SCHEDULING

1. An algorithm for ordering the use of system resources (in particular the CPUs)

2. A means of predicting the worst-case behaviour of the system when the scheduling algorithm is applied.

3. Scheduling schemes:
   - Static
   - Dynamic

Simple process model

By process we mean task here

1. The application is assumed to consist of a fixed set of processes.

2. All processes are periodic, with known periods.

3. The processes are completely independent of each other.

4. All system's overheads, context switching times and so on are ignored (that is, assumed to have zero cost).

5. All processes have a deadline equal to their period (that is, each process must complete before it is next released).

6. All processes have a fixed worst-case execution time.

Nomenclature

Notation	Description
B	Worst-case blocking time
C	Worst-case computation time of the process
D	Deadline of the process
J	Release jitter of a process
N	Number of processes in the system
P	Priority assigned to the process (if applicable)
R	Worst-case response time of the process
T	Minimum time between process releases
U	The utilization of each process (equal to C/T)

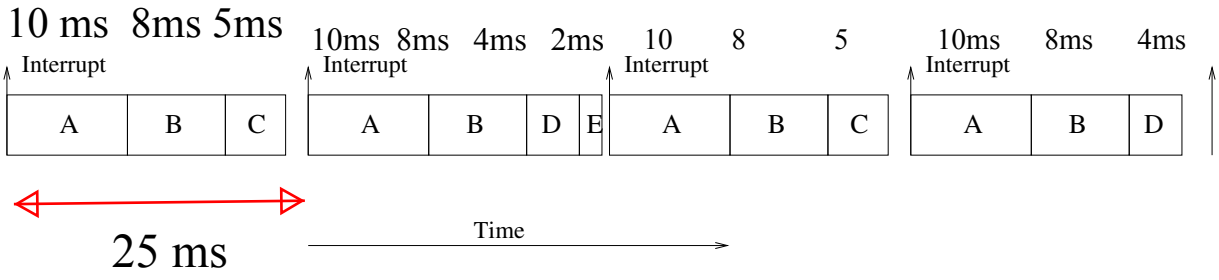
The cyclic executive approach

- Minor Cycle (e.g. 25ms)

- Major Cycle (e.g. 200ms): 4 slots of 25ms each

Construction of a cyclic executive is equivalent to bin packing

Process	Period, T	Computation Time, C
A	25	10
B	25	8
C	50	5
D	50	4
E	100	2




```
count =0;
while (1){
        wait for interrupt (25 ms)
        switch(count++){
            case 0:
                    A();
                    B();
                    C();
                    break;
            case 1:
                    A(); B(); D(); E();
                    break;
            case 2:
                    A(); B(); C();
                    break;
            case 3:
                    A(); B(); D(); break;
            case 4:
                    count =0;
                    break;
                }
        }
```

```
loop
   Wait_For_Interrupt;
   Procedure_For_A;
   Procedure_For_B;
   Procedure_For_C;
   Wait_For_Interrupt;
   Procedure_For_A;
   Procedure_For_B;
   Procedure_For_D;
   Procedure_For_E;
   Wait_For_Interrupt;
   Procedure_For_A;
   Procedure_For_B;
   Procedure_For_C;
   Wait_For_Interrupt;
   Procedure_For_A;
   Procedure_For_B;
   Procedure_For_D;
end loop;
```

Wait for interrupt:

==> ISR sets a register inside CPU equal to 1

(ISR arrives at 25 ms intervals)

==> Code: ldr r0, #1

loop jnz loop

What if the following is used in C and ISR sets a=0?

```
a= 1;
.....
while (a==1);
```

Note

1. No actual processes exist at run-time; each minor cycle is just a sequence of procedure calls

2. The procedures share a common address space and can thus pass data between themselves. This data does not need to be protected (via a semaphore, for example) because concurrent access is not possible

3. All ‘process’ periods must be a multiple of the minor cycle time.

Disadvantages

- The difficulty of incorporating sporadic processes
- The difficulty of incorporating processes with long periods; the major cycle time is the maximum period that can be accommodated without secondary schedules
- The difficulty of actually constructing the cyclic executive
- Any ‘process’ with a sizeable computation time will need to be split into a fixed number of fixed sized procedures

Process-based scheduling

Process states

- runnable
- suspended waiting for a timing event – appropriate for periodic processes
- suspended waiting for a non-timing event – appropriate for sporadic processes

The runnable processes are executed in the order determined by their priority.

