1. Introduction:

In real-time systems, it is imperative to provide time-bound guarantees on task execution time to ensure that the tasks meet their deadline. In particular, they help manage task execution in safety-critical domains such as aerospace, robotics, and industrial automation. The paper analyzes three real-time CPU scheduling Algorithm, Deadline Monotonic Scheduling (DMS), Least Laxity First (LLF), and Minimal Slack Scheduling (MSS). After that, Each algorithm is summarized in terms of logic, working, pros, and cons, followed by a comparative study discussing scenarios where each of them is a good fit.

2. Theoretical Explanation:

i) <u>Deadline Monotonic Scheduling (DMS):</u>

DMS is a fixed-priority algorithm where tasks are prioritized inversely to their relative deadlines. A task's priority is assigned using the rule:

$Priority = \frac{1}{D_i}\ (where\ D_i\ is\ the\ relative\ deadline\ of\ task\ \tau_i)$

Shorter deadlines receive higher priority. For example, in a task set $\{\tau_1(D = 5),\ \tau_2(D = 10)\},\ \tau_1\ will\ always\ preempt\ \tau_2$.

- **Working Principles:**
    - <u>Offline Priority Assignment:</u> Priorities are determined before runtime, simplifying scheduling decisions.
    - <u>Preemption:</u> When a high-priority task is released, it interrupts lower-priority tasks immediately.
    - <u>Schedulability Test:</u> DMS uses the response time analysis (RTA) for constrained-deadline tasks (deadline ≤ period). A task τ_i is schedulable if**:**

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

Where R_i = worst-case response time, C_i = worst-case execution time, hp(i) = set of higher-priority tasks.

- **Advantages:**

  - Low Runtime Overhead: Dynamic calculations are not required in case of static priorities.
  - Determinism: The behavior is predictable against its fixed task set, which is applicable for automotive control systems/IoT devices.
  - Utilization Bounds: Easily analyzed using Liu & Layland's utilization bound $U \leq n(2^{\frac{1}{n}} - 1)$

- **Limitations:**

  - Rigidity: Cannot process requests with non-relational deadlines (deadline > period).
  - Weak adaptability: Not flexible enough in environments where tasks added/removed at run-time.

ii) Least Laxity First (LLF) :

LLF is a dynamic-priority algorithm that schedules tasks based on their laxity (or slack time), defined as:

$$Laxity_i(t) = D_i - t - c_i(t)$$

where D_i = absolute deadline, t = current time, and $c_i(t)$ = remaining execution time of τ_i. The task with the smallest laxity is executed first.

- **Working Principles:**

  - Runtime Priority Updates: Laxity is re-evaluated at every scheduled event (task completion or task arrival).

- Optimality: LLF will yield a feasible schedule if there exists a feasible schedule under the assumptions of a single processor and preemptive execution.

  Example: Consider three tasks at t=0:

  - $\tau_1 : C = 2, D = 5$
  - $\tau_1 : C = 1, D = 4$
  - $\tau_1 : C = 3, D = 6$

  At $t = 0$, laxities are $5 - 0 - 2 = 3$, $4 - 0 - 1 = 3$, and $6 - 0 - 3 = 3$. The scheduler may arbitrarily choose one, but laxities change dynamically as time progresses.

- **Advantages:**

  - Dynamic Adaptability: Adjusts to runtime changes in execution times or deadlines.
  - Optimality: Maximizes deadline meetability in single-core systems.

- **Limitations:**

  - High Overhead: Frequent laxity recalculations lead to excessive context switches. For n tasks, each decision has $O(n)$ complexity.
  - Tie-Breaking Issues: Tasks with equal laxity cause unnecessary preemptions, worsening overhead.

iii) Minimal Slack Scheduling (MSS) :

MSS is identical in functionality to LLF, but focuses on slack time, not laxity.

$$Slack_i(t) = D_i - t - c_i(t)$$

Although LLF and MSS are used interchangeably, there are some implementations that differ in triggering slack/laxity updates (e.g., periodic vs. event-driven).

- **Working Principles:**
    - Slack-specific Monitoring: Slack is updated at a fixed interval or events in tasks.
    - Use Case: Cases where generalized approaches are not preferred and periodic slack updates reduce implementation effort e.g robotics with fixed control loops.

- **Advantages:**
    - Optimal but computationally intensive.
    - Well suited for complex systems with adaptive scheduling needs.
    - Keep a high level of responsiveness with acceptable overhead.
    - Minimizes the constant context switching exhibited in LLF.

- **Limitations:**
    - This requires computational overhead to calculate slack.
    - Complex implementation logic.
    - Slack estimations are prone to inaccuracies that can lead to variance in how the system performs in the real world.

3. Comparative Analysis :

Criterion	DMS	LLF	MSS
Priority Assignment	Static (based on deadlines)	Dynamic (based on laxity)	Dynamic (based on slack)
Overhead	Low (no runtime calculations)	High (frequent laxity updates)	High (frequent slack updates)
Optimality	Suboptimal for arbitrary deadlines	Optimal if feasible schedule exists	Optimal if feasible schedule exists
Context Switches	Minimal	High (due to laxity ties and preemptions)	Moderate
Schedulability Test	Simple (RTA or utilization bounds)	Complex (requires dynamic simulation)	Complex (requires dynamic simulation)
Adaptability	Poor (static priorities)	Excellent (dynamic adjustments)	Excellent (dynamic adjustments)
Use Cases	Embedded systems, automotive ECUs	Robotics, avionics, dynamic real-time systems	Robotics, avionics, dynamic real-time systems

4. Key Insights :

1. DMS is specially designed for deterministic systems with static task sets, such as automobile anti-lock braking systems (ABS), where task deadlines are known in advance and cannot be altered.
2. LLF/MSS are better for workloads that change during execution, such as navigating a drone through built-up areas, where obstacles or environmental changes require modifications at runtime.
3. MSS is at times differentiated from LLF in the literature by its periodic slack updates but the essential idea is similar.

Conclusion :

DMS, LLF, and MSS are compromises between predictability and flexibility. DMS finds ease of use for a static environment while LLF/MSS can adapt at the cost of more computational overhead. The selection is based on system specification: DMS is preferred for deterministic embedded systems while dynamic real-time applications use LLF/MSS. Future work can investigate hybrid algorithms that use static and dynamic priorities to consider the trade-off.

References :

1. Liu, C. L., & Layland, J. W. (1973). Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM.
2. Buttazzo, G. C. (2011). Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Springer.
3. Krishna, C. M., & Shin, K. G. (1997). Real-Time Systems. McGraw-Hill.

________________THE END________________