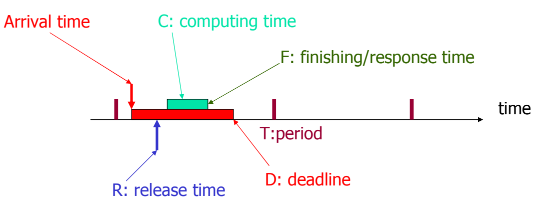
SCHEDULING PERIODIC TASKS

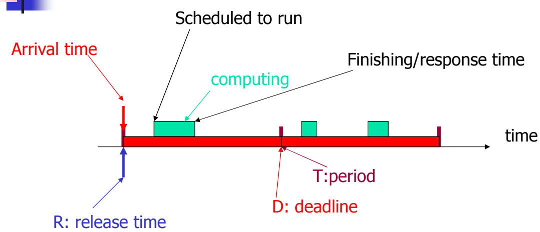
The simple case: Cyclic execution

Repeat a set of aperiodic tasks
at a specific rate (cycle)

Periodic tasks



Periodic tasks (the simplified case)



Assumptions on task sets

- Each task is released at a given constant rate
  - Given by the period T
- All instances of a task have:
  - The same worst case computing time: C (reasonable)
  - The same relative deadline: D=T (not a restriction)
  - The same relative arrival time: A=0 (not a restriction)
  - The same release time R=0, released as soon as they arrive
- All tasks are independent
  - No sharing resources (consider this later)
- All overheads in the kernel are assumed to be zero
  - E.g context switch etc (consider this later)

Periodic task model

- A task = (C, T)
  - C: worst case execution time/computing time (C<=T!)
  - T: period (D=T)
- The simplest case: a task = T
  - C=unknown, D=T
- A task set: (Ci,Ti)
  - All tasks are independent
  - The periods of tasks start at 0 simultaneously (not necessary)

CPU utilization

- C/T is the CPU utilization of a task
- U=Σ Ci/Ti is the CPU utilization of a task set
- Note that the CPU utilization is a measure on how busy the processor could be during the shortest repeating cycle: T1*T2*...*Tn
  - U>1 (overload): some task will fail to meet its deadline no matter what algorithms you use!
  - U<=1: it will depend on the scheduling algorithms
    - If U=1 and the CPU is kept busy (non idle algorithms e.g. EDF), all deadlines will be met

Scheduling periodic tasks

- Assume a set of independent periodic tasks: (Ci,Ti)
- Schedulability analysis:
  - is it possible to meet all deadlines in all periods?
  - A task set is schedulable/feasible if it can be scheduled so that all instances of all tasks meet deadlines
- If yes,
  - how to schedule all task instances to meet all deadlines?
  - Optimal scheduling algorithms?

Solutions: SCS,EDF,EMS,DMS

- Static Cyclic Scheduling (SCS)
- Earliest Deadline First (EDF)
- Rate Monotonic Scheduling (RMS)
- Deadline Monotonic Scheduling (DMS)

Static cyclic scheduling

- Shortest repeating cycle = least common multiple (LCM)
- Within the cycle, it is possible to construct a static schedule i.e. a time table
- Schedule task instances according to the time table within each cycle

Example: the Car Controller

Activities of a car control system. Let

1. C= worst case execution time
2. T= (sampling) period
3. D= deadline

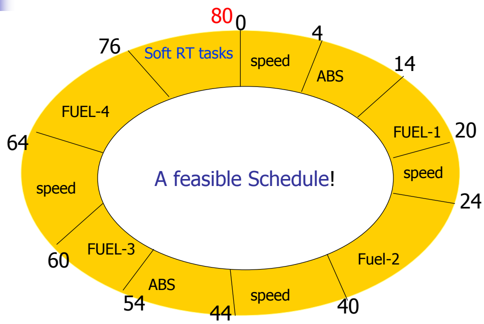
- Speed measurment: C=4ms, T=20ms, D=20ms
- ABS control: C=10ms,T=40ms, D=40ms
- Fuel injection: C=40ms,T=80ms, D=80ms
- Other software with soft deadlines e.g audio, air condition etc

The car controller: static cyclic scheduling

- The cycle = 80ms
- The tasks within the cycle:

