

---

## SCHEDULING Periodic/Sporadic TASKS

1

### Periodic tasks

---

- A task =  $(C, T)$ 
  - $C$ : worst case execution time/computing time ( $C \leq T$ !)
  - $T$ : period i.e. the distance between two instances
  - Deadline is the period
- A task set:  $(C_i, T_i)$ 
  - All tasks are independent
  - $U = \sum C_i/T_i$  is the CPU utilization (the workload)

2

## 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
- Synchronous programming languages: Esterel, Lustre, Signal

3

## 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 =  $7 \cdot 13 \cdot 23 = 2093$  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

4

## 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

5

## Task model (periodic or sporadic)

---

- A task =  $(C, T)$ 
  - $C$ : worst case execution time/computing time ( $C \leq T$ )
  - $T$ : period or the least distance between two instances
  - Deadline is the period
- A task set:  $(C_i, T_i)$ 
  - All tasks are independent
  - $U = \sum C_i / T_i$  is the CPU utilization (the workload)

6

## Earliest Deadline First (EDF)

---

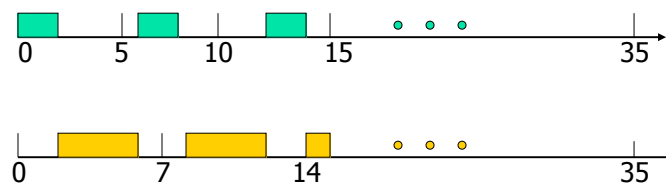
- Task model
  - a set of independent periodic tasks
- EDF:
  - Whenever a new task arrive, sort the ready queue so that the task closest to the end of its period assigned the highest priority
  - Preempt the running task if it is not placed in the first of the queue in the last sorting
- **FACT 1:** EDF is optimal
  - EDF can schedule the task set if any one else can
- **FACT 2** (Schedulability test):
  - $\sum C_i/T_i \leq 1$  iff the task set is schedulable

7

## Example

---

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



8

## 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:  $U \leq 1$  (+)
  - Optimal (+)
  - Best CPU utilization (+)
- **Difficult to implement** in practice. It is not very often adopted due to the dynamic priority-assignment. 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 →

9

## 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

10

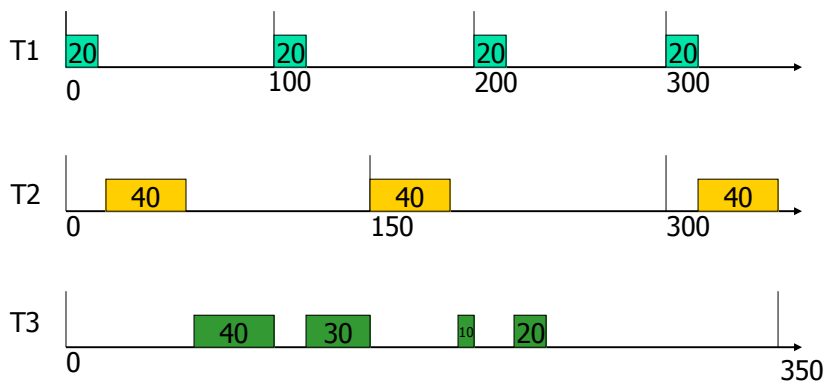
## A classic paper on real-time systems

- C. L. Liu , James W. Layland, Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the ACM (JACM)*, v.20 n.1, p.46-61, Jan. 1973
  - Rate monotonic scheduling

11

### Example

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



12

## Example

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



13

## RMS: schedulability test

- $U < 1$  doesn't imply 'schedulable' with RMS
  - OBS: the previous example is schedulable by EDF, not RMS
- **Idea:** utilization bound
  - Given a task set  $S$ , find a bound on the workload  $U$  (i.e. Utilization) such that  $U \leq B$  if and only if  $S$  is schedulable by RMS (necessary and sufficient test)
  - Note that the bound  $B$  for EDF is 1

14

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

---

- Assume a set of  $n$  independent tasks:
  - $S = \{(C_1, T_1)(C_2, T_2) \dots (C_n, T_n)\}$  and  $U = \sum C_i/T_i$
- **FACT:** if  $U \leq B(n)$ , then  $S$  is schedulable by RMS
  - Where  $B(n) = n \cdot (2^{1/n} - 1)$
- Note that  $B(n)$  depends on the number  $n$  of tasks in the set

15

### Example: Utilization bounds

---

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

Note that  $U(\infty)=0.693$  !

