Realtime Systems - Fixed priority Scheduling

1 Until now

Last time we initiated the discussion of how to excutive a set of parallelly executing processes on a single CPU (pseudo parallellity). We assumed a given periodic taskset

$$\{\tau_1, ., \tau_N\} \tag{1}$$

with constant *relative* deadlines d_i as well as computation times, i.e.

$$\begin{aligned} d_{i,j} &= r_{i,j} + d_i \\ c_{i,j} &= c_i \end{aligned}$$

Given the above we wish to guarantee all deadlines to be met, i.e.

$$C_{i,j} \leq d_{i,j} = r_{i,j} + d_i \tag{2}$$

First we defined the time function $R(t)$ mapping any instant in time to the amount of work (in CPU time-units) remaining in the tasks ready to execute.

The time function $R(t)$ shows at all times how much work is left for the CPU to do of the work requested by the entire task set until time t.

Any time interval $[a, b]$ in which $R(t) > 0$ is called a **busy period**. Any interval in which $R(t) = 0$ is called an idle period.

We defined CPU utilization U as follows:
And concluded that within a busy period

$$R(t) \leq \sum_{i=1}^{N} c_i + (U - 1) \cdot t \tag{3}$$

Where the right hand side is a decreasing function for $U < 1$.

That is for $U < 1$ $R(t)$ will reach 0 before $Tmax = \frac{\sum_{i=1}^{N} c_i}{1-U}$, so that no busy period will last longer than $Tmax$. Any relative deadline d_i less than $Tmax$ will then be met.

However likewise we could show that

$$R(t) \geq (U-1) \cdot t - \sum_{i=1}^{N} c_i \tag{4}$$

so that for $U > 1$ the remaining work increases towards ∞. After some time some deadlines most certainly will fail.

The overall conlusion is that the CPU utilization U determines whether it is possible to meet any deadlines at all or if some deadlines are certain to fail.

It is important to notice that the above analysis is valid however we choose the order of execution, that is whatever scheduling algorithm we may choose to implement.

A number of examples were given where a given taskset was attempted scheduled with: Round Robin, Fixed priority scheduling and Fixed priority scheduling were priorities were set according to the Rate Monotonic Algorithm (RMA), where higher priority was assigned to tasks with lower periods. Only the RMA scheduling was able to solve the problem.

Likewise examples were given, where a taskset was not schedulable with RMA but where it was possible to device a Cyclic Executive solving the problem.

Cyclic executives may outperform both round-robin and RMA but the are not easy to design and they lack the flexibility of the former 2. When we need to add an extra task to some taskset we need to redesign the schedule totally.

Cyclic executives and round robin scheduling was analysed and criteria for schedulability were given in both cases.

2 Fixed priorities.

We will in the following consider fixed priority schemes and in particular RMA and set forth simple criteria for schedulability based on periods and computation times.

2.1 Critical instant

The first thing we need to show is for a given taskset (periods and computation times) what situation constitutes a worst case. The possible situations differ only in the way tasks are mutually phased, i.e. how their ready times are synchonized. We may express phases formally like

