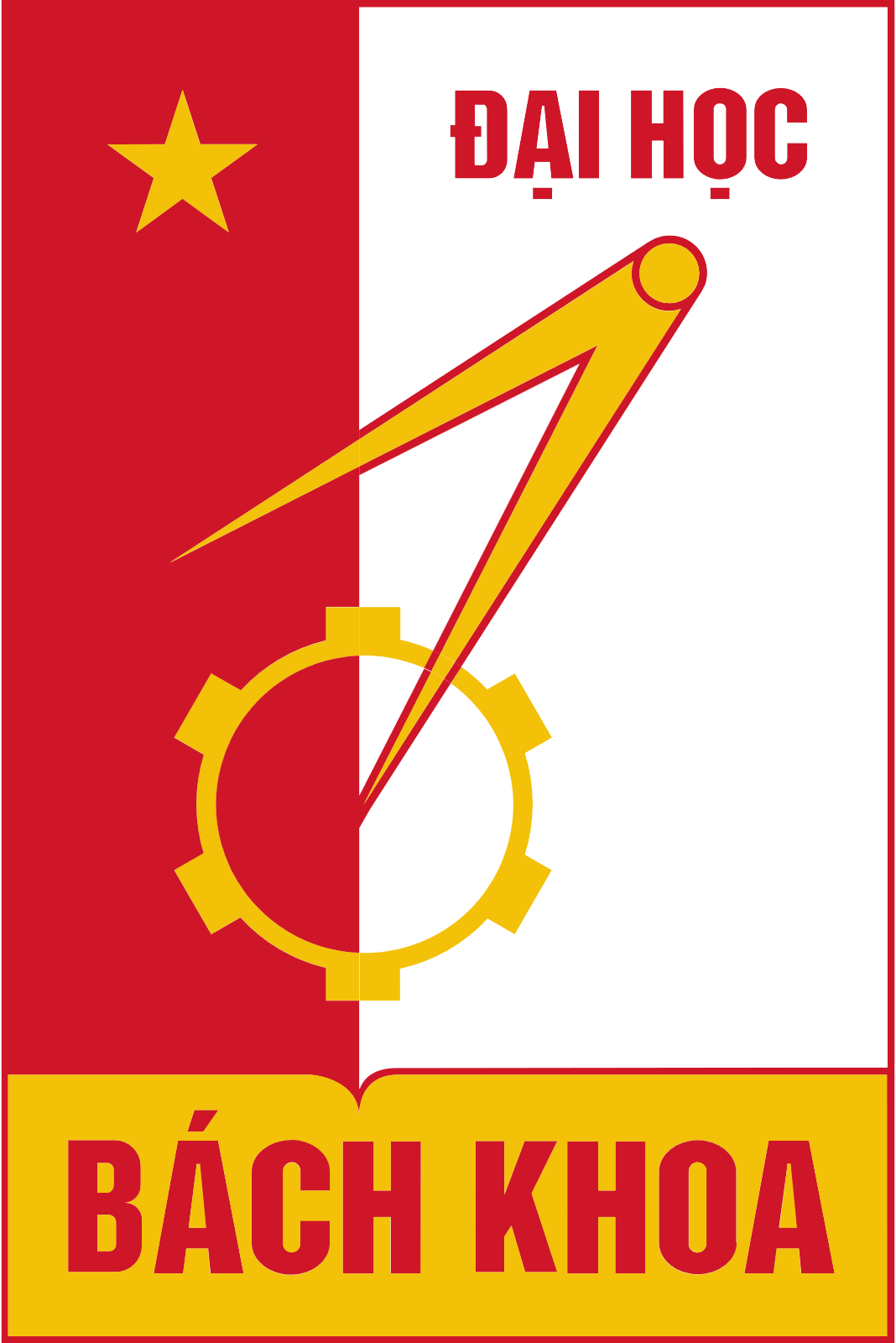
**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

****

**BÁO CÁO BÀI TẬP LỚN**

**ĐỀ TÀI: TÌM HIỂU VÀ CÀI ĐẶT CÁC GIẢI THUẬT  
LẬP LỊCH TRONG HỆ THỐNG THỜI GIAN THỰC**

Học phần: NGUYÊN LÝ HỆ ĐIỀU HÀNH

Mã học phần: IT3070 Mã lớp: 157502

Giảng viên hướng dẫn: TS. Đỗ Quốc Huy

*Nhóm sinh viên thực hiện:*

|  |  |  |
| --- | --- | --- |
| STT | HỌ VÀ TÊN | MSSV |
| 1 | Nguyễn Ngọc Ánh | 20235013 |
| 2 | Lê Minh Hoàng | 20230028 |
| 3 | Nguyễn Thị Chi Mai | 20235149 |
| 4 | Nguyễn Minh Tùng | 20235239 |

**MỤC LỤC**

[A. Giới thiệu đề tài 3](#_Toc199003994)

[I. Khái quát về đề tài 3](#_Toc199003995)

[II. Bài toán mà đề tài giải quyết 3](#_Toc199003996)

[B. Nội dung chính của đề tài 3](#_Toc199003997)

[I. THUẬT TOÁN DEADLINE MONOTONIC (DM) 4](#_Toc199003998)

[**1.1. Giới thiệu thuật toán DM** 4](#_Toc199003999)

[**1.2. Cài đặt chương trình** 4](#_Toc199004000)

