Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment

C. Liu and J. Layland
Contents

1. Introduction and Background
2. The Environment
3. A Fixed Priority Scheduling Algorithm
4. Achievable Processor Utilization
5. The Deadline Driven Scheduling Algorithm
Introduction and Background

➢ The use of computers for
  ▪ Time-critical control and monitoring of industrial processes

➢ Two scheduling algorithms are studied
  ▪ Both are priority driven and preemptive
    • The processing of any task is interrupted by a request for any higher priority task
  ▪ The first one
    • Fixed priority assignment algorithm
  ▪ The second one
    • Dynamic priority assignment
The Environment

Five assumptions

- (A1) The request for all tasks for which hard deadlines exist are periodic
- (A2) Deadlines consist of run-ability constraints only
  - Each task must be completed before the next request for it occurs
- (A3) The tasks are independent in that requests for a certain task do not depend on the initiation or the completion of requests for other tasks
- (A4) Run-time for each task is constant for that task and does not vary with time. Run-time here refers to the time which is taken by a processor to execute the task without interruption
- (A5) Any nonperiodic tasks in the system are special; they are initialization or fault-recovery routines; they displace periodic tasks while themselves are being run, and do not themselves have hard, critical deadlines
A Fixed Priority Scheduling Algorithm (1)

- **Task and scheduling model**
  - **Deadline of a request**
    - The time of the next request for the same task
  - **Overflow of a request**
    - Unfulfilled request by its deadline
  - **Feasible algorithm**
    - If tasks are scheduled so that no overflow ever occurs
  - **Response time of a request**
    - The time span between the request and the end of the response to that request
  - **Critical instant of a task**
    - An instant at which a request for that task will have the largest response time
  - **Critical time zone for a task**
    - The time interval between a critical instant and the deadline of the corresponding request of the task
A Fixed Priority Scheduling Algorithm (2)

➢ Theorem 1. A critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks

➢ Proof.

- Advancing the request time $t_2$ will not speed up the completion of $\tau_m$

![Diagram showing execution of $\tau$, between requests for $\tau_m$](image)
A Fixed Priority Scheduling Algorithm (3)

Important value of Theorem 1

- A simple direct calculation can determine whether or not a given priority assignment will yield a feasible scheduling algorithm

Example: two tasks $\tau_1$ and $\tau_2$

- $T_1 = 2, C_1 = 1$
- $T_2 = 5, C_2 = 1$
A Fixed Priority Scheduling Algorithm (4)

The following inequalities must be met for feasibility

- If $\tau_1$ is the highest priority task (method 1)
  
  \[ \left\lfloor \frac{T_2}{T_1} \right\rfloor C_1 + C_2 \leq T_2. \]  

- If $\tau_2$ is the highest priority task (method 2)
  
  \[ C_1 + C_2 \leq T_1. \]

- It follows from (2) that
  
  \[ \left\lfloor \frac{T_2}{T_1} \right\rfloor C_1 + \left\lfloor \frac{T_2}{T_1} \right\rfloor C_2 \leq \left\lfloor \frac{T_2}{T_1} \right\rfloor T_1 \leq T_2 \]

Whenever $T_1 \leq T_2$ and $\tau_2$ has the highest priority, Ineq.(2) implies Ineq.(1)

- Ineq.(1) is a weaker condition
- If a task set is schedulable by method 2, then it is also schedulable by method 1
A Fixed Priority Scheduling Algorithm (5)

- Reasonable rule of priority assignment
  - Assign priorities according to request rates, independent of run-times
  - Tasks with higher request rates will have higher priorities

- Rate-monotonic priority assignment
  - Optimum in the sense that no other fixed priority assignment rule can schedule a task set which cannot be scheduled by the rate-monotonic priority assignment

- Theorem 2. If a feasible priority assignment exists for some task set, the rate monotonic priority assignment is feasible for that task set
  - Proof. By an inter-change argument
Achievable Processor Utilization (1)

- **Utilization factor**
  - The fraction of processor time spent in the execution of the task set
    - The utilization factor is equal to one minus the fraction of idle processor time
  - $U = \sum_{i=1}^{n} (C_i/T_i)$.
  - Utilization factor can be improved by
    - Increasing run-times and decreasing periods
  - However, utilization factor is upper bounded by the requirement of feasibility (meeting deadlines)
  - How large the processor utilization factor can be?
Achievable Processor Utilization (2)

