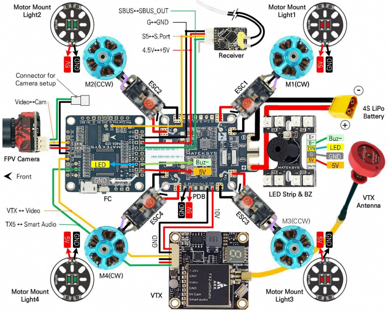
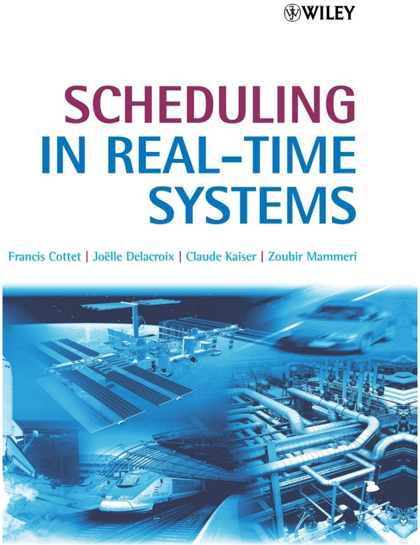


Scheduling of independent tasks

Kizito NKURIKIYEYEZU, Ph.D.

Readings

- Read Chapter 2, section 2.1 (pages 23-33) of Cottet et al. (2002). Scheduling in Real-Time Systems. Skip other sections!
- Topics
  - rate monotonic
  - inverse deadline
  - earliest deadline first
  - least laxity first
  - On-line scheduling



[1] Readings are based on Cottet, F., Delacroix, J., Mammeri, Z., & Kaiser, C. (2002). Scheduling in Real-Time Systems. Wiley.

Review

Recurrent Task Models

- When job with the same computation requirements are released recurrently, they jobs can be modeled by a recurrent task

Recurrent Task Models

- When job with the same computation requirements are released recurrently, they jobs can be modeled by a recurrent task
- Periodic Task τ_i
  - A job is released exactly and periodically by a period T_i

Recurrent Task Models

- When job with the same computation requirements are released recurrently, they jobs can be modeled by a recurrent task
- Periodic Task τ_i
  - A job is released exactly and periodically by a period T_i
  - A phase ϕ_i indicates when the first job is released

Recurrent Task Models

- When job with the same computation requirements are released recurrently, they jobs can be modeled by a recurrent task
- Periodic Task τ_i
  - A job is released exactly and periodically by a period T_i
  - A phase ϕ_i indicates when the first job is released
  - A relative deadline D_i for each job from task τ_i

Recurrent Task Models

- When job with the same computation requirements are released recurrently, they jobs can be modeled by a recurrent task
- Periodic Task τ_i
  - A job is released exactly and periodically by a period T_i
  - A phase ϕ_i indicates when the first job is released
  - A relative deadline D_i for each job from task τ_i
  - $(\phi_i\ , C_i\ , T_i\ , D_i\ )$ is the specification of periodic task τ_i, where C_i is the worst-case execution time. When ϕ_i is omitted, we assume $\phi_i = 0$.

Recurrent Task Models

- When job with the same computation requirements are released recurrently, they jobs can be modeled by a recurrent task
- Periodic Task τ_i
  - A job is released exactly and periodically by a period T_i
  - A phase ϕ_i indicates when the first job is released
  - A relative deadline D_i for each job from task τ_i
  - (ϕ_i, C_i, T_i, D_i) is the specification of periodic task τ_i, where C_i is the worst-case execution time. When ϕ_i is omitted, we assume $\phi_i = 0$.

Recurrent Task Models

- When job with the same computation requirements are released recurrently, they jobs can be modeled by a recurrent task
- Periodic Task τ_i
  - A job is released exactly and periodically by a period T_i
  - A phase ϕ_i indicates when the first job is released
  - A relative deadline D_i for each job from task τ_i
  - $(\phi_i , C_i , T_i , D_i)$ is the specification of periodic task τ_i, where C_i is the worst-case execution time. When ϕ_i is omitted, we assume $\phi_i = 0$.
- Sporadic Task τ_i
  - T_i is the minimal time between any two consecutive job releases

Recurrent Task Models

- When job with the same computation requirements are released recurrently, they jobs can be modeled by a recurrent task
- Periodic Task τ_i
  - A job is released exactly and periodically by a period T_i
  - A phase ϕ_i indicates when the first job is released
  - A relative deadline D_i for each job from task τ_i
  - (ϕ_i , C_i ,T_i ,D_i) is the specification of periodic task τ_i, where C_i is the worst-case execution time. When ϕ_i is omitted, we assume $\phi_i = 0$.
- Sporadic Task τ_i
  - T_i is the minimal time between any two consecutive job releases
  - A relative deadline D_i for each job from task τ_i

Recurrent Task Models

- When job with the same computation requirements are released recurrently, they jobs can be modeled by a recurrent task
- Periodic Task τ_i
  - A job is released exactly and periodically by a period T_i
  - A phase ϕ_i indicates when the first job is released
  - A relative deadline D_i for each job from task τ_i
  - (ϕ_i , C_i ,T_i ,D_i) is the specification of periodic task τ_i, where C_i is the worst-case execution time. When ϕ_i is omitted, we assume $\phi_i = 0$.
- Sporadic Task τ_i
  - T_i is the minimal time between any two consecutive job releases
  - A relative deadline D_i for each job from task τ_i
  - (C_i ,T_i ,D_i) is the specification of sporadic task τ_i, where C_i is the worst-case execution time.

Real-time task model

- r_i, task release time, i.e. the execution request time.

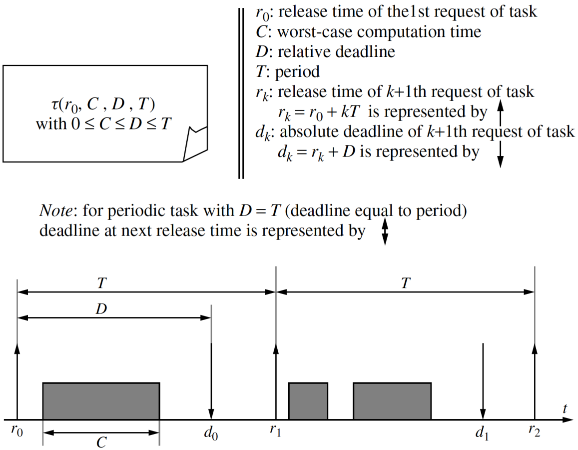


FIG 1. Task model

Real-time task model

- r_i, task release time, i.e. the execution request time.
- C_i, task worst-case computation time.