[**1.3. Nhận xét thuật toán** 6](#_Toc199004001)

[II. THUẬT TOÁN EARLIEST DEADLINE FIRST (EDF) 8](#_Toc199004002)

[**2.1. Giới thiệu thuật toán EDF** 8](#_Toc199004003)

[**2.2. Cài đặt chương trình** 9](#_Toc199004004)

[**2.3. Nhận xét thuật toán** 11](#_Toc199004005)

[III. THUẬT TOÁN LEAST LAXITY FIRST (LLF) 13](#_Toc199004006)

[**3.1. Giới thiệu thuật toán LLF** 13](#_Toc199004007)

[**3.2. Cài đặt chương trình** 14](#_Toc199004008)

[IV. THUẬT TOÁN RATE MONOTONIC (RM) 18](#_Toc199004009)

[**4.1. Giới thiệu thuật toán RM** 18](#_Toc199004010)

[**4.2. Cài đặt chương trình** 19](#_Toc199004011)

[**4.3. Nhận xét thuật toán** 22](#_Toc199004012)

[C. Mã nguồn (source code) 23](#_Toc199004013)

[D. Tài liệu tham khảo 23](#_Toc199004014)

# **A. Giới thiệu đề tài**

## **Khái quát về đề tài**

Trong hệ thống thời gian thực, độ chính xác của hệ thống không chỉ phụ thuộc vào kết quả logic của phép tính mà còn phụ thuộc vào thời gian mà kết quả của các phép tính đó xuất hiện. Trong hệ thống thời gian thực, việc lập lịch được thực hiện bởi những tiêu chí nhất định nhằm đảm bảo quá trình hoàn thành các tác vụ khác nhau của chúng tại các mốc thời gian cụ thể. Chất lượng của các thuật toán lập lịch thời gian thực (real-time scheduling) có ảnh hưởng trực tiếp tới hoạt động của hệ thống thời gian thực. Trong bài báo cáo này, nhóm chúng em sẽ đưa ra các thuật toán lập lịch phổ biến bao gồm: Earliest Deadline First (EDF), Rate Monotonic (RM), Least Laxity First (LTF), Deadline Monotonic (DM) cho các tác vụ định kỳ.

## **Bài toán mà đề tài giải quyết**

Trong các hệ thống thời gian thực, có một số tiến thình cần thực thi với các yêu cầu rất nghiêm ngặt về thời gian hoàn thành (deadline). Mỗi tiến trình trong hệ thống có các thuộc tính sau:

* Period: Chu kỳ của tiến trình, tức là thời gian giữa hai lần thực thi liên tiếp
* Execution time: Thời gian thực thi của tiến trình trong mỗi chu kỳ
* Deadline: Thời gian đến hạn mà tiến trình phải hoàn thành công việc
* Laxity: Khoảng thời gian giữa thời điểm tiến trình có thể bắt đầu và deadline của nó

Cơ chế lập lịch là khái niệm quan trọng trong một hệ thống máy tính, là chiến lược mà hệ thống máy tính sử dụng để quyết định công việc nào sẽ được thực thi vào thời điểm cụ thể. Bài toán mà các giải thuật lập lịch trong hệ thống thời gian thực (real-time scheduling algorithm) là quản lý và phân bố tài nguyên cho các tiến trình trong hệ thống thời gian thực sao cho các tiến trình này hoàn thành đúng thời gian yêu cầu (deadlines).

# **B. Nội dung chính của đề tài**

Có 04 thuật toán được lựa chọn sau đây.

## **THUẬT TOÁN DEADLINE MONOTONIC (DM)**

### **1.1. Giới thiệu thuật toán DM**

* **Mục tiêu** ‑ xác định xem một tập *n* task tuần hoàn/gián đoạn có deadline ràng buộc (*constrained‑deadline, Dᵢ ≤ Tᵢ*) có được lập lịch an toàn trên một CPU đơn hay không, đồng thời cung cấp chính sách lập lịch.
* **Nguyên lý Deadline Monotonic**
* Gán độ ưu tiên tĩnh tỉ lệ nghịch với deadline tương đối *Dᵢ*: deadline càng nhỏ ⇒ ưu tiên càng cao.
* Bộ lập lịch runtime luôn chọn job ready có Deadline nhỏ nhất; trưng dụng ngay khi xuất hiện một job có ưu tiên cao hơn.
* **Điều kiện cần** – tác vụ độc lập, *WCET* đã biết, không share tài nguyên (hoặc dùng mutex với PCP để tránh priority inversion).
* **Kiểm tra khả năng lập lịch**
* **Điều kiện đủ**:



* **Phân tích thời gian đáp ứng (RTA)** – lặp công thức cố định:

A black and white math symbols

AI-generated content may be incorrect.

*Nếu* Rᵢ converge ≤ Dᵢ *∀ i* ⇒ schedulable.

* **Tính chất** – DM là thuật toán cố định ưu tiên tối ưu cho tập task có deadline bị chặn: nếu có một chế độ ưu tiên cố định khác lập lịch được thì DM cũng lập lịch được.