$$r_{i,1} = o_i \quad \text{forall } i \in 1, .., N \tag{5}$$

where the phases are the values $o_i \in]0, T_i[$. Or graphically

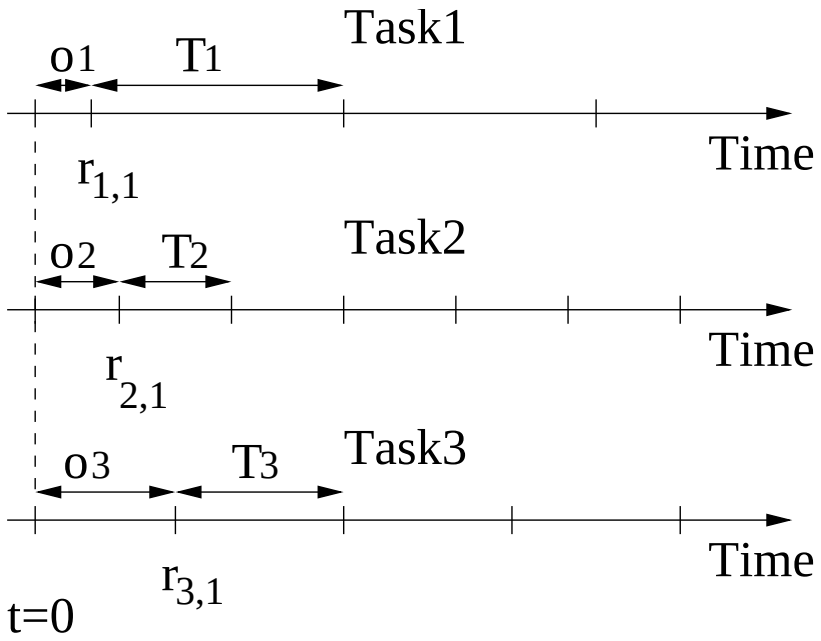

Figure 1: Phasing of periodic tasks

For a periodic taskset all ready times are uniquely determined when knowing periods T_i and phases o_i. However phases are seldomly known since tasks are not initiated simultaneously. What we are looking for is the worst possible values of phases o_i so that if a fixed priority schedule is feasible in this case it is generally feasible. The *critical instant theorem* gives us a simple and intuitively acceptable answer.

Theorem (critical instant)
A job $J_{i,j}$ issued at $r_{i,j}$ will complete no later than in the situation where all other tasks issue their jobs at the exact same time ($r_{i,j}$), that is when $o_i = 0$ forall i

Thus identical phase values constitute the worst case possible. If all jobs can survive a critical instant, i.e. complete before the next job from that job is issued, then all subsequent jobs from that task will also complete in due time, since they will never experience any thing worse than a critical instant.

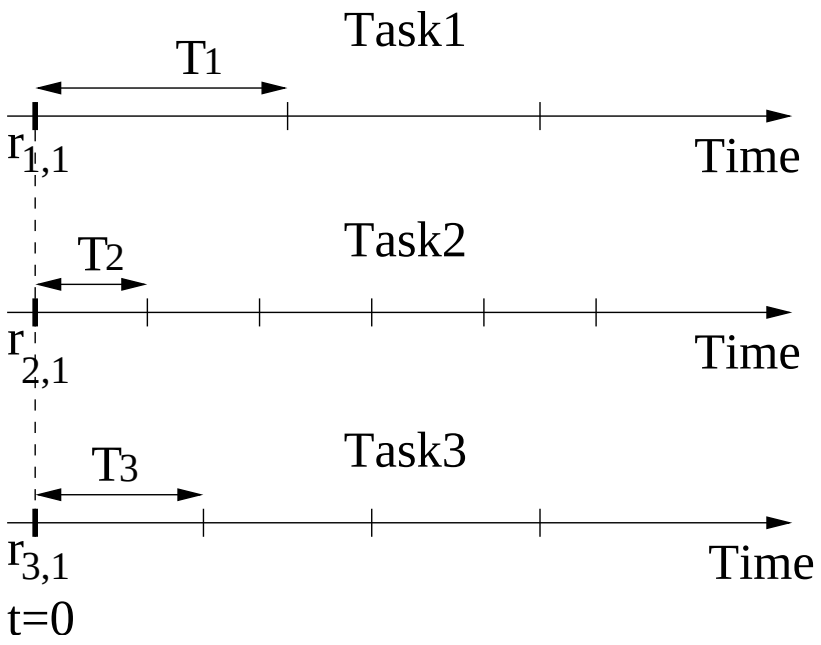

Figure 2: Critical phasing of periodic tasks

2.2 Proof of critical instant theorem (can be omitted)

Given some taskset $\tau_1, .., \tau_N$ with priorities in order, we assume the schedulability of tasks $\tau_1, ..\tau_{i-1}$. Then we wish to find the worst placing of the ready time for τ_i.

