Real-Time Scheduling

Uwe R. Zimmer – The Australian National University

Scheduling schemes

- Static
  - All predictions and schedules are done off-line.
  - Often better predictability = most hard real-time systems.
- Dynamic
  - Run-time situation is taken into account.
  - More flexible, more efficient = most soft real-time systems.

Example: Requested times

Example: Deadlines

Earliest deadline first (EDF)

1. Determine (one of) the processes with the closest deadline.
2. Execute this process.
3. a) Until it finishes.
   b) Or until another process’ deadline is found closer than the current one.

Pre-emption scheme

Dynamic scheme.

Since the dispatched process is selected at run-time, due to the current deadlines.

Real-time scheduling: Earliest Deadline First (EDF)

Fixed Priority Scheduling (FPS), rate monotonic

1. Each process is assigned a hard priority according to its cycle time.

Fixed Priority Scheduling: Earliest Deadline First (EDF)

1. Determine (one of) the processes with the closest deadline.
2. Execute this process.
3. a) Until it finishes.
   b) Or until another process’ deadline is found closer than the current one.

Pre-emption scheme

Dynamic scheme.

Since the dispatched process is selected at run-time, due to the current deadlines.

Dynamic scheduling: Earliest Deadline First (EDF)

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

Earliest deadline first:

- Priority is the amount of time in which the request (from task / CPU) is served.

Earliest deadline first: Maximal utilization

Maximal possible utilization:

\[
\sum \frac{P_i}{C_i} \leq 1 \quad \text{is sufficient & necessary test.}
\]

with \( C_i \) the computation and cycle times of task / (the deadlines \( D_i \) are assumed to be identical with the cycle times \( T_i \), here)

References for this chapter

[Burns01]


[Addison Wesley, third edition, 2001]
calculate the worst case response times for each task individually.

\[ C_i + T_i + \text{computation} + \text{interference} \]

\[ R = C_i \]

\[ \text{for the highest priority task: } R_1 = C_1 \]

\[ \text{for other tasks: } R_j = C_j + \sum_{i=1}^{j-1} \frac{C_i}{T_i} \]

\[ T_i \]

\[ \sum_{i=1}^{N} C_i + \frac{1}{T_i} \]

\[ \text{sufficient, but not necessary!} \]

\[ \text{Utilisation based Analysis for FPS rate monotonic} \]

\[ U \equiv \frac{1}{2} \sum_{i=1}^{N} C_i N_i - 1 \equiv U_{\text{max}} \]

\[ \text{with } C_i \text{ the computation time and } T_i \text{ the length of the period for task } i \text{ of } N \text{ tasks and assuming that the deadline } D_i = T_i \]

\[ \text{guaranteed!} \]

\[ \text{interpretable complexity} \]

\[ \text{sufficient, but not necessary!} \]
Real-Time & Embedded Systems

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

Response time analysis

\[ R_t = C_0 + \sum_{j=1}^{n} C_j T_j \]

\( n \) is the number of tasks.

\( C_0 \) is the system overhead.

\( C_j \) is the execution time of task \( j \).

\( T_j \) is the period of task \( j \).

\[ \sum R_t = \text{fixed point equation!} \]

Example (full requests):

- set of tasks \((T_i, C_i)\) = \{(16, 6), (12, 3), (4, 1)\) at priorities \(1:2:3\); \( R_t = 6 \)
- \( R_t = 8 \times \sum C_j \times \frac{1}{T_j} = 13 \)
- \( R_t = 8 + \sum C_j \times \frac{1}{T_j} = 18 \times \omega \)
- \( R_t = 8 + \sum C_j \times \frac{1}{T_j} = 19 \times \omega \)

Example (reduced requests):

- set of tasks \((T_i, C_i)\) = \{(16, 4), (12, 3), (4, 1)\) at priorities \(1:2:3\); \( R_t = 6 \)
- \( R_t = 8 \times \sum C_j \times \frac{1}{T_j} = 10 \times \omega \)
- \( R_t = 4 + \sum C_j \times \frac{1}{T_j} = 9 \times \omega \)
- \( R_t = 4 + \sum C_j \times \frac{1}{T_j} = 10 \times \omega \)