0	20	40	60	80
Speed ABS Fuel	Speed	Speed ABS	Speed	

The car controller: time table



Static cyclic scheduling: + and –

- Deterministic: predictable (+)
- Easy to implement (+)
- Inflexible (-)
  - Difficult to modify, e.g adding another task
  - Difficult to handle external events
- The table can be huge (-)
  - Huge memory-usage
  - Difficult to construct the time table

Example: shortest repeating cycle

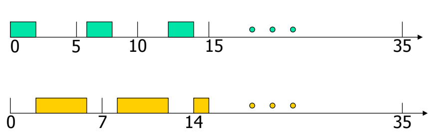
- OBS: The LCM determines the size of the time table
  - LCM =50ms for tasks with periods: 5ms, 10ms and 25ms
  - LCM =7*13*23=2093 ms for tasks with periods: 7ms, 13ms and 23ms (very much bigger)
- So if possible, manipulate the periods so that they are multiples of each other
  - Easier to find a feasible schedule and
  - Reduce the size of the static schedule, thus less memory usage

Earliest Deadline First (EDF)

- Task model
  - a set of independent periodic tasks (not necessarily the simplified task model)
- EDF:
  - Whenever a new task arrive, sort the ready queue so that the task closest to the end of its period assigned the highest priority
  - Preempt the running task if it is not placed in the first of the queue in the last sorting
- FACT 1: EDF is optimal
  - EDF can schedule the task set if any one else can
- FACT 2 (Scedulability test):
  - Σ Ci/Ti <= 1 iff the task set is schedulable

Example

- Task set: {(2,5),(4,7)}
- U = 2/5 + 4/7= 34/35 ~ 0.97 (schedulable!)



EDF: + and –

- Note that this is just the simple EDF algorithm; it works for all types of tasks: periodic or non periodic
  - It is simple and works nicely in theory (+)
  - Simple schedulability test: U <= 1 (+)
  - Optimal (+)
  - Best CPU utilization (+)
- Difficult to implement in practice. It is not very often adopted due to the dynamic priority-assignment (expensive to sort the ready queue on-line), which has nothing to do with the periods of tasks. Note that Any task could get the highest priority (-)
- Non stable: if any task instance fails to meet its deadline, the system is not predictable, any instance of any task may fail (-)

We use periods to assign static priorities: RMS ➔

Rate Monotonic Scheduling: task model

Assume a set of periodic tasks: (Ci,Ti)

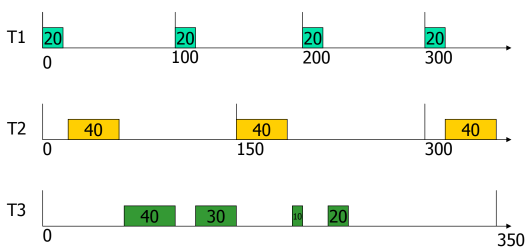
- Di=Ti
- Tasks are always released at the start of their periods
- Tasks are independent

RMS: fixed/static-priority scheduling

- Rate Monotonic Fixed-Priority Assignment:
  - Tasks with smaller periods get higher priorities
- Run-Time Scheduling:
  - Preemptive highest priority first
- FACT: RMS is optimal in the sense:
  - If a task set is schedulable with any fixed-priority scheduling algorithm, it is also schedulable with RMS

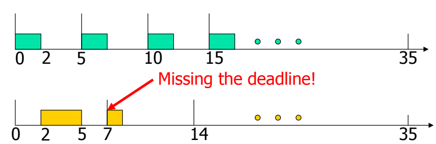
Example

{(20,100),(40,150),(100,350)} Pr(T1)=1, Pr(T2)=2, Pr(T3)=3



Example

- Task set: T1=(2,5), T2=(4,7)
- U = 2/5 + 4/7= 34/35 ~ 0.97 (schedulable?)
- RMS priority assignment: Pr(T1)=1, Pr(T2)=2



RMS: schedulability test

- U<1 doesn't imply 'schedulable' with RMS
  - OBS: the previous example is schedulable by EDF, not RMS
- It is always possible in theory to construct a schedule for the shortest repeating cycle (LCM) and check if the schedule is feasible or not, BUT this may be difficult/impossible in practice if the number of tasks is too big
- Idea: utilization bound
  - Given a task set S, find X(S) such that U<= X(S) if and only if S is schedulable by RMS (necessary and sufficient test)
  - Note that the bound X(S) for EDF is 1

The famous Utilization Bound test (UB test) [by Liu and Layland, 1973: a classic result]

- Assume a set of n independent tasks:
  - S= {(C1,T1)(C2,T2)...(Cn,Tn)} and U = Σ Ci/Ti
- FACT: if $U \le n(2^{1/n}-1)$, then S is schedulable by RMS
- Note that the bound depends only on the size of the task set

UB test is only sufficient, not necessary!