We define the function $L_i(r,t)$ to be the time left for tasks with prio. lower than $i-1$ in the interval $[r, r+t]$. Likewise we define the function $A_{i-1}(t)$ to be 1 when some task in $\tau_1, .., \tau_i$ is running. Then

$$L_i(r,t) = \int_r^{r+t} (1 - A_{i-1}(t))\ dt \tag{6}$$

so that

$$\frac{\partial\ L_i}{\partial\ r} = A_{i-1}(r) - A_{i-1}(r+t) \tag{7}$$

The question of schedulability of τ_i from time r is answered by:

$$L_i(r, d_i) \geq c_i \tag{8}$$

so that the smaller L_i the worst w.r.t. schedulability. We wish to finde time instants in time where L_i is locally smallest.

From 7 we have

- $L_i(\cdot, d_i)$ is non-increasing whenever $A_{i-1}(r)$ is 0
- $L_i(\cdot, d_i)$ is non-decreasing whenever $A_{i-1}(r)$ is 1

Local minimums will be where A_{i-1} jumps from 0 to 1, i.e. the beginning of busy periods for tasks $\tau_1, .., \tau_{i-1}$, as shown in figure 3

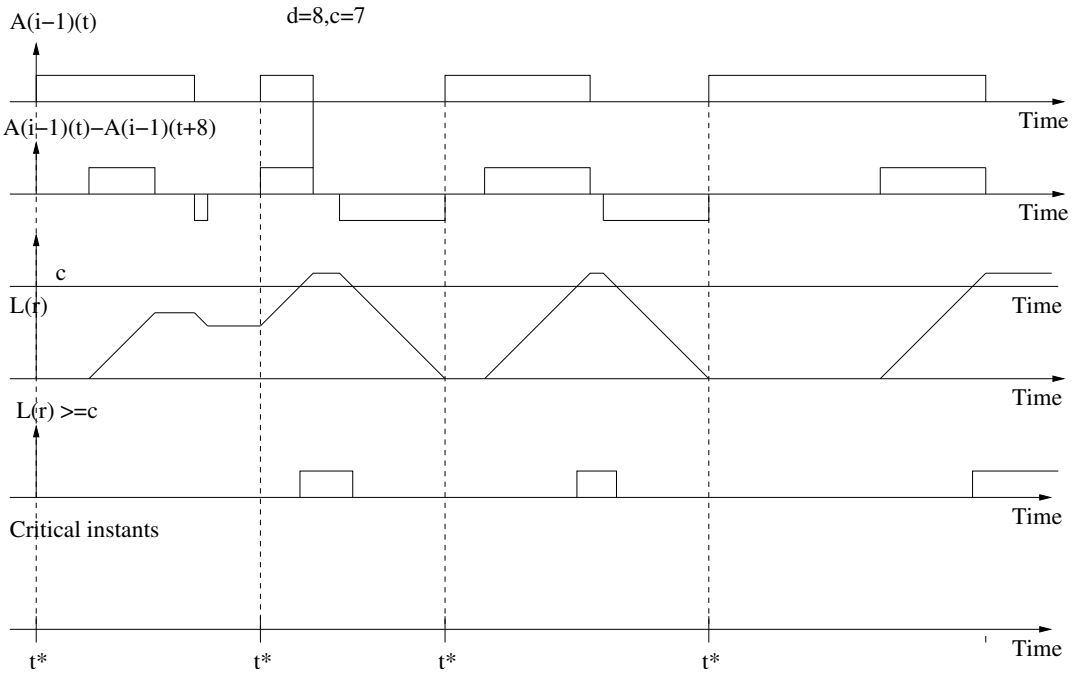

Figure 3: Spare CPU time in a fixed future interval as a function of time

If τ_i is not schedulable from some time t then there is an instant $t*$ where $a_{i-1}(t*)$ jumps from 0 to 1, and where τ_i is not schedulable either.

Thus we need only consider such instants $t*$. In these cases there is no remaining workload from higher priority tasks initiated prior to $t*$.

At any instant t where τ_i completes its 1. job all higher priority tasks have no unfinished work. Therefore the time consumed by higher prio tasks within $[t*, t]$ equals

$$\sum_{j=1}^{i-1} \lceil \frac{t - t* - o_j}{T_j} \rceil \cdot c_j \tag{9}$$

where o_i is the time for the first readytime for τ_j after $t*$.

