Schedulability Analysis for Dual Priority Scheduling

I. Dual Priority

Given a task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, we first make the following **assumptions**:

1) Each task τ_i has a original priority $n + i$ and a promotion priority i.
2) Each task has a fixed promotion point p_i. The concerned job $J_{i,k}$ has its promotion point $P_i = r_{i,k} + p_i$.

Lemma 1 (Worst Case Pattern): Suppose τ could experience deadline miss with the current priority and promotion point assignment, then it must be that there exists a busy time interval $[0, t]$ during which some job of task $\tau_i \in \tau$ would miss its deadline when all other tasks release their first job at time instant 0, and all jobs including those released by τ_i are released as soon as possible with corresponding period.

Proof 1: Among all possible legal event sequences in which τ misses some deadline, let S denote such a sequence where some job of $\tau_i \in \tau$ misses its deadline within the shortest busy interval $[0, t_F]$. As a result no deadline miss would happens earlier than t_F.

Assuming in the event sequence S, there exists some tasks, e.g., τ_j whose first release is not at time instant 0 and the separation between some job release is greater than T_j. Then as we shift $r_{j,1}$ to 0 and reduce time separation between each job releases to T_j, τ_i would consume no more resource than before.

This is because that all jobs of τ_j with deadline before t would still meet its deadline (otherwise we can construct a new sequence S), while the last job of τ_j released before t (e.g., $J_{j,m}$) is more likely to deny processor from τ_i since the promotion point of job $J_{j,m}$ will also decrease. Therefore after we modify the release pattern of τ_j, τ_i would still misses its deadline. Finally by repeating the above steps for all such tasks, the deadline miss of τ_i would still happen at t_F. Therefore Lemma 1 is true.

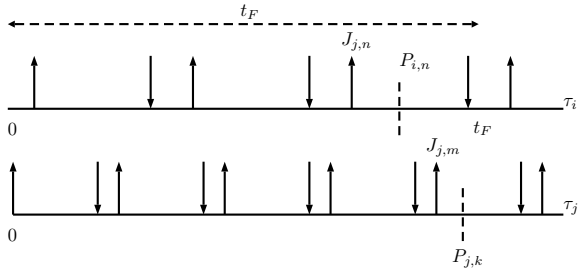

Fig. 1. Worst Case Pattern

From Lemma 1 we know that if τ would experience some deadline miss, then there exists a τ_i that will have its deadline miss with the worst case release pattern defined in Lemma 1. Therefore the following corollary can tell us whether a task system τ is schedulable by the dual priority scheduling algorithm.

Corollary 1: If $\forall \tau_i \in \tau$ no deadline miss happens for all possible time interval length t with the worst case release pattern: 1) some job of τ_i has deadline at t and all jobs of τ_i are released as soon as possible with period T_i; 2) first job of $\tau_j \in \tau - \tau_i$ is released at 0 and all jobs are released as soon as possible with period T_j, then τ is schedulable by the dual priority scheduling algorithm.

Proof 2: This corollary is a direct deduction from Lemma 1.

From Corollary 1, we know that as along as we can guarantee that $\forall\ \tau_i \in \tau : \forall\ t$: no deadline miss happens with the worst case release pattern, then we can declare that the task system τ is schedulable by the dual priority scheduling algorithm. Therefore in the simplest case, an exact but computational expensive schedulability test for dual priority scheduling algorithm could be derived by simulating the behavior of the system with the worst case release pattern.

To reduce the complexity, we need a more efficient test to determine whether $\tau_i \in \tau$ is schedulable

when jobs are released with the worst case release in Lemma 1. Let $ibf_i(\tau_j, t')$ (ibf is short for interference bound function) denote maximum possible resource consumed by $\tau_j \in \tau - \tau_j$ in the system during the time interval $[0, t']$, and let $dbf(\tau_i, t)$ denotes the maximum execution requirement of τ_i during $[0, t]$, when all tasks are released with worst case pattern in Lemma 1.[1] Therefore the following theorem can be used to determine whether $\tau_i \in \tau$ is schedulable when all jobs in the system is released in the worst case release pattern.

