THE UNIVERSITY *of York*

Department of Computer Science

RTS York

Chapter 11: Scheduling Real-Time Systems

Aims

- To understand the role that scheduling and schedulability analysis plays in predicting that real-time applications meet their deadlines
- To understand the cyclic executive approach and its limitations
- To introduce process-based scheduling and distinguish between the various approaches available

Aims

- To introduce Rate Monotonic priority assignment
- To introduce utilization-based schedulability test
- To show why utilization-based necessary are sufficient but not necessary
- To provide and introduction to response time analysis for Fixed Priority scheduling
- Proof of Deadline Monotonic Priority Ordering optimality

Aims

- To understand how response time analysis can be extended to cope with blocking of resources
- To explain priority inversion, priority inheritance and ceiling protocols
- To show that response-time analysis is flexible enough to cope with many application demands by simple extensions
- To cover Earliest Deadline First (EDF) scheduling

Aims

- To briefly consider dynamic scheduling
- To provide an overview of the issues related to Worst-Case Execution Time (WCET) analysis
- Consideration of multiprocessor scheduling
- Consideration of power-aware scheduling
- System overhead estimation

Scheduling

- In general, a scheduling scheme provides two features:
  - An algorithm for ordering the use of system resources (in particular the CPUs)
  - A means of predicting the worst-case behaviour of the system when the scheduling algorithm is applied
- The prediction can then be used to confirm the temporal requirements of the application

Cyclic Executives

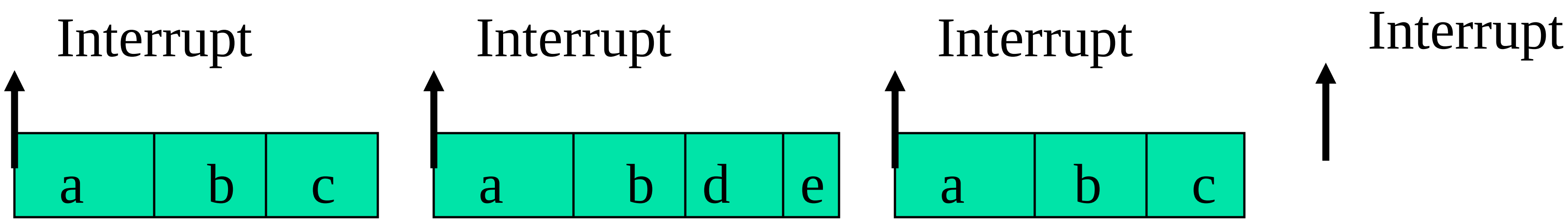
- One common way of implementing hard real-time systems is to use a cyclic executive
- Here the design is concurrent but the code is produced as a collection of procedures
- Procedures are mapped onto a set of minor cycles that constitute the complete schedule (or major cycle)
- Minor cycle dictates the minimum cycle time
- Major cycle dictates the maximum cycle time

Has the advantage of being fully deterministic

Consider Task Set

Task	Period, T	Computation Time, C
a	25	10
b	25	8
c	50	5
d	50	4
e	100	2

Time-line for Task Set



Properties

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

Problems with Cyclic Exec.

- All "*task*" periods must be a multiple of the minor cycle time
- The difficulty of incorporating tasks with long periods; the major cycle time is the maximum period that can be accommodated without secondary schedules
- Sporadic activities are difficult (impossible!) to incorporate
- The cyclic executive is difficult to construct and difficult to maintain — it is a NP-hard problem

Problems with Cyclic Exec.

- Any "task" with a sizable computation time will need to be split into a fixed number of fixed sized procedures (this may cut across the structure of the code from a software engineering perspective, and hence may be error-prone)
- More flexible scheduling methods are difficult to support
- Determinism is not required, but predictability is

Task-Based Scheduling

- Tasks exist at run-time
  - Supported by real-time OS or run-time
- Each task is:
  - Runnable (and possible running), or
  - Suspended waiting for a timing event
  - Suspended waiting for a non-timing event

Task-Based Scheduling

- Scheduling approaches
  - Fixed-Priority Scheduling (FPS)
  - Earliest Deadline First (EDF)
  - Value-Based Scheduling (VBS)

Fixed-Priority Scheduling (FPS)

- This is the most widely used approach and is the main focus of this course
- Each task has a fixed, static, priority which is computer pre-run-time
- The runnable tasks are executed in the order determined by their priority
- In real-time systems, the "priority" of a task is derived from its temporal requirements, not its importance to the correct functioning of the system or its integrity

Earliest Deadline First (EDF)

- The runnable tasks are executed in the order determined by the absolute deadlines of the tasks
- The next task to run being the one with the shortest (nearest) deadline
- Although it is usual to know the relative deadlines of each task (e.g. 25ms after release), the absolute deadlines are computed at run time and hence the scheme is described as dynamic

Value-Based Scheduling (VBS)

- If a system can become overloaded then the use of simple static priorities or deadlines is not sufficient; a more adaptive scheme is needed
- This often takes the form of assigning a value to each task and employing an on-line value-based scheduling algorithm to decide which task to run next

Preemption

- With priority-based scheduling, a high-priority task may be released during the execution of a lower priority one
- In a preemptive scheme, there will be an immediate switch to the higher-priority task
- With non-preemption, the lower-priority task will be allowed to complete before the other executes
- Preemptive schemes enable higher-priority tasks to be more reactive, and hence they are preferred
- Cooperative dispatching (deferred preemption) is a half-way house
- Schemes such as EDF and VBS can also take on a pre-emptive or non pre-emptive form

Scheduling Characteristics

- Sufficient – pass the test will meet deadlines
- Necessary – fail the test will miss deadlines

- Exact – necessary and sufficient

- Sustainable – system stays schedulable if conditions ‘improve’

Simple Task Model

- The application is assumed to consist of a fixed set of tasks
- All tasks are periodic, with known periods
- The tasks are completely independent of each other
- All system's overheads, context-switching times and so on are ignored (i.e, assumed to have zero cost)
- All tasks have a deadline equal to their period (that is, each task must complete before it is next released)
- All tasks have a fixed worst-case execution time

Standard Notation

`B`	Worst-case blocking time for the task (if applicable)
`C`	Worst-case computation time (WCET) of the task
`D`	Deadline of the task
`I`	The interference time of the task
`J`	Release jitter of the task
`N`	Number of tasks in the system
`P`	Priority assigned to the task (if applicable)
`R`	Worst-case response time of the task
`T`	Minimum time between task releases, jobs, (task period)
`U`	The utilization of each task (equal to C/T)
`a-z`	The name of a task

Rate Monotonic Priority Assignment

- Each task is assigned a (unique) priority based on its period; the shorter the period, the higher the priority
- i.e, for two tasks `i` and `j`,

$$T_i < T_j \Rightarrow P_i > P_j$$

- This assignment is optimal in the sense that if any task set can be scheduled (using pre-emptive priority-based scheduling) with a fixed-priority assignment scheme, then the given task set can also be scheduled with a rate monotonic assignment scheme
- Note, priority 1 is the lowest (least) priority

Example Priority Assignment

Process	Period, T	Priority, P
a	25	5
b	60	3
c	42	4
d	105	1
e	75	2

Utilization-Based Analysis

- For D=T task sets only
- A simple sufficient but not necessary schedulability test exists