➢ Full utilization of processor
   - A set of tasks is said to “fully utilize” the processor if the priority assignment is feasible for the set and if an increase in the run-time of any of the tasks in the set will make the priority assignment infeasible

➢ LUB (least upper bound) of utilization factor
   - The minimum of utilization factor over all sets of tasks that fully utilize the processor
   - For all task sets whose utilization factor is below this bound, there exists a fixed priority assignment which is feasible
Achievable Processor Utilization (3)

➢ Schedulable Utilization Bound
Achievable Processor Utilization (4)

- Theorem 3. For a set of two tasks with fixed priority assignment, the least upper bound to the processor utilization factor is \( U = 2(2^{1/2} - 1) \)

- Proof. See Theorem 4
Achievable Processor Utilization (5)

Theorem 4. For a set of $m$ tasks with fixed priority order, and the restriction that the ratio between any two request periods is less than 2, the least upper bound to the processor utilization factor is

$$U = m(2^{1/m} - 1)$$

Proof.

1. We first show that the LUB can be found when
   \[ C_1 = T_2 - T_1 \quad C_k = T_{k+1} - T_k \quad \text{and} \]
   \[ C_m = T_m - 2(C_1 + C_2 + \ldots + C_{m-1}) = 2T_1 - T_m \]

2. We then compute the LUB
Achievable Processor Utilization (6)

We first show that the LUB can be found when
\[ C_1 = T_2 - T_1, \quad C_k = T_{k+1} - T_k, \quad \text{and} \quad C_m = 2T_1 - T_m \]

Proof (1). For \( \Delta \geq 0 \)

1. Suppose that \( C_1 = T_2 - T_1 + \Delta \) and the task set \( S \) fully utilize
   - We can construct another set \( S' \) such that
   - \( C'_1 = T_2 - T_1, \quad C'_2 = C_2 + \Delta, \quad \text{and} \quad C'_k = C_k \)
   - \( S' \) fully utilize since \( C_1 + C_2 = C'_1 + C'_2 \)
   - \( U - U' = \Delta / T_1 - \Delta / T_2 \leq 0 \)

2. Suppose that \( C_1 = T_2 - T_1 - \Delta \) and the task set \( S \) fully utilize
   - We can construct another set \( S' \) such that
   - \( C'_1 = T_2 - T_1, \quad C'_2 = C_2 - 2\Delta, \quad \text{and} \quad C'_k = C_k \)
   - \( S' \) fully utilize
   - \( U - U' = -\Delta / T_1 + 2\Delta / T_2 \geq 0 \)

Similarly, we can show that \( C_k = T_{k+1} - T_k \)
Two Task Case - all fully utilize (7)

1.1/2 + 0.9/3 = 5.1/6

1/2 + 1/3 = 5/6 → LUB case!

0.9/2 + 1.2/3 = 5.1/6
Achievable Processor Utilization (8)

Illustration

Case (1) ➔

\[
\begin{align*}
&\text{\(C'_1 + C'_2 = C_1 + C_2\)} \quad \ldots \quad \text{\(C'_1 + C'_2 = C_1 + C_2\)} \\
&0 \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad T_1 \quad T_2 \quad T_3
\end{align*}
\]

To increase \(C_1\) by \(\Delta\), we need to decrease \(C_2\) by \(2\Delta\).

Case (2) ➔

\[
\begin{align*}
&\text{\(C_1 \quad C_2\)} \quad \ldots \quad \text{\(C_1\)} \\
&0 \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad T_1 \quad T_2
\end{align*}
\]

Case for LUB ➔

\[
\begin{align*}
&\text{\(C_1 \quad C_2 \quad \ldots \quad C_{m-1} \quad C_m \quad C_1 \quad C_2 \quad \ldots \quad C_{m-1}\)} \\
&0 \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad T_1 \quad T_2 \quad T_3 \quad T_{m-1}
\end{align*}
\]
Achievable Processor Utilization (9)

➢ We next compute the LUB

➢ Using the previous results, we can write

\[ U = \frac{(T_2 - T_1)}{T_1} + \frac{(T_3 - T_2)}{T_2} + \ldots + \frac{(2T_1 - T_m)}{T_m} \]

\[ = \frac{T_2}{T_1} - 1 + \frac{T_3}{T_2} - 1 + \ldots + 2 \frac{T_1}{T_m} - 1 \]

\[ = \frac{T_2}{T_1} + \frac{T_3}{T_2} + \ldots + 2 \frac{T_1}{T_m} - m \]