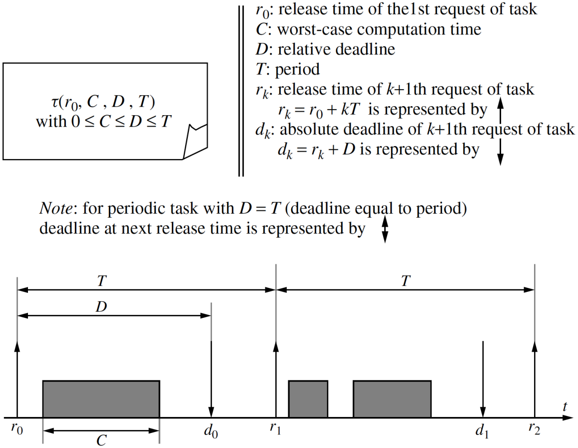


FIG 1. Task model

Real-time task model

- r_i, task release time, i.e. the execution request time.
- C_i, task worst-case computation time.
- D_i, task relative deadline, i.e. the maximum acceptable delay for its processing.

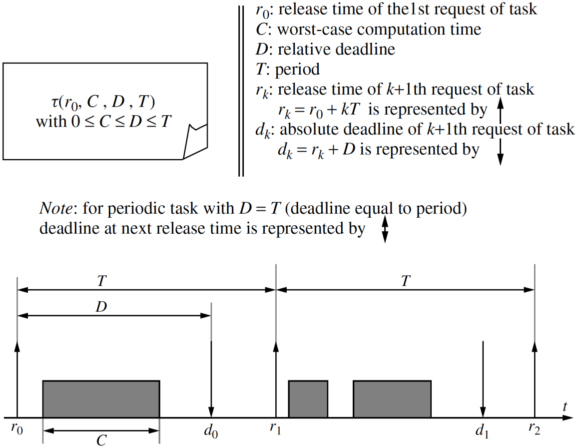


FIG 1. Task model

Real-time task model

- r_i, task release time, i.e. the execution request time.
- C_i, task worst-case computation time.
- D_i, task relative deadline, i.e. the maximum acceptable delay for its processing.
- T_i , task period (valid only for periodic tasks).

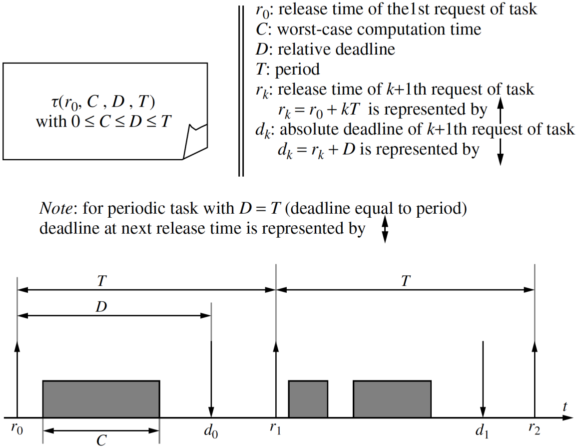


FIG 1. Task model

Real-time task model

- r_i, task release time, i.e. the execution request time.
- C_i, task worst-case computation time.
- D_i, task relative deadline, i.e. the maximum acceptable delay for its processing.
- T_i , task period (valid only for periodic tasks).
- Absolute deadline $d_i = r_i + D_i$—transgression of the absolute deadline causes a timing fault.

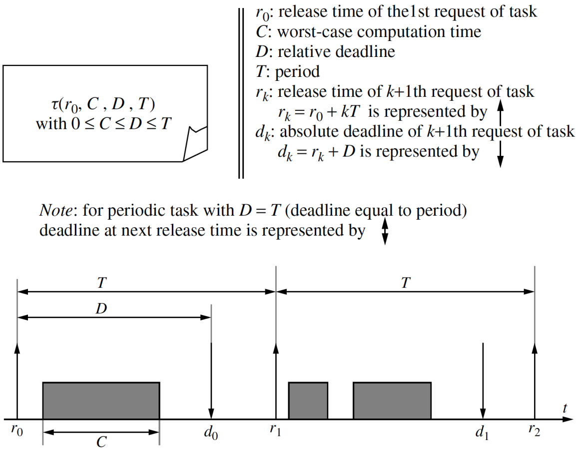


FIG 1. Task model

Relative Deadline vs Period

When we have a task set, we say that the task set is with

- implicity deadline when the relative deadline D_i is equal to the period T_i, i.e., $D_i = T_i$ for every task τ_i constrained deadline when the relative deadline D_i is no more than the period T_i , i.e., $D_i \leq T_i$, for every task τ_i

Relative Deadline vs Period

When we have a task set, we say that the task set is with

- implicity deadline when the relative deadline D_i is equal to the period T_i, i.e., $D_i = T_i$ for every task τ_i constrained deadline when the relative deadline D_i is no more than the period T_i , i.e., $D_i \leq T_i$, for every task τ_i
- arbitrary deadline when the relative deadline D_i could be larger than the period T_i for some task τ_i

Sporadic and Periodic Tasks

- For periodic taks
  - Syncronous system—each task τ_i has a phase of 0, i.e., $\phi_i = 0$

- Hyperperiod: Least common multiple (LCM) of T_i
- Task utilization of task

$$u_i = \frac{C_i}{T_i} \tag{1}$$

- Total system utilization

$$U = \sum_{i=1}^{n} u_i \tag{2}$$

Sporadic and Periodic Tasks

- For periodic taks
  - Syncronous system—each task τ_i has a phase of 0, i.e., $\phi_i = 0$
  - Asynchronous system—the phase are arbitrary
- Hyperperiod: Least common multiple (LCM) of T_i
- Task utilization of task

$$u_i = \frac{C_i}{T_i} \tag{1}$$

- Total system utilization

$$U = \sum_{i=1}^{n} u_i \tag{2}$$

Scheduling of independent tasks

On-line algorithms for periodic tasks

simple rule—that assigns priorities according to temporal parameters of tasks.

- static—the priority is fixed. the priorities are assigned to tasks before execution and do not change over time. For example:
  - rate monotonic (Liu and Layland, 1973)[1]

[1] Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1), 46-61.