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N(2^{1/N} - 1)$$

$$U \leq 0.69 \text{ as } N \rightarrow \infty$$

Utilization Bounds

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

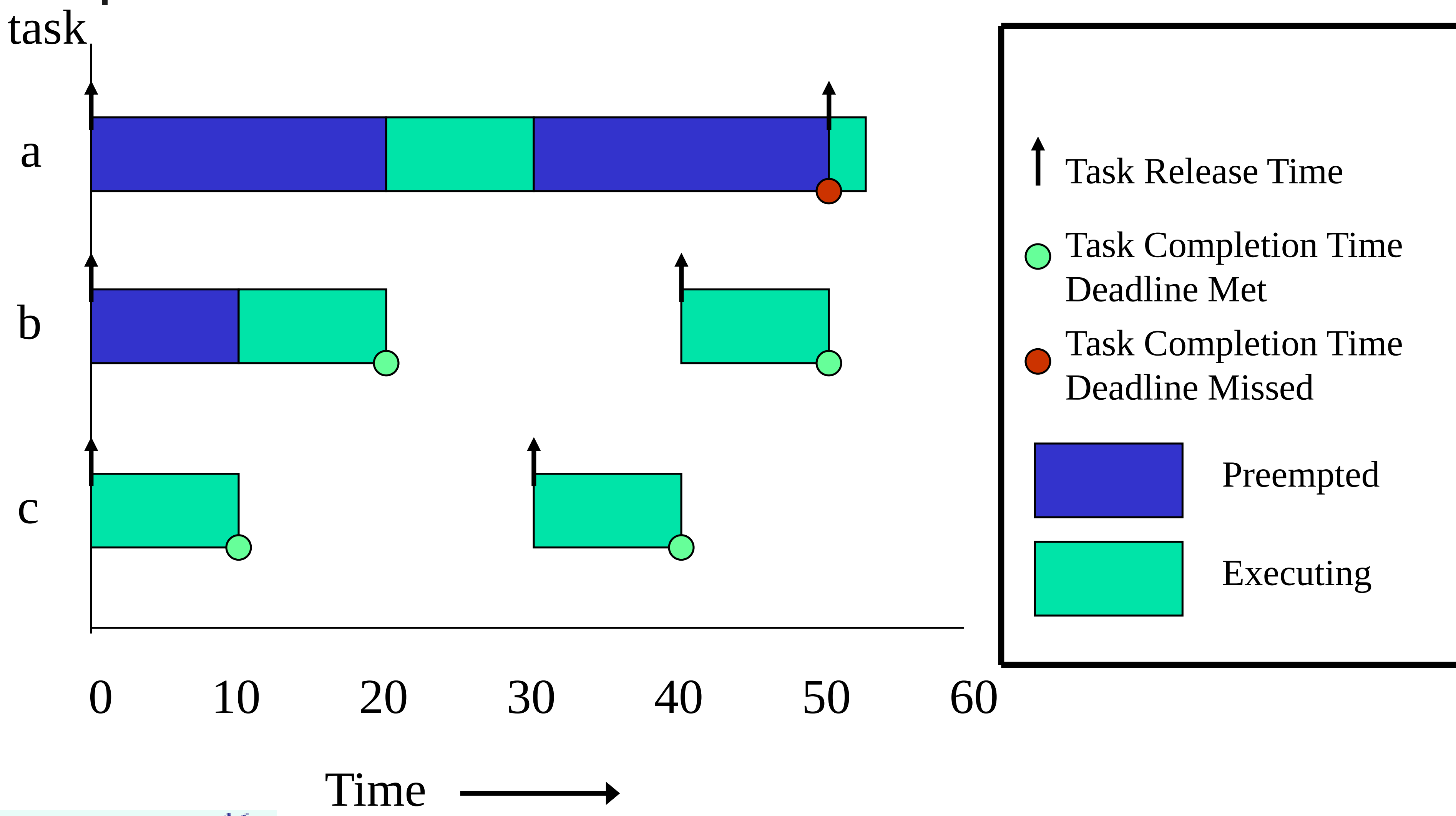
Approaches 69.3% asymptotically

Task Set A

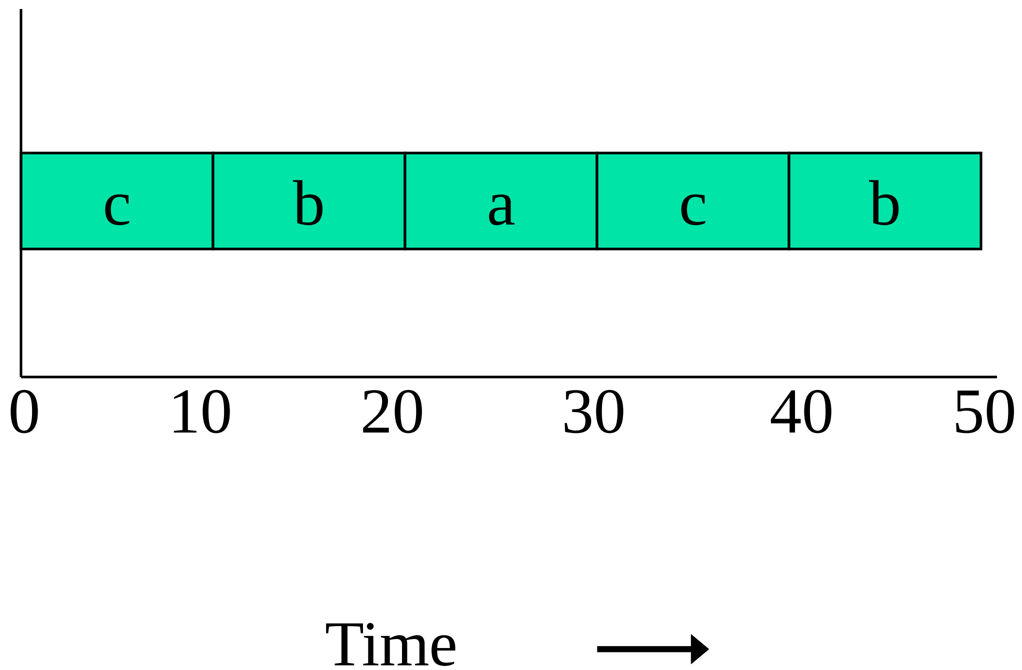
Task	Period T	ComputationTime C	Priority P	Utilization U
a	50	12	1	0.24
b	40	10	2	0.25
c	30	10	3	0.33

- The combined utilization is 0.82 (or 82%)
- This is above the threshold for three tasks (0.78) and, hence, this task set fails the utilization test

Time-line for task Set A



Gantt Chart for Task Set A



Task Set B

Task	Period T	ComputationTime C	Priority P	Utilization U
a	80	32	1	0.400
b	40	5	2	0.125
c	16	4	3	0.250