16



## Example: applying UB Test

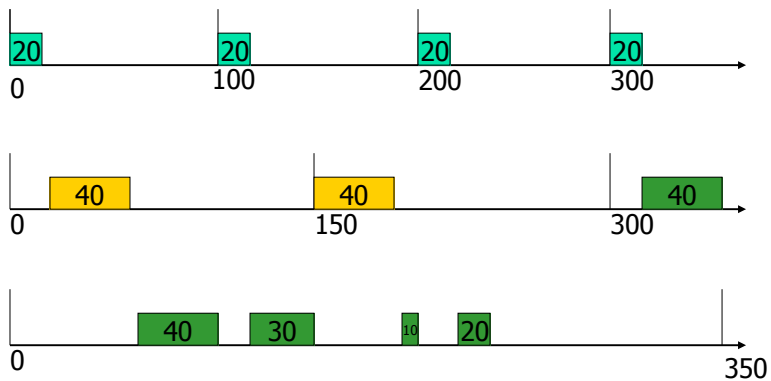
	C	T (D=T)	C/T
Task 1	20	100	0.200
Task 2	40	150	0.267
Task 3	100	350	0.286

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

17

## Example: RM Scheduling

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



18

## UB test is only sufficient, not necessary!

---

- Let  $U = \sum C_i/T_i$  and  $B(n) = n \cdot (2^{1/n} - 1)$
- Three possible outcomes:
  - $0 \leq U \leq B(n)$ : schedulable
  - $B(n) < U \leq 1$ : no conclusion
  - $1 < U$ : overload
- Thus, the test may be too conservative
- (exact test will be given later)

19

## 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  $B(4) = 0.756$ , OBS:  $U > B(4)$
- But the task set is schedulable anyway!

20

## 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?

21

## RMS: Summary

---

- Task model:
  - Sporadic/periodic, independent,  $D=T$ , and a task =  $(C_i, T_i)$
- Fixed-priority assignment:
  - smaller periods = higher priorities
- Run time scheduling: Preemptive HPF
- Sufficient schedulability test:  $U \leq n \cdot (2^{1/n} - 1)$
- Precise/exact schedulability test exists

22

## 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  $D=T$  (-), can be solved
- How about dependent tasks? It works too
- Only sufficient test (-), precise test exists
- 

23

## Critical instant: an important observation

---

- Critical instant of a task is the time point 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 at zero, or at the same time as the other tasks.

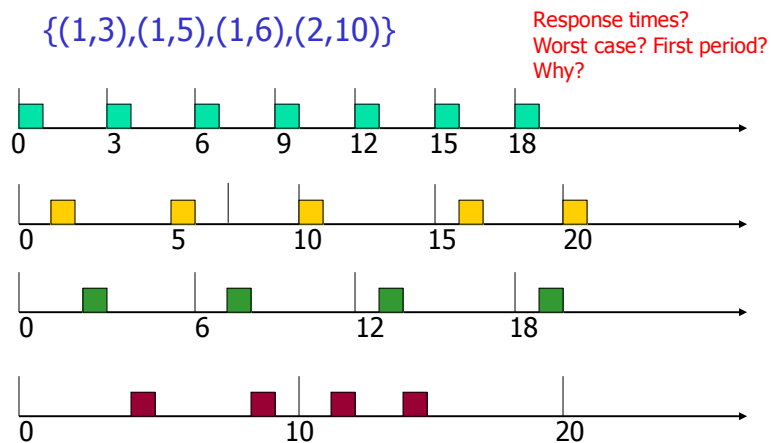
24

## 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  $R \leq D$ , 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!