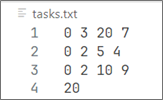
### **1.2. Cài đặt chương trình**

Chương trình mô phỏng DM được viết bằng **C++ 17** trong file DM.cpp.

|  |  |  |  |
| --- | --- | --- | --- |
| **Hàm / Cấu trúc** | **Dữ liệu vào** | **Dữ liệu ra** | **Công việc chính** |
| struct Task | {id, r, C, T, D} | — | Lưu tham số của một tác vụ. r cho phép mỗi task khởi động tại thời điểm tùy ý. |
| struct Job | Tự sinh | — | Thể hiện một instance: id, rem, releaseTime, deadlineTime, Dline. |
| generateJobs(tasks, simTime, jobs) | vector<Task>, int | vector<Job> | Sinh job cho mỗi task: release = r + k·T, deadline = release + D. |
| simulateDM(tasks, simTime) | vector<Task>, int | bool (đủ hạn) + in lịch | Bộ lập lịch DM: luôn chọn công việc có deadline nhỏ nhất, hỗ trợ trưng dụng. |
| responseTimeTest(tasks) | vector<Task> sắp theo D↑ | bool | Phân tích thời gian đáp ứng (RTA) – không phụ thuộc r vì worst‑case synchronous release giả sử tất cả r = 0. |
| main() | — | — | Khởi tạo tập task, chạy RTA, sau đó mô phỏng. |

**1.2.1. Định dạng dữ liệu I/O**

* **Input (file .txt)**

****

* **Output terminal**

A screenshot of a computer

AI-generated content may be incorrect.

**1.2.2. Giải thích chi tiết cấu trúc Task {id, r, C, T, D}**

|  |  |  |
| --- | --- | --- |
| **Trường** | **Ý nghĩa** | **Kiểu dữ liệu** |
| id | Mã định danh duy nhất của task. | int |
| r (*Release Offset*) | Thời điểm phát hành đầu tiên của task; cho phép mô tả khởi động lệch pha. | int |
| C (*WCET*) | Thời gian thực thi tệ nhất của một job. | int |
| T (*Period*) | Khoảng cách giữa hai release liên tiếp. | int |
| D (*Relative Deadline*) | Hạn chót tính từ thời điểm release (yêu cầu D ≤ T). | int |

**1.2.3. Mã nguồn**

[Deadline-Monotonic](https://github.com/adachankawaii/Deadline-Monotonic.git) (C++)

### **1.3. Nhận xét thuật toán**

**Ưu điểm**

* Tối ưu trong lớp lập lịch ưu tiên cố định đối với tác vụ deadline bị chặn.
* Hiệu quả với tải có chu kì dài nhưng deadline ngắn (đảm bảo phản hồi nhanh cho nhiệm vụ khẩn).
* Ổn định hơn RM trong trường hợp overload; các task deadline ngắn (thường quan trọng) vẫn có cơ hội hoàn thành.

**Nhược điểm**

* Phân tích khả năng lập lịch (RTA) phức tạp hơn RM; phải lặp đến hội tụ.
* Cài đặt runtime yêu cầu trưng dụng nhanh → chi phí chuyển ngữ cảnh lớn.
* Khi có deadline > period DM không còn tối ưu; cần thuật toán khác hoặc ưu tiên thủ công.

## **THUẬT TOÁN EARLIEST DEADLINE FIRST (EDF)**

### **2.1. Giới thiệu thuật toán EDF**

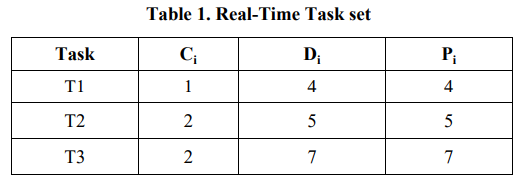
Năm 1973, Liu và Layland đã đề xuất một trong những thuật toán lập lịch thời gian thực phổ biến nhất: Earliest Deadline First (EDF). Thuật toán này có thể được sử dụng cho cả lập lịch thời gian thực **tĩnh và động**.

EDF sử dụng độ ưu tiên để sắp xếp các công việc. Cụ thể, nó gán mức độ ưu tiên cho các tác vụ dựa trên hạn chót tuyệt đối (absolute deadline). Tác vụ nào có hạn chót gần nhất sẽ được ưu tiên cao nhất. Các mức độ ưu tiên này được gán và thay đổi một cách linh hoạt trong quá trình thực thi. EDF được đánh giá là rất hiệu quả so với các thuật toán lập lịch khác trong hệ thống thời gian thực, vì có thể tối ưu hóa mức sử dụng CPU lên đến khoảng 100% trong khi vẫn đảm bảo các tác vụ hoàn thành đúng hạn.