- Let U= Σ Ci/Ti and B(n) = $n*(2^{1/n}-1)$
- Three possible outcomes:
  - 0<= U<= B(n): schedulable
  - B(n)<U<=1: no conclusion
  - 1< U : overload
- Thus, the test may be too conservative, more precise test (in fact, exact test) will be given later
- (Unfortunately, it is not obvious how to calculate the necessary utilization bound for a given task set)

Example: Utilization bounds

B(1)=1.0	B(4)=0.756	B(7)=0.728
B(2)=0.828	B(5)=0.743	B(8)=0.724
B(3)=0.779	B(6)=0.734	U(∞)=0.693

Note that U(∞)=0.693 !

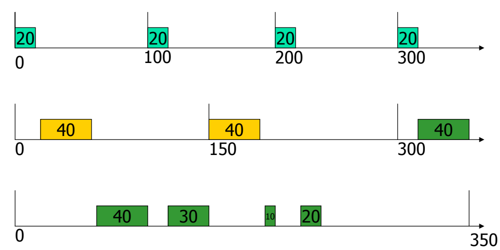
Example: applying UB Test

	C	T (D=T)	C/T
Task 1	20	100	0.200
Task 2	40	150	0.267
Task 3	100	350	0.286

Total utilization: U=0.2+0.267+0.286=0.753<B(3)=0.779!
The task set is schedulable

Example: Run-time RM Scheduling

{(20,100),(40,150),(100,350)}

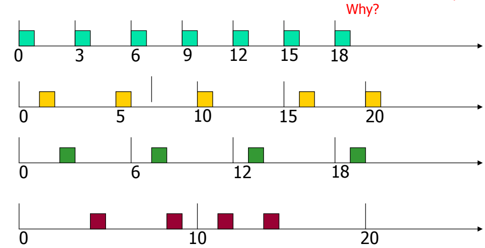


Example: UB test is sufficient, not necessary

- Assume a task set: {(1,3),(1,5),(1,6),(2,10)}
- CPU utilization U= 1/3+1/5+1/6+2/10=0.899
- The utilization bound B(4)=0.756
- The task set fails in the UB test due to U>B(4)
- Question: is the task set schedulable?
- Answer: YES

{(1,3),(1,5),(1,6),(2,10)}

Response times?
Worst case? First period?
Why?



How to deal with tasks with the same period

- What should we do if tasks have the same period?
- Should we assign the same priority to the tasks?
- How about the UB test? Is it still sufficient?
- What happens at run time?

RMS: Summary

- Task model:
  - priodic, independent, D=T, and a task= (Ci,Ti)
- Fixed-priority assignment:
  - smaller periods = higher priorities
- Run time scheduling: Preemptive HPF
- Sufficient schedulability test: $U <= n*(2^{1/n}-1)$
- Precise/exact schedulability test exists

RMS: + and –

- Simple to understand (and remember!) (+)
- Easy to implement (static/fixed priority assignment)(+)
- Stable: though some of the lower priority tasks fail to meet deadlines, others may meet deadlines (+)

- "lower" CPU utilization (-)
- Requires D=T (-)
- Only deal with independent tasks (-)
- Non-precise schedulability analysis (-)
- But these are not really disadvantages;they can be fixed (+++)
  - We can solve all these problems except "lower" utilization

Critical instant: an important observation

- Note that in our examples, we have assumed that all tasks are released at the same time: this is to consider the critical instant (the worst case senario)
  - If tasks meet the first deadlines (the first periods), they will do so in the future (why?)
- Critical instant of a task is the time at which the release of the task will yield the largest response time. It occurs when the task is released simultaneously with higher priority tasks
- Note that the start of a task period is not necessarily the same as any of the other periods: but the delay between two releases should be equal to the constant period (otherwise we have jitters)

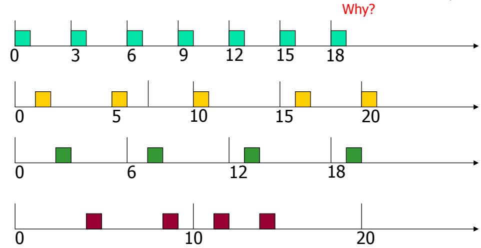
Sufficient and necessary schedulability analysis

- Simple ideas [Mathai Joseph and Paritosh Pandya, 1986]:
  - Critical instant: the worst case response time for all tasks is given when all tasks are released at the same time
  - Calculate the worst case response time R for each task with deadline D. If R<=D, the task is schedulable/feasible. Repeat the same check for all tasks
  - If all tasks pass the test, the task set is schedulable
  - If some tasks pass the test, they will meet their deadlines even the other don't (stable and predictable)
- Question:
  - how to calculate the worst case response times?
    - We did this before!