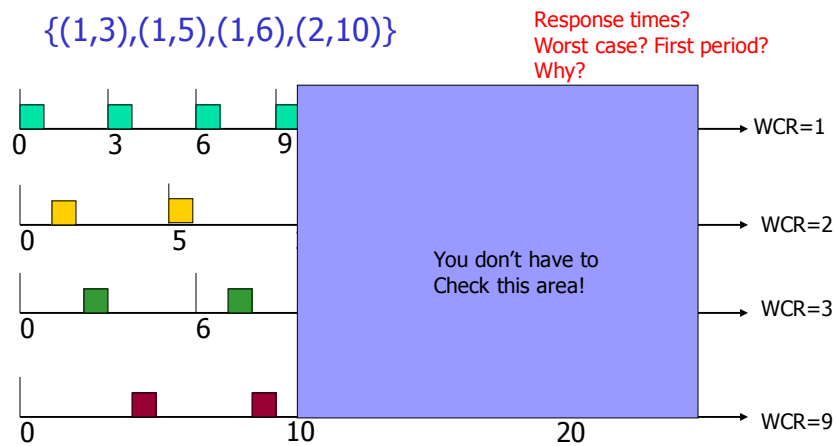
25

## Worst case response time calculation: example



26

## Worst case response time calculation: example



What to do if too many?

27

## Calculation of worst case response times

[Mathai Joseph and Paritosh Pandya, 1986]

- Let  $R_i$  stand for the response time for task  $i$ . Then
  - $R_i = C_i + \sum_j I(i,j)$ 
    - $C_i$  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  $j$
  - $I(i,j) = \lceil R_i/T_j \rceil * C_j$  if task  $i$  has lower priority than  $j$ 
    - $\lceil x \rceil$  denotes the least integer larger than  $x$
    - E.g  $\lceil 3.2 \rceil = 4, \lceil 3 \rceil = 3, \lceil 1.9 \rceil = 2$
- $R_i = C_i + \sum_{j \in HP(i)} \lceil R_i/T_j \rceil * C_j$

28

## Intuition on the equation

---

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

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

29

## Equation solving and schedulability analysis

---

- We need to solve the equation:

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

- This can be done by numerical methods to compute the fixed point of the equation e.g. By iteration: let
  - $R_i^0 = C_i + \sum_{j \in \text{HP}(i)} C_j = C_1 + C_2 + \dots + C_i$  (the first guess)
  - $R_i^{k+1} = C_i + \sum_{j \in \text{HP}(i)} \lceil R_i^k/T_j \rceil * C_j$  (the  $(k+1)$ th guess)
- The iteration stops when either
  - $R_i^{m+1} > T_i$  or  $\rightarrow$  non schedulable
  - $R_i^m < T_i$  and  $R_i^{m+1} = R_i^m \rightarrow$  schedulable
- This is the so called **Precise test**

30

## Example

---

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

31

## Example (combine UB test and precise test)

---

- Consider the same task set:  $\{(1,3),(1,5),(1,6),(3,10)\}$
- 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
- **Question:** is task 4 set schedulable?
  - $R4^0 = C1+C2+C3+C4= 6$
  - $R4^1 = C4+\lceil R4^0/T1 \rceil * C1+\lceil R4^0/T2 \rceil * C2+\lceil R4^0/T3 \rceil * C3$   
 $= 3 + \lceil 6/3 \rceil * 1 + \lceil 6/5 \rceil * 1 + \lceil 6/6 \rceil * 1 = 8$
  - $R4^2 = C4+\lceil R4^1/T1 \rceil * C1+\lceil R4^1/T2 \rceil * C2+\lceil R4^1/T3 \rceil * C3$   
 $= 3 + \lceil 8/3 \rceil * 1 + \lceil 8/5 \rceil * 1 + \lceil 8/6 \rceil * 1$   
 $= 3+3+2+2$   
 $= 10$
  - $R4^3 = C4+\lceil R4^2/T1 \rceil * C1+\lceil R4^2/T2 \rceil * C2+\lceil R4^2/T3 \rceil * C3$   
 $= 3+ 4 + 2 + 2 = 11$  (task 4 is non schedulable!)

32



## 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!

33

## 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!

34

## Extensions to the basic RMS

---

- Deadline  $\leq$  Period
- Non-zero Overheads for context switch
- Non-preemptive sections
- Resource Sharing (priority ceiling protocols)

35

## Tasks with $D \leq T$

---

- 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

36

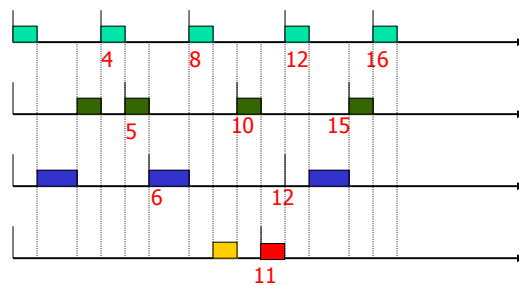
## Deadline Monotonic Scheduling (DMS/RMS) [Leung et al, 1982]

