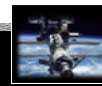




6

Scheduling

Uwe R. Zimmer & Alistair Rendell
The Australian National University



Concurrent & Distributed Systems



References for this chapter

[Bacon98]

J. Bacon
Concurrent Systems
1998 (2nd Edition)
Addison Wesley Longman Ltd,
ISBN 0-201-17767-6

[Stallings2001] – Chapter 3,4

William Stallings
Operating Systems
Prentice Hall, 2001

all references and some links are available on the course page



Concurrent & Distributed Systems

Scheduling



Purpose of scheduling

A scheduling scheme provides two features:

- **Ordering the use of resources** (e.g. CPUs, networks)
- **Predicting the worst-case behaviour** of the system when the scheduling algorithm is applied
... in case that some or all information about the expected resource requests are known

A prediction can then be used

- ☞ at compile-run: to **confirm the overall resource requirements** of the application, or
- ☞ at run-time: to **permit acceptance** of additional usage/reservation requests.



Concurrent & Distributed Systems

Scheduling

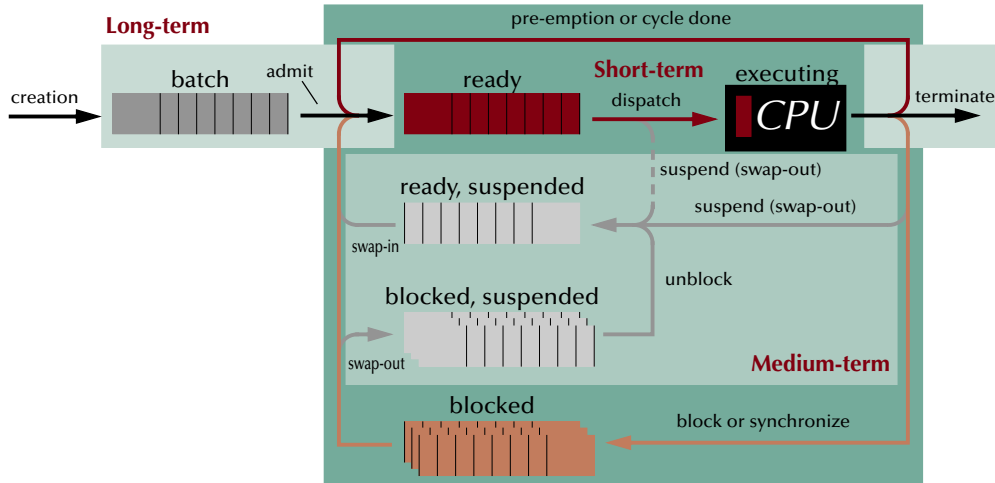


Criteria for scheduling methods

	Performance criteria: minimize the ...	Predictability criteria: minimize the diversion from given
Process / user perspective:		
Waiting time	maximum / average / variance	minimal and maximal waiting times
Response time	maximum / average / variance	minimal and maximal response times
Turnaround time	maximum / average / variance	deadlines
System perspective:		
Throughput	maximum / average / variance of CPU time per process	—
Utilization	CPU idle time	—

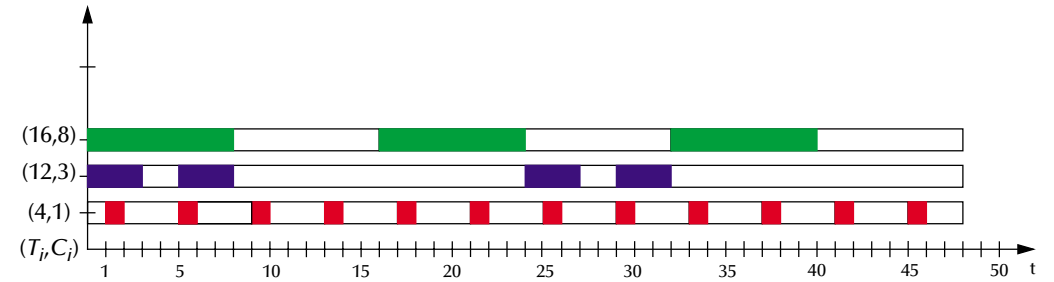
Scheduling

Time scales of scheduling



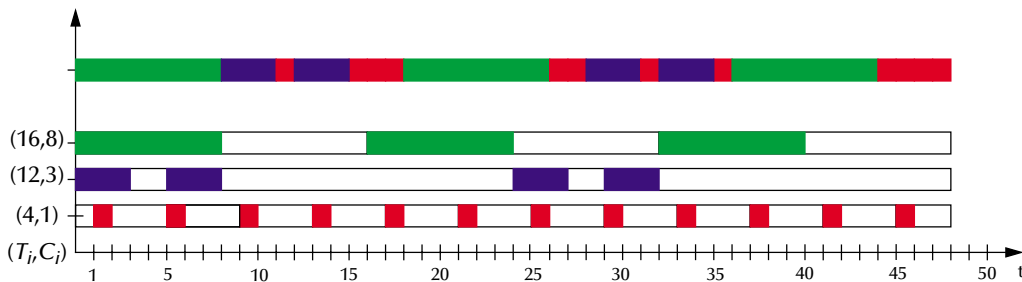
Scheduling

Example: Requested times



Scheduling

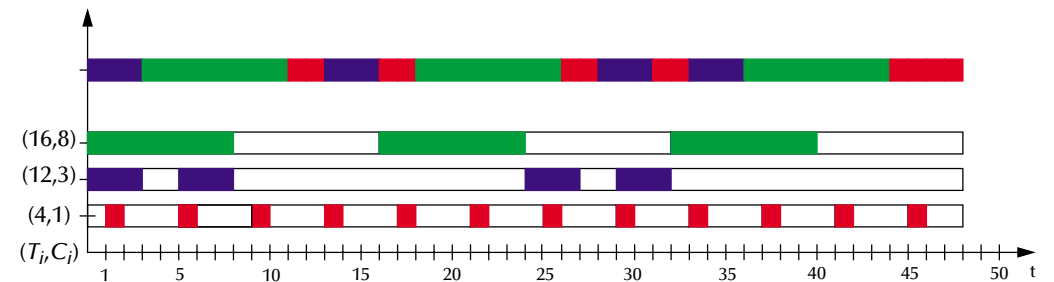
First come, first served (FCFS) – bad case: (arrival order: green, purple, red)