Theorem 1: τ_i would not experience any deadline miss with the worst case release pattern on a single processor by the dual priority scheduling algorithm if the following condition holds:

$$\forall\, t\ : \exists\, t' \in [t - D_i, t] :\ dbf(\tau_i, t) + ibf_i(\tau - \tau_i, t') \leq t' \tag{1}$$

where

$$ibf_i(\tau - \tau_i, t') = \sum_{\tau_j \in \tau - \tau_i} ibf_i(\tau_j, t')$$

.

Proof 3: We can prove the statement that if τ_i is not schedulable with the worst case release pattern, then Equation 1 would not hold. Suppose that τ_i is not schedulable then there are legal event sequences in which τ_i misses some deadline when τ_i is assigned with the current priority and promotion point. Let S' denote such a sequence where τ_i misses deadline at the earliest time at t.

It must be the cases that during $[0, t]$ some tasks are executing because otherwise the time interval between the last idle instant to t could also construct such a sequence. Then it must be that during the time interval $[0, t']$, other tasks have consumed an amount of resource more than

$$ibf_i(\tau - \tau_i, t') > t' - dbf(\tau_i, t)$$

which contradicts the Equation 1.
With Theorem 1, we can use the following test to whether a task system τ is schedulable by the dual priority scheduling algorithm.

Theorem 2: A task system τ is schedulable by the dual priority scheduling on unit-speed uniprocessor if the following hold: $\forall \tau_i \in \tau :\ \forall\ t :\ \exists\ t' \in [t - D_i, t]$:

$$dbf(\tau_i, t) + ibf_i(\tau - \tau_i, t') \leq t' \tag{2}$$

In the worst case release pattern, we can easily calculate the exact value of $dbf(\tau_i, t)$. However the exact value of $ibf_i(\tau - \tau_i, t')$ is not trivial except we simulate the system. As a result, here we use an upper bound of the actual interference to represent $ibf_i(\tau_j, t')$ instead.

II. Interference Bound Function

In the worst case pattern, given a time interval length t, we can easily calculate the release time and promotion point of each job. Here we first present some notations that will be used later in the paper.

TABLE I
NOTATIONS

$m = \lfloor \frac{t'}{T_j} \rfloor$	$r_{j,m} = m \times T_j$	$P_{j,m} = m \times T_j + p_j$
$P_{i,n} = t - D_i + p_i$	$r_{i,n} = P_{i,n} - p_i$	$[a]_0 = \max(a, 0)$
$m' = \lfloor \frac{r_i}{T_j} \rfloor$	$r_{j,m'} = m' \times T_j$	
$m'' = \lfloor \frac{P_{i,n}}{T_j} \rfloor$	$r_{j,m''} = m'' \times T_j$	

A. $ibf_i(\tau_j, t')$ when $j < i$

If the index j is smaller than index i, it means τ_j has higher origin priority than τ_i.

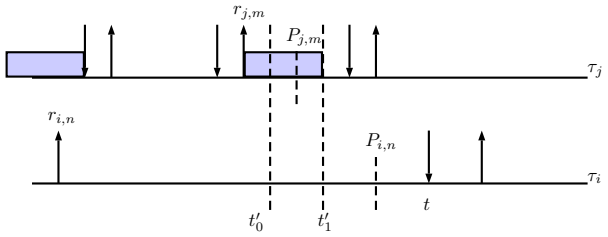

Fig. 2. $P_j \leq P_i$

Case 1 ($P_{j,m} \leq P_{i,n}$**)** as shown in Figure 2: Job $J_{i,n}$ always has lower priority than $J_{j,m}$. Thus the maximum possible resource consumed by τ_j before t' is bounded by

$$rbf_i(\tau_j, t') = m.C_j + \min(C_j, t' - r_{j,m})$$

[1]Note that the two functions are specific for the worst case pattern.