- **Task model:** the same as for RMS but  $D_i \leq T_i$
- **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 referred as Rate Monotonic Scheduling for historical reasons and they are so similar

37

### 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



38

## DMS: Schedulability analysis

---

- UB test (sufficient):

$\sum C_i/D_i \leq n \cdot (2^{1/n} - 1)$  implies schedulable by DMS

- Precise test (exactly the same as for RMS):

Response time calculation:  $R_i = C_i + \sum_{j \in \text{HP}(i)} \lceil R_i/T_j \rceil \cdot C_j$

- $R_i^0 = C_i + \sum_{j \in \text{HP}(i)} C_j = C_1 + C_2 + \dots + C_i \rightarrow$  the first guess

- $R_i^{k+1} = C_i + \sum_{j \in \text{HP}(i)} \lceil R_i^k/T_j \rceil \cdot C_j \rightarrow$  the (k+1)th guess

- The iteration stops when either

- $R_i^{m+1} > D_i$  or  $\rightarrow$  non schedulable

- $R_i^m < D_i$  and  $R_i^{m+1} = R_i^m \rightarrow$  schedulable

39

## Question: other priority-assignments

---

Could we calculate the response times for different priority assignment?

YES!

- Example: tasks with the same period

- assign different priorities

- calculate the response times as usual

40

## Precedence constraints

---

- Tasks released within the same period
  - Easy: priority assignment should take the precedence constraints (DAG) into account, **the priority-ordering satisfying the precedence constraints**
  - Response time calculation is the same as before
  - (how about "sufficient test"?)
  
- Tasks released with different periods?
  - More complex semantics

41

## RMS Summary: Two ways to check schedulability

---

1. UB test (simple but conservative)
2. Response time calculation (precise test)
  - For small task sets: 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

42

More ... on periodic tasks, how about other cases?

EDF: it is always optimal to schedule tasks with deadlines

- o  $D_i = T_i$ :  $U \leq 1$  iff task set is schedulable
- o  $D_i < T_i$  and  $D_i > T_i$ : schedulability analysis for EDF is more complicated (out of scope of the course, further reading [Giorgio Buttazzo's book])

RMS: we know what to do with  $D_i \leq T_i$

- o  $D_i > T_i$  ([Baruah-et al, IEEE TC-2009])

43

## Summary: schedulability analysis

	$D_i = T_i$	$D_i \leq T_i$
Static/Fixed-priority	<p>RMS</p> <p><b>Sufficient test</b></p> $\sum C_i/T_i \leq n \cdot (2^{1/n} - 1)$ <p><b>Precise test</b></p> $R_i = C_i + \sum_{j \in HP(i)} \lceil R_i/T_j \rceil \cdot C_j$ <p><math>R_i \leq T_i</math></p>	<p>DMS</p> <p><b>Sufficient test</b></p> $\sum C_i/D_i \leq n \cdot (2^{1/n} - 1)$ <p><b>Precise test</b></p> $R_i = C_i + \sum_{j \in HP(i)} \lceil R_i/T_j \rceil \cdot C_j$ <p><math>R_i \leq D_i</math></p>
Dynamic priority	<p>EDF</p> <p><b>Precise test</b></p> $\sum C_i/T_i \leq 1$	<p>EDF</p> <p>[Giorgio Buttazzo's book]</p>

44

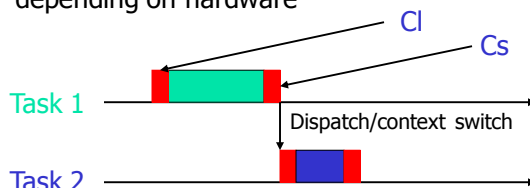
## Summary: schedulability analysis

	$D_i = T_i$	$D_i \leq T_i$	$D_i > T_i$
Static/Fixed-priority	RMS <b>Sufficient test</b> $\sum C_i/T_i \leq n \cdot (2^{1/n} - 1)$ <b>Precise test</b> $R_i = C_i + \sum_{j \in HP(i)} \lceil R_i/T_j \rceil \cdot C_j$ $R_i \leq T_i$	DMS <b>Sufficient test</b> $\sum C_i/D_i \leq n \cdot (2^{1/n} - 1)$ <b>Precise test</b> $R_i = C_i + \sum_{j \in HP(i)} \lceil R_i/T_j \rceil \cdot C_j$ $R_i \leq D_i$	[Baruah-et al, IEEE TC-2009]
Dynamic priority	EDF <b>Precise test</b> $\sum C_i/T_i \leq 1$	EDF [Giorgio Buttazzo's book]	$\sum C_i/T_i \leq 1$