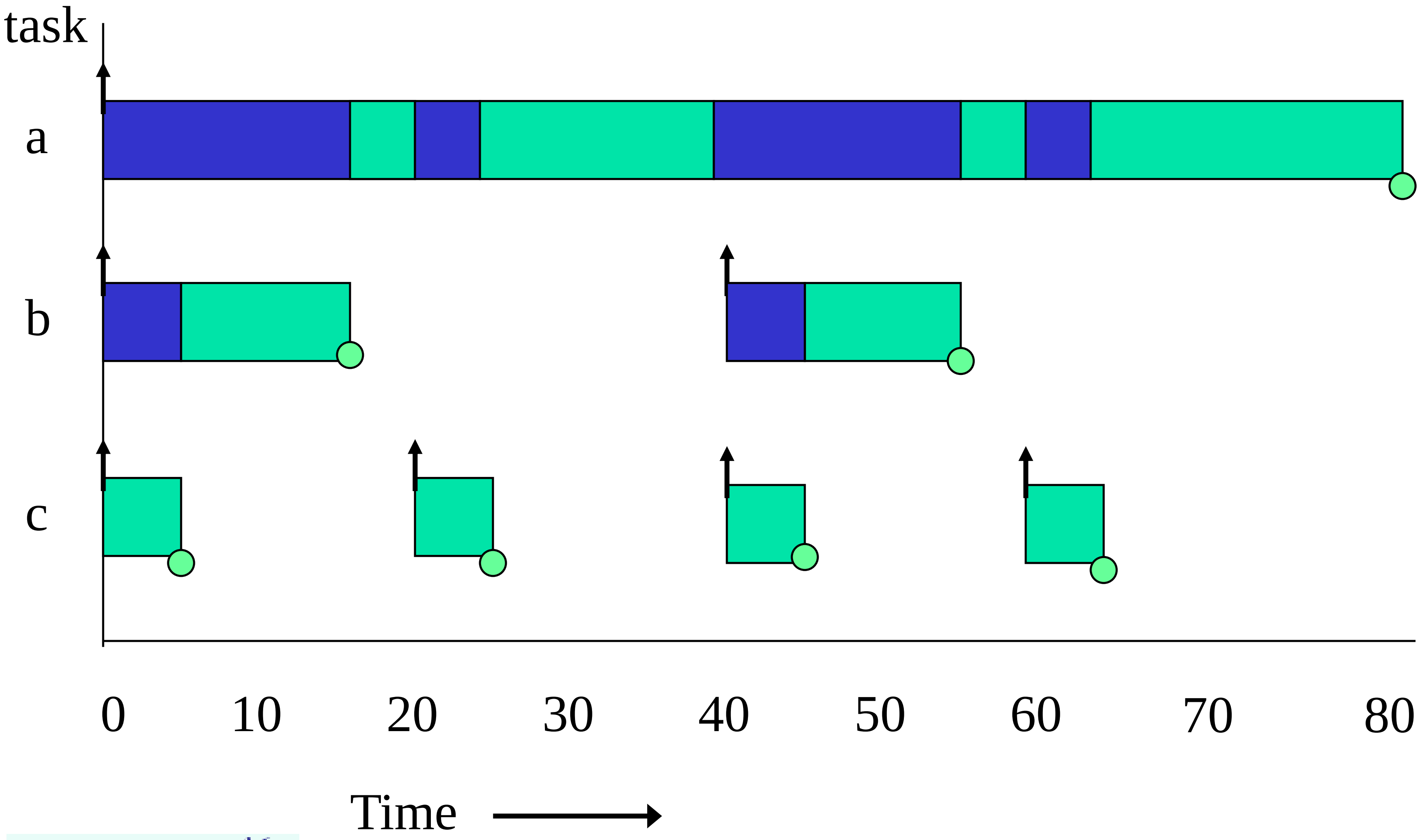
- The combined utilization is 0.775
- This is below the threshold for three tasks (0.78) and, hence, this task set will meet all its deadlines

Task Set C

Task	Period T	ComputationTime C	Priority P	Utilization U
a	80	40	1	0.50
b	40	10	2	0.25
c	20	5	3	0.25

- The combined utilization is 1.0
- This is above the threshold for three tasks (0.78) but the task set will meet all its deadlines

Time-line for Task Set C



Improved Tests - I

- Use families of task (N stands for families), so
  - In Task Set B, let period of Task c change to 14
  - Now utilisation is approximately 0.81
  - Above L&L threshold for 3 tasks
  - But period of a is same as period of b, so N=2
  - L&L bound for N=2 is 0.828

Improved Tests - II

- Alternative formulae

$$\prod_{i=1}^{N}\left(\frac{C_i}{T_i}+1\right)\leq 2$$

Criticism of Tests

- Not exact
- Not general
- BUT it is O(N)

The test is sufficient but not necessary

Response-Time Analysis

- Here task i's worst-case response time, R, is calculated first and then checked (trivially) with its deadline

$$R_i \leq D_i$$

$$R_i = C_i + I_i$$

Where I is the interference from higher priority tasks

Calculating R

During R, each higher priority task j will execute a number of times:

$$\text{Number of Releases} = \left\lceil \frac{R_i}{T_j} \right\rceil$$

The ceiling function $\lceil \ \rceil$ gives the smallest integer greater than the fractional number on which it acts. So the ceiling of 1/3 is 1, of 6/5 is 2, and of 6/3 is 2.

Total interference is given by:

$$\left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Response Time Equation

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Where $hp(i)$ is the set of tasks with priority higher than task i

Solve by forming a recurrence relationship:

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

The set of values $w_i^0, w_i^1, w_i^2, \ldots, w_i^n, \ldots$ is monotonically non decreasing. When $w_i^n = w_i^{n+1}$ the solution to the equation has been found; w_i^0 must not be greater that R_i (e.g. 0 or C_i)

Response Time Algorithm

```
for i in 1..N loop -- for each process in turn
  n := 0
  w_i^n := C_i
loop
    calculate new w_i^{n+1}
    if  w_i^{n+1} = w_i^n  then
      R_i = w_i^n
      exit value found
    end if
    if  w_i^{n+1} > T_i  then
      exit value not found
    end if
    n := n + 1
  end loop
end loop
```

Where: $w_i^n := C_i$; calculate new w_i^{n+1}; if $w_i^{n+1} = w_i^n$ then $R_i = w_i^n$; if $w_i^{n+1} > T_i$ then exit value not found.

Task Set D

Task	Period T	ComputationTime C	Priority P
a	7	3	3
b	12	3	2
c	20	5	1

$$R_a = 3$$

$$w_b^0 = 3$$

$$w_b^1 = 3 + \left\lceil \frac{3}{7} \right\rceil 3 = 6$$

$$w_b^2 = 3 + \left\lceil \frac{6}{7} \right\rceil 3 = 6$$

$$R_b = 6$$

$$w_c^0 = 5$$

$$w_c^1 = 5 + \left\lceil \frac{5}{7} \right\rceil 3 + \left\lceil \frac{5}{12} \right\rceil 3 = 11$$

$$w_c^2 = 5 + \left\lceil \frac{11}{7} \right\rceil 3 + \left\lceil \frac{11}{12} \right\rceil 3 = 14$$

$$w_c^3 = 5 + \left\lceil \frac{14}{7} \right\rceil 3 + \left\lceil \frac{14}{12} \right\rceil 3 = 17$$

$$w_c^4 = 5 + \left\lceil \frac{17}{7} \right\rceil 3 + \left\lceil \frac{17}{12} \right\rceil 3 = 20$$

$$w_c^5 = 5 + \left\lceil \frac{20}{7} \right\rceil 3 + \left\lceil \frac{20}{12} \right\rceil 3 = 20$$

$$R_c = 20$$

Revisit: Task Set C

Process	Period T	ComputationTime C	Priority P	Response time R
a	80	40	1	80
b	40	10	2	15
c	20	5	3	5

- The combined utilization is 1.0
- This was above the utilization threshold for three tasks (0.78), therefore it failed the test
- The response time analysis shows that the task set will meet all its deadlines

Response Time Analysis

- Is sufficient and necessary (exact)
- If the task set passes the test they will meet all their deadlines; if they fail the test then, at run-time, a task will miss its deadline (unless the computation time estimations themselves turn out to be pessimistic)

Sporadic Tasks

- Sporadics tasks have a minimum inter-arrival time
- They also require D<T
- The response time algorithm for fixed priority scheduling works perfectly for values of D less than T as long as the stopping criteria becomes

$$W_i^{n+1} > D_i$$

- It also works perfectly well with any priority ordering — hp(i) always gives the set of higher-priority tasks

Hard and Soft Tasks

- In many situations the worst-case figures for sporadic tasks are considerably higher than the averages
- Interrupts often arrive in bursts and an abnormal sensor reading may lead to significant additional computation
- Measuring schedulability with worst-case figures may lead to very low processor utilizations being observed in the actual running system

General Guidelines

Rule 1 — all tasks should be schedulable using average execution times and average arrival rates

Rule 2 — all hard real-time tasks should be schedulable using worst-case execution times and worst-case arrival rates of all tasks (including soft)

- A consequent of Rule 1 is that there may be situations in which it is not possible to meet all current deadlines
- This condition is known as a transient overload
- Rule 2 ensures that no hard real-time task will miss its deadline
- If Rule 2 gives rise to unacceptably low utilizations for "normal execution" then action must be taken to reduce the worst-case execution times (or arrival rates)

Aperiodic Tasks

- These do not have minimum inter-arrival times
- Can run aperiodic tasks at a priority below the priorities assigned to hard processes, therefore, they cannot steal, in a pre-emptive system, resources from the hard processes
- This does not provide adequate support to soft tasks which will often miss their deadlines
- To improve the situation for soft tasks, a server can be employed