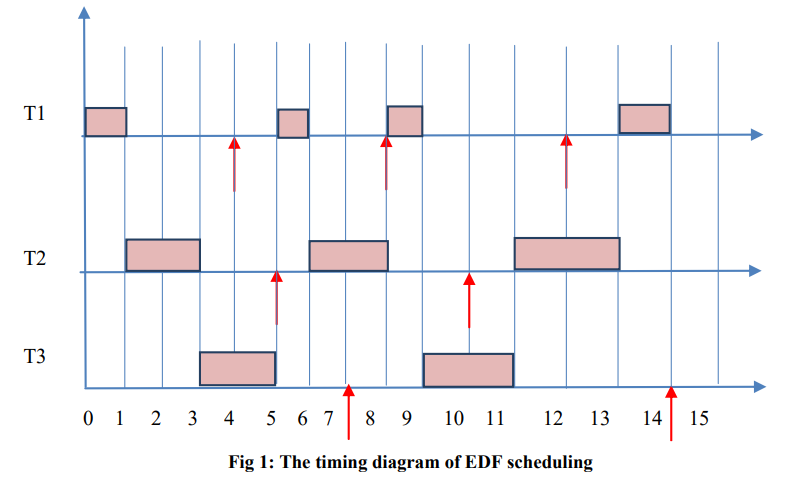
EDF có bao gồm cơ chế xử lý khi hạt nhân hệ điều hành (kernel) bị quá tải. Trong EDF, nếu mức sử dụng CPU dưới 100%, điều đó có nghĩa là tất cả các tác vụ đều hoàn thành trước hạn. EDF sẽ tìm một lịch trình khả thi tối ưu — tức là một lịch trình mà trong đó mọi tác vụ trong hệ thống đều được thực hiện trong thời gian giới hạn. Nếu EDF không thể tìm được một lịch trình khả thi cho toàn bộ tác vụ trong hệ thống thời gian thực, điều đó đồng nghĩa rằng không có thuật toán lập lịch thời gian thực nào khác có thể làm tốt hơn. Tất cả các tác vụ sẵn sàng thực thi phải thông báo hạn chót của mình cho EDF khi trở nên khả thi (runnable).

EDF không yêu cầu các tác vụ hay tiến trình phải mang tính chu kỳ, và cũng yêu cầu trước một thời gian xử lý CPU cố định cho mỗi tác vụ. Trong EDF, bất kỳ tác vụ nào đang thực thi cũng có thể bị tạm dừng (preempted) nếu có một tác vụ khác với hạn chót sớm hơn trở nên khả thi. EDF cho phép tiến trình tạm ngừng (preemption) trong quá trình lập lịch.

Hãy xem Bảng 1, trong đó tập hợp tác vụ này được lập lịch sử dụng thuật toán Earliest Deadline First hoàn toàn cho phép trưng dụng (preemptive), như minh họa trong Hình 1.



Trong đó Ci, Di, Pi lần lượt là thời gian thực thi (execution time) của tác vụ, chu kỳ (period) của tác vụ và hạn chót (deadline) của tác vụ.



### **2.2. Cài đặt chương trình**

**2.2.1. Giải thích cấu trúc**

Chương trình mô phỏng EDF được viết bằng **C++ 17** trong file edf\_scheduler.cpp.

|  |  |  |  |
| --- | --- | --- | --- |
| **Hàm / Cấu trúc** | **Dữ liệu vào** | **Dữ liệu ra** | **Công việc chính** |
| struct Task | {iC, P, D} | — | Lưu tham số của một tác vụ. r cho phép mỗi task khởi động tại thời điểm tùy ý. |
| |  | | --- | | max\_deadline(tasks, n) |  |  | | --- | |  | | Mảng Task, số lượng n | int | Tìm giá trị deadline lớn nhất để làm thời gian mô phỏng. |
| |  | | --- | | schedulability\_test() |  |  | | --- | |  | | Mảng Task, số lượng n | |  | | --- | | In ra đánh giá |  |  | | --- | |  | | Kiểm tra tính khả thi theo tiêu chuẩn EDF |
| |  | | --- | | clone\_tasks(src, n) |  |  | | --- | |  | | Mảng Task gốc | Mảng sao chép | Tạo bản sao các tác vụ để cập nhật quá trình mô phỏng. |
| |  | | --- | | print\_timeline() |  |  | | --- | |  | | Lịch thời gian, số task, hyperperiod | In ra biểu đồ Gantt | In biểu đồ lịch sử dụng CPU theo thời gian. |
| earliest\_deadline\_first() | Mảng Task, số lượng n | In ra lịch mô phỏng | Mô phỏng thuật toán EDF đến deadline lớn nhất, chọn job có deadline tuyệt đối gần nhất để chạy. |

**2.2.2. Định dạng dữ liệu I/O**

* **Input (file .txt)**

File chứa danh sách các task, mỗi dòng gồm 3 số nguyên cách nhau bởi dấu cách, lần lượt là:

A black background with white text

AI-generated content may be incorrect.

* **Output terminal**

**A screenshot of a phone number

AI-generated content may be incorrect.**

### **2.3. Nhận xét thuật toán**

**Ưu điểm**

* **Đáp ứng thời hạn (Meeting Deadlines):**

EDF đảm bảo rằng các tác vụ có thời hạn gần nhất sẽ được thực thi trước. Bằng cách ưu tiên các tác vụ dựa trên thời hạn của chúng, EDF giúp giảm thiểu khả năng trễ hạn và hỗ trợ đáp ứng các yêu cầu thời gian thực.

* **Tối ưu hóa việc sử dụng CPU (Optimal Utilization):**

EDF tối đa hóa việc sử dụng CPU bằng cách cho phép các tác vụ được thực thi ngay khi đến hạn, miễn là CPU đang rảnh. Điều này giúp tối ưu tài nguyên hệ thống và giảm thiểu thời gian nhàn rỗi.

* **Khả năng phản hồi cao (Responsiveness):**

EDF mang lại mức độ phản hồi cao đối với các tác vụ nhạy cảm về thời gian. Nó đảm bảo các tác vụ được lập lịch và thực thi kịp thời, từ đó giảm thời gian phản hồi và nâng cao hiệu năng hệ thống.

* **Tính dự đoán được (Predictability):**

EDF mang lại tính dự đoán trong việc thực thi các tác vụ và thời điểm đến hạn. Các quyết định lập lịch là xác định và có thể phân tích, dự đoán trước – đây là yếu tố then chốt trong các hệ thống thời gian thực.

