versions of some tasks
- Varying in execution time or quality of output they provide
- Primary version for top-quality output,
  alternative for lower-quality

Increased Reward with Increased Service (IRIS):

- Algorithm can be stopped early and output still useful
- Quality of output:
  a monotonically nondecreasing function of the execution time

- Rate Monotonic (RM, static priority):
- Earliest Deadline First (EDF, dynamic priority):
  - C.L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," Journal of ACM, 20(1):46-61, 1973
- Rate Monotonic Deferred Server (DS):
- Multiple Task Versions:
  - J.P. Lehoczky, L. Sha, and J.K. Strosnider, "Enhanced aperiodic responsiveness in hard real-time environments," Proc. IEEE Real-Time Systems Symposium, pp. 261-270, Los Alamitos, CA, 1987
- Precedence and Exclusion Conditions:
  - J. Xu and D.L. Parnas, "Scheduling processes with release times, deadlines, precedence, and exclusion properties," IEEE Trans. Software Engineering, 16(3): 360-369, Mar. 1990
- Increased Reward with Increased Service (IRIS):
  - J.W.S. Liu, K.J. Lin, W.-K. Shih, A.C. Yu, J.Y. Chung, and W. Zhao, "Imprecise computations," Proc. IEEE, 82(1): 83-94, Jan. 1994

- Task Assignment:
  - The optimal assignment of tasks to processors is, in almost all practical cases, an NP-complete problem
  - Do with heuristic procedures:
    - Allocate the tasks
    - Check their feasibility
    - If not feasible, modify the allocation
  - CANNOT guarantee that a feasibly scheduled allocation can be found
  - Need to account for communication costs

- Utilization Balancing Algorithm:
- Next-Fit Algorithm for RM Scheduling:
- Bin-Packing Algorithm for EDF:
- Myopic Offline Scheduling (MOS) Algorithm:
- Focused Addressing & Bidding (FAB) Algorithm:
- Buddy Strategy:
- Assignment with Precedence Constraints:

- Utilization Balancing Algorithm:
  - Tasks: preemptible
  - Assign tasks to processors one by one such that at the end of each step utilizations of various processors nearly balanced

$$\frac{\sum_{i=1}^{p}\left(u_i^B\right)^2}{\sum_{i=1}^{p}\left(u_i^*\right)^2} \le \frac{9}{8}\cdot\frac{p}{p-r+1} \xrightarrow[p>>r]{} 1.125$$

r : Copies of the same tasks

u_i^* : Utilization by using minimizing the sum of squares of process utilization

u_i^B : Utilization under the best-fit algorithm

Next-Fit Algorithm for RM Scheduling:

- Tasks: preemptible
- With RM uniprocessor scheduling algorithm
- Set of tasks → Various classes
- Set of processors → Each task class

Example: 4-Class & 11-Task

Task	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
e_i	5	7	3	1	10	16	1	3	9	17	21
P_i	10	21	22	24	30	40	50	55	70	90	95
u(i)	0.50	0.33	0.14	0.04	0.33	0.40	0.02	0.05	0.13	0.19	0.22
Class	C1	C2	C4	C4	C2	C2	C4	C4	C4	C4	C3

- By RM on each processor

Class	Bound
C1	(0.41,1.00]
C2	(0.26,0.41]
C3	(0.19, 0.26]
C4	(0.00, 0.19]

Processor	Tasks					
p1	T1					
p2	T2		T5	T6		
p3			T11			
p4	T3	T4			T7	T8,T9,T10
p5				T6		

Bin-Packing Algorithm for EDF:

- Tasks: preemptible
- Total utilizations <= a given threshold
- Threshold: the uniprocessor scheduling algorithm is able to schedule the tasks assigned to each processor
- Minimize the number of processors needed
  - > Many algorithms exist for solving it
  - > The First Fit Decreasing (FFD) algorithm

$$\frac{\text{Number of processors used by the FFD algorithm}}{\text{Number of processors used by optimal algorithm}} \longrightarrow \frac{11}{9} = 1.22$$