Execution-time Servers

- A server:
  - Has a capacity/budget of C that is available to its client tasks (typically aperiodic tasks)
  - When a client runs it uses up the budget
  - The server has a replenishment policy
  - If there is currently no budget then clients do not run
  - Hence it protects other tasks from excessive aperiodic activity

Periodic Server (PS)

- Budget C
- Replenishment Period T, starting at say 0
- Client ready to run at time 0 (or T, 2T etc) runs while budget available, is then suspended
- Budget ‘idles away’ if no clients

Deferrable Server (DS)

- Budget C
- Period T – replenished every T time units (back to C)
  - For example 10ms every 50ms
- Anytime budget available clients can execute
- Client suspended when budget exhausted
- DS and SS are referred to as *bandwidth preserving*
  - Retain capacity as long as possible
- PS is not bandwidth preserving

Sporadic Server (SS)

- Initially defined to enforce minimum separation for sporadic tasks
- Parameters C and T
- Request at time t (for a < C) is accepted
  - a is returned to server at time t+T
- Request at time t (for 2C>A>C):
  - C available immediately
  - Replenished at time t+T
  - Remainder (2C-A) available at this time

Task Sets with D < T

- For D = T, Rate Monotonic priority ordering is optimal
- For D < T, Deadline Monotonic priority ordering is optimal

$$D_i < D_j \Rightarrow P_i > P_j$$

D < T Example Task Set

Task	Period T	Deadline D	ComputationTime C	Priority P	Response time R
a	20	5	3	4	3
b	15	7	3	3	6
c	10	10	4	2	10
d	20	20	3	1	20

Proof that DMPO is Optimal

- Deadline monotonic priority ordering (DMPO) is optimal if any task set, `Q`, that is schedulable by priority scheme, `W`, is also schedulable by DMPO

- The proof of optimality of DMPO involves transforming the priorities of `Q` (as assigned by `W`) until the ordering is DMPO
- Each step of the transformation will preserve schedulability

DMPO Proof Continued

- Let `i` and `j` be two tasks (with adjacent priorities) in `Q` such that under `W`: $P_i > P_j \wedge D_i > D_j$
- Define scheme `W'` to be identical to `W` except that tasks `i` and `j` are swapped

Consider the schedulability of `Q` under `W'`

- All tasks with priorities greater than P_i will be unaffected by this change to lower-priority tasks
- All tasks with priorities lower than P_j will be unaffected; they will all experience the same interference from `i` and `j`
- Task `j`, which was schedulable under `W`, now has a higher priority, suffers less interference, and hence must be schedulable under `W'`

DMPO Proof Continued

- All that is left is the need to show that task i, which has had its priority lowered, is still schedulable
- Under `W`

$$R_j < D_j, D_j < D_i \; and \; D_i \le T_i$$

- Hence task i only interferes once during the execution of j
- It follows that:

$$R'_i = R_j \le D_j < D_i$$

- It can be concluded that task `i` is schedulable after the switch
- Priority scheme `W'` can now be transformed to `W"` by choosing two more tasks that are in the wrong order for DMP and switching them

Task Interactions and Blocking

- If a task is suspended waiting for a lower-priority task to complete some required computation then the priority model is, in some sense, being undermined

- It is said to suffer priority inversion

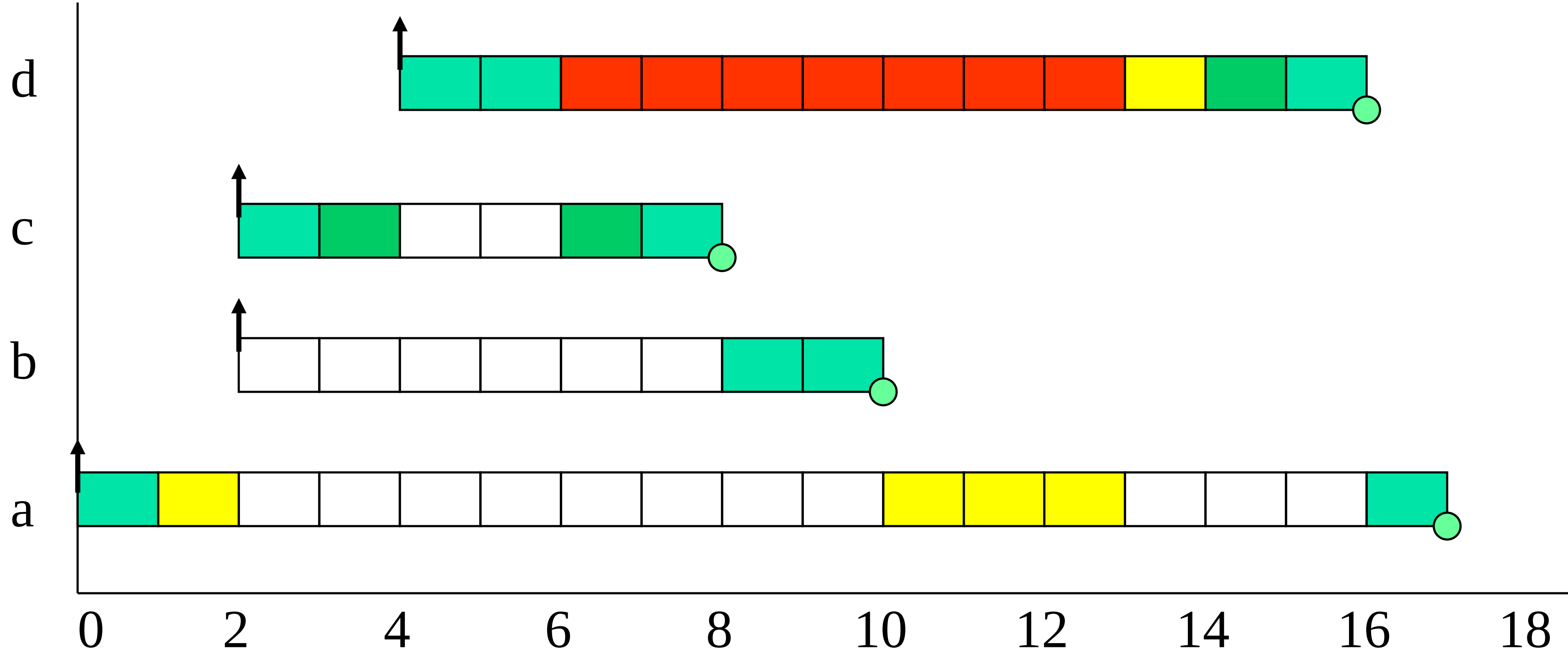
- If a task is waiting for a lower-priority task, it is said to be blocked

Priority Inversion

- To illustrate an extreme example of priority inversion, consider the executions of four periodic tasks: `a`, `b`, `c` and `d`; and two resources: `Q` and `V`

Task	Priority	Execution Sequence	Release Time
a	1	EQQQQE	0
b	2	EE	2
c	3	EVVE	2
d	4	EEQVE	4