Worst case response time calculation: example

{(1,3),(1,5),(1,6),(2,10)}

Response times?
Worst case? First period?
Why?

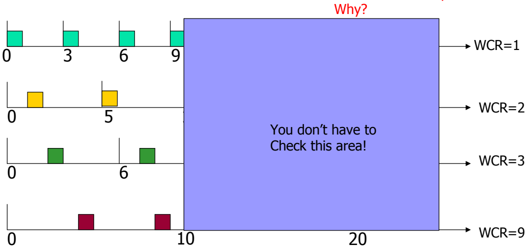


What to do if too many?

Worst case response time calculation: example

{(1,3),(1,5),(1,6),(2,10)}

Response times?
Worst case? First period?
Why?



What to do if too many?

Calculation of worst case response times

[Mathai Joseph and Paritosh Pandya, 1986]

- Let Ri stand for the response time for task i. Then
  $Ri = Ci + \sum_j I(i,j)$
  - Ci is the computing time
  - I(i,j) is the so-called interference of task j to i
  - I(i,j) = 0 if task i has higher priority than j
- $I(i,j) = \lceil Ri/Tj \rceil * Cj$ if task i has lower priority than j
  - $\lceil x \rceil$ denotes the least integer larger than x
  - E.g $\lceil 3.2 \rceil = 4$, $\lceil 3 \rceil = 3$, $\lceil 1.9 \rceil = 2$
- $Ri = Ci + \sum_{j \in HP(i)} \lceil Ri/Tj \rceil * Cj$

Intuition on the equation

$Ri = Ci + \sum_{j \in HP(i)} \lceil Ri/Tj \rceil * Cj$

- $\lceil Ri/Tj \rceil$ is the number of instances of task j during Rj
- $\lceil Ri/Tj \rceil * Cj$ is the time needed to execute all instances of task j released within Rj
- $\sum_{j \in HP(i)} \lceil Ri/Tj \rceil * Cj$ is the time needed to execute instances of tasks with higher priorities than task i, released during Rj
- Rj is the sum of the time required for executing task instances with higher priorities than task j and its own computing time

Equation solving and schedulability analysis

- We need to solve the equation:
  $Ri = Ci + \sum_{j \in HP(i)} \lceil Ri/Tj \rceil * Cj$
- This can be done by numerical methods to compute the fixed point of the equation e.g. By iteration: let
  - $Ri^0 = Ci + \sum_{j \in HP(i)} Cj = C1+C2+\ldots+Ci$ (the first guess)
  - $Ri^{k+1} = Ci + \sum_{j \in HP(i)} \lceil Ri^k/Tj \rceil * Cj$ (the (k+1)th guess)
- The iteration stops when either
  - $Ri^{m+1} > Ti$ or → non schedulable
  - $Ri^m < Ti$ and $Ri^{m+1} = Ri^m$ → schedulable
- This is the so called Precise test

Example (response time calculation)

- Assume a task set: {(1,3),(1,5),(1,6),(2,10)}
- Question: is the task set schedulable?
- Answer: YES
- Because
  - $R1^1 = R1^0 = C1 = 1$ (done)
  - $R2^0 = C2 + C1 = 2$,
    $R2^1 = C2 + \lceil R2^0/T1 \rceil * C1 = 1 + \lceil 2/3 \rceil * 1 = 2$ (done)

Exercise

- Calculate R3 and R4 for the above example
- Construct the run-time RMS schedule and check if your calculation is correct

Example (combine UB test and precise test)