* **Tính linh hoạt (Flexibility):**

EDF có thể xử lý cả các tác vụ định kỳ lẫn không định kỳ, phù hợp với nhiều loại hệ thống thời gian thực khác nhau. Nó cho phép tạo và lập lịch động các tác vụ mà không ảnh hưởng đến việc thực thi các tác vụ hiện tại.

**Nhược điểm**

* **Vấn đề quá tải tạm thời (Transient Overload Problem):**

Khi hệ thống bị quá tải trong một thời gian ngắn, EDF có thể khiến nhiều tác vụ bị trễ hạn do không có cơ chế kiểm soát mức sử dụng CPU.

* **Vấn đề chia sẻ tài nguyên (Resource Sharing Problem):**

EDF không xử lý hiệu quả việc chia sẻ tài nguyên giữa các tác vụ. Nếu hai tác vụ cạnh tranh cùng một tài nguyên, việc ưu tiên theo deadline có thể dẫn đến tình trạng phong tỏa hoặc đảo ưu tiên.

* **Vấn đề trong việc triển khai hiệu quả (Efficient Implementation Problem):**

Việc hiện thực EDF một cách hiệu quả yêu cầu các cấu trúc dữ liệu phức tạp để theo dõi và sắp xếp các deadline, đặc biệt khi có nhiều tác vụ hoặc khi deadline thay đổi động.

## **THUẬT TOÁN LEAST LAXITY FIRST (LLF)**

Thuật toán LLF (Least Laxity First) là một thuật toán điều phối **ưu tiên động**, nghĩa là mức độ ưu tiên của các tiến trình sẽ thay đổi theo thời gian thực. Tại mỗi thời điểm, hệ thống sẽ ưu tiên tiến trình có **độ linh động (laxity)** nhỏ nhất. Thuật toán này cho phép ưu tiên những tiến trình đang có nguy cơ trễ hạn, giúp cải thiện tỷ lệ hoàn thành đúng deadline.

### **3.1. Giới thiệu thuật toán LLF**

**3.1.1. Khái niệm độ linh động (Laxity)**

Laxity là khoảng thời gian mà một tiến trình có thể trì hoãn thực hiện mà vẫn hoàn thành đúng hạn.

Laxity của một tiến trình tại thời điểm t được tính theo công thức:

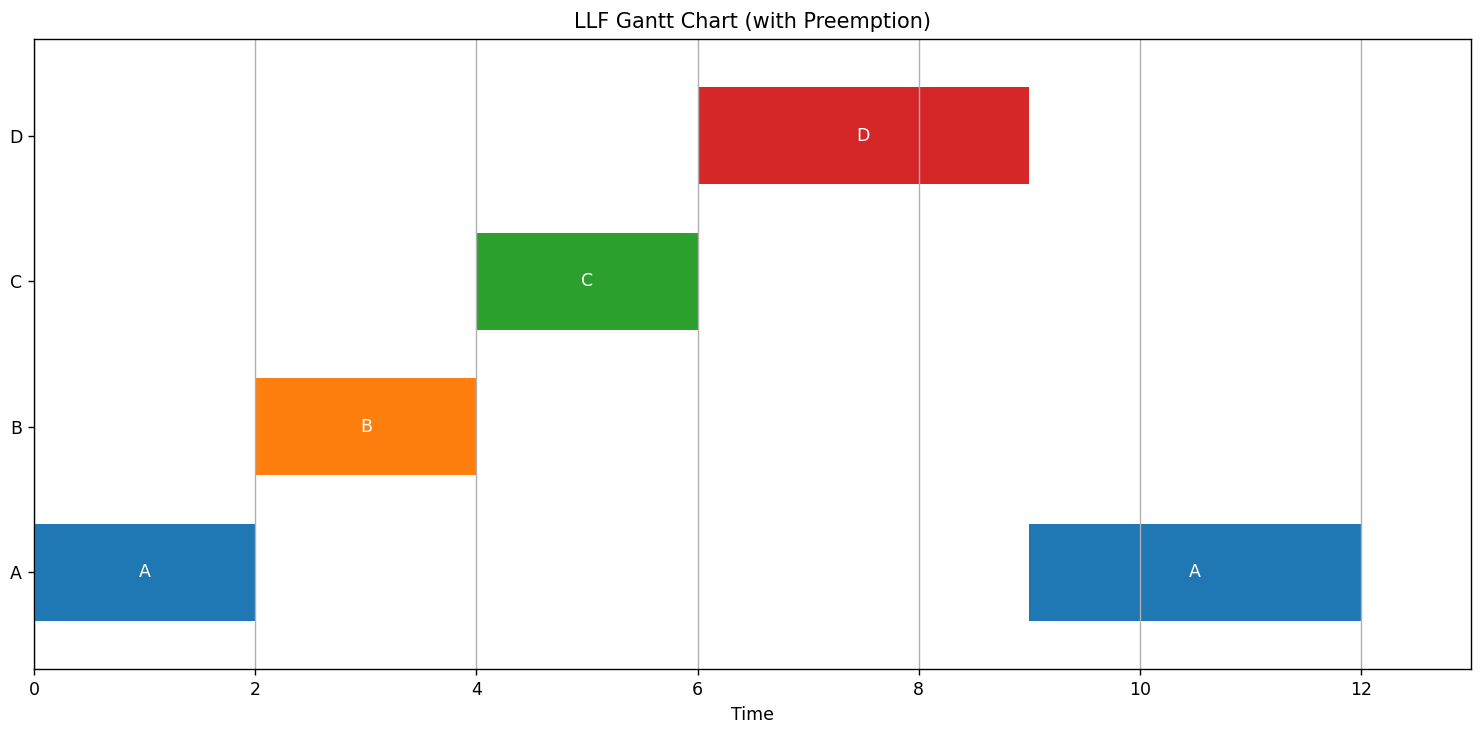
*Laxity = Deadline − Current Time − Remaining Execution Time*