Therefore the remaining CPU-time is found by:

$$L_i(t*, t) = (t - t*) - \sum_{j=1}^{i-1} \lceil \frac{t - t* - o_j}{T_j} \rceil \cdot c_j \tag{10}$$

which is smallest for all t whenever $o_j = 0$ for all j, which completes the proof.

3 Completion time analysis

The worst case completion time $C_{i,j}$ of som job $J_{i,j}$ may be found if we consider $J_{i,j}$ to be issued at a critical instant $t = 0$. Now we consider later times $t > 0$ where the CPU is idle from higher priority tasks. At such a time higher priority tasks completed all jobs issued prior to t. Thus the time $L_i(t)$ left over from higher priority tasks within $[0, t]$ can be found from

$$L_i(t) = t - \sum_{j=1}^{i-1} \lceil \frac{t}{T_j} \rceil \cdot c_j \tag{11}$$

which simply states that the time left over from higher priority tasks, is the CPU time offered in $[0, t]$ subtracted the time used by higher priority tasks.

At the completion time $C_{i,j}$ for $J_{i,j}$ the time left over should match the time spent by $J_{i,j}$, i.e.

$$c_i = L_i(C_{i,j}) \tag{12}$$

so that $C_{i,j}$ is the smallest number t fulfilling the equation

$$c_i = L_i(t) = t - \sum_{j=1}^{i-1} \lceil \frac{t}{T_j} \rceil \cdot c_j \tag{13}$$

or rewritten

$$t = c_i + \sum_{j=1}^{i-1} \lceil \frac{t}{T_j} \rceil \cdot c_j \tag{14}$$

Equation 14 is not analytically solvable but as time is a 1-dimensional variable, graphical methods turn out to work fine as we shall se in the following examples.

3.1 Example 3.1

Taskset τ_1, τ_2, τ_3 with periods $T_1 = 4, T_2 = 12, T_3 = 64$ and computation times $c_1 = 2, c_2 = 2, c_3 = 6$ and relative deadlines equal periods. This is the example from last lecture which was not schedulable with Round Robin or fixed priorities (1,3,2), but schedulable with RMA priorities (1,2,3).

We will first analyse the situation for priorities $(1, 3, 2)$ where τ_1 obviously completes at $t = 2$. Turning to τ_3 we (having the 2. highest priority) may solve equation 14 graphically which gives us figure (4)