In real-time systems, the ‘priority’ of a process is derived from its temporal requirements, not its importance or integrity.

Rate monotonic priority assignment

Each process is assigned a (unique) priority based on its period; the shorter the period, the higher the priority (that is, $T_i < T_j \Rightarrow P_i > P_j$).

Process	Period, T	Priority, P
A	25	5
B	60	3
C	42	4
D	105	1
E	75	2

Recap: Task model is as follows
* Periodic
* Independent
* Negligible overhead in context switches
* Deadline = period
* Constant worst case execution time

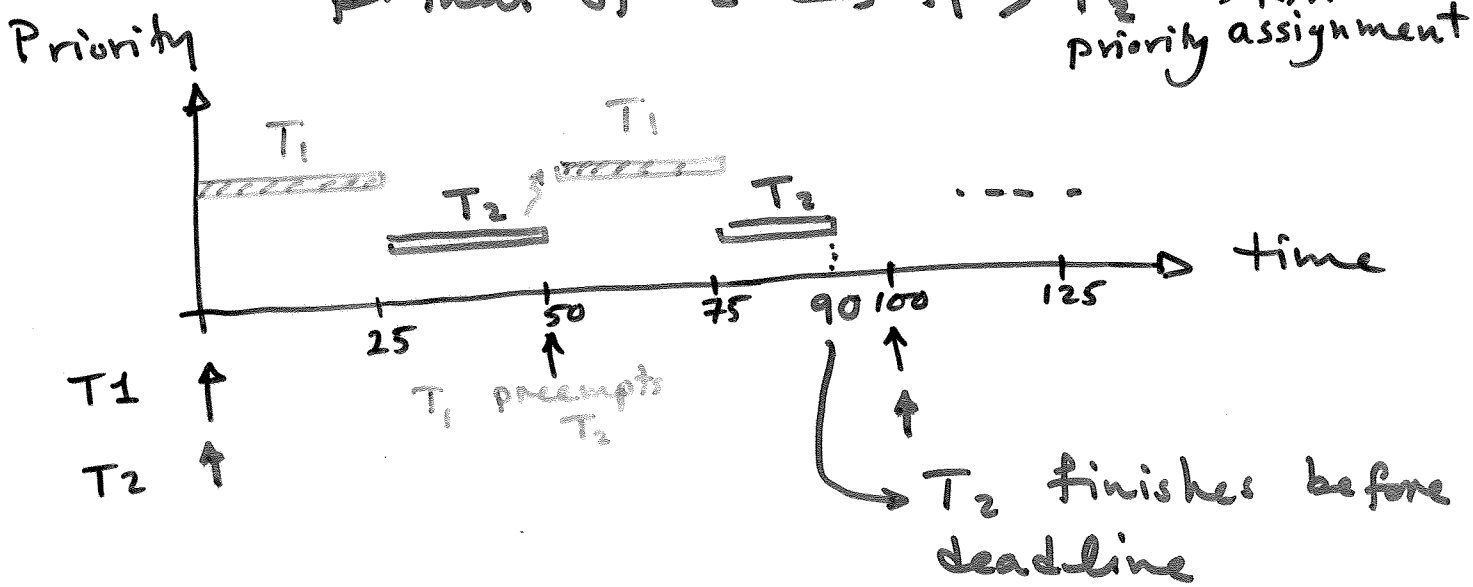
* The importance of a scheduling algorithm

Example Consider tasks 1 & 2 as follows

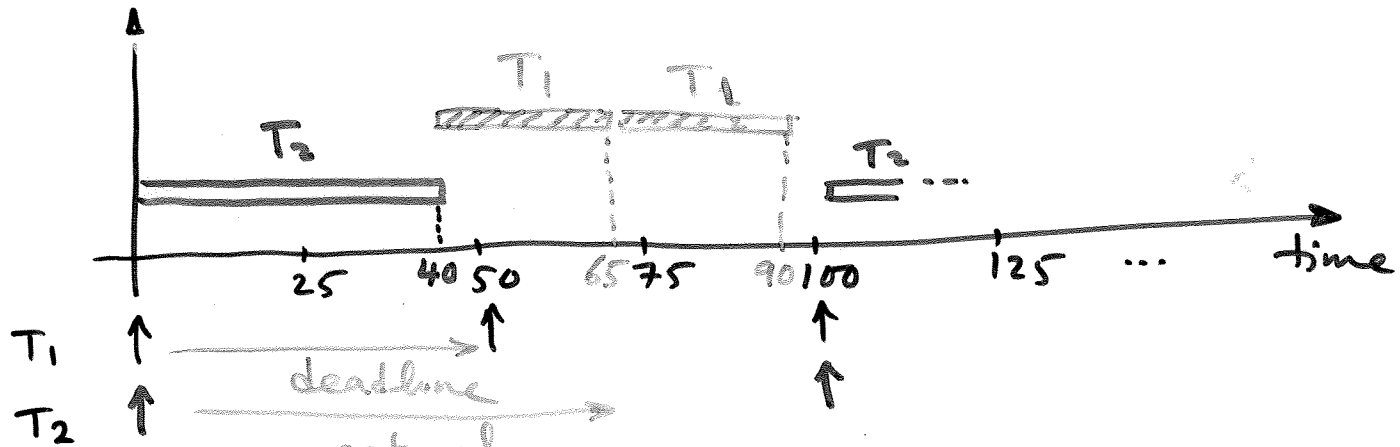
Task	Period	Computation time
1	50 ms	25 ms
2	100 ms	40 ms