Response time analysis (reduced requests)

\[ R_t = C_0 + \sum_{j=1}^{n} C_j \frac{R_t^\alpha}{T_j} \]

\( \alpha \) is the work load factor.

\[ \sum R_t = \text{fixed point equation!} \]

Example (further reduced requests):

- set of tasks \((T_i, C_i)\) = \{(16, 4), (12, 3), (4, 1)\) at priorities \(1:2:3\); \( R_t = 4 \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 8 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 9 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 10 \times \omega \)

Dynamic scheduling: Earliest Deadline First (EDF)

Response time analysis

\[ R_t = C_0 + \sum_{j=1}^{n} \frac{R_t^\alpha}{T_j} [\text{max}(1, \frac{C_j}{T_j}) + 1] \]

\( \alpha \) is the work load factor.

Example (reduced requests):

- set of tasks \((T_i, C_i)\) = \{(16, 4), (12, 3), (4, 1)\) at priorities \(1:2:3\); \( R_t = 4 \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 8 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 9 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 10 \times \omega \)

Example (further reduced requests):

- set of tasks \((T_i, C_i)\) = \{(16, 4), (12, 3), (4, 1)\) at priorities \(1:2:3\); \( R_t = 4 \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 8 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 9 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 10 \times \omega \)

Response time analysis (full requests)

\[ R_t = C_0 + \sum_{j=1}^{n} C_j \frac{R_t^\alpha}{T_j} \]

\( \alpha \) is the work load factor.

\[ \sum R_t = \text{fixed point equation!} \]

Example (reduced requests):

- set of tasks \((T_i, C_i)\) = \{(16, 4), (12, 3), (4, 1)\) at priorities \(1:2:3\); \( R_t = 4 \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 8 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 9 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 10 \times \omega \)

Example (further reduced requests):

- set of tasks \((T_i, C_i)\) = \{(16, 4), (12, 3), (4, 1)\) at priorities \(1:2:3\); \( R_t = 4 \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 8 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 9 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 10 \times \omega \)

Response time analysis (full requests)

\[ R_t = C_0 + \sum_{j=1}^{n} C_j \frac{R_t^\alpha}{T_j} \]

\( \alpha \) is the work load factor.

\[ \sum R_t = \text{fixed point equation!} \]

Example (reduced requests):

- set of tasks \((T_i, C_i)\) = \{(16, 4), (12, 3), (4, 1)\) at priorities \(1:2:3\); \( R_t = 4 \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 8 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 9 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 10 \times \omega \)

Response time analysis (reduced requests)

\[ R_t = C_0 + \sum_{j=1}^{n} C_j \frac{R_t^\alpha}{T_j} \]

\( \alpha \) is the work load factor.

\[ \sum R_t = \text{fixed point equation!} \]

Example (further reduced requests):

- set of tasks \((T_i, C_i)\) = \{(16, 4), (12, 3), (4, 1)\) at priorities \(1:2:3\); \( R_t = 4 \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 8 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 9 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 10 \times \omega \)

Response time analysis (full requests)

\[ R_t = C_0 + \sum_{j=1}^{n} C_j \frac{R_t^\alpha}{T_j} \]

\( \alpha \) is the work load factor.

\[ \sum R_t = \text{fixed point equation!} \]

Example (reduced requests):

- set of tasks \((T_i, C_i)\) = \{(16, 4), (12, 3), (4, 1)\) at priorities \(1:2:3\); \( R_t = 4 \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 8 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 9 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 10 \times \omega \)

Response time analysis (reduced requests)

\[ R_t = C_0 + \sum_{j=1}^{n} C_j \frac{R_t^\alpha}{T_j} \]

\( \alpha \) is the work load factor.

\[ \sum R_t = \text{fixed point equation!} \]

Example (further reduced requests):

- set of tasks \((T_i, C_i)\) = \{(16, 4), (12, 3), (4, 1)\) at priorities \(1:2:3\); \( R_t = 4 \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 8 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 9 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 10 \times \omega \)