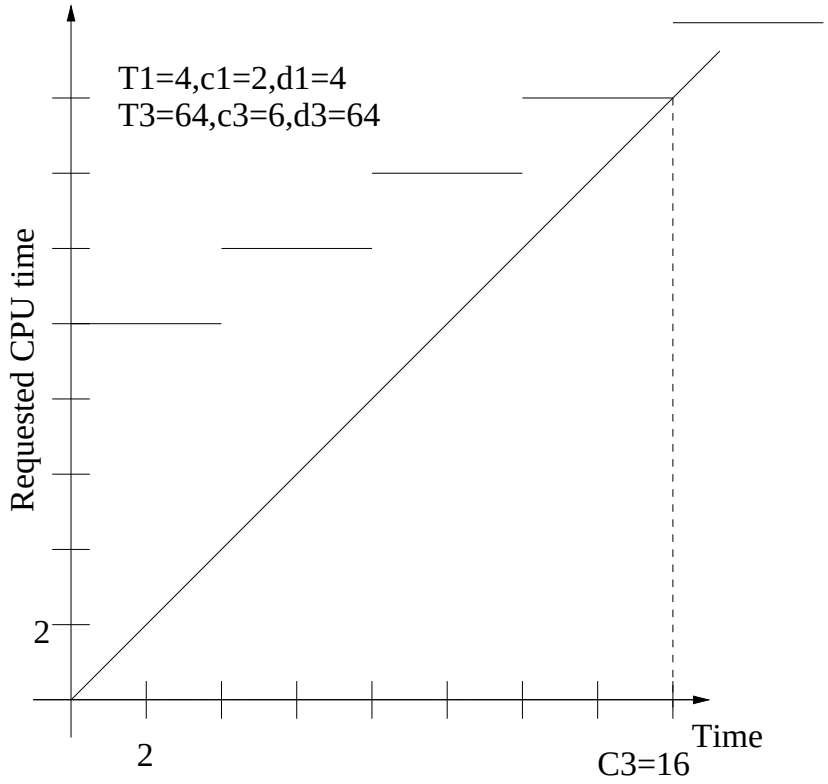

Figure 4: Scheduling 2. priority task in order (1,3,2)

showing τ_3 to complete in due time.

When trying to schedule τ_2 having the lowest priority the result is as shown in figure (5), $C_2 = 16 > d_2 = 12$ implying that τ_2 is not schedulable.

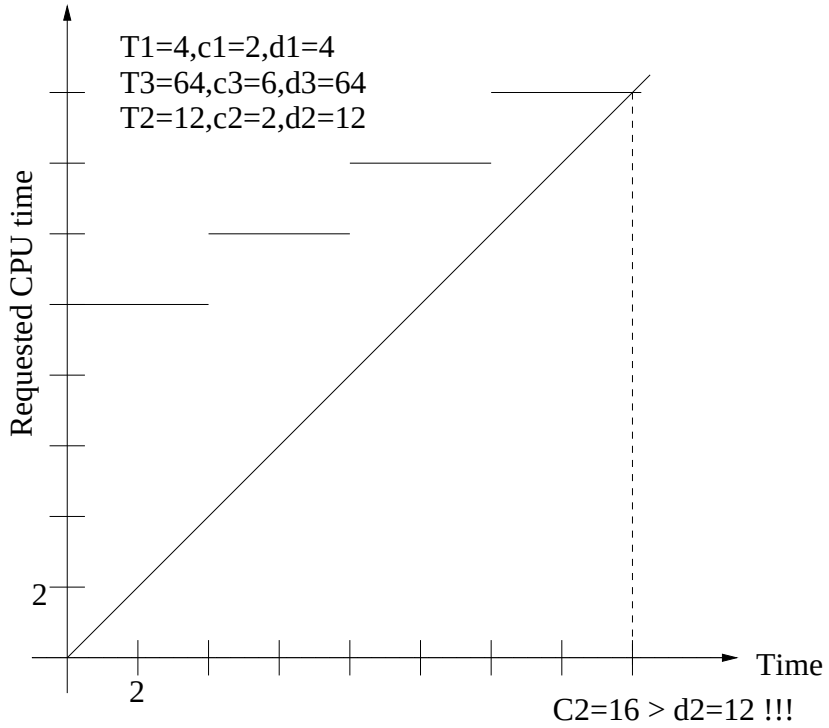

Figure 5: Scheduling lowest priority task in order (1,3,2)

Now for priorities (1,2,3) as according to the RMA rule we get $C_2 = 4$, and $C_3 = 20 < 64$ (figure (6))

The analysis shows the same result as before; priority order (1,3,2) is not schedulable whereas (1,2,3) is.