Example of Priority Inversion

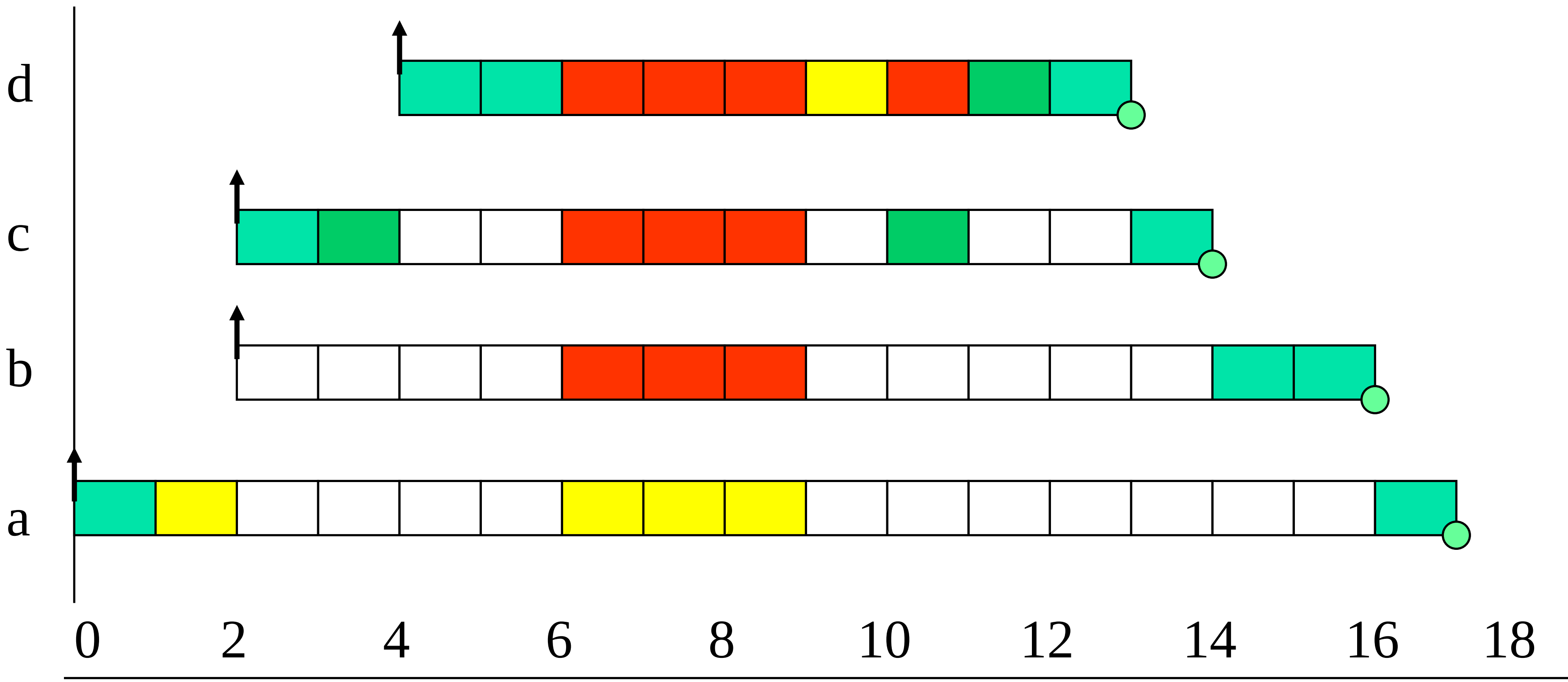


Symbols

■ (cyan)	Executing	□	Preempted
■ (yellow)	Executing with Q locked	■ (red)	Blocked
■ (green)	Executing with V locked		

Priority Inheritance

- If task `p` is blocking task `q`, then `q` runs with `p`'s priority



Mars Path-Finder

- A problem due to priority inversion nearly caused the lose of the Mars Path-finder mission

- As a shared bus got heavily loaded critical data was not been transferred
- Time-out on this data was used as an indication of failure and lead to re-boot

- Solution was a patch that turned on priority inheritance, this solved the problem

Calculating Blocking

- If a task has m critical sections that can lead to it being blocked then the maximum number of times it can be blocked is m
- If B is the maximum blocking time and K is the number of critical sections, then task i has an upper bound on its blocking given by:

$$B_i = \sum_{k=1}^{K} usage(k,i)C(k)$$

Response Time and Blocking

$$R_i = C_i + B_i + I_i$$

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$$w_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

Priority Ceiling Protocols

Two forms

- Original ceiling priority protocol
- Immediate ceiling priority protocol

On a Single Processor

- A high-priority task can be blocked at most once during its execution by lower-priority tasks
- Deadlocks are prevented
- Transitive blocking is prevented
- Mutual exclusive access to resources is ensured (by the protocol itself)

OCPP

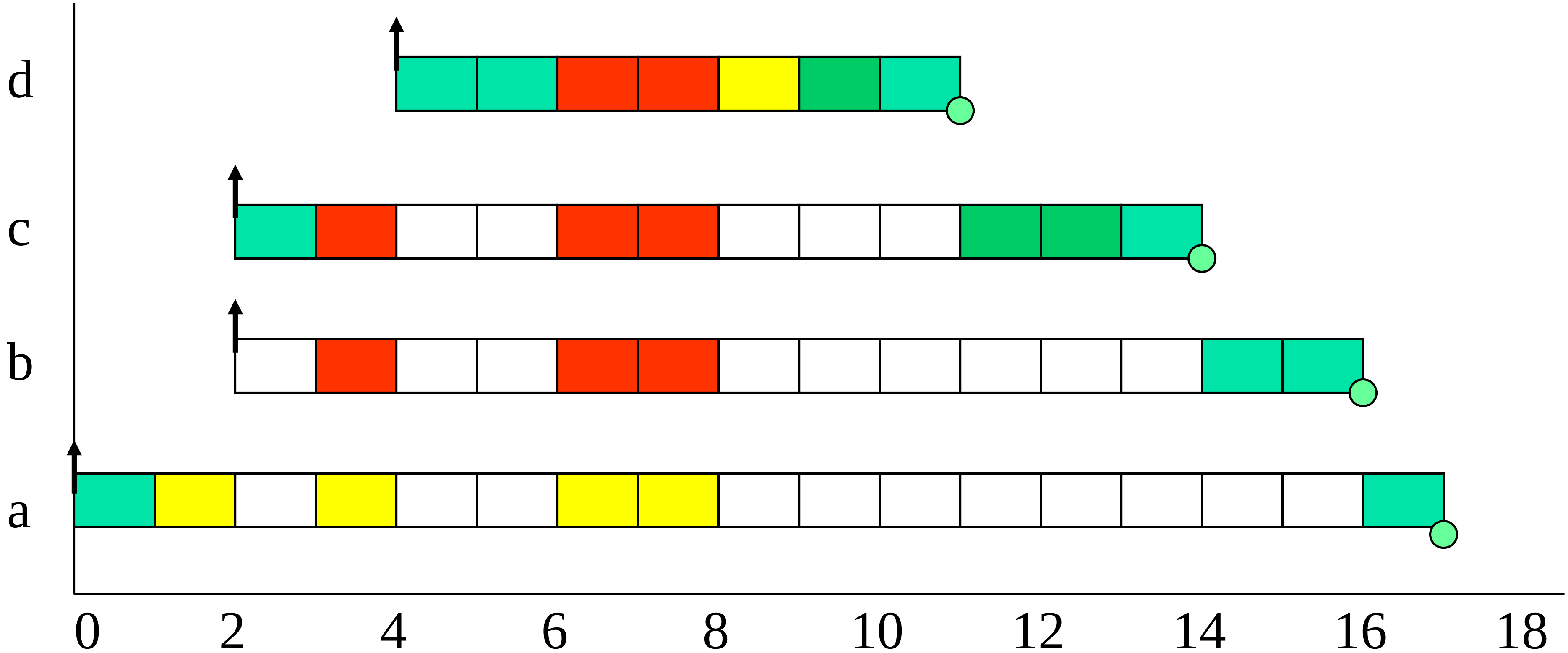
- Each task has a static default priority assigned (perhaps by the deadline monotonic scheme)
- Each resource has a static ceiling value defined, this is the maximum priority of the tasks that use it
- A task has a dynamic priority that is the maximum of its own static priority and any it inherits due to it blocking higher-priority tasks
- A task can only lock a resource if its dynamic priority is higher than the ceiling of any currently locked resource (excluding any that it has already locked itself)

OCPP Analysis

$$B_i = \max_{k=1}^{k} usage(k,i)C(k)$$

- Even though blocking has a 'single term' impact, good design practice is to keen all critical sections small