➢ Let \( R_i = \frac{T_{i+1}}{T_i} \)

\[ \frac{T_1}{T_m} = \frac{T_1}{T_m} \frac{T_{m-1} \ldots T_3 T_2}{T_{m-1} \ldots T_3 T_2} = \frac{T_{m-1} T_{m-2} \ldots T_3 T_1}{T_{m-1} T_{m-2} \ldots T_3 T_2} = \frac{1}{R_{m-1} R_{m-2} \ldots R_1} \]

\[ U = R_1 + R_2 + \ldots + R_i + \ldots + 2 \frac{1}{(R_{m-1} \ldots R_1)} - m \]
Achievable Processor Utilization (10)

Given

\[ U = R_1 + R_2 + \ldots + R_i + \ldots + 2/(R_{m-1} \ldots R_1) - m \]

To find the minimum, we solve the following

\[
\frac{\partial U}{\partial R_i} = 1 - 2(R_{m-1} \ldots R_1)/(R_{m-1} \ldots R_i \ldots R_1)^2 = 0
\]

\[ R_i = 2/(R_{m-1} R_{m-2} \ldots R_2 R_1) \]

This implies \( R_1 = R_2 = \ldots = R_{m-1} \), thus \( R_i = 2^{1/m} \)

Finally it follows that

\[
U = (m-1)2^m + 2/(2^{m-1}) - m = (m-1)2^m + 2/(2^{1-1}) - m
\]

\[ = (m-1)2^m + 2^m - m = m(2^m - 1) \]
Achievable Processor Utilization (11)

- For \( m = 3 \), \( U = 3(2^{1/3} - 1) \approx 0.78 \)
- For large \( m \), \( U \approx \ln 2 \)

- The restriction that the largest ratio between request period less than 2 can be removed

- Theorem 5. For a set of \( m \) tasks with fixed priority order, the least upper bound to processor utilization is \( m(2^{1/m} - 1) \)
Achievable Processor Utilization (12)

Outline of proof of Theorem 5

- The idea is that if a set of tasks fully utilizes the processor and for some $i$, $i < m$
  - $T_m/T_i \geq 2$,

- then we can always construct another set of tasks that will
  - (1) fully utilize the processor,
  - (2) $T_m/T_i < 2$, and
  - (3) the utilization of the new set is less than the original one
The Deadline Driven Scheduling Algorithm

- Priorities are assigned to tasks according to the deadlines of their current requests
  - A task will be assigned the highest priority if the deadline of its current request is the nearest
  - At any instant, the task with the highest priority and yet unfulfilled request will be executed
  - This method of assigning priorities is a dynamic one

- We want to establish a necessary and sufficient condition for the feasibility of the deadline driven scheduling algorithm
The Deadline Driven Scheduling Algorithm

➢ Theorem 6. When the deadline driven scheduling algorithm is used to schedule a set of tasks on a processor, there is no processor idle time prior to an overflow

➢ Proof. Shifting $a, b, \ldots, c$ to $t_2$ leads to a contradiction
The Deadline Driven Scheduling Algorithm

Theorem 7. For a given set of m tasks, the deadline driven scheduling algorithm is feasible if and only if

\[ U = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \ldots + \frac{C_m}{T_m} \leq 1 \]

Proof.

(1) To show the necessity, compute the total demand of computation time between \([0, T_1, T_2, \ldots, T_m]\)

\[
(T_2T_3 \cdots T_m)C_1 + (T_1T_3 \cdots T_m)C_2 + \cdots + (T_1T_2 \cdots T_{m-1})C_m.
\]

\[
(T_2T_3 \cdots T_m)C_1 + (T_1T_3 \cdots T_m)C_2 + \cdots + (T_1T_2 \cdots T_{m-1})C_m > T_1T_2 \cdots T_m
\]

\[
(C_1/T_1) + (C_2/T_2) + \cdots + (C_m/T_m) > 1
\]
Proof of Theorem 7.