Waiting time: 0...11; average: 5.95 – Turnaround time: 3...12; average: 8.47

Scheduling

First come, first served (FCFS) – nice case: (arrival order: red, purple, green)

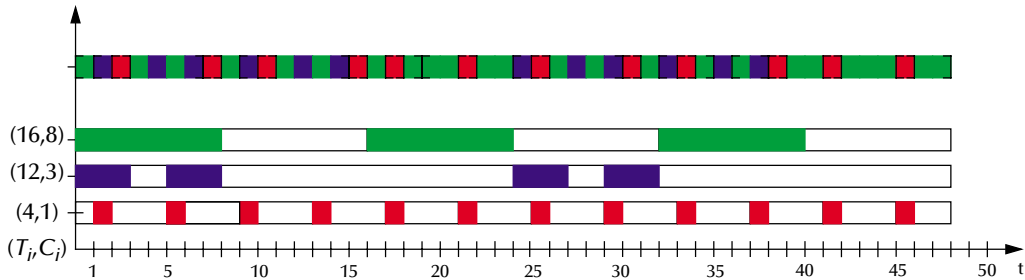


Waiting time: 0...11; average: 5.47 – Turnaround time: 3...12; average: 8.00

☛ The actual average waiting time for FCFS may vary here between: 5.47 and 5.95

Scheduling

Round robin (RR) – pre-emption



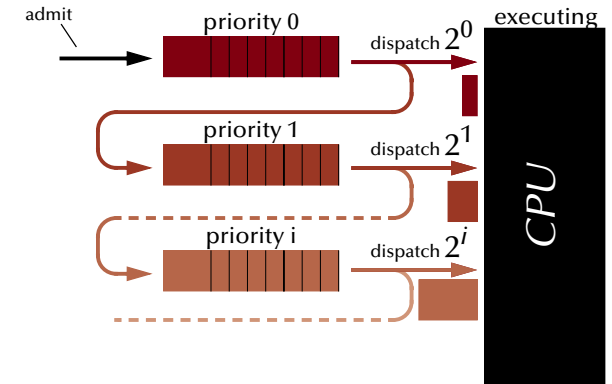
Waiting time: 0...4; average: 1.21 – Turnaround time: 1...19; average: 5.63

- ☞ Waiting and average turnaround time is going down, but maximal turnaround time is going up ... assuming that task-switching is free and always possible

Scheduling

Feedback with 2^i pre-emption intervals – pre-emption

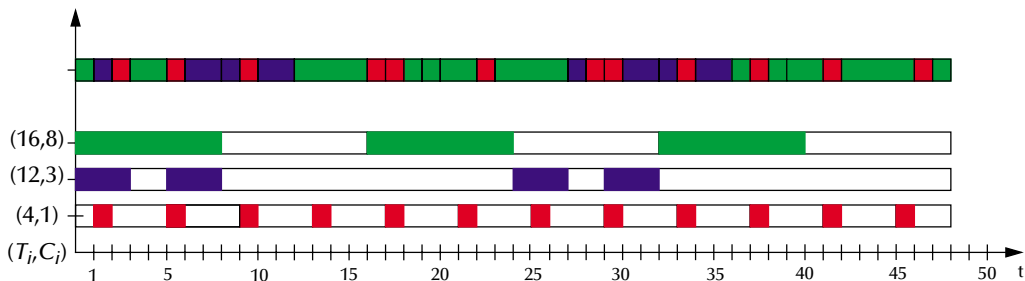
- implement multiple hierarchical ready-queues
- fetch processes from the highest filled ready queue
- dispatch more CPU time for lower priorities (2^i units)



- ☞ processes on lower ranks may suffer starvation
- ☞ new and short tasks will be preferred

Scheduling

Feedback with 2^i pre-emption intervals – pre-emption

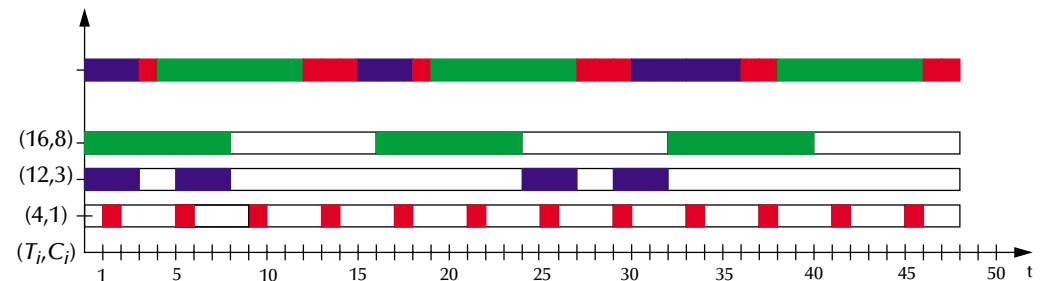


Waiting time: 0...6; average: 1.79 – Turnaround time: 1...21; average: 5.63

- ☞ less task switches than RR, but long processes can suffer starvation!

