SCHEDULING PERIODIC TASKS

Periodic tasks

- Arrival time
- C: computing time
- F: finishing/response time
- R: release time
- D: deadline
- T: period

Periodic tasks (the simplified case)

- Arrival time
- Scheduled to run
- Finishing/response time
- Time
- R: release time
- D: deadline

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 execution time: C
  - The same relative deadline: D=T (not a restriction)
  - The same relative arrival time: A=0 (not a restriction)
- 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)
- A task set: (Ci, Ti)
  - All tasks are independent
  - The periods of tasks start at 0 simultaneously

CPU utilization

- C/T is the CPU utilization of a task
- U=Σ C/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 Algorithms

- 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
- Synchronous programming languages: Esterel, Lustre, Signal

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 measurement: $C=4\text{ms}$, $T=20\text{ms}$, $D=20\text{ms}$
- ABS control: $C=10\text{ms}$, $T=40\text{ms}$, $D=40\text{ms}$
- Fuel injection: $C=40\text{ms}$, $T=80\text{ms}$, $D=80\text{ms}$
- Other software with soft deadlines e.g. audio, air condition etc

The car controller: static cyclic scheduling

- The shortest repeating cycle = 80ms
- All task instances within the cycle:

<table>
<thead>
<tr>
<th>0</th>
<th>20</th>
<th>40</th>
<th>60</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>ABS</td>
<td>Speed</td>
<td>ABS</td>
<td>Speed</td>
</tr>
<tr>
<td>64</td>
<td>40</td>
<td>20</td>
<td>34</td>
<td>14</td>
</tr>
</tbody>
</table>

Try any method to schedule the tasks

The car controller: 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 = 2093ms 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 arrives, 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):
  - $\sum \frac{C_i}{T_i} \leq 1$ iff the task set is schedulable

Example

- Task set: $(2,5), (4,7)$
- $U = \frac{2}{5} + \frac{4}{7} = \frac{34}{35} \approx 0.97$ (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: $U \leq 1$ (+)
  - 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: $(C_i, T_i)$
- $D_i = T_i$
- 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)\}

\[ Pr(T1)=1, \quad Pr(T2)=2, \quad Pr(T3)=3 \]

Example

- Task set: \( T1=(2,5), \quad T2=(4,7) \)
- \( U = \frac{2}{5} + \frac{4}{7} = \frac{34}{35} \approx 0.97 \) (schedulable?)
- RMS priority assignment: \( Pr(T1)=1, \quad Pr(T2)=2 \)

Example: Utilization bounds

<table>
<thead>
<tr>
<th>( B(1) )</th>
<th>( B(4) )</th>
<th>( B(7) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>0.756</td>
<td>0.728</td>
</tr>
<tr>
<td>0.828</td>
<td>0.743</td>
<td>0.724</td>
</tr>
<tr>
<td>0.779</td>
<td>0.734</td>
<td>( U(\infty) = 0.693 )</td>
</tr>
</tbody>
</table>

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

Example: applying UB Test

<table>
<thead>
<tr>
<th>Task</th>
<th>C</th>
<th>( D=T )</th>
<th>C/T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>20</td>
<td>100</td>
<td>0.200</td>
</tr>
<tr>
<td>Task 2</td>
<td>40</td>
<td>150</td>
<td>0.267</td>
</tr>
<tr>
<td>Task 3</td>
<td>100</td>
<td>350</td>
<td>0.286</td>
</tr>
</tbody>
</table>

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

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

300

UB test is only sufficient, not necessary!

- Let \( U = \sum C_i / T_i \) and \( B(n) = n \times (2^{1/n} - 1) \)
- Three possible outcomes:
  - \( 0 < U \leq B(n) \): schedulable
  - \( B(n) < U < 1 \): no conclusion
  - \( 1 < U \): overload

- Thus, the test may be too conservative
- (exact test will be given later)

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

This is only for the first periods! But we will see that this is enough to tell that the task set is schedulable.