Example: 4-Class & 11-Task

Task	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
e_i	5	7	3	1	10	16	1	3	9	17	21
P_i	10	21	22	24	30	40	50	55	70	90	95
u(i)	0.50	0.33	0.14	0.04	0.33	0.40	0.02	0.05	0.13	0.19	0.22

L = (T1, T6, T2, T5, T11, T10, T3, T9, T8, T4, T7)

U = (U1, U2, U3, U4, …),
containing the total utilizations of processor p_i in Ui

L = (T1, T6, T2, T5, T11, T10, T3, T9, T8, T4, T7)

Step	Ti	u(i)	Pi	U = (U_1, U_2, U_3)
1	T1	0.50	p1	(0.50)
2	T6	0.40	P1	(0.90)
3	T2	0.33	p2	(0.90, 0.33)
4	T5	0.33	p2	(0.90, 0.66)
5	T11	0.22	p2	(0.90, 0.88)
6	T10	0.18	p3	(0.90, 0.88, 0.18)
7	T3	0.14	p3	(0.90, 0.88, 0.32)
8	T9	0.13	p3	(0.90, 0.88, 0.45)
9	T8	0.06	p1	(0.96, 0.88, 0.45)
10	T4	0.04	p1	(1.00, 0.88, 0.45)
11	T7	0.02	p2	(1.00, 0.90, 0.45)

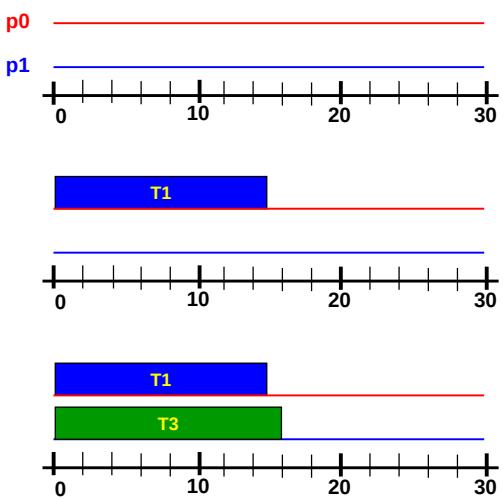
Myopic Offline Scheduling (MOS) Algorithm:

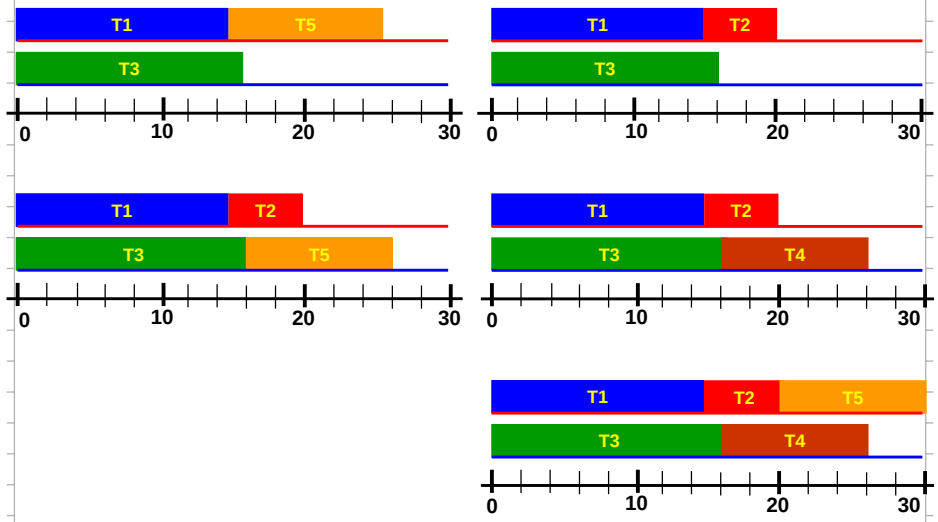
- Can deal with nonpreemptible tasks
- Build up a schedule tree and based on a search process to find feasible schedule minimizing a heuristic function H such as execution time, deadline, start time, laxity, etc.
- For n tasks, the schedule tree has n+1 levels (including the root)