- Consider the task set: {(1,3),(1,5),(1,6),(3,10)}
  - (this is not the previous example!)
- CPU utilization U= 1/3+1/5+1/6+3/10=0.899> B(4)= 0.756
  - Fail the UB test!
- But U(3)= 1/3+1/5+1/6=0.699<B(3)=0.779
  - This means that the first 3 tasks are schedulable
  - Thus we do not need to calculate for R1, R2, R3!
- Question: is task 4 set schedulable?
  - $R4^0 = C1+C2+C3+C4 = 6$
  - $R4^1 = C4+\lceil R4^0/T1 \rceil * C1+\lceil R4^0/T2 \rceil * C2+\lceil R4^0/T3 \rceil * C3$
    $= 3 + \lceil 6/3 \rceil *1+\lceil 6/5 \rceil *1+\lceil 6/6 \rceil *1=8$
  - $R4^2 = C4+\lceil R4^1/T1 \rceil * C1+\lceil R4^1/T2 \rceil * C2+\lceil R4^1/T3 \rceil * C3$
    $= 3 + \lceil 8/3 \rceil *1+\lceil 8/5 \rceil *1+\lceil 8/6 \rceil *1$
    $= 3+4+2+2$
    $= 11$ (task 4 is non schedulable!)

Combine UB and Precise tests

- Order tasks according to their priorities (periods)
- Use UB test as far as you can until you find the first non-schedulable task
- Calculate response time for the task and all tasks with lower priority

Example

	C	T	C/T
Task 1	40	100	0.400
Task 2	40	150	0.267
Task 3	100	350	0.286

Total utilization: U=0.4+0.267+0.286= 0.953>B(3)=0.779!
UB test is inclusive: we need Precise test
but we do have U(T1)+U(T2)= 0.4+0.267= 0.667<U(2)=0.828
so we need to calculate R3 only!

Calculate response time for task 3

- $R3^0 = C1+C2+C3= 180$
- $R3^1 = C3+\lceil R3^0/T1 \rceil * C1+\lceil R3^0/T2 \rceil * C2$
  $=100+\lceil 180/100 \rceil *40+\lceil 180/150 \rceil *40$
  $=100+2*40+2*40=260$
- $R3^2 = C3+\lceil R3^1/T1 \rceil * C1+\lceil R3^1/T2 \rceil * C2$
  $=100+\lceil 260/100 \rceil *40+\lceil 260/150 \rceil *40=300$
- $R3^3 = C3+\lceil R3^2/T1 \rceil * C1+\lceil R3^2/T2 \rceil * C2$
  $=100+\lceil 300/100 \rceil *40+\lceil 300/150 \rceil *40=300$ (done)

Task 3 is schedulable and so are the others!

Question: other priority-assignments

- Could we calculate the response times by the same equation for different priority assignment?

Precedence constraints

How to handle precedence constraints?

- We can always try the 'old' method: static cyclic scheduling!
- Alternatively, take the precedence constraints (DAG) into account in priority assignment: the priority-ordering must satisfy the precedence constraints
  - Precise schedulability test is valid: use the same method as beforee to calculate the response times.

Summary: Three ways to check schedulability

1. UB test
2. Response time calculation
3. Construct a schedule for the first periods
   - assume the first instances arrive at time 0 (critical instant)
   - draw the schedule for the first periods
   - if all tasks are finished before the end of the first periods, schedulable, otherwise NO

Summary: UB and precise test

- UB test is simple but conservative
- Response time test is precise
- Both share the same limitations:
  - D=T (deadline = period)
  - Independent tasks
  - No interrupts
  - Zero context switch: OH=0
  - No synchronization between tasks!

Extensions to the basic RMS

- Deadline <= Period
- Interrupt handling
- Non zero OH for context switch
- Non preemptive sections
- Sharing resources

RMS for tasks with D <= T

- RMS is no longer optimal (example?)
- Utilization bound test must be modified
- Response time test is still applicable
  - Assuming that fixed-priority assignment is adopted
  - But considering the critical instant and checking the first deadlines principle are still applicable

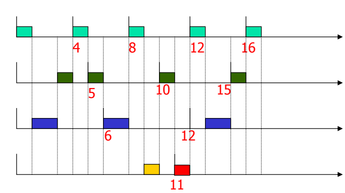
Deadline Monotonic Scheduling (DMS) [Leung et al, 1982]

- Task model: the same as for RMS but Di<=Ti
- Priority-Assignment: tasks with shorter deadline are assigned higher priorities
- Run-time scheduling: preemptive HPF

- FACTS:
  - DMS is optimal
  - RMS is a special case of DMS
- DMS is often refered as Rate Monotonic Scheduling for historical reasons and they are so similar

