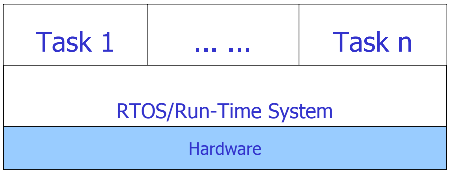
Overall Stucture of Real Time Systems



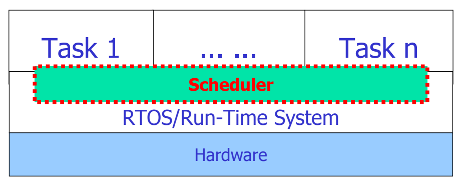
So far, we have talked about

- Programming Languages to implement the Tasks
- Run-TIme/Operating Systems to run the Tasks

Question

How to schedule the Tasks such that given timing constraints are satisfied?

Overall Stucture of Real Time Systems



Today's topic:

REAL TIME SCHEDULING (BASICS)

Task models

- Non periodic/Aperiodic (three parameters)
  - A: arrving time
  - C: computing time
  - D: deadline (relative deadline)

Constraints on task sets

- Timing constraints: deadline for each task,
  - Relative to arriving time or absolute deadline
- Other constraints
  - Precedence constraints
    - Precedence graphs imposed e.g by input/output relation
  - Resource constraints: mutual exclusion
    - Resource access protocols

Scheduling Problems

Given a set of tasks (ready queue)

1. Check if the set is schedulable
2. If yes, construct a schedule to meet all deadlines
3. If yes, construct an optimal schedule e.g. minimizing response times

Tasks with the same arrival time

Assume a list of tasks

(A,C1, D1)(A,C2, D2) ...(A,Cn,Dn)

that arrive at the same time i.e. A

- How to find a feasible schedule?
- (OBS: there may be many feasible schedules)

Earlist Due Date first (EDD) [Jackson 1955]

- EDD: order tasks with nondecreasing deadlines.
  - Simple form of EDF (earlist deadline first)
- Example: (1,10)(2,3)(3,5)
  - Schedule: (2,3)(3,5)(1,10)
- FACT: EDD is optimal
  - If EDF cann't find a feasible schedule for a task set, then no other algorithm can, i.e. The task set is non schedulable.

EDD: Schedulability test

- If C1+C2...+Ck <=Dk for all k<=n for the schedule with nondescreasing ordering of deadlines, then the task set is schedulable
- Response time for task i, Ri =C1+...+Ci
- Prove that EDD is optimal ?

EDD: Examples

- (2, 4)(1,5)(6,10) is schedulable:
  - Feasible schedule: (2,4)(1,5)(6,10)
  - Note that (1,5)(2,4)(6,10) is also feasible
- (1,10)(3,3)(2,5) is schedulable
  - The feasible schedule: (3,3)(2,5)(1,10)
  - Why not shortest task first?
- (4,6)(1,10)(3,5) is not schedulable
  - (3,5)(4,6)(1,10) is not feasible: 3+4 > 6!

EDD: optimality

- Assume that Ri is the finishing time (relative to the release time) of task i. Note that R means response time. Let Li = Ri-Di (the lateness for task i)
- FACT: EDD is optimal with respect to minimizing the maximum lateness Lmax= MAXi(Li) (the general form of optimality of EDD)
- Note that even a task set is non schedulable, EDD may minimize the maximal lateness (minimizes loss for soft tasks?)

EDD: Exercises

- Prove: EDD is optimal in finding a feasible schedule
- Program the schedulability test for EDD

Tasks with different arrival times

- Assume a list of tasks
  - S= (A1,C1, D1)(A2,C2, D2) ...(An,Cn,Dn)
- Preemptive EDF [Horn 1974]:
  - Whenever new tasks arrive, sort the ready queue according to earlist deadlines first at the moment
  - Run the first task of the queue if it is non empty
- FACT: Preemptive EDF is optimal [Dertouzos 1974] in finding feasible schedules.