OCPP Inheritance



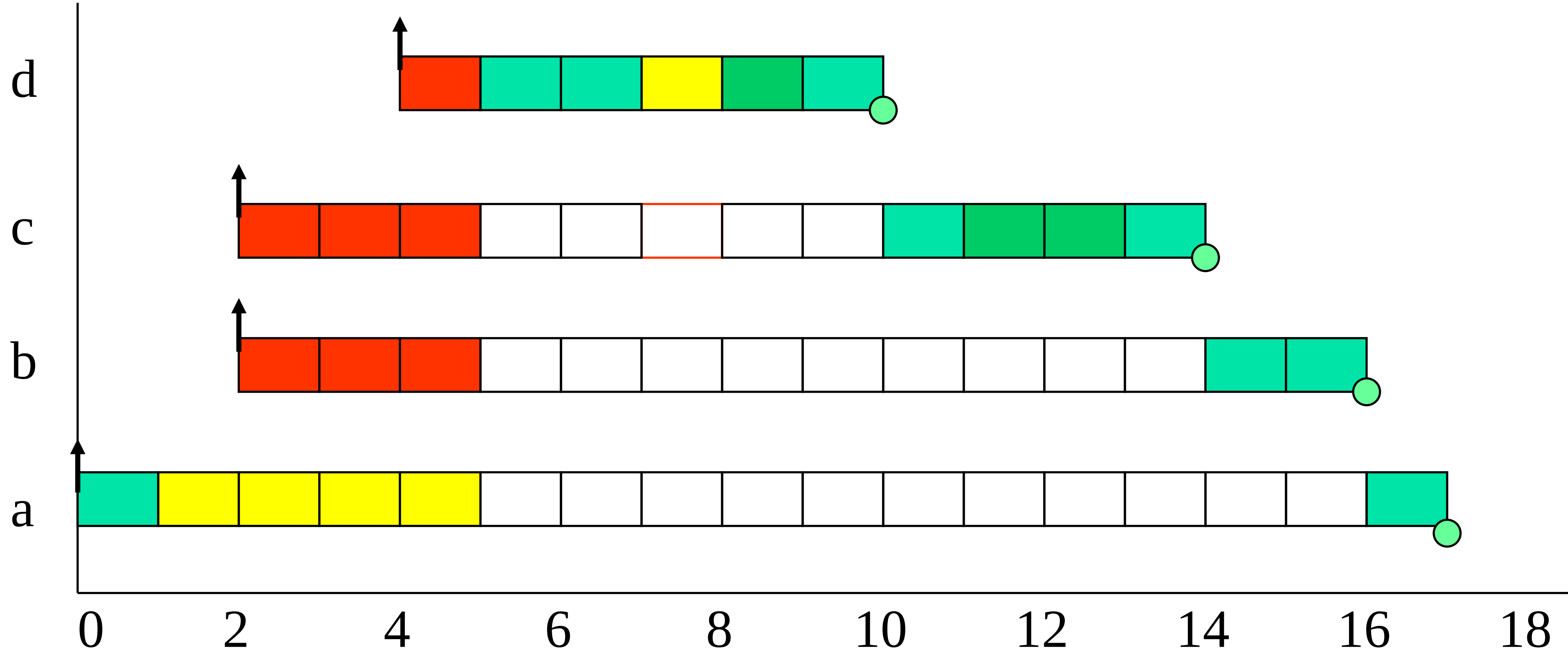
ICPP

- Each task has a static default priority assigned (perhaps by the deadline monotonic scheme)
- Each resource has a static ceiling value defined, this is the maximum priority of the tasks that use it
- A task has a dynamic priority that is the maximum of its own static priority and the ceiling values of any resources it has locked

ICPP - Properties

- As a consequence of ICPP, a task will only suffer a block at the very beginning of its execution
- Once the task starts actually executing, all the resources it needs must be free; if they were not, then some task would have an equal or higher priority and the task's execution would be postponed

ICPP Inheritance



OCPP versus ICPP

- Although the worst-case behaviour of the two ceiling schemes is identical (from a scheduling view point), there are some points of difference:
  - ICCP is easier to implement than the original (OCPP) as blocking relationships need not be monitored
  - ICPP leads to less context switches as blocking is prior to first execution
  - ICPP requires more priority movements as this happens with all resource usage
  - OCPP changes priority only if an actual block has occurred

OCPP versus ICPP

- Note that ICPP is called Priority Protect Protocol in POSIX and Priority Ceiling Emulation in Real-Time Java

An Extendible Task Model

So far:

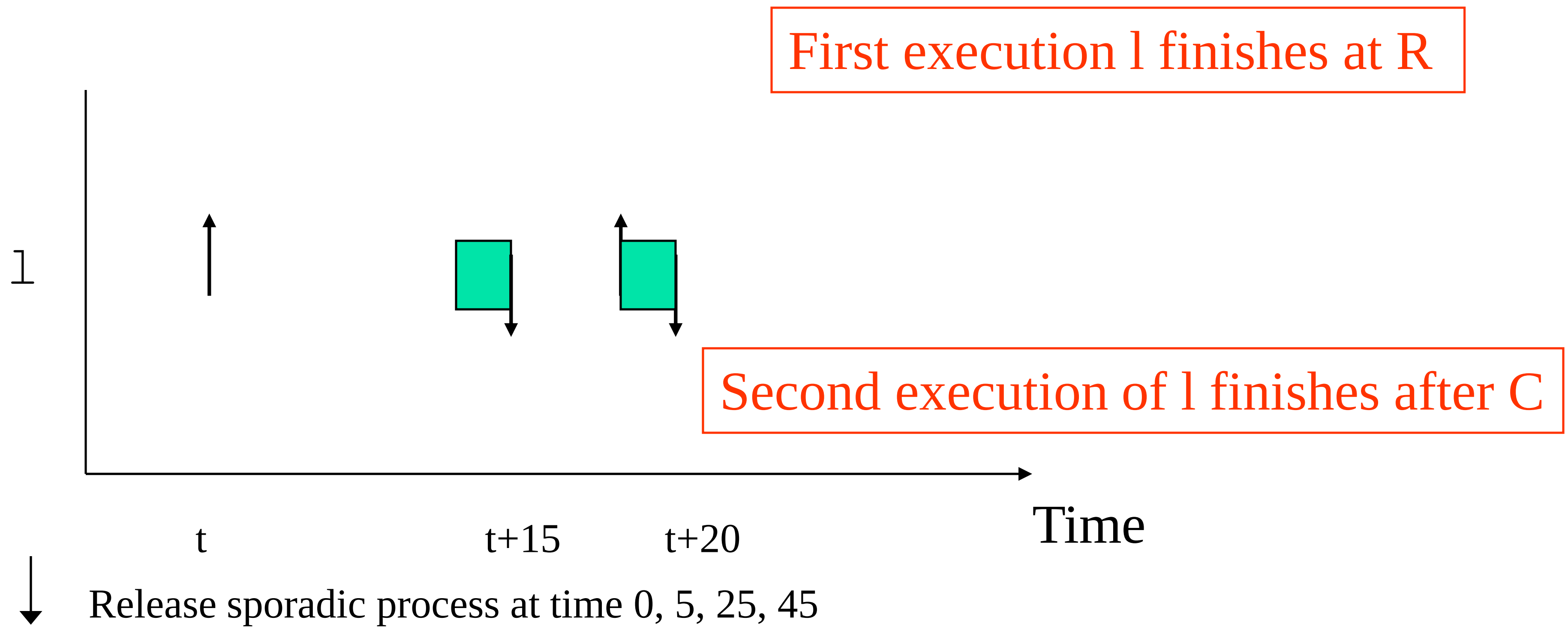
- Deadlines can be less than period (D<T)
- Sporadic and aperiodic tasks, as well as periodic tasks, can be supported
- Task interactions are possible, with the resulting blocking being factored into the response time equations

Extensions

- Release Jitter
- Arbitrary Deadlines
- Cooperative Scheduling
- Fault Tolerance
- Offsets
- Optimal Priority Assignment
- Execution-time Servers

Release Jitter

- A key issue for distributed systems
- Consider the release of a sporadic task on a different processor by a periodic task, `l`, with a period of 20



Release Jitter

- Sporadic is released at `0, T-J, 2T-J, 3T-J`
- Examination of the derivation of the schedulability equation implies that task `i` will suffer
  - one interference from task `s` if $R_i \in [0, T-J)$
  - two interferences if $R_i \in [T-J, 2T-J)$
  - three interference if $R_i \in [2T-J, 3T-J)$
- This can be represented in the response time equations
- If response time is to be measured relative to the real release time then the jitter value must be added