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:
  - 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: \( U \leq n \times (2^{1/n} - 1) \)
- 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 scenario)
  - 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.
  - 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**

- Let \( R_i \) stand for the response time for task \( i \). Then \( R_i = C_i + \sum_{j \in HP(i)} \lfloor \frac{R_i}{T_j} \rfloor C_j \)
  - \( C_i \) is the computing time
  - \( I(i,j) \) is the so-called interference of task \( j \) to \( i \)
  - \( I(i,j) = 0 \) if task \( j \) has higher priority than \( i \)
  - \( I(i,j) = \lfloor \frac{R_i}{T_j} \rfloor C_j \) if task \( j \) has lower priority than \( i \)
  - \( \lfloor x \rfloor \) denotes the least integer larger than \( x \)
  - \( E.g., \lfloor 3.2 \rfloor = 3, \lfloor 3 \rfloor = 3, \lfloor 1.9 \rfloor = 2 \)
  - \( R_i = C_i + \sum_{j \in HP(i)} \lfloor \frac{R_i}{T_j} \rfloor C_j \)
Intuition on the equation

\[ R_i = C_i + \sum_{j \in HP(i)} \left( \frac{R_i}{T_j} \right) C_j \]

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

Equation solving and schedulability analysis

- We need to solve the equation:
  \[ R_i = C_i + \sum_{j \in HP(i)} \left( \frac{R_i}{T_j} \right) C_j \]
- This can be done by numerical methods to compute the fixed point of the equation e.g. By iteration:
  \[ R_i^0 = C_i + \sum_{j \in HP(i)} C_j \]
  \[ R_i^{k+1} = C_i + \sum_{j \in HP(i)} \left( \frac{R_i^k}{T_j} \right) C_j \]
- 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

Example

- Assume a task set: \{(1,3),(1,5),(1,6),(2,10)\}
- Question: is the task set schedulable?
- Answer: YES
- Because
  - \( R_1^1 = R_1^0 = C_1 = 1 \) (done)
  - \( R_2^0 = C_2 + C_1 = 2 \)
  - \( R_2^1 = C_2 + \left[ \frac{R_2^0}{T_1} \right] C_1 = 1 + \left[ \frac{2}{3} \right] 1 = 2 \) (done)

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 the tasks with lower priority

Example (combine UB test and precise test)

- Consider the same task set: \{(1,3),(1,5),(1,6),(3,10)\}
- CPU utilization \( U = \frac{1}{3} + \frac{1}{5} + \frac{1}{6} + \frac{3}{10} = 0.899 > B(4) = 0.756 \)
  - Fail the UB test!
- But \( U(3) = \frac{1}{3} + \frac{1}{5} = 0.669 < B(3) = 0.779 \)
  - This means that the first 3 tasks are schedulable
- Question: is task 4 set schedulable?
  - \( R_4^0 = C_4 + C_2 + C_3 = 6 \)
  - \( R_4^1 = C_4 + \left[ \frac{R_4^0}{T_1} \right] C_1 = 4 + \left[ \frac{6}{3} \right] 1 = 4 + 2 = 6 \)
  - \( R_4^2 = C_4 + \left[ \frac{R_4^1}{T_1} \right] C_1 = 4 + \left[ \frac{6}{3} \right] 1 = 4 + 2 = 6 \)
  - \( R_4^3 = C_4 + \left[ \frac{R_4^2}{T_1} \right] C_1 = 4 + \left[ \frac{6}{3} \right] 1 = 4 + 2 = 6 \)
  - \( R_4^4 = C_4 + \left[ \frac{R_4^3}{T_1} \right] C_1 = 4 + \left[ \frac{6}{3} \right] 1 = 4 + 2 = 6 \)
  - \( R_4^5 = C_4 + \left[ \frac{R_4^4}{T_1} \right] C_1 = 4 + \left[ \frac{6}{3} \right] 1 = 4 + 2 = 6 \)
  - \( R_4^6 = C_4 + \left[ \frac{R_4^5}{T_1} \right] C_1 = 4 + \left[ \frac{6}{3} \right] 1 = 4 + 2 = 6 \)
  - \( R_4^7 = C_4 + \left[ \frac{R_4^6}{T_1} \right] C_1 = 4 + \left[ \frac{6}{3} \right] 1 = 4 + 2 = 6 \)
  - \( R_4^8 = C_4 + \left[ \frac{R_4^7}{T_1} \right] C_1 = 4 + \left[ \frac{6}{3} \right] 1 = 4 + 2 = 6 \)
  - \( R_4^9 = C_4 + \left[ \frac{R_4^8}{T_1} \right] C_1 = 4 + \left[ \frac{6}{3} \right] 1 = 4 + 2 = 6 \)
  - \( R_4^{10} = C_4 + \left[ \frac{R_4^9}{T_1} \right] C_1 = 4 + \left[ \frac{6}{3} \right] 1 = 4 + 2 = 6 \)
  - \( R_4^{11} = C_4 + \left[ \frac{R_4^{10}}{T_1} \right] C_1 = 4 + \left[ \frac{6}{3} \right] 1 = 4 + 2 = 6 \)
  - Task 4 is non schedulable!
Calculate response time for task 3