Preemptive EDF: Schedulability test

- At time Ai, if the list ordered according to EDF
  (A'1,C'1,D'1)(A'2,C'2,D'2)...(A'i,C'i,D'i)
  satisfies C'1+...+C'k <=D'k for all k=1,2...i, then S is schedulable at time Ai
- If S is schedulable at all Ai's, S is schedulable

Preemptive EDF: Example

Consider (1, 5, 11)(2,1,3)(3, 4,8)

- Deadlines are relative to arrival times

- At 1, (5,11)
- At 2, (1,3)(4,10)
- At 3, (4,8)(4,9)

Preemptive EDF: Response time calculation

- Complicated
- But possible

Preemptive EDF: Exercises

- Write a program to calculate the response times for (non)preemptive EDF

19

Preemptive EDF: Optimality

- Assume that Ri is the finishing time (relative to the release time) of task i. Note that R means response time. Let Li = Ri-Di (the lateness for task i)
- FACT: preemptive EDF is optimal with respect to minimizing the maximum lateness Lmax= MAXi(Li) (the general form of optimality of preemptive EDF)

20

Non preemptive EDF (on-line version)

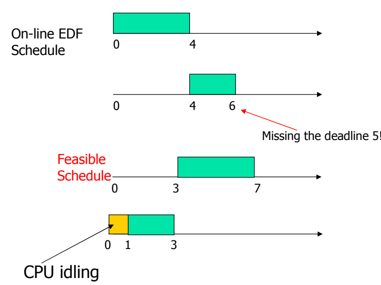
- Alternative 1: Run a task until it's finished and then sort the queue according to EDF
  - + The algorithm may be run on-line, easy to implement, less overhead (no more context switch than necessay)
  - - However it is not optimal, it may not find the feasible schedule even it exists e.g (0,5,20)(1,1,3)(6,7,30): the second task misses its deadline. Note that the feasible schedule: (1,1,3)(0,5,20)(6,7,30)

21

On-line non preemptive EDF: example

	T1	T2
A	0	1
C	4	2
D	7	5

Assume that D is absolute deadline



22

On-line non preemptive EDF: Optimal?

- If we only consider non-idle algorithms (CPU waiting only no task to run), is EDF is optimal?
- Unfortunately no!
- Example
  - T1= (0, 10, 100)
  - T2= (0,1,101)
  - T3= (1,4,4)
  - Run T1,T3,T2: the 3rd task will miss its deadline
  - Run T2,T3,T1: it is a feasible schedule

23

Off-line Non preemptive EDF (complete search)

- Alternative 2: the decision should be made according to all the parameters in the whole list of tasks
- Consider the example: (0,5,20)(1,1,3)(6,7,30)

24

Off-line Non preemptive EDF (NP-hard)

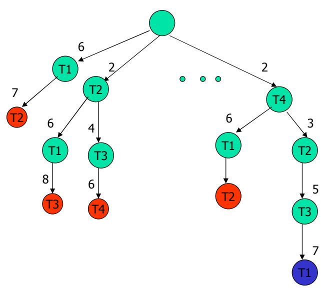
- Unfortunately, to find a feasible non-preemptive schedule for task set with different arrival times is not easy
- The worst case is to test all possible combinations of n tasks (NP-hard, difficult for large n)

Practical methods: Bratley's algorithm

- Search until a non-schedulable situation occur, then backtrack [Bratley's algorithm]
  - simple and easy to implement but may not find a schedule if n is too big (worst case)

Example (Bratley's alg.)

	T1	T2	T3	T4
A	4	1	1	0
C	2	1	2	2
D	7	5	6	4



Heuristic methods: Spring algorithm

- Similar to Bratley's alg. But
  - Use heuristic function H to guide the search until a feasible schedule is found, otherwise backtrack: add a new node in the search tree if the node has smallest value according to H e.g H(task i) = Ci, Ai, Di etc [Spring alg.]
  - However it may be difficult to find the right H

Example Heuristics

- H(Ti) = Ai FIFO
- H(Ti) = Ci SJF
- H(Ti) = Di EDF
- H(Ti) = Di +w*Ci EDF+SJF
- ...

EDF: + and –

- Simple (+)
- Preemptive EDF, Optimal (+)
- No need for computing times (+)
- On-line and off-line (+)
- Preemptive schedule easy to find (+)
- But preemptive EDF is "difficult" to implement efficiently (-)
  - Must use a list of "timers", one per task
- Nonpreemptive (feasible) schedule difficult to find (-)
  - But minimal context switch (+)
  - And the only way to schedule non preemtive tasks

Other scheduling algorithms

- Classical ones
  - HPF (priorities = degrees of importance of tasks)
  - Weighted Round Robin
- LRT (Latest Release Time or reverse EDF)
- LST (Least Slack Time first)