[2] Leung, J. Y. T., & Merrill, A. M. (1980). A note on preemptive scheduling of periodic, real-time tasks. Information processing letters, 11(3), 115-118

[3] Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1), 46-61.

[4] Dhall, S. K. (1977). Scheduling periodic-time-critical jobs on single processor and multiprocessor computing systems. University of Illinois at Urbana-Champaign.

On-line algorithms for periodic tasks

simple rule—that assigns priorities according to temporal parameters of tasks.

- static—the priority is fixed. the priorities are assigned to tasks before execution and do not change over time. For example:
  - rate monotonic (Liu and Layland, 1973)[1]
  - inverse deadline or deadline monotonic (Leung and Merrill, 1980)[2]

---

[1] Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1), 46-61.

[2] Leung, J. Y. T., & Merrill, A. M. (1980). A note on preemptive scheduling of periodic, real-time tasks. Information processing letters, 11(3), 115-118

[3] Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1), 46-61.

[4] Dhall, S. K. (1977). Scheduling periodic-time-critical jobs on single processor and multiprocessor computing systems. University of Illinois at Urbana-Champaign.

On-line algorithms for periodic tasks

simple rule—that assigns priorities according to temporal parameters of tasks.

- static—the priority is fixed. the priorities are assigned to tasks before execution and do not change over time. For example:
  - rate monotonic (Liu and Layland, 1973)[1]
  - inverse deadline or deadline monotonic (Leung and Merrill, 1980)[2]

---

[1] Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1), 46-61.

[2] Leung, J. Y. T., & Merrill, A. M. (1980). A note on preemptive scheduling of periodic, real-time tasks. Information processing letters, 11(3), 115-118

[3] Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1), 46-61.

[4] Dhall, S. K. (1977). Scheduling periodic-time-critical jobs on single processor and multiprocessor computing systems. University of Illinois at Urbana-Champaign.

On-line algorithms for periodic tasks

simple rule—that assigns priorities according to temporal parameters of tasks.

- static—the priority is fixed. the priorities are assigned to tasks before execution and do not change over time. For example:
  - rate monotonic (Liu and Layland, 1973)[1]
  - inverse deadline or deadline monotonic (Leung and Merrill, 1980)[2]
- dynamic—scheduling algorithm is based on variable parameters, i.e. absolute task deadlines
  - earliest deadline first (Liu and Layland, 1973)[3]

---

[1] Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1), 46-61.

[2] Leung, J. Y. T., & Merrill, A. M. (1980). A note on preemptive scheduling of periodic, real-time tasks. Information processing letters, 11(3), 115-118

[3] Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1), 46-61.

[4] Dhall, S. K. (1977). Scheduling periodic-time-critical jobs on single processor and multiprocessor computing systems. University of Illinois at Urbana-Champaign.

On-line algorithms for periodic tasks

simple rule—that assigns priorities according to temporal parameters of tasks.

- static—the priority is fixed. the priorities are assigned to tasks before execution and do not change over time. For example:
  - rate monotonic (Liu and Layland, 1973)[1]
  - inverse deadline or deadline monotonic (Leung and Merrill, 1980)[2]
- dynamic—scheduling algorithm is based on variable parameters, i.e. absolute task deadlines
  - earliest deadline first (Liu and Layland, 1973)[3]
  - least laxity first (Dhall, 1977; Sorenson, 1974)[4]

---

[1] Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1), 46-61.

[2] Leung, J. Y. T., & Merrill, A. M. (1980). A note on preemptive scheduling of periodic, real-time tasks. Information processing letters, 11(3), 115-118

[3] Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1), 46-61.

[4] Dhall, S. K. (1977). Scheduling periodic-time-critical jobs on single processor and multiprocessor computing systems. University of Illinois at Urbana-Champaign.

Rate monotonic scheduling

summary—with the rate monotonic (RM) algorithm, tasks with shorter periods (higher request rates) get higher priorities. Task with smallest time period have highest priority and a task with longest time period have the lowest priority

- Priorities are fixed and are decided before start of execution and does not change over time

Rate monotonic scheduling

summary—with the rate monotonic (RM) algorithm, tasks with shorter periods (higher request rates) get higher priorities. Task with smallest time period have highest priority and a task with longest time period have the lowest priority

- Priorities are fixed and are decided before start of execution and does not change over time
- Priority of a task is inversely proportional to its timer period.

Rate monotonic scheduling

summary—with the rate monotonic (RM) algorithm, tasks with shorter periods (higher request rates) get higher priorities. Task with smallest time period have highest priority and a task with longest time period have the lowest priority

- Priorities are fixed and are decided before start of execution and does not change over time
- Priority of a task is inversely proportional to its timer period.
- For a set of n periodic tasks, a feasible RM schedule exists if the CPU utilization, U, is below a specific bound (Equation (3))

$$U = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \cdot n \left(2^{\frac{1}{n}} - 1\right) \tag{3}$$

where:

Rate monotonic scheduling

summary—with the rate monotonic (RM) algorithm, tasks with shorter periods (higher request rates) get higher priorities. Task with smallest time period have highest priority and a task with longest time period have the lowest priority

- Priorities are fixed and are decided before start of execution and does not change over time
- Priority of a task is inversely proportional to its timer period.
- For a set of n periodic tasks, a feasible RM schedule exists if the CPU utilization, U, is below a specific bound (Equation (3))

$$U = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \cdot n \left(2^{\frac{1}{n}} - 1\right) \tag{3}$$

where:

- U—utilization factor

Rate monotonic scheduling

summary—with the rate monotonic (RM) algorithm, tasks with shorter periods (higher request rates) get higher priorities. Task with smallest time period have highest priority and a task with longest time period have the lowest priority

- Priorities are fixed and are decided before start of execution and does not change over time
- Priority of a task is inversely proportional to its timer period.
- For a set of n periodic tasks, a feasible RM schedule exists if the CPU utilization, U, is below a specific bound (Equation (3))

$$U = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \cdot n \left(2^{\frac{1}{n}} - 1\right) \tag{3}$$

where:

- U—utilization factor
- C_i —computation time for task τ_i

Rate monotonic scheduling

summary—with the rate monotonic (RM) algorithm, tasks with shorter periods (higher request rates) get higher priorities. Task with smallest time period have highest priority and a task with longest time period have the lowest priority