(deadline = period)

Case A Assume priority of 1 is greater that that of 2 → $P_1 > P_2$ → RM priority assignment



CASE B Reverse priorities, i.e. $P_1 < P_2$



Utilization-based schedulability tests (RMA)

If $$\sum_{i=1}^{N} \left(\frac{C_i}{T_i}\right) < N(2^{1/N} - 1)$$

Then all tasks meet their deadlines.

N	Utilization Bound
1	100.0%
2	82.8%
3	78.0%
4	75.7%
5	74.3%
10	71.8%

In the limit: 69.3%

** If tasks meet their deadlines we cannot conclude that the above inequality is valid (may or may not be valid)

** RMA is optimal for fixed priorities, i.e., if a task set cannot be scheduled using RMA, it cannot be scheduled using other arrangement of fixed priorities

	Period T	Computation Time, C	Priority P	Utilization U
Task_1	50	12	1	0.24
Task_2	40	10	2	0.25
Task_3	30	10	3	0.33

highest priority

$$\sum_{i=1}^{3} \frac{C_i}{T_i} = \frac{12}{50} + \frac{10}{40} + \frac{10}{30} = 0.82$$

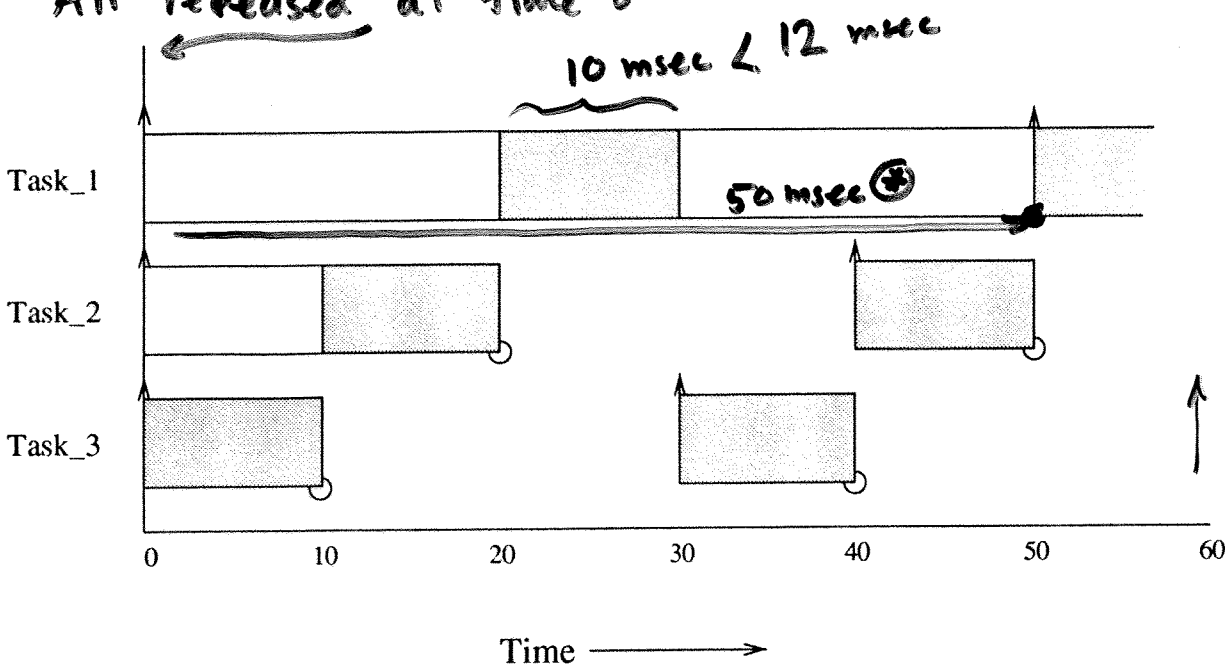
$$3(2^{1/3} - 1) = 0.78$$

$0.82 \not< 0.78$ → fails RMA utilization test

* Can we conclude that the task set is NOT schedulable?

let's see if the deadlines are met.