Response time analysis (full requests)

\[ R_t = C_0 + \sum_{j=1}^{n} C_j \frac{R_t^\alpha}{T_j} \]

\( \alpha \) is the work load factor.

\[ \sum R_t = \text{fixed point equation!} \]

Example (reduced requests):

- set of tasks \((T_i, C_i)\) = \{(16, 4), (12, 3), (4, 1)\) at priorities \(1:2:3\); \( R_t = 4 \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 8 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 9 \times \omega \)
- \( R_t = 4 \times \sum C_j \times \frac{1}{T_j} = 10 \times \omega \)

Response time analysis (reduced requests)

\[ R_t = C_0 + \sum_{j=1}^{n} C_j \frac{R_t^\alpha}{T_j} \]

\( \alpha \) is the work load factor.

\[ \sum R_t = \text{fixed point equation!} \]
Introducing a server task to EDF

EDF can handle higher (full) utilization than FPS.

FPS is easier to implement and implies less real-time overhead.

Scheduling

Constraints which we used up to here:

• tasks are periodic
• deadlines are identical with task’s period time \((D = T)\)
• pre-emptive scheduling
• worst case execution times are known
• tasks are independent

Scheduling — real-world considerations

Extending which we will introduce:

• tasks are periodic
  • we will introduce sporadic and aperiodic processes
  • deadlines are identical with task’s period time \((D = T)\)
  • we will introduce arbitrary deadlines
  • non-pre-emptive scheduling
  • we will introduce (briefly) cooperative scheduling

• tasks are independent
  • we will introduce schedules for interacting tasks
  • processes with lower priorities will always miss their deadlines first.

Graceful degradation features (resource is over-booked):

• EDF can handle higher (full) utilization than FPS.
• FPS is easier to implement and implies less real-time overhead.

Scheduling

Constraints which we used up to here:

• tasks are periodic
• deadlines are identical with task’s period time \((D = T)\)
• pre-emptive scheduling
• worst case execution times are known
• tasks are independent
Tasks with D < T
(Deadline not less than intercritical period)
In fixed priority scheduling FPS change from Rate Monotonic Priority Ordering (RMPO)
to:
Deadline Monotonic Priority Ordering (DMPO)

Lemma
Any task set Q which is schedulable by a FPS scheme W is also schedulable by DMPO!

Static scheduling: Fixed Priority Scheduling (FPS), DMPO

\( t_j \) is still schedulable in W' because:

in W: \( R_i \leq D_i < D_j \leq T_j \Rightarrow R_i < T_j \), i.e. \( t_j \), interfered only once with \( t_j \)
in W: \( t_j \), released once in \( R_i \) and \( R_i < R_j \) = in W'

\( \sum_{j \in \text{Q}} Pi \) = \( \sum_{j \in \text{Q}} Pj \) \( \geq \) \( t_k \)
\( \sum_{j \in \text{Q}} PiPj \) \( \geq \) \( t_kPj \) Pi
\at least one \( t_j \) with \( \sum_{j \in \text{Q}} Pj \) \( \geq \) \( t_k \)
\at least one \( t_j \) with \( \sum_{j \in \text{Q}} PjPj \) \( \geq \) \( t_kPj \)

Proof
1. \( t_i, t_j \) are two tasks in Q, with \( P_i < P_j \) and \( D_j > D_i \) in W = DMPO
2. Generate W' by swapping \( P_i \) and \( P_j \): \( D_j < D_i \) \( \Rightarrow \) \( D_j > D_i \) \( \Rightarrow \) W' = DMPO
3. W' is scheduling Q because:
   a. all \( t_j \) \( \in \) Q with \( P_j \leq P_i \) or \( P_j < P_i \) are unaffected
   b. \( t_j \) is schedulable in W' because \( P_j > P_i \Rightarrow R_i' \leq D_j \)
   c. \( t_j \) is schedulable in W' because
      in W: \( R_i \leq D_i < D_j \leq T_j \Rightarrow R_i < T_j \) i.e. \( t_j \) interfered only once with \( t_j \)
in W: \( t_j \), released once in \( R_i \) and \( R_i < R_j \) = in W'
      in W': \( t_j \), interferes only once with \( t_j \), \( \Rightarrow R_i' \leq D_j \)