- Priorities are fixed and are decided before start of execution and does not change over time
- Priority of a task is inversely proportional to its timer period.
- For a set of n periodic tasks, a feasible RM schedule exists if the CPU utilization, U, is below a specific bound (Equation (3))

$$U = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \cdot n \left(2^{\frac{1}{n}} - 1\right) \tag{3}$$

where:

- U—utilization factor
- C_i —computation time for task τ_i
- T_i —release period for task τ_i

Rate monotonic scheduling

summary—with the rate monotonic (RM) algorithm, tasks with shorter periods (higher request rates) get higher priorities. Task with smallest time period have highest priority and a task with longest time period have the lowest priority

- Priorities are fixed and are decided before start of execution and does not change over time
- Priority of a task is inversely proportional to its timer period.
- For a set of n periodic tasks, a feasible RM schedule exists if the CPU utilization, U, is below a specific bound (Equation (3))

$$U = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \cdot n \left(2^{\frac{1}{n}} - 1\right) \tag{3}$$

where:

- U—utilization factor
- C_i —computation time for task τ_i
- T_i —release period for task τ_i
- n —number of tasks to be scheduled.

Rate monotonic scheduling

- For two tasks (i.e., $n = 2$), the upper bounds on utilization is (Equation (4))

$$n\left(2^{\frac{1}{2}} - 1\right) = 2\left(\sqrt{2} - 1\right) = 0.828 \tag{4}$$

Rate monotonic scheduling

- For two tasks (i.e., $n = 2$), the upper bounds on utilization is (Equation (4))

$$n\left(2^{\frac{1}{2}} - 1\right) = 2\left(\sqrt{2} - 1\right) = 0.828 \tag{4}$$

- For a large number of tasks (i.e., $n \rightarrow \infty$), the upper bound is

$$U \leq \lim_{n\to\infty} n\left(2^{\frac{1}{n}} - 1\right) = ln(2) = 0.693 \tag{5}$$

Rate monotonic scheduling

- For two tasks (i.e., $n = 2$), the upper bounds on utilization is (Equation (4))

$$n\left(2^{\frac{1}{2}} - 1\right) = 2\left(\sqrt{2} - 1\right) = 0.828 \tag{4}$$

- For a large number of tasks (i.e., $n \to \infty$), the upper bound is

$$U \leq \lim_{n \to \infty} n\left(2^{\frac{1}{n}} - 1\right) = ln(2) = 0.693 \tag{5}$$

- As a general rule, when $n > 10$, the RMS can meet its deadlines if $U < 70\%$

Rate monotonic scheduling—Example 1

- According to RM scheduling algorithm task with shorter period has higher priority so τ_2 has the highest priority, τ_3 an intermediate priority and τ_1 the lowest priority

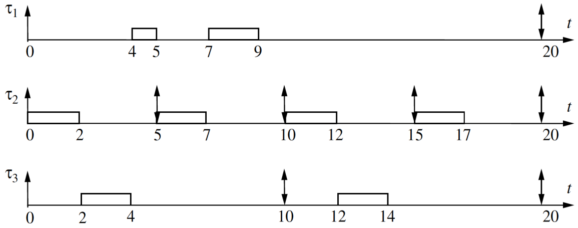


FIG 2. Example of a rate monotonic schedule with three periodic tasks: $\tau_1(0, 3, 20, 20)$, $\tau_2(0, 2, 5, 5)$ and $\tau_3(0, 2, 10, 10)$

Rate monotonic scheduling—Example 1

- According to RM scheduling algorithm task with shorter period has higher priority so τ_2 has the highest priority, τ_3 an intermediate priority and τ_1 the lowest priority
- At $t = 0$, all the tasks are released. Now τ_2 (highest priority task)executes first till $t = 2$.

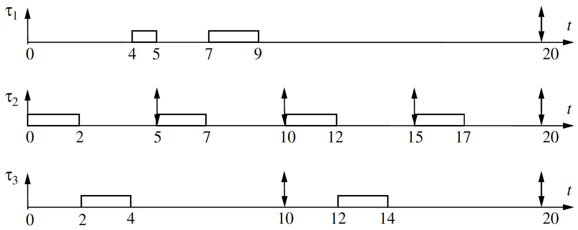


FIG 2. Example of a rate monotonic schedule with three periodic tasks: $\tau_1(0, 3, 20, 20)$, $\tau_2(0, 2, 5, 5)$ and $\tau_3(0, 2, 10, 10)$

Rate monotonic scheduling—Example 1

- According to RM scheduling algorithm task with shorter period has higher priority so τ_2 has the highest priority, τ_3 an intermediate priority and τ_1 the lowest priority
- At $t = 0$, all the tasks are released. Now τ_2 (highest priority task)executes first till $t = 2$.
- At $t = 2$ τ_3 (intermediate priority) executes second until $t = 4$

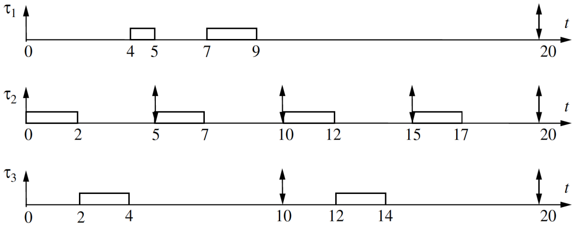


FIG 2. Example of a rate monotonic schedule with three periodic tasks: $\tau_1(0, 3, 20, 20)$, $\tau_2(0, 2, 5, 5)$ and $\tau_3(0, 2, 10, 10)$

Rate monotonic scheduling

- After τ_2 completes, the lowest priority task, τ_1, executes until $t = 5$

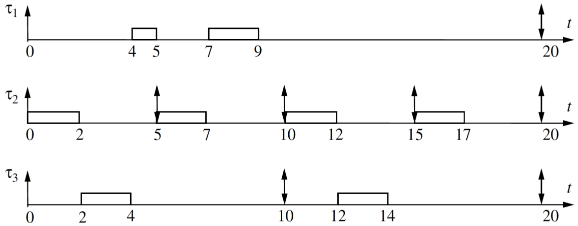


FIG 3. Example of a rate monotonic schedule with three periodic tasks: $\tau_1(0, 3, 20, 20)$, $\tau_2(0, 2, 5, 5)$ and $\tau_3(0, 2, 10, 10)$