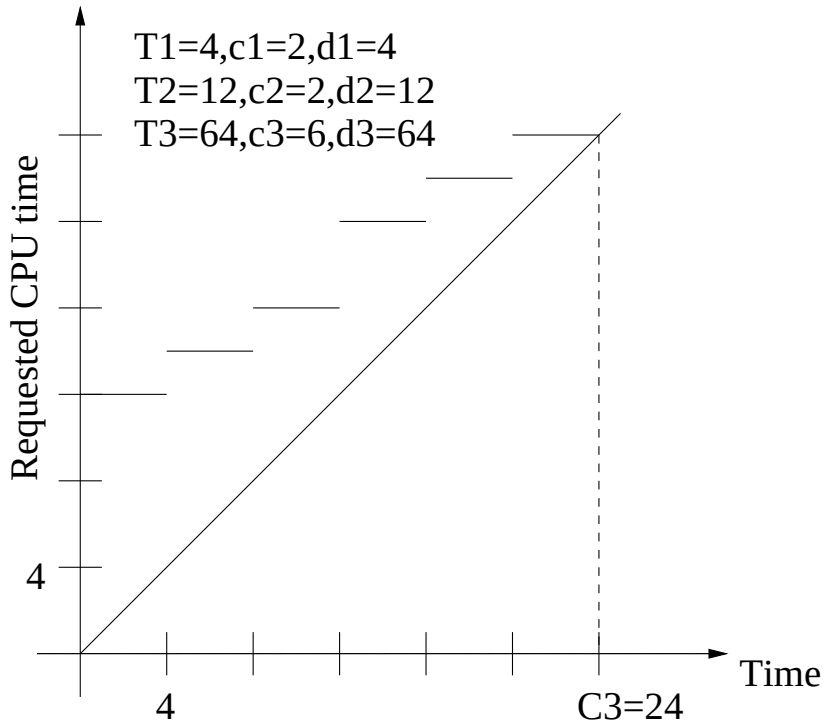

Figure 6: Scheduling lowest priority task in order (1,2,3)

4 Optimal priority assignment

Now we know how to examine feasibility of fixed priority schedules but how to assign priorities ? As we have seen fixed priority scheduling may work for some priority order and not for some other. Is there a priority order that is better than others ? How does it depend on the taskset ? The following theorem establishes a valuable answer

Theorem
For a given taskset $\tau_1, .., \tau_N$ *with deadlines* $d_i \leq T_i$ *and where* $d_i \leq d_{i+1}$ *the Deadline Monotonic (DMA) priority order* $1, 2., .., N$ *(the lower the deadline the higher priority) is optimal in the following sense: If some other priority order is feasible then DMA is also feasible*

Thus when relative deadline are below or equal to periods the DMA priority assignment order is optimal. When deadlines equal periods $d_i = T_i$ the DMA order is exactly the Rate Monotonic order from before. When deadlines exceed periods $d_i > T_i$ we have no general result.

4.1 Proof of DMA optimality (may be omitted)

Now returning to equation 14 we may try to consider some priority order $p_1, .., p_N$ of a general periodic taskset $\tau_1, .., \tau_N$, where relative deadlines are equal to or below periods, i.e. $d_i \leq T_i$

Now assume that the taskset is schedulable with the above order of priorities and for some i and $k < i$:

$$d_i < d_j \quad \forall j \in k, .., i-1 \tag{15}$$

Then we consider another priority order $p_1, .., p_{k-1}, p_k + 1, .., p_{i-1} + 1, p_i - (i-k), p_{i+1}, .., p_N$ corresponding to taskorder $\tau_1, .., \tau_{k-1}, \tau_i, \tau_k, \tau_{k+1}, .., \tau_{i-1}, \tau_{i+1}, ..\tau_N$. Then how about feasibility.

Now for tasks $\tau_1, .., \tau_{k-1}$ the situation is unchanged. Likewise for tasks $\tau_{i+1}, .., \tau_i$. Obviously τ_i is still schedulable since it was assigned a higher priority. Since τ_i was schedulable prior to the priority change we have equation 14 stating that there is a time $C_i \in [0, d_i]$ so that:

$$\begin{aligned} C_i &= c_i + \sum_{j=1}^{i-1} \lceil \frac{C_i}{T_j} \rceil \cdot c_j \\ &= \sum_{j=1}^{i} \lceil \frac{C_i}{T_j} \rceil \cdot c_j \quad \text{since} \quad C_i \leq d_i \leq T_i \\ &\geq \sum_{j=1}^{p-1} \lceil \frac{C_i}{T_j} \rceil \cdot c_j + \lceil \frac{C_i}{T_i} \rceil \cdot c_i \end{aligned} \tag{16}$$

Where to latter inequality is exactly the criterion for τ_p, $p \in k, .., i-1$ to complete before $C_i \leq d_i \leq d_p$ and in turn making τ_p schedulable.

So interchanging priorities by moving tasks with lower relative deadlines ahead of larger ones, preserves schedulability. Such a process can be continued until priorities follow deadline order, i.e. $p_i < p_k \equiv d_i < d_k$ and the proof is completed.