45

## Handling context switch overheads in schedulability analysis

- Assume that
  - $C_l$  is the extra time required to load the context for a new task (load contents of registers etc from TCB)
  - $C_s$  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,  $C_l = C_s$ , which is a parameter depending on hardware



46

## Handling context switch overheads ?

---

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

47

## Handling context switch

---

- $R_i = C_i' + \sum_{j \in HP(i)} \lceil R_i/T_j \rceil * C_j'$   
 $= C_i + 2C_s + \sum_{j \in HP(i)} \lceil R_i/T_j \rceil * (C_j + 2C_s)$ 
  - This is wrong!
- $R_i = C_i + 2C_s + \sum_{j \in HP(i)} \lceil R_i/T_j \rceil * C_j$   
 $+ \sum_{j \in HP(i)} \lceil R_i/T_j \rceil * 4C_s$   
(each preemption  $\rightarrow$  2 context switches)  
 $= C_i + 2C_s + \sum_{j \in HP(i)} \lceil R_i/T_j \rceil * (C_j + 4C_s)$ 
  - This is right

48



## Handling non-preemptive sections

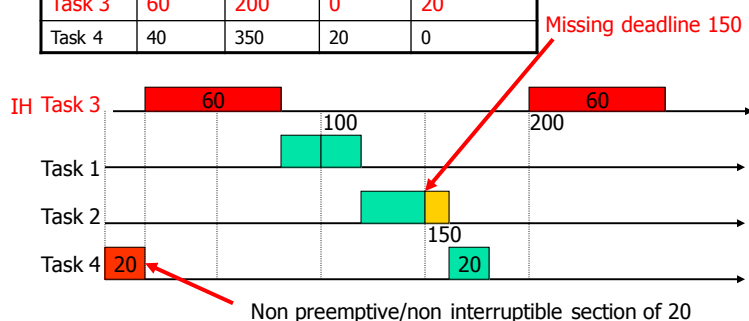
- So far, we have assumed that all tasks are preemptive regions of code. This not always the case e.g code for context switch though it may be short, and the short part of the interrupt handler as we considered before
  - Some section of a task is non preemptive
- In general, we may assume an extra parameter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

49

## 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



50

## Handling non-preemptive 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
  - Note that a task preempts only one task with lower priority within each period

51

So now, we have an equation:

---

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

52

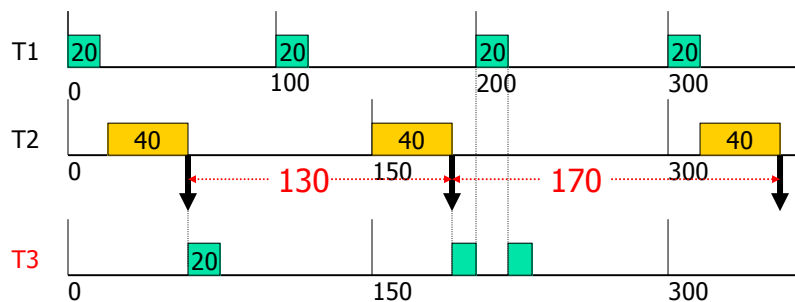
## The Jitter Problem (only for periodic tasks)

- Periodic tasks are released (or invoked) at a constant rate periodically
- There are situations where a task is released but can not be executed immediately because it is waiting for input data or other resources
- Such variations or jitters of tasks may cause tasks with lower priority to miss their deadlines.