Trong đó:

* *Deadline* là thời hạn hoàn thành tiến trình
* *Current Time* là thời gian hiện tại (thời gian thực)
* *Remaining Execution Time* là thời gian thực hiện của tiến trình

Laxity trả về một số lớn hơn hoặc bằng 0. Nếu giá trị nó nhỏ hơn 0, tức là chắc chắn tiến trình đó sẽ bị chậm deadline.

**3.1.2. Nguyên tắc hoạt động của LLF**



* *Task("A", arrival=0, exec\_time=5, deadline=12),*
* *Task("B", arrival=2, exec\_time=2, deadline=4),*
* *Task("C", arrival=3, exec\_time=2, deadline=6),*
* *Task("D", arrival=6, exec\_time=3, deadline=9),*

*Hình 3.1 – Biểu đồ Gantt ví dụ cho giải thuật LLF*

* Tại mỗi đơn vị thời gian, tính **laxity** cho tất cả các tiến trình đang sẵn sàng.
* Chọn tiến trình có **laxity nhỏ nhất** để thực hiện. Nếu CPU đang có tiến trình mà có tiến trình khác có lexity nhỏ hơn tiến trình trong CPU, CPU sẽ ưu tiên thực hiện tiến trình có lexity nhỏ hơn trước.
* Nếu có nhiều tiến trình có cùng laxity nhỏ nhất, chọn tiến trình đến trước.
* Cập nhật lexity của tất cả các tiến trình. Tiến trình nào có lexity nhỏ hơn 0 (chắc chắn chậm deadline), **loại nó khỏi hàng đợi.**

### **3.2. Cài đặt chương trình**

Trong chương trình, mỗi tiến trình được biểu diễn dưới dạng một cấu trúc Task, định nghĩa như sau: **typedef struct {**

* **char** id**;**  // Tên hoặc ký hiệu định danh tiến trình (ví dụ: 'A', 'B', ...)
* **int** arrival**;** // Thời điểm đến (arrival time) của tiến trình trong hệ thống
* **int** exec\_time**;** // Thời gian cần để thực thi hoàn toàn (burst time)
* **int** deadline**;** // Thời điểm cuối cùng tiến trình phải kết thúc
* **int** remaining**;** // Thời gian còn lại để thực thi (cập nhật sau mỗi lần chạy)
* **int** completed**;** // Cờ đánh dấu đã hoàn tất (1), chưa (0), sẽ quá deadline (-1)
* **int** start\_time**;** // Thời điểm bắt đầu thực thi lần đầu tiên

**} Task;**

- **int** laxity(**Task\*** task, **int** current\_time)

* Đầu vào: task: con trỏ tới một tiến trình; current\_time: thời điểm hiện tại.
* Đầu ra: Giá trị laxity (độ linh động): deadline - current\_time - remaining, dùng để chọn tiến trình ưu tiên nhất trong LLF.

- **int** all\_done(**Task** tasks[], **int** n)

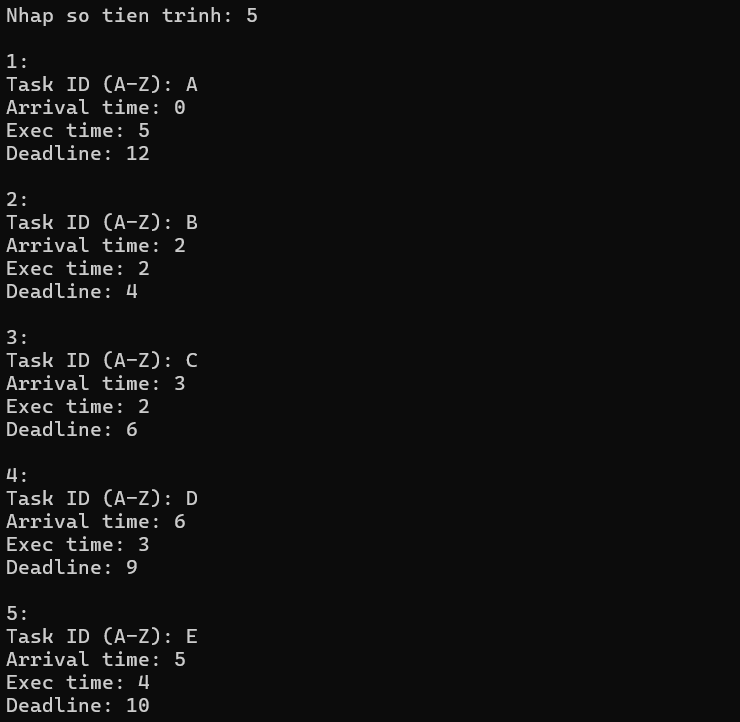
* Đầu vào: tasks[]: danh sách các tiến trình; n: số lượng tiến trình.
* Đầu ra: Trả về 1 nếu tất cả tiến trình đã hoàn tất (hoặc bị trễ deadline), ngược lại trả về 0.