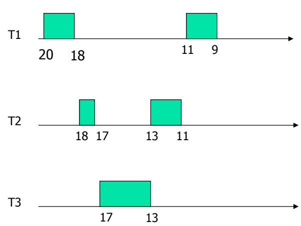
Latest Release Time (reversed EDF)

- Release time = arrival time
- Idea: no advantage to completing any hard task sooner than necessary. We may want to postpone the execution of hard tasks e.g to improve response times for soft tasks.
- LRT: Schedule tasks from the latest deadline to earliest deadline. Treat deadlines as 'release times' and arrival times as 'Deadlines'. The latest 'Deadline' first
- FACT: LRT is optimal in finding feasible schedule (for preemptive tasks)

LRT: Example

	T1	T2	T3
A	0	11	12
C	4	3	4
D	20	18	17

(D=absolute deadline)



Reverse time: we get the schedule:
T1(9,11)T2(11,13)T3(13,17)T2(17,18)T1(18,20)
OBS: from 0 to 9, soft tasks may be running!

LRT: + and -

- It needs Arrival times (-)
- It got to be an off-line algorithm (-)
- Only for preemptive tasks (-)
- It could optimize Response times for soft tasks (+)

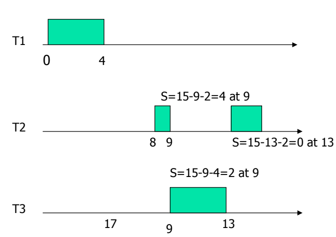
Least slack time first (LST)

- Let Si = Di-Ci (the Slack time for task i)
  - Si is the maximal (tolerable) time that task i can be delayed
- Idea: there is no point to complete a task earlier than its deadline. Other (soft) tasks may be executed first
  - Slack stealing
- LST: order the queue with nondecreasing slack times
- FACT: preemptive LST is optimal in finding feasible schedules

LST: Example

	T1	T2	T3
A	0	8	9
C	4	3	4
D	20	15	15

(D=absolute deadline)



Comment: a task should run until a Slack reaches 0 (to avoid context switch)
And if more than one 0-slack: nonschedulable

LST: + and –

- It needs Computing times (-)
- Only for preemptive tasks (-)
- Not easy to implement! (-)
- But it can run on-line (+)
- and it may improve response times?

Independent tasks

- OBS! we have assumed that tasks are independent!
  - meaning that we can compute them in arbitrary orderings only if the orderings (schedules) are feasible
- All algorithms we have studied so far are applicable only to independent tasks

Summary: scheduling independent tasks

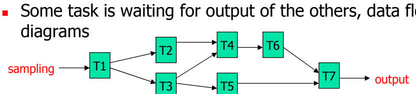
Task types	Same arrival times	Preepmtive Different arrival times	Non preemptive Different arrival times
Algorithms For Independent tasks	EDD,Jackson55 O(n log n), optimal	EDF, Horn 74 O(n**2), Optimal LST, LRT optimal	Tree search Bratley'71 O(n n!), optimal Spring, Stankovic et al 87 O(n**2), Heuristic

Dependent tasks

- In practice, tasks are dependent. We often have conditions or constraints e.g.
  - A must be computed before B
  - B must be computed before C and D
- Such conditions are called precedence constraints which can be represented as *Directed Acyclic Graphs* (DAG) known as Precedence graphs
- Such graphs are also known as **"Task Graph"**

Dependent tasks: Examples

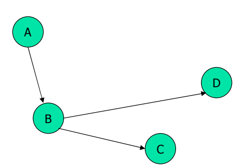
- Input/output relation
  - Some task is waiting for output of the others, data flow diagrams



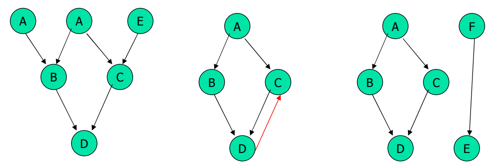
- Synchronization
  - Some task must be finished before the others e.g. It is holding a shared resource
- Other dependence relations (e.g priority-orderings?)

Precedence graph: Example

- A must be computed before B
- B must be computed before C and D



Precedence graph: Examples



Not a precedence graph!

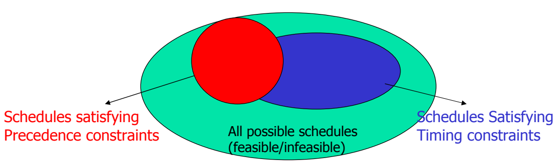
Conjunct and Disjunct join: We will only consider conjunct join!