Rate monotonic scheduling

- After τ_2 completes, the lowest priority task, τ_1, executes until $t = 5$
- At $t = 5$, τ_2 is released, and since it has higher priority that τ_1, it preempts τ_1 and starts its execution until completion at $t = 7$

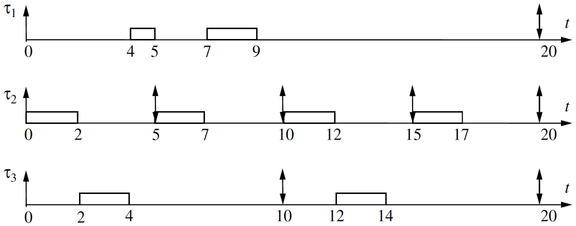


FIG 3. Example of a rate monotonic schedule with three periodic tasks: $\tau_1(0, 3, 20, 20)$, $\tau_2(0, 2, 5, 5)$ and $\tau_3(0, 2, 10, 10)$

Rate monotonic scheduling

- After τ_2 completes, the lowest priority task, τ_1, executes until $t = 5$
- At $t = 5$, τ_2 is released, and since it has higher priority that τ_1, it preempts τ_1 and starts its execution until completion at $t = 7$
- etc...

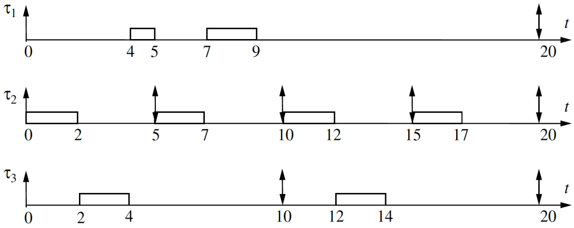


FIG 3. Example of a rate monotonic schedule with three periodic tasks: $\tau_1(0, 3, 20, 20)$, $\tau_2(0, 2, 5, 5)$ and $\tau_3(0, 2, 10, 10)$

Rate monotonic scheduling

- After τ_2 completes, the lowest priority task, τ_1, executes until $t = 5$
- At $t = 5$, τ_2 is released, and since it has higher priority that τ_1, it preempts τ_1 and starts its execution until completion at $t = 7$
- etc...

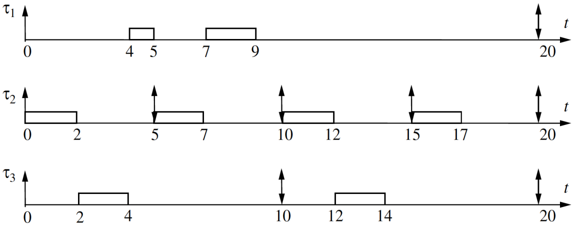


FIG 3. Example of a rate monotonic schedule with three periodic tasks: $\tau_1(0, 3, 20, 20)$, $\tau_2(0, 2, 5, 5)$ and $\tau_3(0, 2, 10, 10)$

Rate monotonic scheduling

- After τ_2 completes, the lowest priority task, τ_1, executes until $t = 5$
- At $t = 5$, τ_2 is released, and since it has higher priority that τ_1, it preempts τ_1 and starts its execution until completion at $t = 7$
- etc...

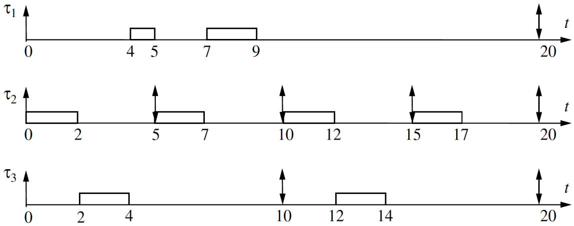


FIG 3. Example of a rate monotonic schedule with three periodic tasks: $\tau_1(0, 3, 20, 20)$, $\tau_2(0, 2, 5, 5)$ and $\tau_3(0, 2, 10, 10)$

The three tasks meet their deadline since the utilization factors

$$U = \frac{3}{20} + \frac{2}{5} + \frac{2}{10} = 0.75 \leq 3(2^{\frac{1}{3}} - 1) = 0.779 \tag{6}$$

Rate monotonic scheduling —Example 2

- In this example, we have a set of three periodic tasks for which the relative deadline is equal to the period

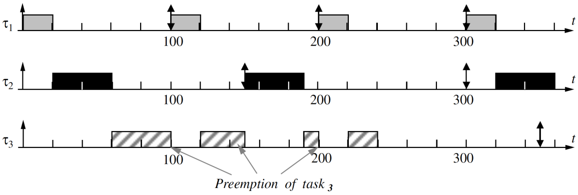


FIG 4. Example of a rate monotonic schedule with three periodic tasks: τ_1 (0, 20, 100, 100), τ_2(0, 40, 150, 150) and τ_3(0, 100, 350, 350)

Rate monotonic scheduling —Example 2

- In this example, we have a set of three periodic tasks for which the relative deadline is equal to the period
- Task τ_1 has the highest priority and task τ_3 has the lowest priority.

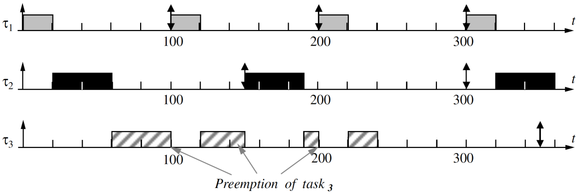


FIG 4. Example of a rate monotonic schedule with three periodic tasks: τ_1 (0, 20, 100, 100), τ_2(0, 40, 150, 150) and τ_3(0, 100, 350, 350)

Rate monotonic scheduling —Example 2

- In this example, we have a set of three periodic tasks for which the relative deadline is equal to the period
- Task τ_1 has the highest priority and task τ_3 has the lowest priority.
- The major cycle of the task set is LCM(100, 150, 350) = 2100.

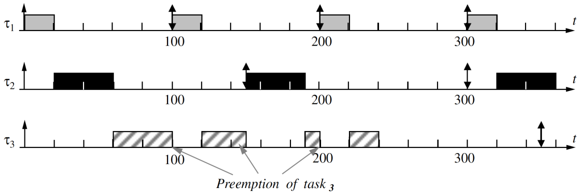


FIG 4. Example of a rate monotonic schedule with three periodic tasks: τ_1 (0, 20, 100, 100), τ_2(0, 40, 150, 150) and τ_3(0, 100, 350, 350)

Rate monotonic scheduling —Example 2

- In this example, we have a set of three periodic tasks for which the relative deadline is equal to the period
- Task τ_1 has the highest priority and task τ_3 has the lowest priority.
- The major cycle of the task set is LCM(100, 150, 350) = 2100.