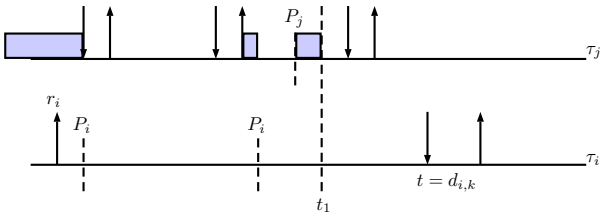

Fig. 3. $P_j > P_i$

Case 2 ($P_j > P_i$) as shown in Figure 3 τ_i has higher priority than τ_j's last job during $[\max(r_j, P_i), P_j]$ if $P_i \leq P_j$:

$$rbf_i(\tau_j, t') = \lfloor \frac{t'}{T_j} \rfloor C_j + \min\left(C_j, t' - r_j - (\min\{t', P_j\} - \max\{r_j, P_i\})\right)$$

B. $ibf_i(\tau_j, t')$ *when* $j > i$

Case 1.1 ($P_i \leq P_j \wedge r_j \leq r_i$) as shown in Figure 4: τ_j may execute during $[r_j, r_i]$, and τ_j would not execute after r_j or P_i unless τ_i has finished.

However we should assume all deadlines before t' is meet. whether we should assume it is meet.

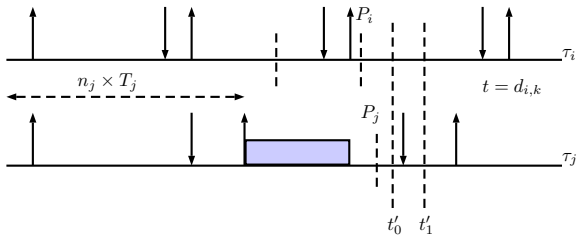

Fig. 4. $P_i \leq P_j \wedge r_j \leq r_i$

$$ibf_i(\tau_j, t') = (\lfloor \frac{t'}{T_j} \rfloor) \times C_j + \min(C_j, r_i - r_j)$$

Case 1.2 ($P_i \leq P_j \wedge r_j > r_i$) as shown in Figure 5: the last job of τ_j would not execute unless τ_i finishes. However all previous jobs are assumed to finish because otherwise the deadline miss should be already found. The $n_j - 1$ job must already finish by $\min(P_i, r_j - T_j + D_j)$.

$$rbf_i(\tau_j, t') = (n_j) \times C_j$$

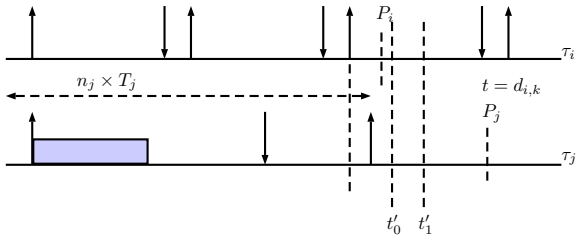

Fig. 5. $P_i \leq P_j \wedge r_j > r_i$

Case 2.1 ($P_i > P_j \wedge r_j \leq r_i$) as shown in Figure ??:the last job of τ_j can execute until $\min(t', P_i)$

$$rbf_i(\tau_j, t') = \min\left(C_j, r_i - r_j + [\min(t', P_i) - \max(P_j, r_i)]_0\right) + (n_j)C_j$$

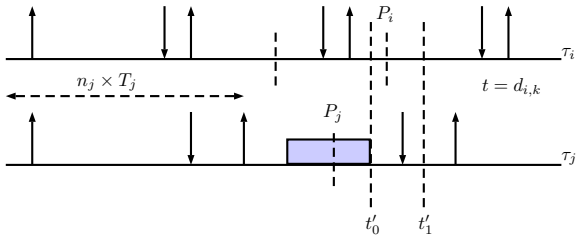

Fig. 6. $P_i > P_j \wedge r_j \leq r_i$

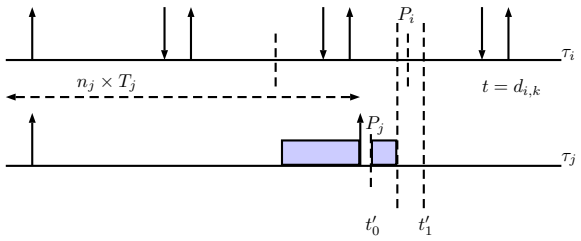

Fig. 7. $P_i > P_j \wedge r_j > r_i$

Case 2.2 ($P_i > P_j \wedge r_j > r_i$) as shown in Figure ??:

$$rbf_i(\tau_j, t') = (n_j)C_j + \min\left(C_j, [\min(t', P_i) - P_j]_0\right)$$