Example

	C	T	D
Task 1	1	4	3
Task 2	1	5	5
Task 3	2	6	4
Task 4	1	11	10



R1=1
R2=4
R3=3
R4=10

DMS: Schedulability analysis

- UB test (sufficient):

  Σ Ci/Di <= n*($2^{1/n}$-1) implies schedulable by DMS
- Prescise test (exactly the same as for RMS):

  Response time calculation: Ri= Ci + $\Sigma_{j \in HP(i)} \lceil Ri/Tj \rceil$*Cj
  - Ri^0 = Ci + $\Sigma_{j \in HP(i)}$ Cj = C1+C2+...+Ci (the first guess)
  - Ri^{k+1} = Ci + $\Sigma_{j \in HP(i)} \lceil Ri^k/Tj \rceil$*Cj (the (k+1)th guess)
  - The iteration stops when either
    - Ri^{m+1}>Di or → non schedulable
    - Ri^m<Di and $Ri^{m+1} = Ri^m$ → schedulable

DMS: Schedulability analysis

- Assume that all tasks (or their first instances) arrive at time 0: critical instant
- Construct a schedule for the first periods: draw the diagram!
- If all tasks meet their deadlines within the first periods, schedulable, otherwise not!

Summary: 3 ways for DMS schedulability check

- UB test (sufficient, inconclusive)
- Response time calculation
- Draw the schedule for the first periods

EDF for tasks with D <= T

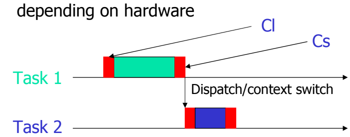
- You can always use EDF and it is always optimal to schedule tasks with deadlines
  - We have a precise UB test for EDF for tasks with Di=Ti: U<=1 iff task set is schedulable
  - Unfortunately, for tasks with Di<=Ti, schedulability analysis is more complicated (out of scope of the course, further reading [Giorgio Buttazzo's book])
    - We can always check the whole LCM

Summary: schedulability analysis

	Di=Ti	Di<=Ti
Static/Fixed-priority	RMS Sufficient test Σ Ci/Ti <= n*($2^{1/n}$-1) Precise test Ri= Ci + $\Sigma_{j \in HP(i)} \lceil Ri/Tj \rceil$*Cj Ri<=Ti	DMS Sufficient test Σ Ci/Di <= n*($2^{1/n}$-1) Precise test Ri= Ci + $\Sigma_{j \in HP(i)} \lceil Ri/Tj \rceil$*Cj Ri<=Di
Dynamic priority	EDF Precise test Σ Ci/Ti <=1	EDF ?

Handling context switch overhands in schedulability analysis

- Assume that
  - Cl is the extra time required to load the context for a new task (load contents of registers etc from TCB)
  - Cs is the extra time required to save the context for a current task (save contents of registers etc to TCB)
  - Note that in most cases, Cl=Cs, which is a parameter depending on hardware



Handling context switch overheads ?

- Thus, the real computing time for a task should be $Ci^* = Ci + Cl + Cs$
- The schedulability analysis techniques we studied so far are applicable if we use the new computing time C^*.
  - Unfortunately this is not right

Handling context switch

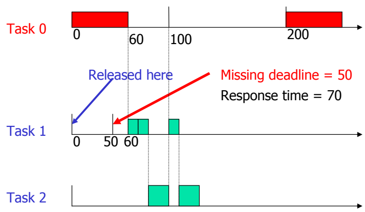
- $Ri = Ci + 2Ccs + \sum_{j \in HP(i)} \lceil Ri/Tj \rceil * (Cj + 2Ccs)$
  - This is wrong!
- $Ri = Ci + 2Ccs + \sum_{j \in HP(i)} \lceil Ri/Tj \rceil * Cj + \sum_{j \in HP(i)} \lceil Ri/Tj \rceil * 4Ccs$

  $= Ci + 2Ccs + \sum_{j \in HP(i)} \lceil Ri/Tj \rceil * (Cj + 4Ccs)$
  - This is right

Handling interrupts: problem and example

Task 0 is the interrupt handler with highest priority

	C	T=D
IH, task 0	60	200
Task 1	10	50
Task 2	40	250



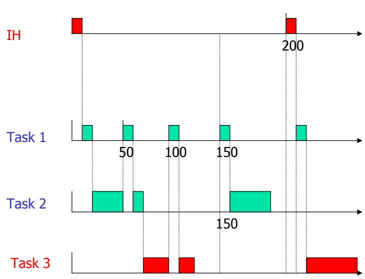
Handling interrupts: solution

- Whenever possible: move code from the interrupt handler to a special application task with the same rate as the interrupt handler to make the interrupt handler (with high priority) as shorter as possible
- Interrupt processing can be inconsistent with RM priority assignment, and therefore can effect schedulability of task set (previous example)
  - Interrupt handler runs with high priority despites its period
  - Interrupt processing may delay tasks with shorter periods (deadlines)
  - how to calculate the worst case response time ?