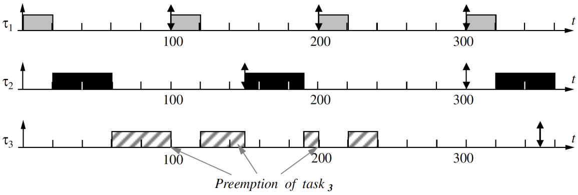


FIG 4. Example of a rate monotonic schedule with three periodic tasks: τ_1 (0, 20, 100, 100), τ_2(0, 40, 150, 150) and τ_3(0, 100, 350, 350)

Rate monotonic scheduling —Example 2

- In this example, we have a set of three periodic tasks for which the relative deadline is equal to the period
- Task τ_1 has the highest priority and task τ_3 has the lowest priority.
- The major cycle of the task set is LCM(100, 150, 350) = 2100.

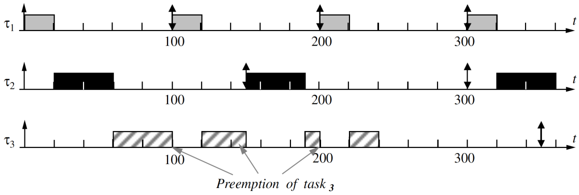


FIG 4. Example of a rate monotonic schedule with three periodic tasks: τ_1 (0, 20, 100, 100), τ_2(0, 40, 150, 150) and τ_3(0, 100, 350, 350)

The processor utilization factor is:

$$U = \frac{20}{100} + \frac{40}{150} + \frac{100}{350} = 0.75 < 3 \cdot (\sqrt[3]{2} - 1) = 0.779 \tag{7}$$

So this task set is schedulable. All the three tasks meet their deadlines.

Rate monotonic scheduling

- Due to priority assignment based on the periods of tasks, the RM algorithm should be used to schedule tasks with relative deadlines equal to periods.

Rate monotonic scheduling

- Due to priority assignment based on the periods of tasks, the RM algorithm should be used to schedule tasks with relative deadlines equal to periods.
- This is the case where the sufficient condition (Equation (3)) can be used.

Rate monotonic scheduling

- Due to priority assignment based on the periods of tasks, the RM algorithm should be used to schedule tasks with relative deadlines equal to periods.
- This is the case where the sufficient condition (Equation (3)) can be used.
- For tasks with relative deadlines not equal to periods, the inverse deadline algorithm should be used.

Rate monotonic scheduling

- Due to priority assignment based on the periods of tasks, the RM algorithm should be used to schedule tasks with relative deadlines equal to periods.
- This is the case where the sufficient condition (Equation (3)) can be used.
- For tasks with relative deadlines not equal to periods, the inverse deadline algorithm should be used.
- The RMS can meet all of the deadlines if total CPU utilization, $U \leq 70\%$. The other 30% of the CPU can be dedicated to lower-priority, non-real-time tasks.

Rate monotonic scheduling

- Due to priority assignment based on the periods of tasks, the RM algorithm should be used to schedule tasks with relative deadlines equal to periods.
- This is the case where the sufficient condition (Equation (3)) can be used.
- For tasks with relative deadlines not equal to periods, the inverse deadline algorithm should be used.
- The RMS can meet all of the deadlines if total CPU utilization, $U \leq 70\%$. The other 30% of the CPU can be dedicated to lower-priority, non-real-time tasks.
- For smaller values of n or in cases where U is close to this estimate, the calculated utilization bound should be used.

Inverse (monotonic) deadline algorithm

Deadline-monotonic scheduling

summary—Deadline-monotonic priority assignment is a priority assignment policy used with fixed-priority pre-emptive scheduling[5]

- Allows a weakening of the condition which requires equality between periods and deadlines in static-priority schemes.

[5] https://en.wikipedia.org/wiki/Fixed-priority_pre-emptive_scheduling

[6] Audsley, N. C., Burns, A., & Wellings, A. J. (1993). Deadline monotonic scheduling theory and application. Control Engineering Practice, 1(1), 71–78. https://doi.org/10.1016/0967-0661(93)92105-D

Deadline-monotonic scheduling

summary—Deadline-monotonic priority assignment is a priority assignment policy used with fixed-priority pre-emptive scheduling[5]

- Allows a weakening of the condition which requires equality between periods and deadlines in static-priority schemes.
- The task with the shortest relative deadline is assigned the highest priority[6]

[5] https://en.wikipedia.org/wiki/Fixed-priority_pre-emptive_scheduling

[6] Audsley, N. C., Burns, A., & Wellings, A. J. (1993). Deadline monotonic scheduling theory and application. Control Engineering Practice, 1(1), 71–78. https://doi.org/10.1016/0967-0661(93)92105-D

Deadline-monotonic scheduling

summary—Deadline-monotonic priority assignment is a priority assignment policy used with fixed-priority pre-emptive scheduling[5]

- Allows a weakening of the condition which requires equality between periods and deadlines in static-priority schemes.
- The task with the shortest relative deadline is assigned the highest priority[6]
- For an arbitrary set of n tasks with deadlines shorter than periods, a sufficient condition is given in Equation (8)

$$U = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq n \cdot \left(2^{\frac{1}{n}} - 1\right) \tag{8}$$

[5] https://en.wikipedia.org/wiki/Fixed-priority_pre-emptive_scheduling

[6] Audsley, N. C., Burns, A., & Wellings, A. J. (1993). Deadline monotonic scheduling theory and application. Control Engineering Practice, 1(1), 71–78. https://doi.org/10.1016/0967-0661(93)92105-D

Deadline-monotonic scheduling—Example

- Task τ_2 has the highest priority and task τ_3 the lowest.