AND/OR-precedence graphs

- AND-node, all incomming edges must be finished first
- OR-node: some of the incomming edges must be finished

Scheduling under Timing and Precedence constraints

- Feasible schedules should meet
  - Timing constraints: deadlines and also
  - Precedence constraints: Precedence graphs
- Overlapping area of blue and red is what we need
- Precedence constraints restrict the search area (Guiding!)



Dependent tasks with the same arrival times

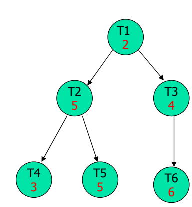
- Assume a list of tasks:
  (A,C1,D1)(A,C2,D2) ...(A,Cn,Dn)
- In addition to the deadlines D1...Dn, the tasks are also constrained by a DAG
- Solution: Latest Deadline First (LDF), Lawler 1973
- FACT: LDF is optimal (in finding feasible schedules)

Latest Deadline First (LDF)

- It constructs a schedule from tail to head using a queue:
  1. Pick up a task from the current DAG, that
     - Has the latest deadline and
     - Does not precede any other tasks (a leaf!)
  2. Remove the selected task from the DAG and put it to the queue
- Repeat the two steps until the DAG contains no more tasks. Then the queue is a potentilly feasible schedule. The last task selected should be run first.
- Note that this is similar to LRT

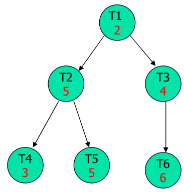
LDF: Example

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6



LDF: Example

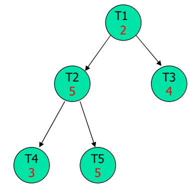
	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6



LDF: T6

LDF: Example

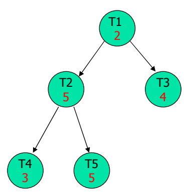
	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6



LDF: T6

LDF: Example

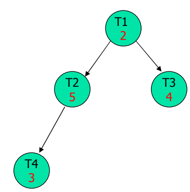
	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6



LDF: T6,T5

LDF: Example

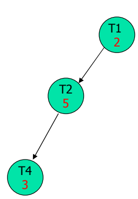
	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6



LDF: T6,T5

LDF: Example

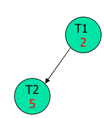
	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6



LDF: T6,T5,T3

LDF: Example

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6



LDF: T6,T5,T3,T4

LDF: Example

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6

LDF: T6,T5,T3,T4,T2

LDF: Example

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6

LDF: T6,T5,T3,T4,T2,T1

LDF: Example

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6

LDF: T6,T5,T3,T4,T5,T1

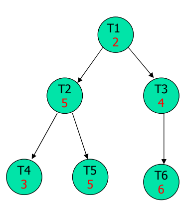
Feasible Schedule

Earlest Deadline First (EDF)

- It is a variant of LDF, but start with the root of the DAG:
  1. Pick up a task with earlest deadline among all nodes that have no fathers (the roots)
  2. Remove the selected task from the DAG and put it to the queue
- Repeat the two steps until the DAG contains no more tasks. Then the queue is a feasible schedule.
- Unfortunately, EDF is not optimal (see the following example)

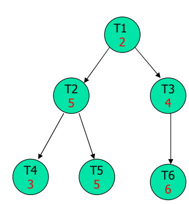
LDF: Example

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6



LDF: Example

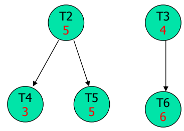
	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6



EDF: T1

EDF: Example

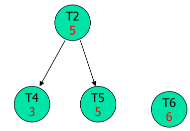
	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6



EDF: T1

LDF: Example

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6



EDF: T1,T3

LDF: Example

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6

EDF: T1,T3,T2

LDF: Example

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6

EDF: T1,T3,T2,T4,T5,T6

LDF: Example

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6

T4 will miss its Deadline: 3

LDF: T6,T5,T3,T4,T2,T1
Feasible

EDF: T1,T3,T2,T4,T5,T6
Infeasible

Dependent tasks with different arrival times

- Assume a list of tasks:
  S = (A1,C1,D1)(A2,C2,D2)...(A3,Cn,Dn)
- In addition to the deadlines D1...Dn, the tasks are also constrained by a DAG
- Solution: The Complete Search guided by the DAG
  - The Bratley's algorithm
  - The Spring algorithm

Better algorithms?