- \( R_3^0 = C_1 + C_2 + C_3 = 180 \)
- \( R_3^1 = C_3 + \frac{R_3^0}{T_1} \cdot C_1 + \frac{R_3^0}{T_2} \cdot C_2 \)
  \[ = 100 + \frac{180}{100} \cdot 40 + \frac{180}{150} \cdot 40 \]
  \[ = 100 + 2 \cdot 40 + 2 \cdot 40 = 260 \]
- \( R_3^2 = C_3 + \frac{R_3^1}{T_1} \cdot C_1 + \frac{R_3^1}{T_2} \cdot C_2 \)
  \[ = 100 + \frac{260}{100} \cdot 40 + \frac{260}{150} \cdot 40 = 300 \]
- \( R_3^3 = C_3 + \frac{R_3^2}{T_1} \cdot C_1 + \frac{R_3^2}{T_2} \cdot C_2 \)
  \[ = 100 + \frac{300}{100} \cdot 40 + \frac{300}{150} \cdot 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 before to calculate the response times.

Summary: Three ways to check schedulability

1. UB test (simple but conservative)
2. Response time calculation (precise test)
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

Extensions to the basic RMS

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

RMS for tasks with \( D \leq T \)

- 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 referred as Rate Monotonic Scheduling for historical reasons and they are so similar

**DMS: Schedulability analysis**

- UB test (sufficient):
  $\Sigma C_i/D_i <= n^{*(2^{1/n}-1)}$ implies schedulable by DMS

- Precise test (exactly the same as for RMS):
  Response time calculation:
  $R_i = C_i + \Sigma_{j \in HP(i)} \left\lceil R_i/T_j \right\rceil C_j$
  - $R_i^0 = C_i + \Sigma_{j \in HP(i)} C_j = C_1 + C_2 + ... + C_i$ → the first guess
  - $R_i^{k+1} = C_i + \Sigma_{j \in HP(i)} \left\lceil R_i^{k}/T_j \right\rceil C_j$ → the $(k+1)$th guess
  - The iteration stops when either
    - $R_i^{m+1} > Di$ or non schedulable
    - $R_i^{m} < Di$ and $R_i^{m+1} = R_i^{m}$ → schedulable

**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 $Di = Ti$; $U <= 1$ if 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**

<table>
<thead>
<tr>
<th>Static/Fixed-priority</th>
<th>D=Ti</th>
<th>D&lt;e=Ti</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMS</td>
<td>Sufficient test</td>
<td>$\Sigma C_i/T_i &lt;= n^{*(2^{1/n}-1)}$</td>
</tr>
<tr>
<td>Precise test</td>
<td>$R_i = C_i + \Sigma_{j \in HP(i)}\left\lceil R_i/T_j \right\rceil C_j$</td>
<td></td>
</tr>
<tr>
<td>$R_i &lt;= Ti$</td>
<td>$R_i = C_i + \Sigma_{j \in HP(i)}\left\lceil R_i/T_j \right\rceil C_j$</td>
<td></td>
</tr>
<tr>
<td>Dynamic priority</td>
<td>EDF</td>
<td>Precise test</td>
</tr>
<tr>
<td></td>
<td>$\Sigma C_i/T_i &lt;= 1$</td>
<td>$\Sigma C_i/T_i &lt;= 1$</td>
</tr>
</tbody>
</table>

$R_1=4$
$R_2=1$
$R_3=3$
$R_4=10$
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.