Scheduling

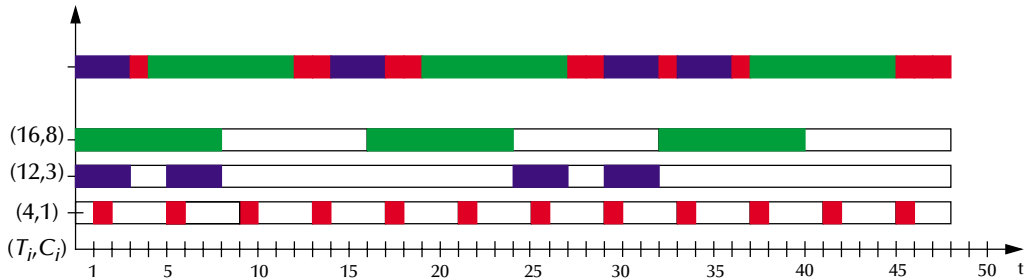
Shortest job first (SJF) – C_i is known



Waiting time: 0...10; average: 3.47 – Turnaround time: 1...14; average: 6.00

- ☞ on average this is doing better than FCFS

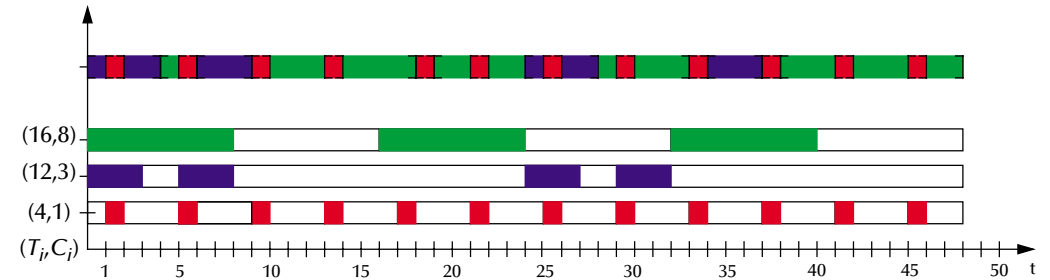
Highest response ratio first (HRRF) – C_i is known



Response ratio: $(W_i + C_i)/C_i$ – Waiting time: 0...9; average: 4.11 – Turnaround time: 1...13; average 6.63

☞ on average this is doing worse than SJF, but the maximal waiting and turnaround times and variance might be reduced!

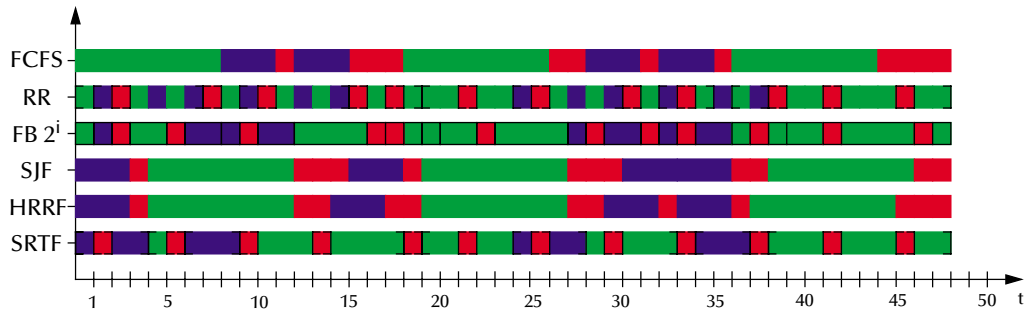
Shortest remaining time first (SRTF) – C_i is known + pre-emption



Waiting time: 0...6; average: 1.05 – Turnaround time: 1...18; average 4.42

☞ on average this is doing better than FCFS, SJF or HRRF, but the maximal turnaround time is going up and there are many task-switches!

Non-realtime scheduling methods



☞ CPU utilization: 100% in all cases.

☞ Pre-emptive methods perform better, assuming that the overhead is negligible.

☞ Knowledge of C_i (computation times) has a significant impact on scheduler performance.

	Selection	Pre-emption	Waiting in high load situations	Turnaround	Preferred processes	Starvation possible?
FCFS	$\max(W_i)$	no	possibly long	possibly long	long	no
RR	equal share	yes	bound	possibly long	none	no
Feedback	priority queues	yes	short on average	very short on average, large maximum	short	yes
SJF	$\min(C_i)$	no	short on average	short on average	short	yes
HRRF	$\max\left(\frac{W_i + C_i}{C_i}\right)$	no	short on average, lower variance	short on average, lower variance	balanced, towards short	no
SRTF	$\min(C_i - E_i)$	yes	very short on average	very short on average, large maximum	short	yes

Concurrent & Distributed Systems

Predictable scheduling

Towards predictable scheduling ...

- ☞ Task behaviours are more specified (restricted).
 - ☞ Task requirements are more specific (time scopes).
 - ☞ Task sets are often fully or mostly static.
 - ☞ Sporadic and urgent requests (e.g. user interaction, alarms) need to be addressed.
- CPU-utilization and throughput (system oriented performance measures) are not important!

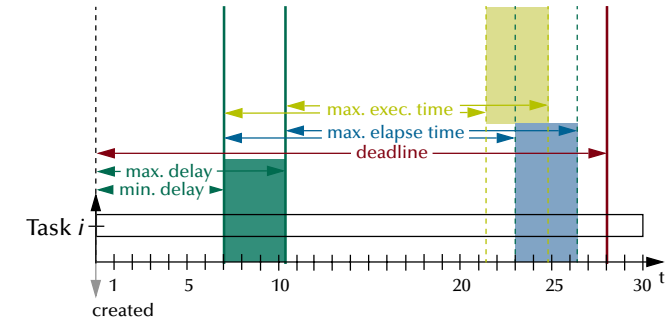
Concurrent & Distributed Systems

Specifying timing requirements

Temporal scopes

Common attributes:

- Minimal & maximal delay after creation
- Maximal elapsed time
- Maximal execution time
- Absolute deadline



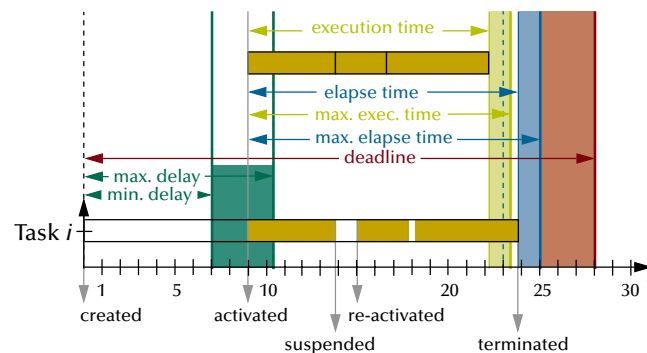
Concurrent & Distributed Systems

Specifying timing requirements

Temporal scopes

Common attributes:

- Minimal & maximal delay after creation
- Maximal elapsed time
- Maximal execution time
- Absolute deadline