All released at time 0



Process Release Time

Process Completion Time - Deadline met

Deadline Missed

Executing

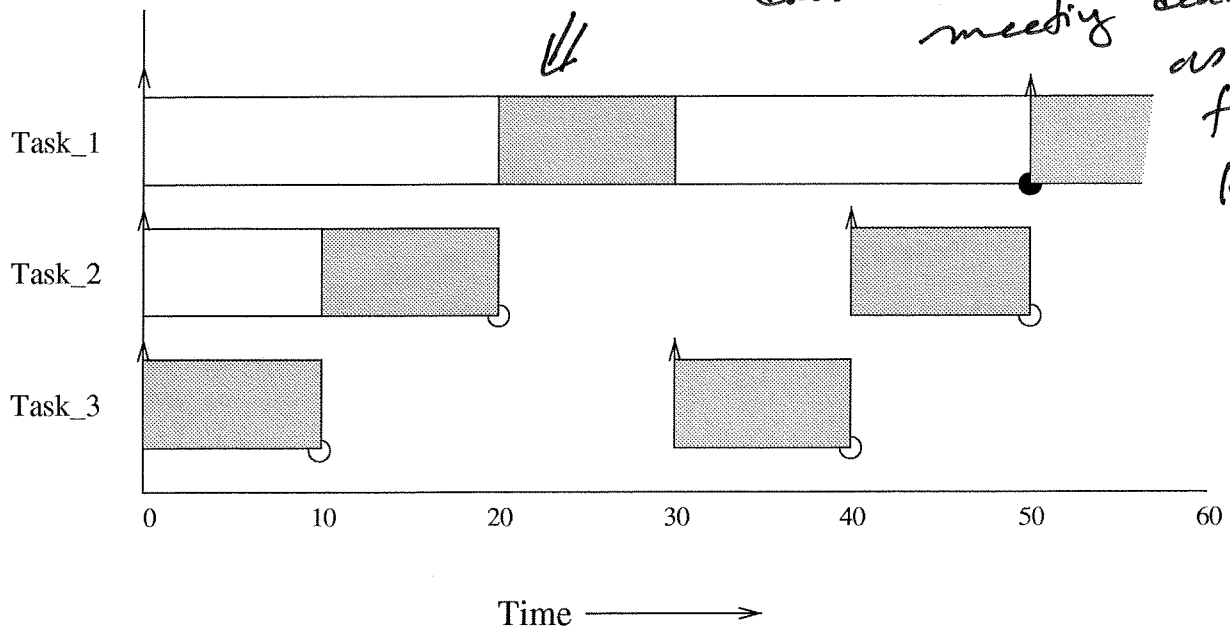
Preempted

* Task-1 misses its deadline
(another 2 msec remaining)

NOTE: We cannot conclude from the test that the set is not schedulable using RMA.