Example: 5-(nonpreemptive)-Task & 2-Processor

Task	T1	T2	T3	T4	T5
r_i	0	10	0	15	0
e_i	15	5	16	9	10
D_i	15	20	18	25	50

$H(i) = r_i$





Focused Addressing & Bidding Algorithm:

- Tasks arrive at the individual processors
- If one processor finds itself unable to meet the deadline or other constraints,
- Then it tries to offload some of its workload onto other processors
- By announcing which task(s) it would like to offload and waiting for other processors to offer to take them up

Buddy Strategy:

- Roughly the same as the focused addressing algorithm
- Processor load: under-loaded, fully loaded, overloaded
- Overloaded ask under-loaded to take over some

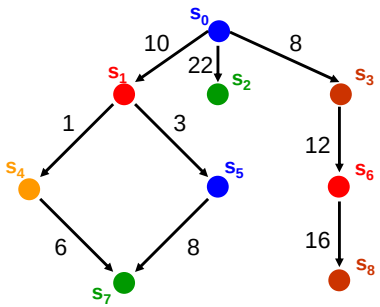
Assignment with Precedence Constraints:

- Take precedence into account
- Use a trial-and-error process to assign tasks that communicate heavily with one another
- So that communication costs are minimized

Example: 1-Task & 8-Subtask, 2-Processor

Subtask	e_i	D_i	LFT
s0	4	-	7
s1	10	-	24
s2	15	22	22
s3	4	-	26
s4	18	-	42
s5	3	-	42
s6	6	-	32
s7	3	45	45
s8	8	40	40

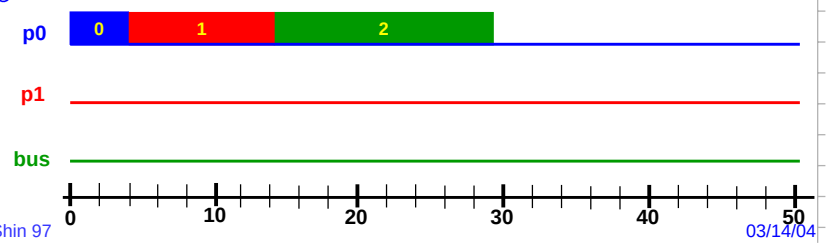
LFT: Latest finishing time

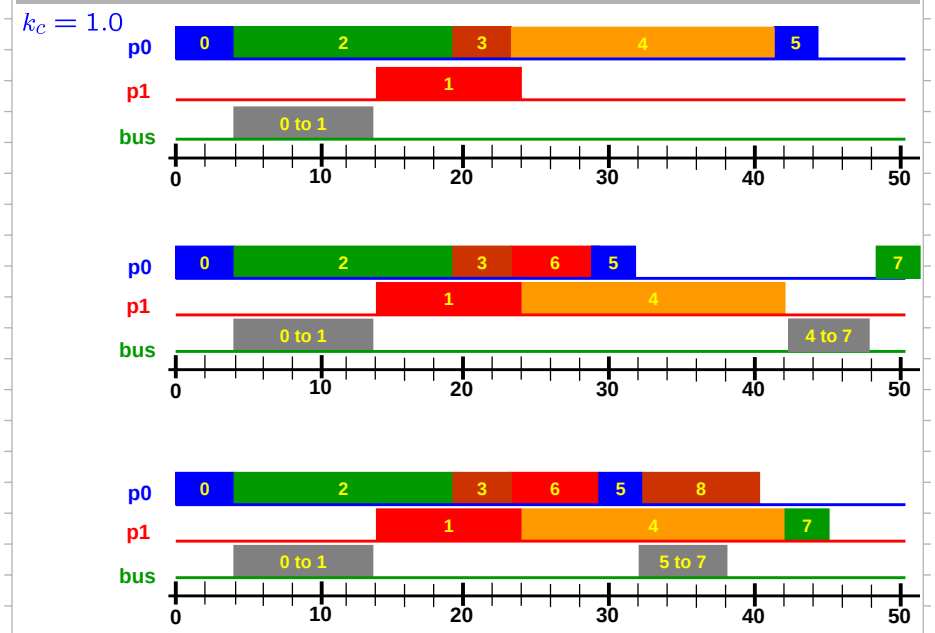


Pair si,sj	e_i+e_j	c_{ij}	$(e_i+e_j)/c_{ij}$
s0 s1	14	10	1.40
s0 s2	19	22	0.86
s0 s3	8	8	1.00
s1 s4	28	14	2.00
s1 s5	13	3	4.33
s4 s7	21	6	3.50
s5 s7	6	8	0.75
s3 s6	10	12	0.83
s6 s8	14	16	0.88