Concurrent & Distributed Systems

Specifying timing requirements

Some common scope attributes

Temporal Scopes can be:

Periodic	– e.g. controllers, samplers, monitors
Aperiodic	– e.g. ‘periodic on average’ tasks, burst requests
Sporadic / Transient	– e.g. mode changes, occasional services

Deadlines (absolute, elapse, or execution time) can be:

Hard	– single failure leads to severe malfunction
Firm	– results are meaningless after the deadline
Soft	– only multiple or permanent failures threaten the whole system
	– results may still be useful after the deadline



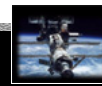
Concurrent & Distributed Systems



Predictable scheduling

A simple process model

- The number of processes in the system is fixed.
 - All processes are periodic and all periods are known.
 - All deadlines are identical with the process cycle times (periods).
 - The worst case execution time is known for all processes.
 - All processes are independent.
 - All processes are released at once.
 - The task-switching overhead is negligible.
- ☞ this model can only be applied to a very specific group of systems.
(more real-world extensions to this model will be discussed in other courses).

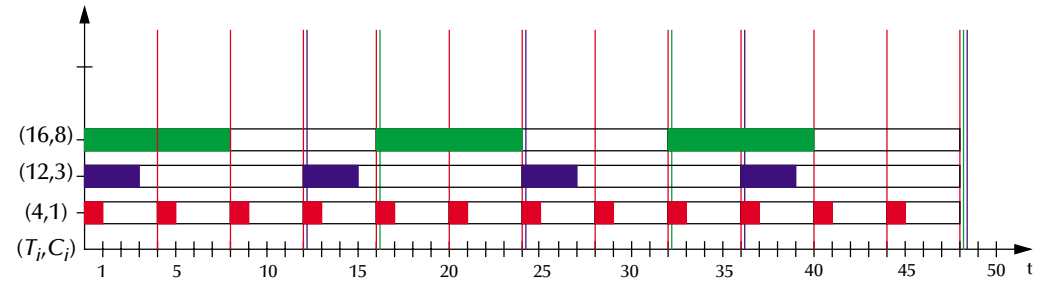


Concurrent & Distributed Systems



Predictable scheduling

Introducing deadlines



Concurrent & Distributed Systems



Dynamic scheduling

Earliest deadline first (EDF)

1. Determine (one of) the processe(s) with the closest deadline.
 2. Execute this process
 - 2-a until it finishes
 - 2-b or until another process' deadline is found closer than the current one.
- ☞ Pre-emptive scheme
- ☞ Dynamic scheme, since the dispatched process is selected at run-time, due to the current deadlines.

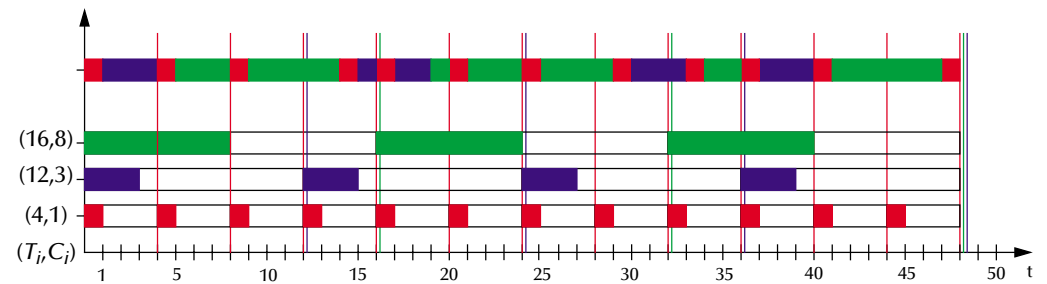


Concurrent & Distributed Systems



Dynamic scheduling: Earliest Deadline First (EDF)

Earliest deadline first

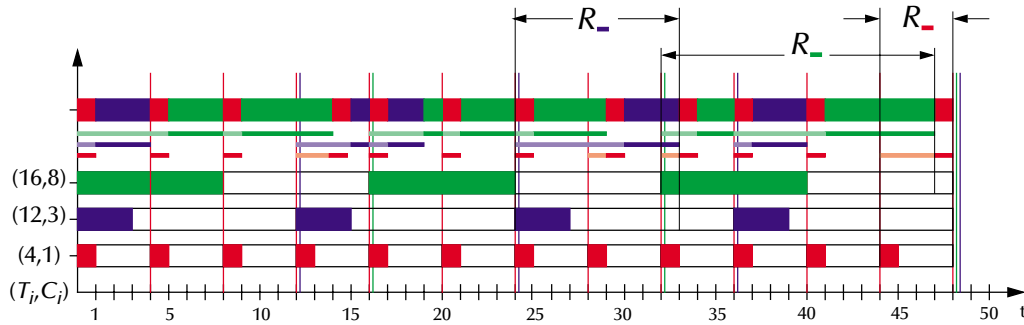


1. Schedule the earliest deadline first
2. Avoid task switches (in case of equal deadlines)

Concurrent & Distributed Systems

Dynamic scheduling: Earliest Deadline First (EDF)

Earliest deadline first: Response times



worst case response times R_i (maximal time in which the request from task T_i is served):

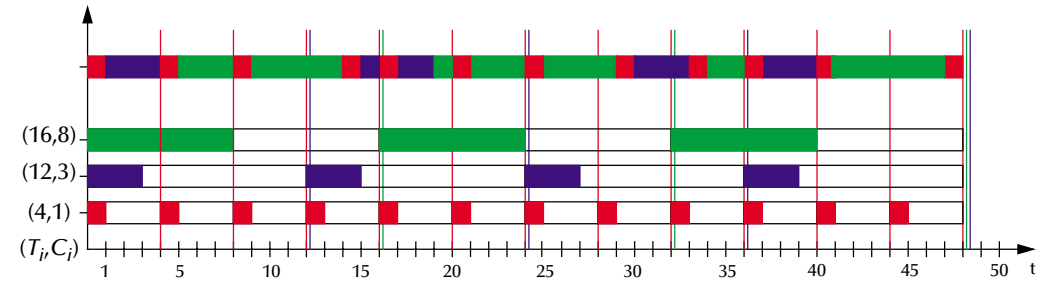
☞ can be close or identical to deadlines.