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i + J_j}{T_j} \right\rceil C_j \qquad R_i^{periodic} = R_i + J_i$$

Arbitrary Deadlines

- To cater for situations where D (and hence potentially R) > T

$$w_i^{n+1}(q) = B_i + (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n(q)}{T_j} \right\rceil C_j$$

$$R_i(q) = w_i^n(q) - qT_i$$

- The number of releases is bounded by the lowest value of q for which the following relation is true: $R_i(q) \le T_i$
- The worst-case response time is then the maximum value found for each q:

$$R_i = \max_{q=0,1,2,\ldots} R_i(q)$$

Arbitrary Deadlines

- When formulation is combined with the effect of release jitter, two alterations to the above analysis must be made
- First, the interference factor must be increased if any higher priority processes suffers release jitter:

$$w_i^{n+1}(q) = B_i + (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n(q) + J_j}{T_j} \right\rceil C_j$$

- The other change involves the task itself. If it can suffer release jitter then two consecutive windows could overlap if response time plus jitter is greater than period

$$R_i(q) = w_i^n(q) - qT_i + J_i$$

Cooperative Scheduling

- True preemptive behaviour is not always acceptable for safety-critical systems
- Cooperative or deferred preemption splits tasks into slots
- Mutual exclusion is via non-preemption
- The use of deferred preemption has two important advantages
  - It increases the schedulability of the system, and it can lead to lower values of C
  - With deferred preemption, no interference can occur during the last slot of execution

Cooperative Scheduling

- Let the execution time of the final block be F_i

$$w_i^{n+1} = B_{MAX} + C_i - F_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

- When this converges that is, $w_i^n = w_i^{n+1}$, the response time is given by: $R_i = w_i^n + F_i$

BUT - example

- First task: T=D=6, C=2, F=2 (ie one block)
- Second task: T=D=8,C=6, F=3 (2 blocks)
- Is this schedulable?
- What is utilisation of this task set?

Fault Tolerance

- Fault tolerance via either forward or backward error recovery always results in extra computation
- This could be an exception handler or a recovery block
- In a real-time fault tolerant system, deadlines should still be met even when a certain level of faults occur
- This level of fault tolerance is know as the fault model
- If the extra computation time that results from an error in task `i` is C_i^f

$$R_i = C_i + B_i + \sum_{j \in hep(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \max_{k \in hep(i)} C_k^f$$

- where `hep(i)` is set of tasks with priority equal to or higher than `i`

Fault Tolerance

- If `F` is the number of faults allowed

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \max_{k \in hep(i)} F C_k^f$$

- If there is a minimum arrival interval T_f

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \max_{k \in hep(i)} \left(\left\lceil \frac{R_i}{T_f} \right\rceil C_k^f \right)$$

Offsets

- So far assumed all tasks share a common release time (critical instant)

Task	T	D	C	R
a	8	5	4	4
b	20	10	4	8
c	20	12	4	16

- With offsets

Task	T	D	C	O	R
a	8	5	4	0	4
b	20	10	4	0	8
c	20	12	4	10	8

Arbitrary offsets are not amenable to analysis

Non-Optimal Analysis

- In most realistic systems, task periods are not arbitrary but are likely to be related to one another
- As in the example just illustrated, two tasks have a common period. In these situations it is ease to give one an offset (of `T/2`) and to analyse the resulting system using a transformation technique that removes the offset — and, hence, critical instant analysis applies
- In the example, tasks `b` and `c` (having the offset of 10) are replaced by a single notional process with period 10, computation time 4, deadline 10 but no offset

Non-Optimal Analysis

Process	T	D	C	O	R
a	8	5	4	0	4
n	10	10	4	0	8

Non-Optimal Analysis

- This notional task has two important properties:
  - If it is schedulable (when sharing a critical instant with all other tasks) then the two real tasks will meet their deadlines when one is given the half period offset
  - If all lower priority tasks are schedulable when suffering interference from the notional task (and all other high-priority tasks) then they will remain schedulable when the notional task is replaced by the two real tasks (one with the offset)
- These properties follow from the observation that the notional task always uses more (or equal) CPU time than the two real tasks

Notional Task Parameters

$$T_n = \frac{T_a}{2} = \frac{T_b}{2}$$

$$C_n = Max(C_a, C_b)$$

$$D_n = Min(D_a, D_b)$$

$$P_n = Max(P_a, P_b)$$

Can be extended to more than two processes

Other Requirements

- Mode changes
- Task where C values are variable
  - Optimal solution not possible for FPS
- Where there are gaps in the execution behaviour
  - Optimal solution not possible for FPS or EDF
- Communication protocols
  - For example CAN
- Dual priority

Priority Assignment

Theorem (Audsley' algorithm)

- If task p is assigned the lowest priority and is feasible then, if a feasible priority ordering exists for the complete task set, an ordering exists with task p assigned the lowest priority

Priority Assignment

```
procedure Assign_Pri (Set : in out Process_Set; N : Natural;
                      Ok : out Boolean) is
begin
  for K in 1..N loop
    for Next in K..N loop
      Swap(Set, K, Next);
      Process_Test(Set, K, Ok);
      exit when Ok;
    end loop;
    exit when not Ok;  -- failed to find a schedulable process
  end loop;
end Assign_Pri;
```

Insufficient Priorities

- If insufficient priorities then tasks must share priority levels
- If task *a* shares priority with task *b*, then each must assume the other interferes
- Priority assignment algorithm can be used to pack tasks together
- Ada requires 31, RT-POSIX 32 and RT-Java 28

Execution-time Servers - FPS

- Periodic Servers act as periodic tasks
- Sporadic Servers act as periodic tasks
  - Are directly supported by POSIX
- Deferrable Servers worst case behaviour is when
  - Budget available but not used until end of replenishment period
  - Budget is used, replenished and used again (ie 2C in one go)
  - Analysed as task with release jitter

EDF Scheduling

- Always run task with earliest absolute deadline
- Will consider
  - Utilisation based tests
  - Processor demand criteria
  - QPA
  - Blocking
  - Servers

Utilization-based Test for EDF

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$

A much simpler test than that for FPS

- Superior to FPS (0.69 bound in worst-case); it can support high utilizations. However,
- Bound only applicable to simple task model
  - Although EDF is always as good as FPS, and usually better

FPS v EDF

- FPS is easier to implement as priorities are static
- EDF is dynamic and requires a more complex run-time system which will have higher overhead
- It is easier to incorporate tasks without deadlines into FPS; giving a task an arbitrary deadline is more artificial
- It is easier to incorporate other factors into the notion of priority than it is into the notion of deadline

FPS v EDF

- During overload situations
  - FPS is more predictable; Low priority process miss their deadlines first
  - EDF is unpredictable; a domino effect can occur in which a large number of processes miss deadlines
- But EDF gets more out of the processor!