- Assume a list of tasks:
  S = (A1,C1,D1)(A2,C2,D2)...(A3,Cn,Dn)
- In addition to the deadlines D1...Dn, the tasks are also constrained by a DAG
- Idea:
  - Transform the task set S (constrained by the DAG) to an Independent task set S* such that
    S is schedulable under DAG iff S* is schedulable

Idea: how to transform S to S*?

- Idea:
  If Ti ->Tj is in the DAG i.e. Ti must be executed before Tj, we replace the arrival time for Tj and deadline for Ti with
  - Aj* = max(Aj, Ai+Ci)
    - Tj can not be computed before the completion of Ti
  - Di*=min(Di,Dj-Cj)
    - Ti should be finished early enough to meet the deadline for Tj

Algorithm (EDF*): transform S to S*

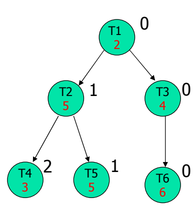
- Let arrival times and deadlines be 'absolute times'
- Step 1: Transform the arrival times from roots to leafs
  - For all initial (root) nodes Ti, let Ai* = Ai
  - REPEAT:
    - Pick up a node Tj whose fathers arrival times have been modified. If no such node, stop. Otherwise:
    - Let Aj* =max(Aj, max{Ai*+Ci: Ti->Tj})
- Step 2: Transform the deadlines from leafs to roots
  - For all terminal (leafs) nodes Tj, let Dj* = Dj
  - REPEAT:
    - Pick up a node Ti all whose sons deadlines have been modified. If no such node, stop. Otherwise:
    - Let Di* =min(Di, min{Dj*-Cj: Ti->Tj})
- Step 3: use EDF to schedule S*=(A1*,C1,D1*)...(An*.Cn,Dn*)

EDF*: optimality

FACT:
- S is schedulable under a DAG iff S* is schedulable
- EDF* is optimal in finding a feasible schedule

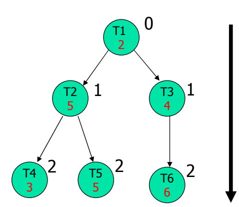
Example

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6
A	0	1	0	2	1	0



EDF*: Example(1)

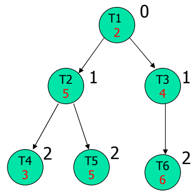
	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6
A	0	1	0	2	1	0



Step 1: Modifying the arrival times (top-down)

EDF*: Example(1)

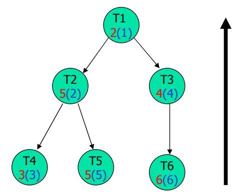
	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6
A*	0	1	1	2	2	2



Step 1: Modifying the arrival times (top-down)

EDF*: Example(2)

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D	2	5	4	3	5	6
A*	0	1	1	2	2	2

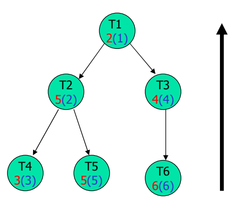


Step 2: Modifying the deadlines (bottom-up)

EDF*: Example(2)

S*

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D*	1	2	4	3	5	6
A*	0	1	1	2	2	2

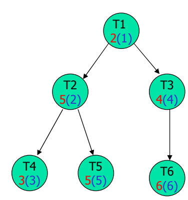


Step 2: Modifying the deadlines (bottom-up)

EDF*: Example(3)

S*

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D*	1	2	4	3	5	6
A*	0	1	1	2	2	2



Step 3: now we don’t need the DAG any more!

EDF*: Example(3)

S*

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D*	1	2	4	3	5	6
A*	0	1	1	2	2	2

Step 3: schedule S* using EDF

EDF*: Example(3)

S*

	T1	T2	T3	T4	T5	T6
C	1	1	1	1	1	1
D*	1	2	4	3	5	6
A*	0	1	1	2	2	2

Finally we have a schedule: T1,T2,T4,T3,T5,T6

Summary: scheduling aperiodic tasks

Task types	Same arrival times	Preepmtive Different arrival times	Non preemptive Different arrival times
Algorithms for Independent tasks	EDD,Jackson55 O(n log n), optimal	EDF, Horn 74 O(n**2), Optimal LST, optimal LRT, optimal	Tree search Bratley'71 O(n n!), optimal Spring, Stankovic et al 87 O(n**2) Heuristic
Algorithms for Dependent tasks	LDF, Lawler 73 O(n**2) Optimal	EDF* Chetto et al 90 O(n**2) optimal	Spring As above