- **void** llf\_scheduler(**Task** tasks[], **int** n)

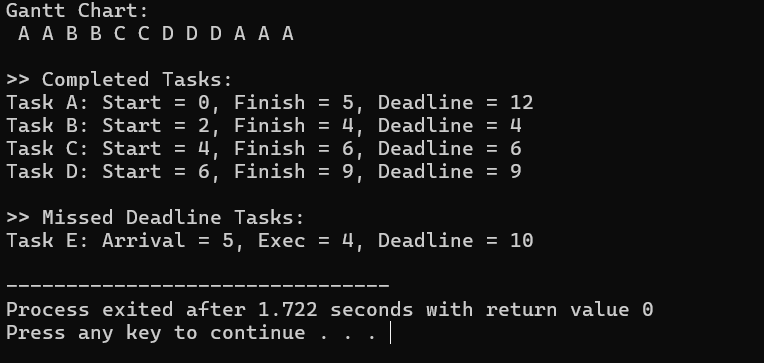
* Hàm nhận vào một mảng các tiến trình (kiểu Task) gồm tối đa n phần tử.
* Chức năng: Hàm thực hiện lập lịch các tiến trình theo thuật toán LLF, trong đó ở mỗi đơn vị thời gian, tính toán laxity của từng tiến trình.
  + Tiến trình có độ lỏng thời hạn (laxity) nhỏ nhất sẽ được chọn để thực thi.
  + Hàm xử lý theo mô hình preemptive, cho phép chuyển CPU sang tiến trình khác nếu xuất hiện tiến trình có laxity thấp hơn. Nếu tại thời điểm hiện tại không có tiến trình khả dụng, hệ thống sẽ ở trạng thái nhàn rỗi. Quá trình lập lịch tiếp tục cho đến khi tất cả tiến trình được xử lý hoặc bị loại do laxity âm.
  + Kết thúc quá trình, hàm sẽ in ra biểu đồ Gantt thể hiện lịch sử lập lịch, danh sách tiến trình hoàn thành đúng hạn, cũng như những tiến trình bị bỏ qua do không kịp deadline.

- **int** main(): chứa danh sách khởi tạo các tiến trình và gọi hàm llf\_scheduler.

**Kết quả minh họa:**

****

*Hình 3.2 – Danh sách tiến trình đầu vào, số lượng tiến trình, gọi hàm main*

****

*Hình 3.3 – Danh sách tiến trình hoàn thành kịp deadline / không kịp deadline và biểu đồ Gantt*

### 

## **THUẬT TOÁN RATE MONOTONIC (RM)**

### **4.1. Giới thiệu thuật toán RM**

Rate Monotonic (RM – Lập lịch tỷ lệ đơn điệu) là một thuật toán trưng dụng, lập lịch tĩnh trên hệ thống bộ xử lý đơn.

**4.1.1. Tác vụ định kỳ (periodic task)**

Các tác vụ định kỳ (periodic task) T được định nghĩa như sau:

T = (C, P, D)

Với C là thời gian cần thực thi, P là chu kỳ và D là hạn định của T.

**4.1.2. Giả thiết phục vụ cho thuật toán**

Phương pháp lập lịch này dựa trên một số giả thiết sau đây:

* Tất cả các tác vụ tham gia hệ thống phải có hạn định (deadline) kiểu chu kỳ.
* Thời gian thực hiện của các tác vụ biết trước và không đổi.
* Hạn chót cho các tác vụ là vào cuối mỗi chu kỳ (D = T).
* Các tác vụ phải sẵn sàng thực hiện vào đầu mỗi chu kỳ.
* Các tác vụ độc lập với nhau, nghĩa là không có quyền ưu tiên giữa các tác vụ và chúng không chặn lẫn nhau.
* Chi phí cho chuyển đổi ngữ cảnh, hoán đổi, v.v... rất nhỏ và có thể bỏ qua.

Thuật toán RM dựa trên độ ưu tiên. Dựa trên chu kỳ, tác vụ nào có chu kỳ càng nhỏ thì mức ưu tiên càng cao. Điều này có nghĩa rằng, hạn định của mỗi tác vụ là không đổi.

**4.1.3. Điều kiện về đầu vào**

Để tập các tác vụ đã cho có thể lên lịch, phương trình sau cần được thỏa mãn:

Trong đó:

* U là mức sử dụng của bộ xử lý.
* n là số lượng tác vụ.
* thứ tự là thời gian cần thực thi và thời gian chu kỳ của tác vụ thứ i.

### **4.2. Cài đặt chương trình**

**4.2.1. Cấu trúc chương trình**

1. bool is\_schedulable(const vector<pair<int, int>>& tasks)

* Kiểm tra xem tasks có thể được lập lịch mà không có sự trễ deadline hay không.
* Dựa vào công thức được nêu ở trên.