→ HAVE TO VERIFY IT BY OTHER METHODS

changing priorities does not guarantee meeting deadlines as it fails RMA



Process Release Time

Process Completion Time - Deadline met

Deadline Missed

Executing

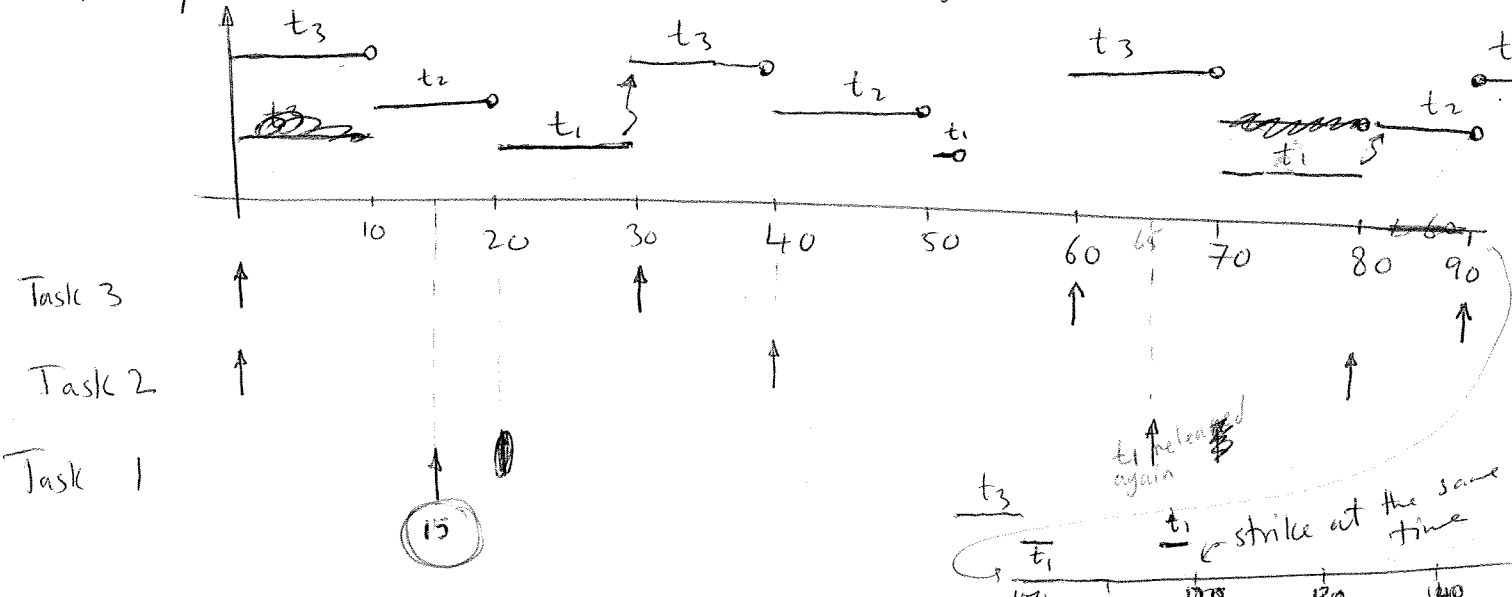
Preempted

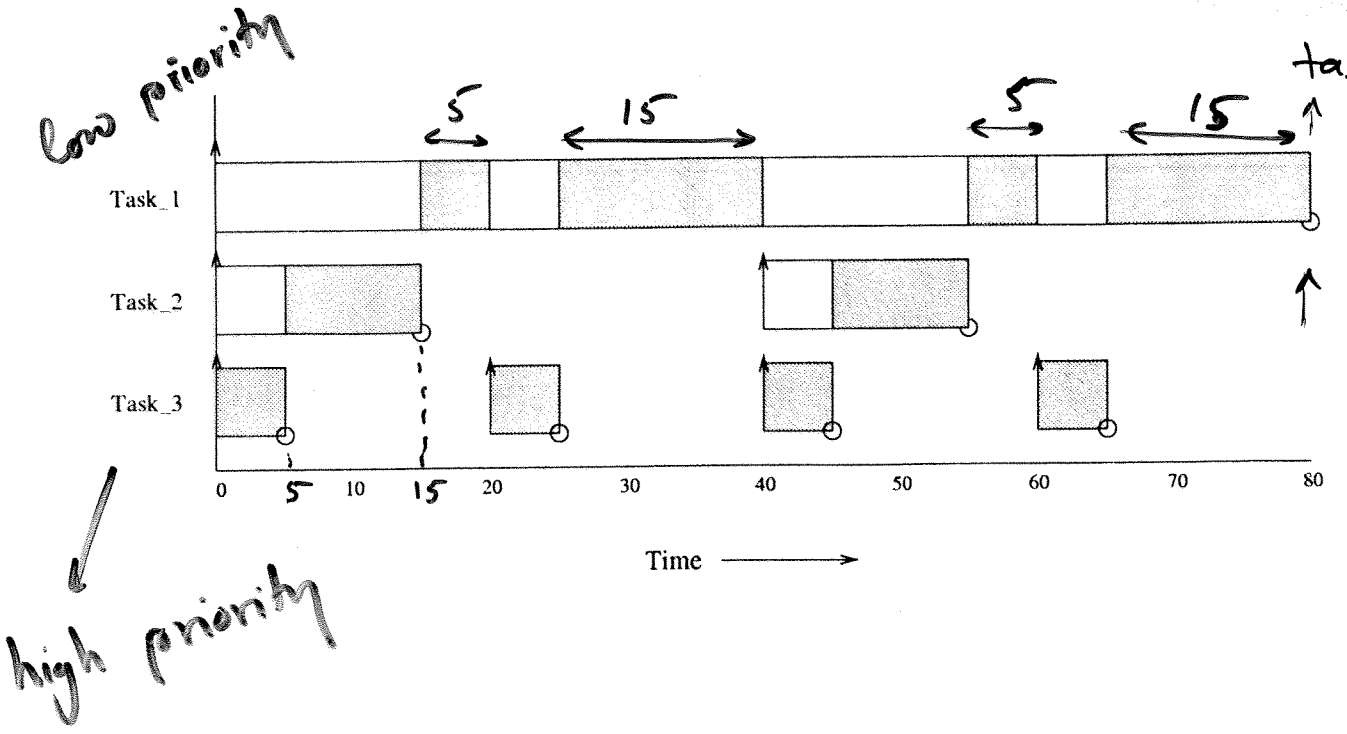
Assignment

Check what happens if Task 1 is released at time 15 e.g. 15, 20

meets deadline

priority





* How far must the time-line be drawn?

* (Liu & Layland, 1973)

For task sets that share a common release time, a time-line equal to the size of the longest period is sufficient.

i.e. If tasks meet their 1st deadline they'll meet all future deadline

	Period T	Computation Time, C	Priority P	Utilization U
Task_1	80	40	1	0.50
Task_2	40	10	2	0.25