Thus, the real computing time for a task should be:
\[
C_l' = C_l + C_s + \sum_{j} \left( \frac{R_i}{T_j} \right) \times C_j
\]

The schedulability analysis techniques we studied so far are applicable if we use the new computing time \( C_l' \).
Unfortunately this is not right.

Handling context switch overhands?

Handling interrupts: problem and example

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

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 despite its period
- Interrupt processing may delay tasks with shorter periods (deadlines)
- how to calculate the worst case response time?
Handling non-preemptive sections

- So far, we have assumed that all tasks are preemptive regions of code. This is 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 $B_i$ 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 $i$

Handling non-preemptive sections: Response time calculation

- The equation for response time calculation:
  $$R_i = B_i + C_i + \sum_{j \in HP(i)} \left( \frac{R_i}{T_j} \right) * C_j$$
- 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.

So now, we have an equation:

$$R_i = B_i + C_i + 2Ccs + \sum_{j \in HP(i)} \left( \frac{R_i}{T_j} \right) * (C_j + 4* Ccs)$$

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

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

- \( J(\text{biggest}) = \text{maximal delay from period-start} \)
- \( J(\text{smallest}) = \text{minimal delay from period-start} \)
- 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

**Jitter: Example**

\( \{(20, 100), (40, 150), (20, T_3)\} \)

\( T_1 \):

- One release
- \( R_{low} = 0 \)
- \( T_{low} = 20 \)

\( T_2 \):

- One more release due to the jitter
- \( J_{high} = 90 \)
- \( R_{low} + J_{high} = 90 \)
- \( T_{low} = 200 \)

\( T_3 \):

- One more release due to the jitter
- \( J_{high} = 210 \)
- \( R_{low} + J_{high} = 210 \)
- \( T_{low} = 300 \)

**The number of preemptions due to Jitter**

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

**The number of preemptions/blocking when jitters occur**

- Task \( L \) will be preempted at least 2 times if \( R_{low} > T_{low} - J_{high} \)
- Task \( L \) will be preempted at least 3 times if \( R_{low} > 2 * T_{low} - J_{high} \)
- ...
- Task \( L \) will be preempted at least \( n \) times if \( R_{low} > (n-1) * T_{low} - J_{high} \)

Thus \( (R_{low} + J_{high}) / T_{high} > n \)

the largest \( n \) satisfying the condition is given by

\( n = \lceil (R_{low} + J_{high}) / T_{high} \rceil \)
Handling Jitters in schedulability analysis

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

Now, we have an equation:

$R_i = C_i + 2C_{cs} + B_i + \sum_{j \in HP(i)} \left(\frac{(R_i + J_j)}{T_j}\right) \times (C_j + 4C_{cs})$

The response time for task $i$

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

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

Finally, we have an equation (why?):

$R_i = C_i + 2C_{cs} + B_i + R_{si} + \sum_{j \in HP(i)} \left(\frac{(R_i + J_j)}{T_j}\right) \times (C_j + 4C_{cs})$

Resource Sharing with HLP and PCP (and BIP)

- Let $CS(k,S)$ denote the computing time for the critical section that task $k$ uses semaphore $S$.
- $Use(S)$ is the set of tasks using $S$
- Then for HLP and PCP, the maximal blocking time $R_{si}$ and response time $R_i$ for task $i$ is as follows:
  - $R_{si} = \max\{CS(k,S) | (i,k) \in Use(S), pr(k) < pr(i) \leq C(S)\}$
  - $R_i = R_{si} + C_i + \sum_{j \in HP(i)} \left(\frac{R_i}{T_j}\right) \times C_j$

How about BIP?

- $R_i = \text{Sum}(CS(k,S) | (i,k) \in Use(S), pr(k) < pr(i) \leq C(S))$
- $R_i = R_{si} + C_i + \sum_{j \in HP(i)} \left(\frac{R_i}{T_j}\right) \times C_j$

Summary: + and -

- Static Cyclic Scheduling (SCS)
  - Simple, and reliable, may be difficult to construct the time table and difficult to modify (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)
- Deadline Monotonic Scheduling (DMS)
  - Similar to RMS
- Handling overheads, blocking, resource sharing (priority ceiling protocols)