53

### 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

54

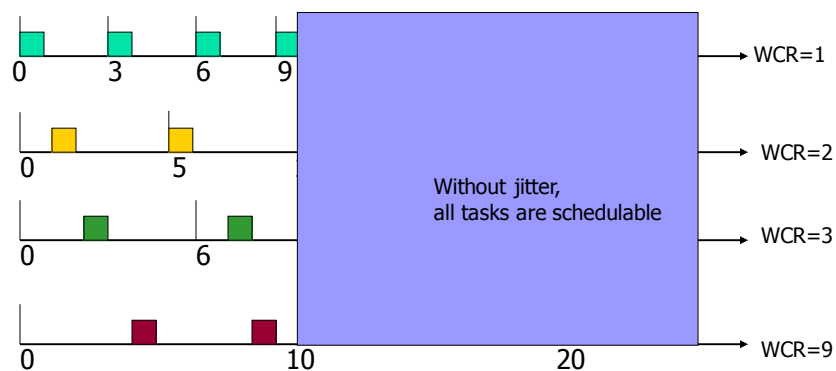
## Jitter for periodic tasks: Definition

- $J(\text{biggest}) = \text{maximal delay from period-start}$
- $J(\text{smallest}) = \text{minimal delay from period-start}$
- $\text{Jitter} = J(\text{biggest}) - J(\text{smallest})$
- Jitter = the maximal length of the interval in which a task may be released **non-deterministically**
- **If  $J(\text{biggest}) = J(\text{smallest})$ , then NO JITTER** and therefore no influence on the other tasks with lower priorities

55

## Problem with jitter: example

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

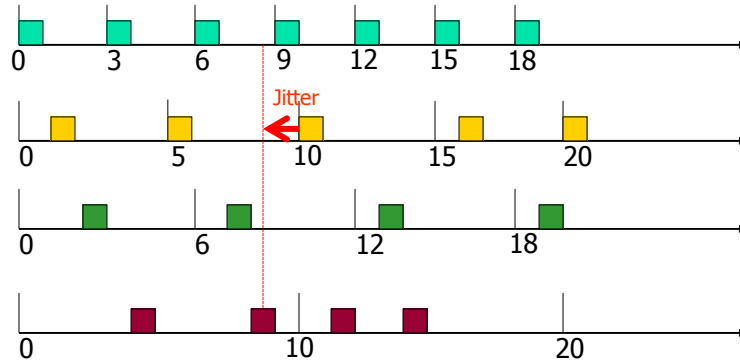


What to do if too many?