☞ small or none spare capacity, if any task misses its expected computation time.

Concurrent & Distributed Systems

Dynamic scheduling: Earliest Deadline First (EDF)

Earliest deadline first: Maximal utilization



☞ maximal possible utilization: $\sum_{i=1}^n \frac{C_i}{T_i} \leq 1$ ☞ sufficient & necessary test!
with C_i, T_i the computation and cycle times of task i
(the deadlines D_i are assumed to be identical with the cycles times T_i here)

Concurrent & Distributed Systems

Static scheduling

Fixed Priority Scheduling (FPS), rate monotonic

1. Each process is assigned a fixed priority according to its cycle time T_i :

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

2. At any point in time: dispatch the process with the highest priority

☞ Pre-emptive scheme

☞ Static scheme, since the dispatch order of processes is fixed and calculated off-line.

Concurrent & Distributed Systems

Static scheduling

Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic ordering is **optimal**
(in the framework of fixed priority schedulers)

i.e. **if** a process set is schedulable under a FPS-scheme, **then** it is also schedulable by applying rate monotonic priorities.

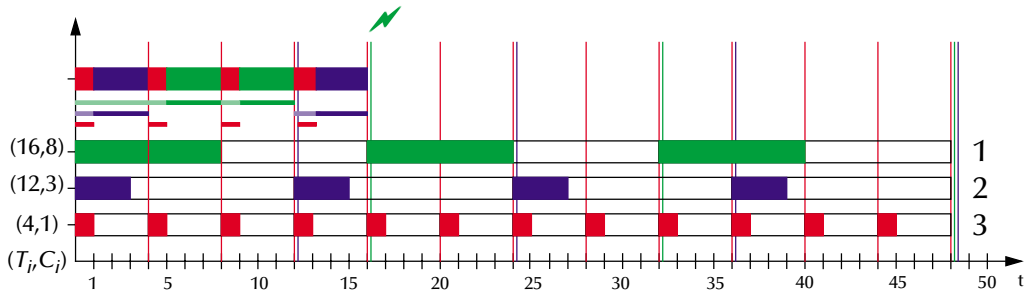


Concurrent & Distributed Systems

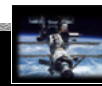


Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities



max. utilization test: $\sum_{i=1}^n \frac{C_i}{T_i} \leq N \left(2^{\frac{1}{N}} - 1 \right)$ ☞ sufficient, but not necessary test!

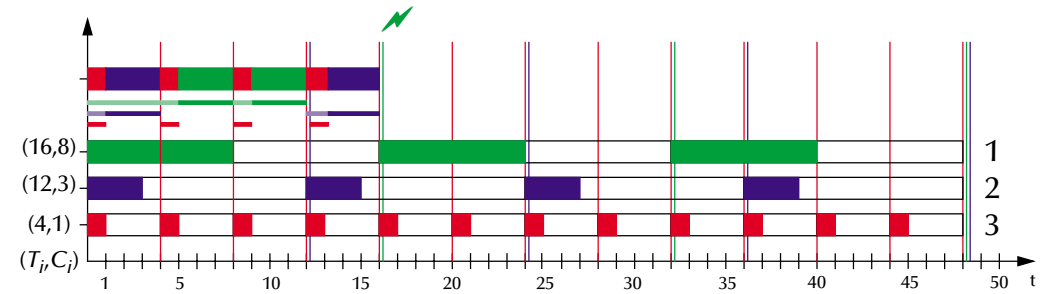


Concurrent & Distributed Systems



Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities



utilization test: $\sum_{i=1}^n \frac{C_i}{T_i} = 1 > 0.779 \approx N \left(2^{\frac{1}{N}} - 1 \right)$ ☞ not guaranteed!

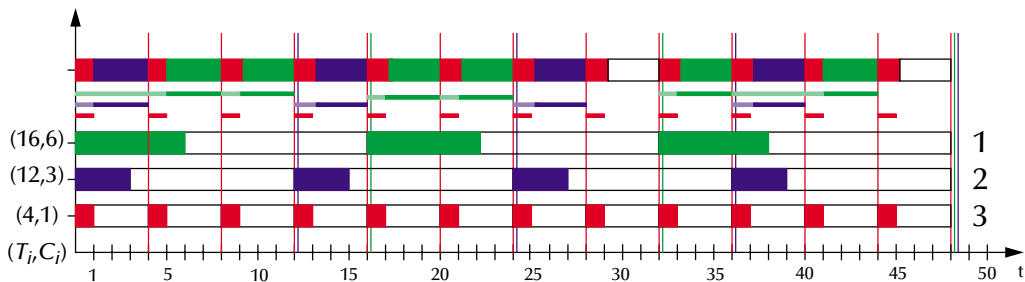


Concurrent & Distributed Systems

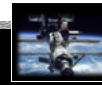


Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities (reduced requests)



☞ utilization: $\frac{6}{16} + \frac{3}{12} + \frac{1}{4} = 0.875 > 0.779 \approx 3 \left(2^{\frac{1}{3}} - 1 \right)$; $\sum_{i=1}^n \frac{C_i}{T_i} \leq N \left(2^{\frac{1}{N}} - 1 \right)$ ☞ not guaranteed!

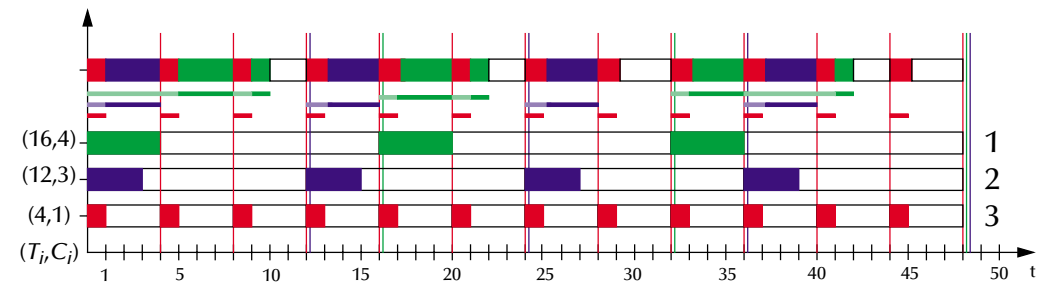


Concurrent & Distributed Systems



Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities (further reduced requests)



☞ utilization: $\frac{4}{16} + \frac{3}{12} + \frac{1}{4} = 0.75 \leq 0.779 \approx 3 \left(2^{\frac{1}{3}} - 1 \right)$; $\sum_{i=1}^n \frac{C_i}{T_i} \leq N \left(2^{\frac{1}{N}} - 1 \right)$ ☞ guaranteed!