$$\frac{e_i + e_j}{c_{ij}} < k_c$$

$k_c = 1.5$





- Static algorithms:
  - Periodic tasks with hard deadlines
  - Not applicable to aperodic tasks b/c timing info unknown
- Dynamic algorithms:
  - Centralized
    - > All tasks distributed by one central processor into others
    - > So, processors' load is known and deadlines are guaranteed
  - Distributed
    - > Tasks arrive independently at each processor
    - > Transfer policy: guarantee constraints of incoming tasks
    - > Location policy: find other processors if not schedulable
    - > Information policy: collect & maintain state info of others

- Utilization Balancing Algorithm:
  - Tasks: preemptible
  - Assign tasks to processors one by one such that at the end of each step utilizations of various processors nearly balanced
- Next-Fit Algorithm for RM Scheduling:
  - Tasks: preemptible
  - With RM uniprocessor scheduling algorithm
  - Set of tasks → Various classes
  - Set of processors → Each task class

- Bin-Packing Algorithm for EDF:
  - Tasks: preemptible
  - Total utilizations <= a given threshold
  - Threshold: the uniprocessor scheduling algorithm is able to schedule the tasks assigned to each processor
- Myopic Offline Scheduling (MOS) Algorithm:
  - Can deal with nonpreemptible tasks
  - Build up a schedule tree and based on a search process to find feasible schedule minimizing a heuristic function such as execution time, deadline, start time, laxity, etc.

- Focused Addressing & Bidding (FAB) Algorithm:
  - Tasks arrive at the individual processors
  - If one processor finds itself unable to meet the deadline or other constraints,
    - > Then it tries to offload some of its workload onto other processors
  - By announcing which task(s) it would like to offload and waiting for other processors to offer to take them up
- Buddy Strategy:
  - Roughly the same as the focused addressing algorithm
  - Processor load: under-loaded, fully loaded, overloaded
  - Overloaded ask under-loaded to take over some

- Assignment with Precedence Constraints:
  - Take precedence into account
  - Use a trial-and-error process to assign tasks that communicate heavily with one another
  - So that communication costs are minimized

- Utilization Balancing Algorithm:
  - J.A. Bannister and K.S. Trivedi, “Task allocation in fault-tolerant distributed systems,” Acta Informatica, 20(3): 261-281, Sep. 1983
- Next-Fit Algorithm for RM Scheduling:
  - S. Davari and S.K. Dhall, “An on line algorithm for real-time tasks allocation,” Proc. IEEE Real-Time Systems Symposium, pp. 194-200, Los Alamitos, CA, 1986
- Bin-Packing Algorithm for EDF:
  - E.G. Coffman, Computer and Job-Shop Scheduling Theory, Wiley, New York, 1976
- Myopic Offline Scheduling (MOS) Algorithm:
  - K.J. Ramamritham, A. Stankovic, and P.-F. Shiah, “Efficient scheduling algorithms for real-time multiprocessor systems,” IEEE Trans. on Parallel and Distributed Systems, 1(2): 184-194, Apr. 1990
- Focused Addressing & Bidding (FAB) Algorithm:
  - K.J. Ramamritham, A. Stankovic, and W. Zhao, “Distributed scheduling of tasks with deadlines and resource requirements,” IEEE Trans. on Computers, 38(8): 1110-1123, Aug. 1989
- Buddy Strategy:
  - K.G. Shin and Y.-C. Chang, “Load sharing in distributed real-time systems with state-change broadcasts,” IEEE Trans. on Computers, 38(8): 1124-1142, Aug. 1989