Task_3	20	5	3	0.25

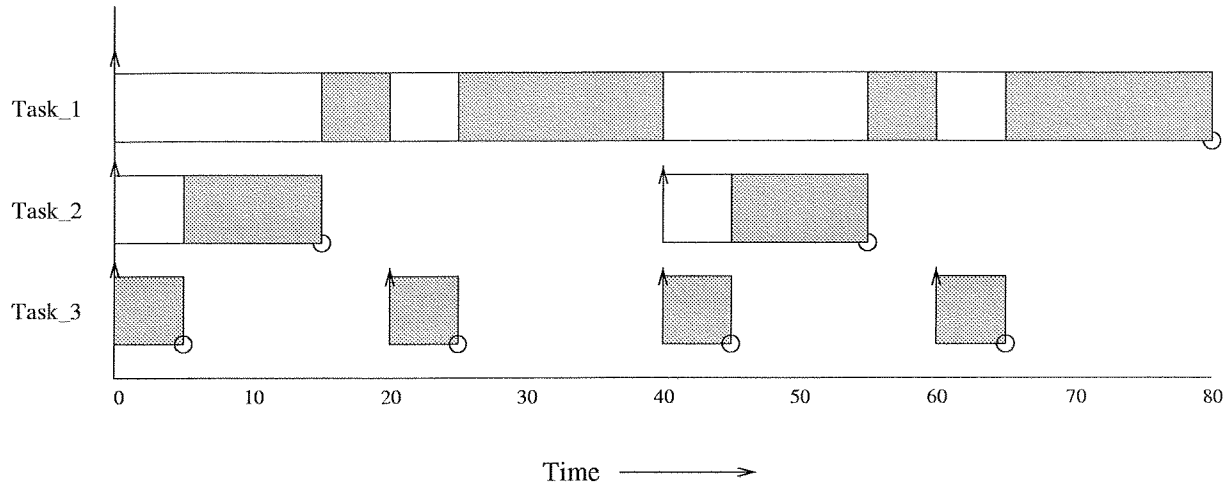
$$\sum \frac{C_i}{T_i} = \frac{40}{80} + \frac{10}{40} + \frac{5}{20} = 1 \not\leq 3(2^{1/3}-1)$$

∴ Fails the RM test!

However, the tasks meet their deadlines (using a time-line analysis)

↓

(See Figure ~~P18~~)



To achieve 100% utilization when using fixed priorities

Assign periods so that all tasks are harmonic

e.g. 10, 20, 40 are preferred over 10, 20, 50

	Period T	Computation Time, C	Priority P	Utilization U
Task_1	80	32	1	0.400
Task_2	40	5	2	0.125
Task_3	16	4	3	0.250

$$\sum \frac{C_i}{T_i} = \frac{32}{80} + \frac{5}{40} + \frac{4}{16} = 0.4 + 0.125 + 0.25$$
$$= 0.775$$
$$< (2^{1/3} - 1) \times 3 = 0.78$$

Guaranteed to meet its deadlines ⇐

(Draw figure)