Proof
1. \( t_i, t_j \) are two tasks in Q, with \( P_i > P_j \) and \( D_j > D_i \) in W = DMPO
2. Without priority inheritance
3. W' is scheduling Q because:
   a. all \( t_j \) \( \in \) Q with \( P_j < P_i \) or \( P_j < P_i \) are unaffected
   b. \( t_j \) is schedulable in W' because \( P_j > P_i \Rightarrow R_i' < D_j \)
   c. \( t_j \) is schedulable in W' because
      in W': \( t_j \), interferes only once with \( t_j \), \( \Rightarrow R_i' < D_j \)

Proof
1. \( t_i, t_j \) are two tasks in Q, with \( P_i > P_j \) and \( D_j > D_i \) in W = DMPO
2. Without priority inheritance
3. W' is scheduling Q because:
   a. all \( t_j \) \( \in \) Q with \( P_j < P_i \) or \( P_j < P_i \) are unaffected
   b. \( t_j \) is schedulable in W' because \( P_j > P_i \Rightarrow R_i' < D_j \)
   c. \( t_j \) is schedulable in W' because
      in W': \( t_j \), interferes only once with \( t_j \), \( \Rightarrow R_i' < D_j \)
at the end of each code-block, i.e. 'offering' a task-switch.

All critical sections are completely enclosed in one code-block, with no pre-emption of the critical section by any lower priority task.

Schedules are simpler with deferred pre-emption.

Execution times are (a bit) easier to predict.

Maximal blocking time:

\[ B_{\text{max}} \leq t_i \cdot C_i \]

Immediately ceiling priority protocol (POSIX, Ada, RT-Java)

\[ t_i \cdot P_i \geq t_k \cdot C_k \]

Each resource (lock, monitor) has a static ceiling priority, which is inherited by all tasks holding that resource.

\[ \text{Priority Inversion} \]

One additional lock request.

\[ \text{Priority Inversion} \]

Scheduling: Interdependencies

- Immediate ceiling priority protocol
- Priority ceiling protocols
- Pre-emptive dispatching
- Maximal blocking times can be accepted by all tasks
- Immediate ceiling priority protocol

Deferral

\[ \text{Priority Inversion} \]

- Immediate ceiling priority protocol
- Priority ceiling protocols
- Pre-emptive dispatching
- Maximal blocking times can be accepted by all tasks
- Immediate ceiling priority protocol

Scheduling: Interdependencies

- Immediate ceiling priority protocol
- Priority ceiling protocols
- Pre-emptive dispatching
- Maximal blocking times can be accepted by all tasks
- Immediate ceiling priority protocol
Defered pre-emption – Cooperative Scheduling

Response times:
\[ R = B + F + C_k \]

Where:
- \( R \) is the response time
- \( B \) is the load on the system
- \( F \) is the service time
- \( C_k \) is the context switch time

\[ R = B + F + C_k \]

In the simplified case:
\[ C_k = T \]

And \( F \) is the execution time of the final code-block

\[ R = B + T \]

And with \( T = 0 \):

\[ R = B \]

Tasks with \( D > T \)

(Deadline later than inter-arrival period)

Since the inter-arrival time \( D \) is now longer than the cycle time \( T \), more than one release of the task \( i \) needs to be considered.

\[ R_j(q) = R_i(q-1) + T \]

Where \( q \) is the number of releases

\[ R_j(q) = R_i(q-1) + T \]

With \( q \) number of releases

\[ R_j(q) = R_i(q) - (q-1)T \]

\[ R_j(q) = R_i(q) - (q-1)T \]

by introducing further assumptions about the granularity of the period times and deadlines:

Some tasks can be scheduled by introducing offsets to the release times, but …

\[ \text{without any further restrictions this problem is NP} \]-hard!

\[ \text{without any further restrictions this problem is NP} \]-hard!