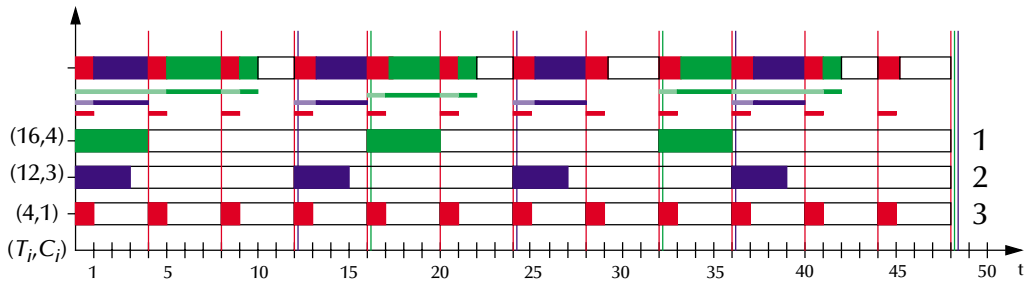


Concurrent & Distributed Systems

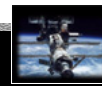


Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis (further reduced requests)



calculate the worst case response times for each task individually.

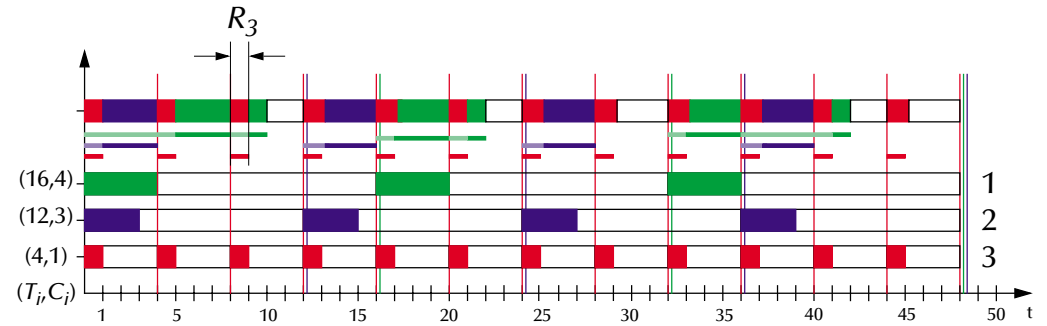


Concurrent & Distributed Systems



Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis (further reduced requests)



for the highest priority task: $R_3 = C_3$

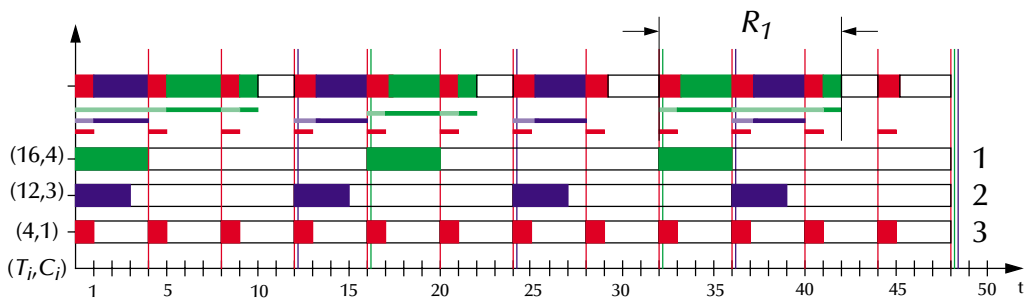


Concurrent & Distributed Systems

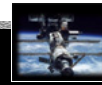


Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis (further reduced requests)



for other tasks: $R_j = C_j + I_j = \text{computation } C_j + \text{interference } I_j$

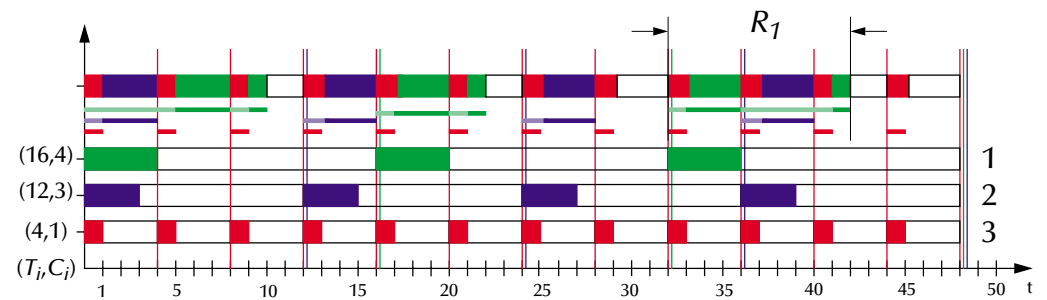


Concurrent & Distributed Systems



Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis (further reduced requests)

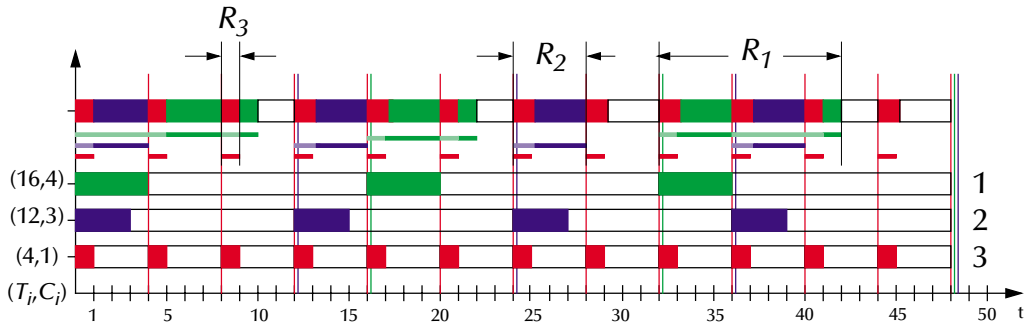


for other tasks: $R_j = C_j + \sum_{i>j} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$