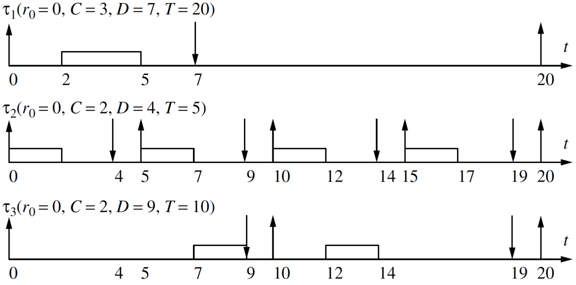


FIG 5. Inverse deadline schedule for a set of three periodic tasks τ_1 $(r_0 = 0, C = 3, D = 7, T = 20)$, $\tau_2(r_0 = 0, C = 2, D = 4, T = 5)$ and $\tau_3(r_0 = 0, C = 2, D = 9, T = 10)$

$$U = \frac{3}{7} + \frac{2}{4} + \frac{2}{9} = 1.15 > 3(\sqrt[3]{2} - 1) = 0.779 \qquad (9)$$

Deadline-monotonic scheduling—Example

- Task τ_2 has the highest priority and task τ_3 the lowest.
- The sufficient condition in Equation (8) is not satisfied because the processor load factor is $1.15 > 0.779$ (Equation (9))

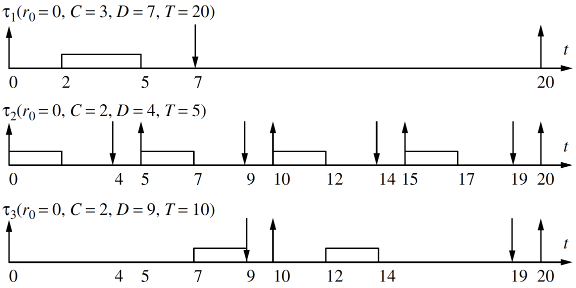


FIG 5. Inverse deadline schedule for a set of three periodic tasks τ_1 ($r_0 = 0, C = 3, D = 7, T = 20$), $\tau_2(r_0 = 0, C = 2, D = 4, T = 5)$ and $\tau_3(r_0 = 0, C = 2, D = 9, T = 10)$

$$U = \frac{3}{7} + \frac{2}{4} + \frac{2}{9} = 1.15 > 3(\sqrt[3]{2} - 1) = 0.779 \qquad (9)$$

Deadline-monotonic scheduling—Example

- Task τ_2 has the highest priority and task τ_3 the lowest.
- The sufficient condition in Equation (8) is not satisfied because the processor load factor is $1.15 > 0.779$ (Equation (9))
- However, the task set is schedulable because the schedule is given within the major cycle of the task set.

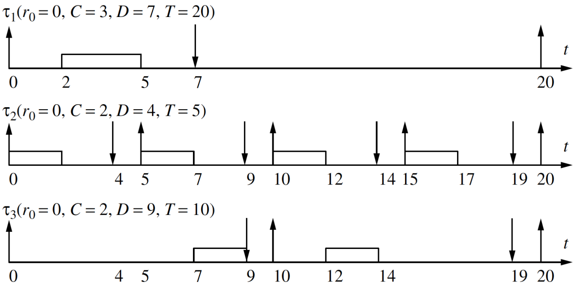


FIG 5. Inverse deadline schedule for a set of three periodic tasks τ_1 ($r_0 = 0$, $C = 3$, $D = 7$, $T = 20$), $\tau_2(r_0 = 0, C = 2, D = 4, T = 5)$ and $\tau_3(r_0 = 0, C = 2, D = 9, T = 10)$

$$U = \frac{3}{7} + \frac{2}{4} + \frac{2}{9} = 1.15 > 3(\sqrt[3]{2} - 1) = 0.779 \tag{9}$$

Earliest deadline first algorithm

Earliest deadline first algorithm

summary—the earliest deadline first (EDF) algorithm assigns priority to tasks according to their absolute deadline: the task with the earliest deadline will be executed as the highest priority.

- The EDF algorithm does not make any assumption about the periodicity of the tasks; hence it can be used for scheduling periodic as well as aperiodic tasks.

Earliest deadline first algorithm

summary—the earliest deadline first (EDF) algorithm assigns priority to tasks according to their absolute deadline: the task with the earliest deadline will be executed as the highest priority.

- The EDF algorithm does not make any assumption about the periodicity of the tasks; hence it can be used for scheduling periodic as well as aperiodic tasks.
- A necessary and sufficient schedulability condition exists for periodic tasks with deadlines equal to periods.

Earliest deadline first algorithm

summary—the earliest deadline first (EDF) algorithm assigns priority to tasks according to their absolute deadline: the task with the earliest deadline will be executed as the highest priority.

- The EDF algorithm does not make any assumption about the periodicity of the tasks; hence it can be used for scheduling periodic as well as aperiodic tasks.
- A necessary and sufficient schedulability condition exists for periodic tasks with deadlines equal to periods.
- A set of periodic tasks with deadlines equal to periods is schedulable with the EDF algorithm if and only if the processor utilization factor is less than or equal to 1 (Equation (10))

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{10}$$

Earliest deadline first algorithm

summary—the earliest deadline first (EDF) algorithm assigns priority to tasks according to their absolute deadline: the task with the earliest deadline will be executed as the highest priority.

- The EDF algorithm does not make any assumption about the periodicity of the tasks; hence it can be used for scheduling periodic as well as aperiodic tasks.
- A necessary and sufficient schedulability condition exists for periodic tasks with deadlines equal to periods.
- A set of periodic tasks with deadlines equal to periods is schedulable with the EDF algorithm if and only if the processor utilization factor is less than or equal to 1 (Equation (10))

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{10}$$

- A hybrid task set is schedulable with the EDF algorithm if (Equation (11)):

$$U = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1 \tag{11}$$

Earliest deadline first algorithm—Example

- At time $t = 0$, the three tasks are ready to execute and the task with the smallest absolute deadline is τ_2.

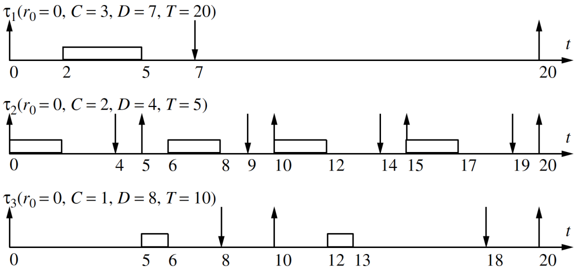


FIG 6. EDF EDF schedule for a set of three periodic tasks $\tau_1(r_0 = 0, C = 3, D = 7, 20 = T)$,$\tau_2(r_0 = 0, C = 2, D = 4, T = 5)$, $\tau_3(r_0 = 0, C = 1, D = 8, T = 10)$

Earliest deadline first algorithm—Example

- At time $t = 0$, the three tasks are ready to execute and the task with the smallest absolute deadline is τ_2.
- τ_2 is executed.