Handling interrupts: example

Task 0 is the interrupt handler with highest priority

	C	T=D
IH	10	200
Task 1	10	50
Task 2	40	150
Task 3	50	200



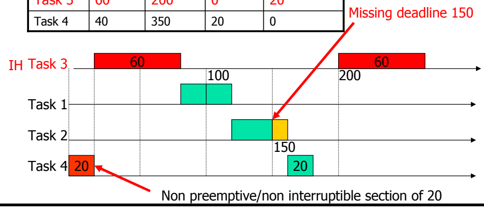
Handling non-preemtive sections

- So far, we have assumed that all tasks are preemptive regions of code. This not always the case e.g code for context switch though it may be short, and the short part of the interrupt handler as we considered before
  - Some section of a task is non preemptive
- In general, we may assume an extra parameter B in the task model, which is the computing time for the non preemtive section of a task.
  - Bi = computing time of non preemptive section of task i

Handling non preemptive sections: Problem and Example

Task 3 is an interrupt handler with highest priority
Task 4 has a non preemptive section of 20 sec

	C	T=D	blocking	blocked
Task 1	20	100	0	20
Task 2	40	150	0	20
Task 3	60	200	0	20
Task 4	40	350	20	0



Handling non-preemtive sections: Response time calculation

- The equation for response time calculation:

  $R_i = B_i + C_i + \sum_{j \in HP(i)} \lceil R_i/T_j \rceil * C_j$

- Where B_i is the longest time that task i can be blocked by lower-priority tasks with non preemptive section
  - $B_i = \max\{C_{bj} \mid \text{task } j \text{ has lower priority than task } i\}$
  - Note that a task preempts only one task with lower priority within each period

So now, we have an equation:

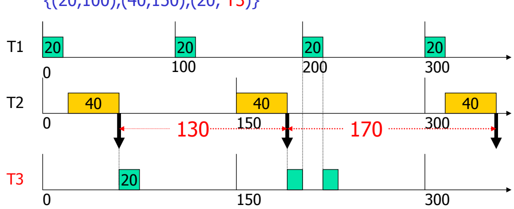
$$R_i = B_i + C_i + 2C_{cs} + \sum_{j \in HP(i)} \lceil R_i/T_j \rceil * (C_j + 4*C_{cs})$$

The Jitter Problem

- So far, we have assumed that tasks are released at a constant rate (at the start of a constant period)
- This is true in practice and a realistic assumption
- However, there are situations where the period or rather the release time may 'jitter' or change a little, but the jitter is bounded with some constant J
- The jitter may cause some task missing deadline

Jitter: Example

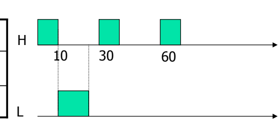
{(20,100),(40,150),(20, T3)}



T3 is activated by T2 when it finishes within each period
Note that because the response time for T2 is not a constant,
the period between two instances of T3 is not a constant: 170, 130

Effect of Jitters

	C	D	T	R
H	10	20	30	10
L	15	25	1000	25



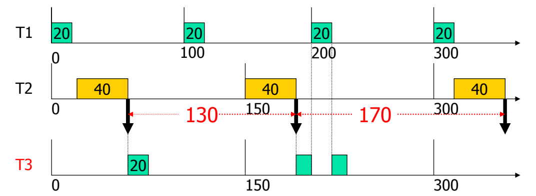
Jitter: Definition

- J(biggest)=maximal delay from period-start
- J(smallest)=minimal delay from period-start
- Jitter= J(biggest)-J(smallest)

- If J(biggest)=J(smallest), then no influence on the other tasks with lower priorities