5 Utilization criterion

But how about utilization. Is there an upper limit $\bar{U}$, so that whenever

$$U \leq \bar{U} \tag{17}$$

then the taskset is schedulable by RMA for relative deadlines equal to periods.

Liu and Layland reported a maximum utilization $\bar{U}$ so that when

$$U \leq \bar{U} = N \cdot (2^{\frac{1}{N}} - 1) \tag{18}$$

then the taskset is schedulable by RMA for relative deadlines equal to periods.

The problem is that $\bar{U}_N$ approaches $log(2) = 0.69$ for large N. That is for large tasksets we may experience a utilization as low as 0.7 for the taskset to be schedulable.

Likewise even though $U > \bar{U}$ the taskset may be schedulable with RMA and $T_i = d_i$. This bound may be rather conservative.

6 Non pre emptive systems

As for cyclic executives and round robin scheduling one may imagine a non preemptive fixed priority scheduler. For hard real time tasks a criterion similar to inequality 14 may be stated

$$\begin{aligned} t_{\text{idle}} &= c_k + \sum_{j=1}^{i-1} \lceil \frac{t_{\text{idle}}}{T_j} \rceil \cdot c_j \\ C_{i,j} &= c_i + t_{\text{idle}} \\ k &= argmax\{c_j \mid j > i\} \end{aligned} \tag{19}$$

Where t_{idle} is the first instant following a critical one where the CPU is idle w.r.t. to higher priority tasks and therefore the instant where $J_{i,j}$ is scheduled. Obviously every task has to wait for the completion of any lower priority task τ_k. Completion of $J_{i,j}$ is acheived at $C_{i,j} = c_i + t_{\text{idle}}$ due to nonpreemptiveness.

7 Context switching

In any real time platform the switch between different tasks take a *hopefully* small amount of time which nevertherless has to be taken into account w.r.t. feasibility, especially when the system is heavily loaded. Such a time is called a context switch and is assumed allways to last T_C time units.

In fixed priority scheduling, context switching happens only when jobs are issued and completed. In pre-emptive systems higher priority tasks with low periods may pre-empt lower priority tasks multiple times each time introducing 2 context switches. One for pre-emption and one for resumption. Thus context switching may for pre-emptive fixed priority scheduling be accounted for by adding $2 \cdot T_C$ to all computation times c_i.

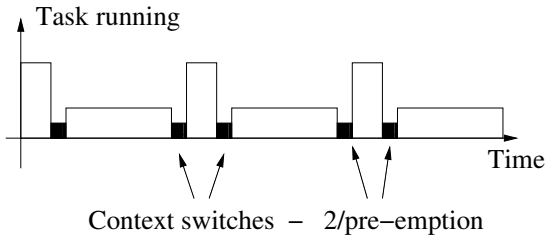

Figure 7: Context switching under preemptive fixed priority scheduling

In non preemptive fixed priority scheduling context switching is allways associated to scheduling the next job in line. Thus context switching only happens when jobs are scheduled. In this case context switching is accounted for by adding T_C to each computation time.

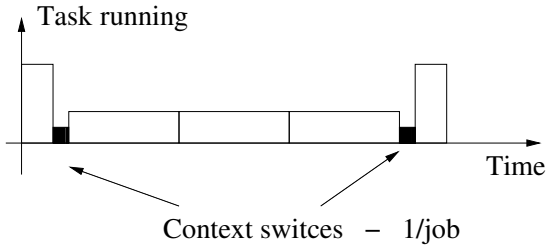

Figure 8: Context switching under nonpreemptive fixed priority scheduling

8 Summary

- Critical instants appear when all tasks launch jobs simultaneously.
- Only critical instants need to be analysed by completion time analysis.
- DMA priority assignment is optimal.

- RMA is optimal when relative deadlines equal periods.
- A utilization based criterion exists but is conservative.
- Maximally acheivable utilization may be as low as 0.7.
- Context switching may be taken into account in pre emptive and non preemptive cases.