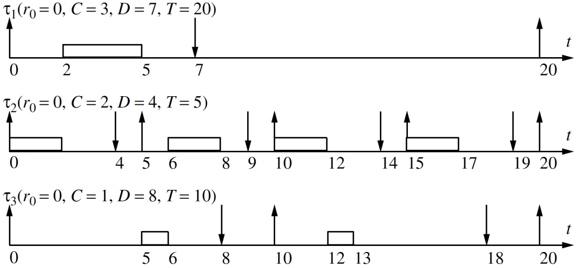


FIG 6. EDF EDF schedule for a set of three periodic tasks $\tau_1(r_0 = 0, C = 3, D = 7, 20 = T)$,$\tau_2(r_0 = 0, C = 2, D = 4, T = 5)$, $\tau_3(r_0 = 0, C = 1, D = 8, T = 10)$

Earliest deadline first algorithm—Example

- At time $t = 0$, the three tasks are ready to execute and the task with the smallest absolute deadline is τ_2.
- τ_2 is executed.
- At time $t = 2$,task τ_2 completes.

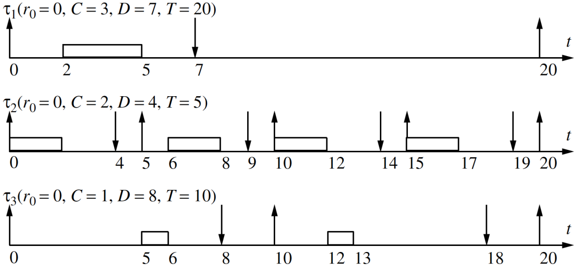


FIG 6. EDF EDF schedule for a set of three periodic tasks $\tau_1(r_0 = 0, C = 3, D = 7, 20 = T)$,$\tau_2(r_0 = 0, C = 2, D = 4, T = 5)$, $\tau_3(r_0 = 0, C = 1, D = 8, T = 10)$

Earliest deadline first algorithm—Example

- At time $t = 0$, the three tasks are ready to execute and the task with the smallest absolute deadline is τ_2.
- τ_2 is executed.
- At time $t = 2$,task τ_2 completes.
- The task with the smallest absolute deadline is now τ_1, which executes until completion at $t = 5$

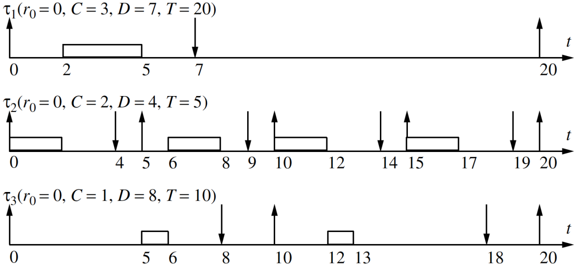


FIG 6. EDF EDF schedule for a set of three periodic tasks $\tau_1(r_0 = 0, C = 3, D = 7, 20 = T)$,$\tau_2(r_0 = 0, C = 2, D = 4, T = 5)$, $\tau_3(r_0 = 0, C = 1, D = 8, T = 10)$

Earliest deadline first algorithm—Example

- At time $t = 0$, the three tasks are ready to execute and the task with the smallest absolute deadline is τ_2.
- τ_2 is executed.
- At time $t = 2$,task τ_2 completes.
- The task with the smallest absolute deadline is now τ_1, which executes until completion at $t = 5$
- At this point, task τ_2 is again ready. However, the task with the smallest absolute deadline is now τ_3, which begins to execute.

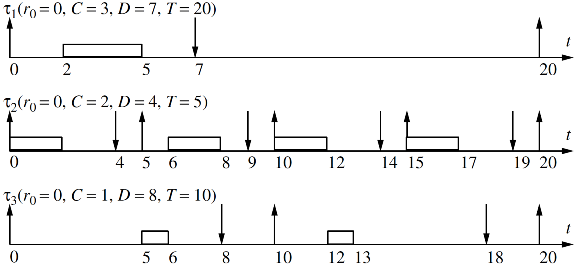


FIG 6. EDF EDF schedule for a set of three periodic tasks $\tau_1(r_0 = 0, C = 3, D = 7, 20 = T)$,$\tau_2(r_0 = 0, C = 2, D = 4, T = 5)$, $\tau_3(r_0 = 0, C = 1, D = 8, T = 10)$

Least laxity first algorithm

Least laxity first algorithm

summary—the least laxity first (LLF) algorithm assigns priority to tasks according to their relative laxity: the task with the smallest laxity will be executed at the highest priority

- When a task is executed, its relative laxity is constant.

Least laxity first algorithm

summary—the least laxity first (LLF) algorithm assigns priority to tasks according to their relative laxity: the task with the smallest laxity will be executed at the highest priority

- When a task is executed, its relative laxity is constant.
- However, the relative laxity of ready tasks decreases.

Least laxity first algorithm

summary—the least laxity first (LLF) algorithm assigns priority to tasks according to their relative laxity: the task with the smallest laxity will be executed at the highest priority

- When a task is executed, its relative laxity is constant.
- However, the relative laxity of ready tasks decreases.
- Thus, when the laxity of the tasks is computed only at arrival times, the LLF schedule is equivalent to the EDF schedule.

Least laxity first algorithm

summary—the least laxity first (LLF) algorithm assigns priority to tasks according to their relative laxity: the task with the smallest laxity will be executed at the highest priority

- When a task is executed, its relative laxity is constant.
- However, the relative laxity of ready tasks decreases.
- Thus, when the laxity of the tasks is computed only at arrival times, the LLF schedule is equivalent to the EDF schedule.
- However if the laxity is computed at every time t , more context-switching will be necessary.

Least laxity first algorithm

summary—the least laxity first (LLF) algorithm assigns priority to tasks according to their relative laxity: the task with the smallest laxity will be executed at the highest priority

- When a task is executed, its relative laxity is constant.
- However, the relative laxity of ready tasks decreases.
- Thus, when the laxity of the tasks is computed only at arrival times, the LLF schedule is equivalent to the EDF schedule.
- However if the laxity is computed at every time t , more context-switching will be necessary.
- Please take a closer look at example Figure 2.9 on page 32 of the textbook

The end