Jitter: Example

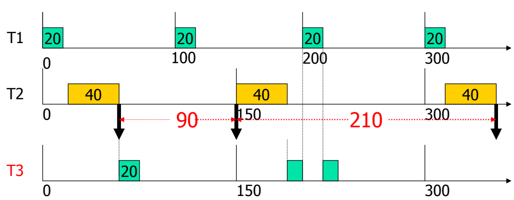
{(20,100),(40,150),(20, T3)} Pr(T1)=1, Pr(T2)=2, Pr(T3)=3



T3 is activated by T2 by the end of each instance
J(biggest)= R2(worst case), J(smallest)= R2(best case)
Jitter = J(biggest)- J(smallest)=60-40=20

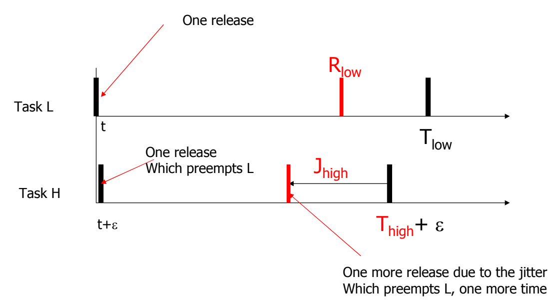
Jitter: Example

{(20,100),(40,150),(20, T3)}

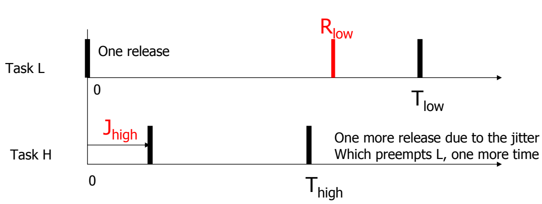


T3 is activated by T2 at any time during its execution of an instance
J(biggest)= R2(worst case), J(smallest)= R2(best case)-C2
Jitter = J(biggest)- J(smallest)=60-0=60

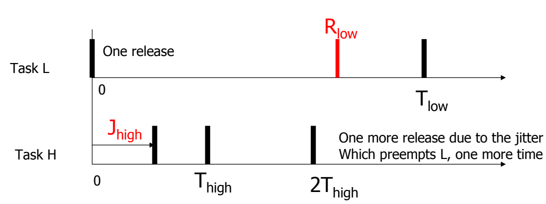
The number of preemptions due to Jitter



Task L will be preempted at least 2 times if $R_{low} > T_{high} - J_{high}$



Task L will be preempted at least 3 times if $R_{low} > 2T_{high} - J_{high}$



The number of preemptions/blocking when jitters occur

- Task L will be preempted at least 2 times if $R_{low} > T_{high} - J_{high}$
- Task L will be preempted at least 3 times if $R_{low} > 2 * T_{high} - J_{high}$
- ...
- Task L will be preempted at least n times if

  $R_{low} > (n-1) * T_{high} - J_{high}$
- Thus $(R_{low} + J_{high})/Tj > n-1$
- the largest n satisfying the condition is given by

  $n = \lceil (R_{low} + J_{high}) / T_{high} \rceil$

Handling Jitters in schedulability analysis

- $Ri = Ci + \sum_{j \in HP(i)}$ "number of preemptions" $*Cj$
  - $Ri^* = Ri + Ji(biggest)$
- if $Ri^* < Di$, task i is schedulable otherwise no

Handling Jitters in schedulability analysis

- $R_i = C_i + \sum_{j \in HP(i)} \lceil (R_i + J_j)/T_j \rceil * C_j$
  - $R_i^* = R_i + J_i(biggest)$

  why $R_i + J_i(biggest)$?
- if $R_i^* < D_i$, task i is schedulable, otherwise no

Finally, we have an equation (why?):

$R_i = C_i + 2Ccs + B_i + \sum_{j \in HP(i)} \lceil (R_i + J_j)/T_j \rceil * (C_j + 4Ccs)$

The response time for task i

$R_i^* = R_i + J_i(biggest)$

$J_i(biggest)$ is the "biggest jitter" for task i

Summary: + and -

- Static Cyclic Scheduling (SCS)
  - Simple, and reliable, may be difficult to construct the time table and difficult to modify and (inflexible)
- Earliest Deadline First (EDF)
  - Simple in theory, but difficult to implement, non-stable
  - no precise analysis for tasks D<T
- Rate Monotonic Scheduling (RMS)
  - Simple in theory and practice, and easy to implement
- Deadline Monotonic Scheduling (DMS)
  - Similar to RMS
- Handling overheads, blocking