Concurrent & Distributed Systems

Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis (further reduced requests)

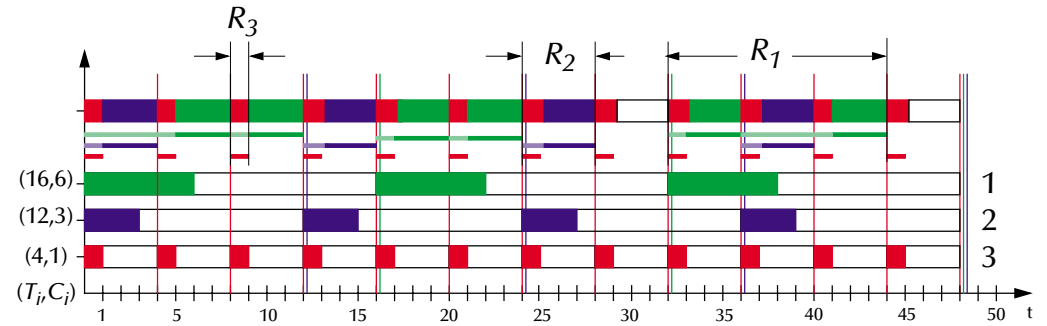


$$\checkmark R_3 = 1\checkmark; R_2 = 4\checkmark; R_1 = 10\checkmark \text{ and } \sum_{i=1}^n \frac{C_i}{T_i} \leq N \left(2^{\frac{1}{N}} - 1 \right) \checkmark$$

Concurrent & Distributed Systems

Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis (reduced requests)

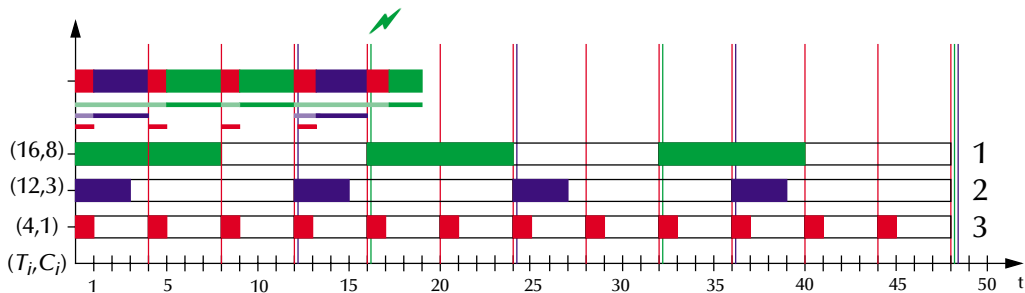


$$\checkmark R_3 = 1\checkmark; R_2 = 4\checkmark; R_1 = 12\checkmark \text{ but } \sum_{i=1}^n \frac{C_i}{T_i} > N \left(2^{\frac{1}{N}} - 1 \right) \times$$

Concurrent & Distributed Systems

Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis (full requests)

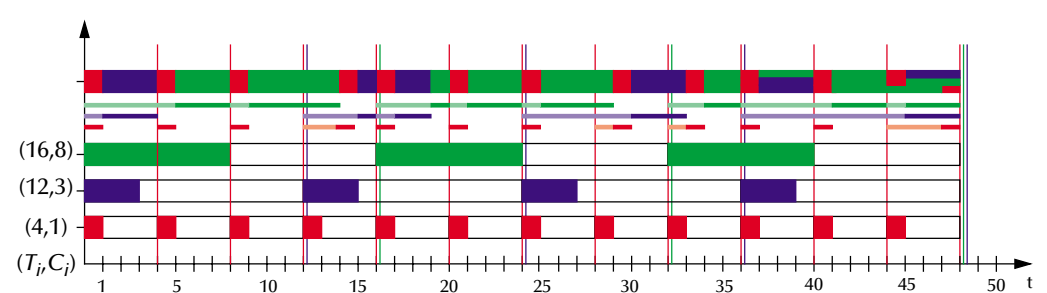


$$\checkmark R_3 = 1\checkmark; R_2 = 4\checkmark; R_1 = 19 \times \text{ and } \sum_{i=1}^n \frac{C_i}{T_i} > N \left(2^{\frac{1}{N}} - 1 \right) \times$$

Concurrent & Distributed Systems

Dynamic scheduling: Earliest Deadline First (EDF)

Response time analysis (full requests)

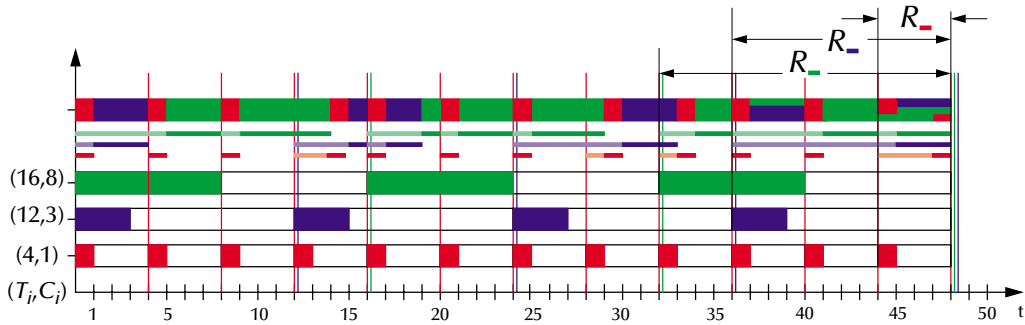


testing all combinations in a hyper-period: LCM of $\{T_i\}$ — here: 48

Concurrent & Distributed Systems

Dynamic scheduling: Earliest Deadline First (EDF)

Response time analysis (full requests)