(2) To show the sufficiency, assume that the algorithm is not feasible and

\[ U = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \ldots + \frac{C_m}{T_m} \leq 1 \]

- There will be an overflow at \( t = T \) in the interval \([0, T1T2\ldots Tm]\).
- There is no idle time in the interval \([0, T]\).
- Let \( a_1, a_2, a_3, \ldots, b_1, b_2, b_3 \ldots \) denote the request times immediately before \( T \).
- Let \( a_1, a_2, a_3, \ldots \) are the request times of tasks with deadlines at \( T \).
- Let \( b_1, b_2, b_3, \ldots \) are the request times of tasks with deadlines beyond \( T \).
The Deadline Driven Scheduling Algorithm

Proof of Theorem 7.

- Two cases must be considered
  - Case 1. None of the computation times of requested at $b_1, b_2, b_3, ...$ was carried out before $T$
  - Case 2. Some of the computation times of requested at $b_1, b_2, b_3, ...$ was carried out before $T$
Proof of Theorem 7.

Case 1.

- The total demand of computation time in the interval \([0, T]\):
  \[
  \lfloor T/T_1 \rfloor C_1 + \lfloor T/T_2 \rfloor C_2 + \cdots + \lfloor T/T_m \rfloor C_m
  \]

- Since there is no idle period:
  \[
  \lfloor T/T_1 \rfloor C_1 + \lfloor T/T_2 \rfloor C_2 + \cdots + \lfloor T/T_m \rfloor C_m > T
  \]

- Thus:
  \[
  (T/T_1) C_1 + (T/T_2) C_2 + \cdots + (T/T_m) C_m > T
  \]

  \[
  (C_1/T_1) + (C_2/T_2) + \cdots + (C_m/T_m) > 1
  \]

- This is a contradiction.
The Deadline Driven Scheduling Algorithm

Proof of Theorem 7.

Case 2.

• There must exist a point $T'$ such that none of requests at $b_1, b_2, b_3, \ldots$ was carried out in the interval $[T', T]$.

• In other words, only those requests with deadline at or before $T$ will be executed.

• Note that since some of the requests at $b_1, b_2, b_3, \ldots$ can be executed until $T'$, all those requests initiated before $T'$ with deadlines at or before $T$ have been fulfilled before $T'$.

• Therefore, the total demand of computation time in the interval $[T', T]$ is less than or equal to

$$L(T - T')/T_1 \lor C_1 + L(T - T')/T_2 \lor C_2 + \cdots + L(T - T')/T_m \lor C_m$$

• Since an overflow occurs,

$$L(T - T')/T_1 \lor C_1 + L(T - T')/T_2 \lor C_2 + \cdots + L(T - T')/T_m \lor C_m > T - T',$$

• $(C_1/T_1) + (C_2/T_2) + \cdots + (C_m/T_m) > 1$, which is a contradiction.
The Deadline Driven Scheduling Algorithm

Fig. 5. Processing overflow at time $T$ without execution of $\{b_i\}$ following $T'$
Note!

➢ Task $b_j$ cannot arrive more than once in the interval $(T', T)$, since this means task $b_j$ has a deadline earlier than $T$ and must be completed before $T$ in $(T', T)$.

➢ Thus, there is no execution of task $b_j$ in $(T', T)$ and this means only tasks $a_i$ can execute in $(T', T)$.

➢ Let the computation time demand of all tasks $a_i$ be $X$ and $X$ is less than

\[ \sum_{i=1}^{m} C_i \]  

- Terms for task $b_j$ are 0

➢ $X > T - T'$ since there is a deadline miss, thus

\[ \sum_{i=1}^{m} C_i \]  

\[ > T - T' \]
The Deadline Driven Scheduling Algorithm

- Deadline driven scheduling algorithm is optimum
  - If a set of tasks can be scheduled by any algorithm, it can be scheduled by the deadline driven driven algorithm

- This claim comes from Theorem 7
  - If a set of tasks can be scheduled by any algorithm, their CPU utilization is no greater than one
  - Therefore, the task set can also be scheduled by the deadline driven scheduling algorithm
Other Proof of Optimality

- We show that any feasible schedule can be systematically transformed to EDF schedule.
- Assume that parts of two jobs $J_i$ and $J_k$ are scheduled in non-EDF order:

```
  J_i  J_k
```

This can be easily resolved by swapping the jobs:

```
  J_k  J_i
```

Note that this operation cannot cause deadline miss.