56

Assume the yellow task has a jitter of 1.5

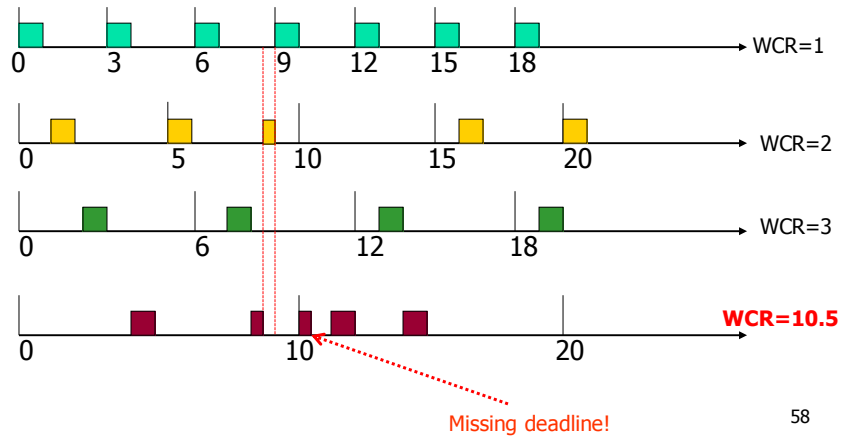
$\{(1,3),(1,5),(1,6),(2,10)\}$



57

The jitter will cause a deadline miss for the last task

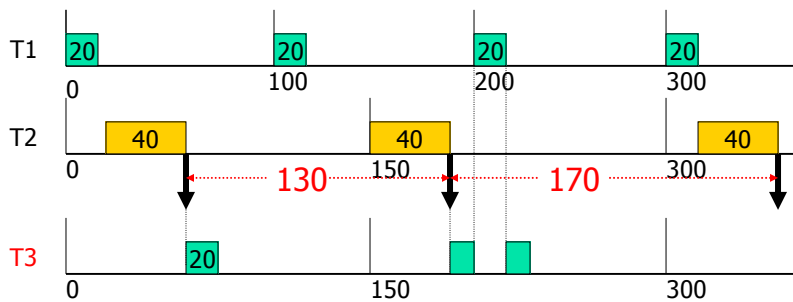
$\{(1,3),(1,5),(1,6),(2,10)\}$



58

## Jitter: Example

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

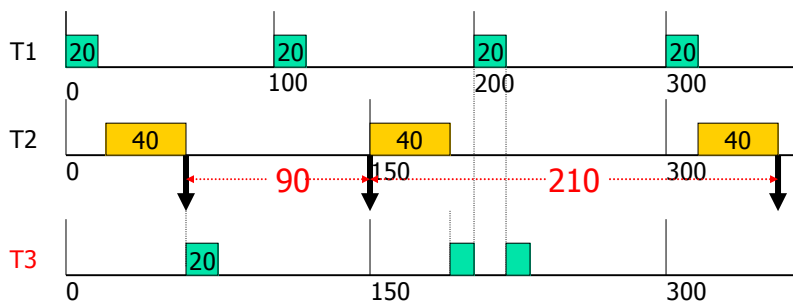


T3 is activated by T2 by the end of each instance  
 $J(\text{biggest}) = R2(\text{worst case}), J(\text{smallest}) = R2(\text{best case})$   
 $\text{Jitter} = J(\text{biggest}) - J(\text{smallest}) = 60 - 40 = 20$

59

## Jitter: Example

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



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

60

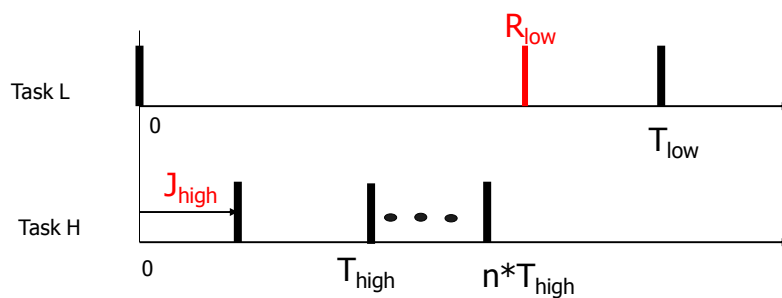
## Handling Jitters in schedulability analysis

- $R_i = C_i + \sum_{j \in HP(i)} \text{"number of preemptions"} * C_j$ 
  - $R_i^* = R_i + J_i(\text{biggest})$  is the worst case response time of task  $i$
  - $J_i(\text{biggest})$  is the worst case jitter
  
- if  $R_i^* < D_i$ , task  $i$  is schedulable otherwise no

61

## Response time analysis for lower-priority tasks tasks that suffer from jitters by high-priority tasks

Task L will be preempted at most  $\lceil (R_{low} + J_{high}) / T_{high} \rceil$  times



62

## Handling Jitters in schedulability analysis

---

- $R_i = C_i + \sum_{j \in \text{HP}(i)} \lceil (R_i + J_j) / T_j \rceil * C_j$ 
  - $R_i^* = R_i + J_i(\text{biggest})$
- if  $R_i^* < D_i$ , task  $i$  is schedulable, otherwise no

63

Now, we have an equation:

---

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

The response time for task  $i$

$$R_i^* = R_i + J_i(\text{biggest})$$

$J_i(\text{biggest})$  is the "biggest jitter" for task  $i$

64



## Resource Sharing with HLP and PCP (and BIP)

---

- Let
  - $CS(k,S)$  denote the computing time for the critical section that task  $k$  locks  $S$ .
  - $Use(S)$  is the set of tasks locking  $S$
- For HLP and PCP, the maximal blocking
  - $RS_i = \max\{CS(k,S) \mid i,k \text{ in } Use(S), pr(k) < pr(i) \text{ for all } k, S\}$
- How about BIP?
  - $RS_i = \text{Sum}\{CS(k,S) \mid i,k \text{ in } Use(S), pr(k) < pr(i) \text{ for all } k, S\}$
- The response time for task  $i$ 
  - $R_i = RS_i + C_i + \sum_{j \in HP(i)} \lceil R_i/T_j \rceil * C_j$

65

Finally, we have an equation (why?):

---

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

66

## 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 for over-loaded systems
  - $D_i < T_i$  or  $D_i > T_i$ 
    - Precise test exits (later, Martin's lecture)
- **Rate Monotonic Scheduling (RMS)**
  - Simple in theory and practice, and easy to implement
- **Deadline Monotonic Scheduling (DMS)**
  - Similar to RMS
- **Handling overheads, blocking, resource sharing (priority ceiling protocols)**

67