• Task and interrupt priorities (static, dynamic, active)
• Prioritized entry queues
• Priority ceiling locking (PCL)
• Schedules (at least EDF within priorities pre-emptions are requested)

Ada95 provides:
- Task and interrupt priorities (static, dynamic, active)
- Task attributes
- Prioritized entry queues
- Priority ceiling locking (PCL)
- Schedules (at least EDF within priorities pre-emptions are requested)

Ada95 does not provide:
- Earliest Deadline First (EDF)
- Specific servers (Ada95 implementation of a specific server is on the course page)
- Direct execution time measurements e.g. POSIX or V��ble times

POSIX provides:
- Task and interrupt priorities (static, dynamic, active)
- Prioritized entry queues
- Priority ceiling locking (PCL)
- Schedules, priority based with at least:
  - EDF
  - Round Robin
  - Specific server, possibly others

Threads can be:
- System controlled or
- Process controlled (priority scheduling similar to the case)

Times

Real-Time Java provides:
- Task priorities (static, dynamic, active)
- Prioritized message queues
- Priority ceiling locking (PCL)
- Schedulable objects associated with threads with:
  - Memory, wide, and scheduling parameters
- Pre-emptive priority-oriented dispatching, possibly with a feasibility analysis
- An extendible scheduler class = dynamic scheduling

Real-Time Java does not (necessarily) provide:
- Earliest Deadline First (EDF)
- Specific servers
- Direct task execution time measurements (might be provided)

Real-Time Java

Value of the schedulability analysis: complexity can be reduced to be realistic.
public abstract class SchedulingParameters
{
    public SchedulingParameters ();
}

public class PriorityParameters extends SchedulingParameters
{
    public PriorityParameters (int priority);
    public int getPriority ();
    public void setPriority (int priority) throws …;
}

Priority is the only default scheduling parameter

public abstract class ReleaseParameters
{

}

Cost is an estimate of the max. execution time
Measuring execution time is not requested, i.e. the overrunHandler might never be activated!

public class PeriodicParameters extends ReleaseParameters
{
    public PeriodicParameters              (HighResolutionTime start,
                                            RelativeTime       period,
                                            RelativeTime       cost,
                                            RelativeTime       deadline,
                                            AsyncEventHandler  overrunHandler,
                                            AsyncEventHandler  missHandler);
    public RelativeTime       getPeriod ();
    public HighResolutionTime getStart ();
    public void               setPeriod (RelativeTime period);
    public void               setStart (HighResolutionTime start);
}

most frequently used release parameters

public class AperiodicParameters extends ReleaseParameters
{
    public AperiodicParameters             (RelativeTime      cost,
                                            RelativeTime      deadline,
                                            AsyncEventHandler overrunHandler,
                                            AsyncEventHandler missHandler);
}

these are the minimum release parameters (while cost might be used for feasibility analysis only)

public class SporadicParameters extends AperiodicParameters
{
    public SporadicParameters             (RelativeTime      minInterarrival,
                                            RelativeTime      cost,
                                            RelativeTime      deadline,
                                            AsyncEventHandler overrunHandler,
                                            AsyncEventHandler missHandler);
    public RelativeTime getMinimumInterarrival ();
    public void         setMinimumInterarrival (RelativeTime minimum);
}

Sporadic events are not allowed to come in bursts!

Summary

• Basic real-time scheduling
  • Fixed Priority Scheduling (FPS) with
    • Rate Monotonic (RM) Deadline Monotonic Priority Ordering (DMPO)
  • Earliest Deadline First (EDF)

• Real-world extensions
  • Aperiodic, sporadic, soft real-time tasks
  • Deadlines shorter than period
  • Cooperative and deferred pre-emption scheduling
  • Real-time service in terms of exception handling considerations
  • Synchronized tasks (priority inheritance, priority ceiling protocols)

• Language support
  • Ada95, POSIX = static, off-line analysis mostly — RT-Java = on-line, dynamic scheduling