Processor Demand Criteria

- Arbitrary EDF system (D less than or greater than T) is schedulable if
  - Load at time t is less than t, for all t
  - There is a bound or how big t needs to be
  - Load expressed as h(t)
  - Only points in time that refer to actual task deadlines need be checked

PDC

$$h(t) = \sum_{i=1}^{N} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i$$

If D>T then floor function is constrained to have minimum 0

$$\forall t > 0, h(t) \leq t$$

Upper Bound for PDA

$$L_a = \max\left\{ D_1, \ldots, D_N, \frac{\sum_{i=1}^{N} (T_i - D_i) C_i / T_i}{1 - U} \right\}$$

U is the utilisation of the task set, note upper bound not defined for U=1

Upper Bound for PDC

$$w^0 = \sum_{i=1}^{N} C_i$$

$$w^{j+1} = \sum_{i=1}^{N} \left\lceil \frac{w^j}{T_i} \right\rceil C_i$$

This is the processor busy period and is bounded for U no greater than 1

$$L_b = w^j = w^{j+1}$$

$$L = \min(L_a, L_b)$$

QPA Test

- Refers to Quick Processor demand Analysis
- Start at L, and work backwards towards 0
  - If at time t, h(t)>t then unschedulable
  - else let t equal h(t)
    - If h(t) =t then let t equal t-1 (or smallest absolute deadline <t)
  - If t < smallest D, then schedulable
- Typically QPA requires only 1% of the effort of PDA but is equally necessary and sufficient

EDF and Blocking

- SRP – Stack Resource Policy – is a generalisation of the priority ceiling protocol
- Two notions:
  - Preemption level for access to shared objects
  - Urgency for access to processor
- With FPS, priority is used for both
- With EDF, ‘priority’ is used for preemption level, and earliest absolute deadline is used for urgency

EDF and Servers

- There are equivalent servers under EDF to those defined for FPS: periodic server, deferrable server and sporadic server
- There are also EDF-specific servers
  - If, for example, an aperiodic task arrives at time 156 and wants 5ms of computation from a server with budget 2ms and period 10ms
  - Task will need 3 allocations, and hence is given an absolute deadline of 186
  - EDF scheduling rules are then applied

Online Analysis

- There are dynamic soft real-time applications in which arrival patterns and computation times are not known *a priori*
- Although some level of offline analysis may still be applicable, this can no longer be complete and hence some form of on-line analysis is required
- The main role of an online scheduling scheme is to manage any overload that is likely to occur due to the dynamics of the system's environment
- During transient overloads EDF performs very badly. It is possible to get a cascade effect in which each task misses its deadline but uses sufficient resources to result in the next task also missing its deadline

Admission Schemes

- To counter this detrimental domino effect many on-line schemes have two mechanisms:
  - an admissions control module that limits the number of tasks that are allowed to compete for the processors, and
  - an EDF dispatching routine for those tasks that are admitted
- An ideal admissions algorithm prevents the processors getting overloaded so that the EDF routine works effectively

Values

- If some tasks are to be admitted, whilst others rejected, the relative importance of each task must be known
- This is usually achieved by assigning value
- Values can be classified
  - Static: the task always has the same value whenever it is released
  - Dynamic: the task's value can only be computed at the time the task is released (because it is dependent on either environmental factors or the current state of the system)
  - Adaptive: here the dynamic nature of the system is such that the value of the task will change during its execution

Values

- To assign static values requires the domain specialists to articulate their understanding of the desirable behaviour of the system

Worst-case Execution Time

- WCET values are critical for all forms of analysis
  - C values in all the equations of this chapter
  - Known as Timing Analysis
- Found either by static analysis or measurement
  - Static analysis is pessimistic and hard to undertake with modern processors with cache(s), pipelines, out-of-order execution, branch prediction etc
  - Measurement is potentially optimistic – was the worst-case path measured?
- Timing analysis usually represents each task as a directed graph of basic blocks (or basic paths)

Basic Blocks

- Once the worst-case time of each basic block is obtained (via measurement or a model of the processor) then
- Direct graph is collapsed with maximum loop parameters and worst-case branches assumed to be taken

Example

```
for i in 1..10 loop
  if Cond then
    -- basic block of cost 100
  else
    -- basic block of cost 10
  end if;
end loop;
```

With no further semantic information must assume total cost is 1000, But if Cond can only be true 4 times then cost is 460

ILP – Integer Linear Programming – and Constraint Satisfaction programs are being employed to model basic block dependencies

Multiprocessor

- Issues:
  - Homogeneous or heterogeneous processors
  - Globally scheduling or partitioned
    - Or other alternatives between these two extremes
  - Optimality?
    - EDF is not optimal
    - EDF is not always better than FPS
    - Global is not always better than partitioned
- Partitioning is an allocation problem followed by single processor scheduling

Example

Task	T	D	C
a	10	10	5
b	10	10	5
c	12	12	8

2 processors, global

EDF and FPS would run a and b on the 2 processors

No time left for c (7 on each, 14 in total, but not 8 on one)

Example

Task	T	D	C
d	10	10	9
e	10	10	9
f	10	10	2

2 processors, partitioned

Task f can not sit on just one processor, it needs to migrate and for d and e to cooperate

Results

- For the simple task model introduced earlier
  - A *pfair* scheme can theoretically schedule up to a total utilisation of M (for M processors), but overheads are excessive
  - For FPS with partitioned and first-fit on utilisation

$$U \leq M(\sqrt{2}-1)$$

  - ie 0.414M

Results

- EDF partitioned first-fit

$$U \le \frac{\beta M + 1}{\beta + 1}$$

$$\beta = \left\lfloor \frac{1}{U_{\max}} \right\rfloor$$

- For high maximum utilisation can get lower than 0.25M, but can get close to M for some examples

Results

- EDF global

$$U \le M - (M-1)U_{\max}$$

- Again for high Umax can be as low as 0.2M but also close to M for other examples

Results

- Combinations:
  - Fixed priority to those tasks with U greater than 0.5
  - EDF for the rest

$$U \leq \left(\frac{M+1}{2} \right)$$

Power Aware

- For battery-power embedded systems it is useful to run the processor with the lowest speed commensurate with meeting all deadlines
- Halving the speed may quadruple the life
- Sensitivity analysis will, for a schedulable system, enable the maximum scaling factor for all C values to be ascertained – with the system remaining schedulable

Overheads

- To use in an industrial context, the temporal overheads of implementing the system must be taken in to account
  - Context switches (one per job)
  - Interrupts (one per sporadic task release)
  - Real-time clock overheads
- Will consider FPS only here

Context switches

Rather than

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$$R_i = CS^1 + C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (CS^1 + CS^2 + C_j)$$

Where the new terms are the cost of switching to the task and the cost of switching away from the task

Interrupt Handling

Cost of handling interrupts: $$\sum_{k \in \Gamma_s} \left\lceil \frac{R_i}{T_k} \right\rceil IH$$

Where Γ_s is the set of sporadic tasks

And IH is the cost of a single interrupt (which occurs at maximum priority level)

Clocks Overheads

- There is a cost per clock interrupt
  - and a cost for moving one task from delay to run queue
  - and a (reduced) cost of moving groups of tasks

Let CT_c be the cost of a single clock interrupt, Γ_p be the set of periodic tasks, and CT_s be the cost of moving one task the following equation can be derived

Full model

$$R_i = CS^1 + C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (CS^1 + CS^2 + C_j)$$

$$+ \sum_{k \in \Gamma_s} \left\lceil \frac{R_i}{T_k} \right\rceil IH + \left\lceil \frac{R_i}{T_{clk}} \right\rceil CT_c + \sum_{g \in \Gamma_p} \left\lceil \frac{R_i}{T_g} \right\rceil CT_s$$

Summary

- A scheduling scheme defines an algorithm for resource sharing and a means of predicting the worst-case behaviour of an application when that form of resource sharing is used
- With a cyclic executive, the application code must be packed into a fixed number of minor cycles such that the cyclic execution of the sequence of minor cycles (the major cycle) will enable all system deadlines to be met
- The cyclic executive approach has major drawbacks many of which are solved by priority-based systems

Summary

- Rate monotonic priority ordering is optimal for the simple task model
- Simple utilization-based schedulability tests are not exact for FPS
- Response time analysis is flexible and caters for:
  - Any priority ordering
  - Periodic and sporadic processes
  - Is necessary and sufficient for many situations

Summary

- Blocking can result in serious priority inversion
- Priority inheritance (in some form) is needed to bound blocking
- Priority ceiling protocols bound blocking to a single block, supply mutual exclusion and prevent deadlocks
- Response time analysis can easily be extended to cope with blocking caused by ceiling protocols

Summary

- Examples have been given of how to expand RTA to many different application requirements
- RTA is a framework, it must be expanded to deal with particular situations
- Analysis of EDF scheduling systems has been covered
- Multiprocessor scheduling is still not mature, some current results are outlined
- Power aware scheduling was briefly cover
- Finally, how to take system overheads into account in response time analysis was addressed