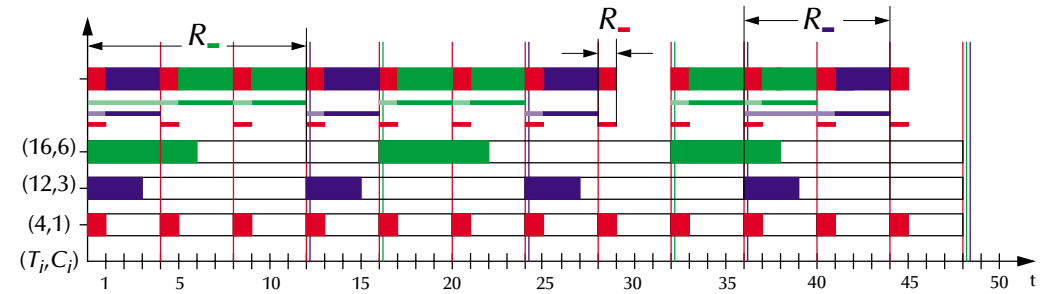
testing all combinations in a hyper-period: LCM of $\{T_i\}$ – here: 48

$$R_-: 16 \leq 16v = T_-; \quad R_-: 12 \leq 12v = T_-; \quad R_-: 4 \leq 4v = T_-$$

Concurrent & Distributed Systems

Dynamic scheduling: Earliest Deadline First (EDF)

Response time analysis (reduced requests)



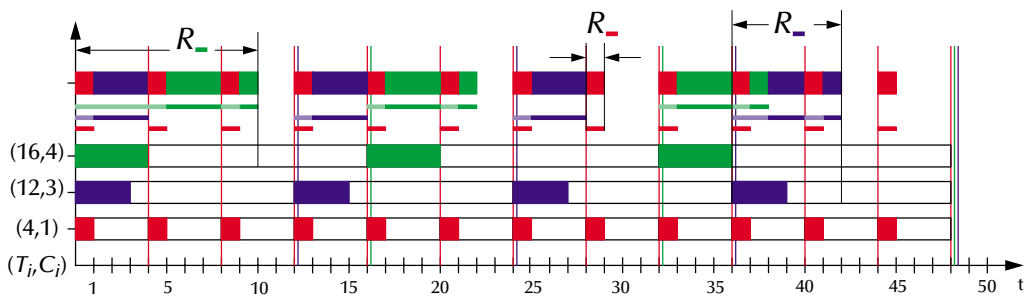
relaxed task-set changes:

$$R_-: 16 \rightarrow 12 \leq 16v = T_-; \quad R_-: 12 \rightarrow 8 \leq 12v = T_-; \quad R_-: 4 \rightarrow 1 \leq 4v = T_-$$

Concurrent & Distributed Systems

Dynamic scheduling: Earliest Deadline First (EDF)

Response time analysis (further reduced requests)



further relaxed task-set changes:

$$R_-: 12 \rightarrow 10 \leq 16v = T_-; \quad R_-: 8 \rightarrow 6 \leq 12v = T_-; \quad R_-: 1 \rightarrow 1 \leq 4v = T_-$$

Concurrent & Distributed Systems

Real-time scheduling

Response time analysis (comparison)

	Fixed Priority Scheduling		Earliest Deadline First	
	utilization test	response times $\{R_j\}$	utilization test	response times $\{R_j\}$
$\{(T_i, C_i)\} = \{(16, 8); (12, 3); (4, 1)\}$	✗ (1.000)	$\{\infty, 4, 1\}$	✓ (1.000)	$\{16, 12, 4\}$
$\{(T_i, C_i)\} = \{(16, 6); (12, 3); (4, 1)\}$	✗ (0.875)	$\{12, 4, 1\}$	✓ (0.875)	$\{12, 8, 1\}$
$\{(T_i, C_i)\} = \{(16, 4); (12, 3); (4, 1)\}$	✓ (0.750)	$\{10, 4, 1\}$	✓ (0.750)	$\{10, 6, 1\}$
	$\sum_{i=1}^n \frac{C_i}{T_i} \leq N \left(2^{\frac{1}{N}} - 1 \right)$	$C_i + \sum_{j>i} \left\lceil \frac{R_j}{T_j} \right\rceil C_j$	$\sum_{i=1}^n \frac{C_i}{T_i} \leq 1$	check full hyper-cycle

Real-time scheduling



Fixed Priority Scheduling ↔ Earliest Deadline First

- EDF can handle higher (full) utilization than FPS.
- FPS is easier to implement and implies less run-time overhead
- Graceful degradation features (resource is over-booked):
 - FPS: processes with lower priorities will always miss their deadlines first.
 - EDF: any process can miss its deadline and can trigger a cascade of failed deadlines.
- Response time analysis and utilization tests:
 - FPS: $O(n)$ utilization test — response time analysis: fixed point equation
 - EDS: $O(n)$ utilization test — response time analysis: fixed point equation in hyper-cycle

Summary



Scheduling

• Basic performance based scheduling

- C_i is not known: first-come-first-served (FCFS), round robin (RR), and feedback-scheduling
- C_i is known: shortest job first (SJF), highest response ration first (HRRF), shortest remaining time first (SRTF)-scheduling

• Basic predictable scheduling

- Fixed Priority Scheduling (FPS) with Rate Monotonic (RMPO)
- Earliest Deadline First (EDF)

	Selection	Pre-emption	Waiting	Turnaround	Preferred processes	Starvation possible?
FCFS	$\max(W_i)$	no	possibly long	possibly long	long	no
RR	equal share	yes	bound	possibly long	none	no
Feedback	priority queues	yes	short on average	very short on average, large maximum	short	yes
SJF	$\min(C_i)$	no	short on average	short on average	short	yes
HRRF	$\max((W_i + C_i)/C_i)$	no	short on average, lower variance	short on average, lower variance	balanced	no
SRTF	$\min(C_i - E_i)$	yes	very short on average	very short on average, large maximum	short	yes
FPS	$\max(P_i)$	yes	priority based	priority based	higher priority	yes
EDF	$\min(D_i)$	yes	deadline based	often close to deadlines	most urgent	no