1. vector<int> rm\_schedule(const vector<pair<int, int>>& tasks, int duration = -1)

* Hàm chính, lập lịch cho các tasks.
* Sắp xếp các tasks theo thứ tự tăng dần về chu kỳ.
* Chọn ra và cập nhật các tasks được đem vào quá trình xử lý.
* Cấu trúc dữ liệu được sử dụng:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên biến** | **Kiểu dữ liệu** | **Kích thước** | **Vai trò chính** |
| tasks | vector<pair<int, int>> | n | Tiến trình đầu vào: (execution time, period) |
| sorted\_tasks | vector<pair<int, int>> | n | Bản sao tasks, đã sắp xếp theo period tăng |
| duration | int | - | Tổng thời gian mô phỏng lập lịch |
| timeline | vector<int> | duration | Lập lịch: tiến trình nào chạy tại thời điểm nào |
| remaining\_time | vector<int> | n | Thời gian xử lý còn lại cho mỗi tiến trình |
| release\_time | vector<int> | n | Thời điểm phát hành mới nhất của mỗi tiến trình |

**4.2.2. Nhập input và hiển thị output**

A screenshot of a computer

Description automatically generated

*Hình 4.1 – Cách thức nhập input và hiển thị output*

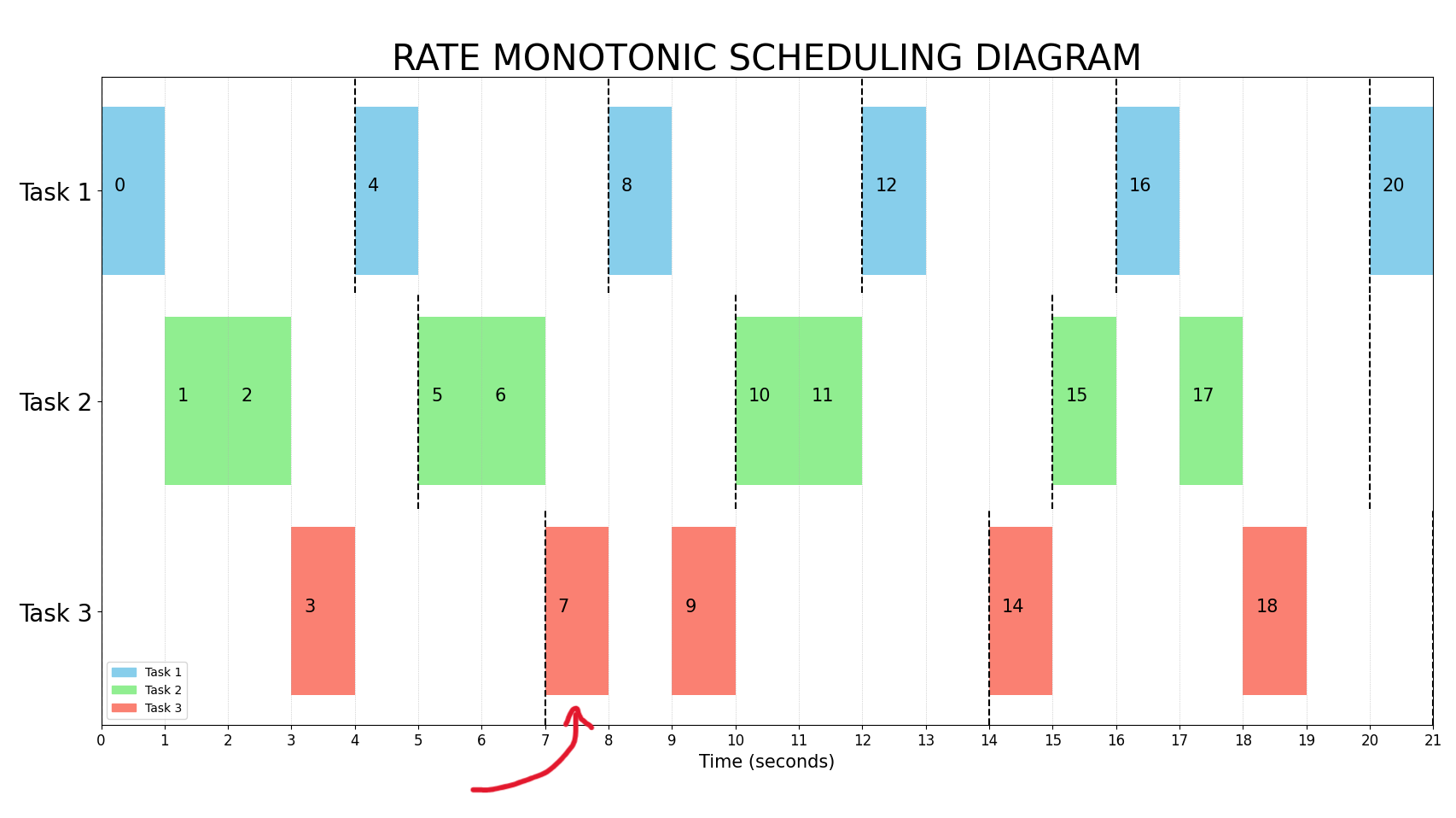
**4.2.3. Ví dụ minh họa**

***Ví dụ 1:*** Xét 03 tác vụ được xác định theo bảng sau:

|  |  |  |  |
| --- | --- | --- | --- |
| Tác vụ | Thời gian thực hiện | Chu kỳ | Hạn định |
| T1 | 1 | 4 | 4 |
| T2 | 2 | 5 | 5 |
| T3 | 2 | 7 | 7 |

Ta có: U = 1/4 + 2/5 + 2/7 = 0.9357 > 0.7977 =

Sơ đồ thời gian làm việc của bộ xử lý:



*Hình 4.2 – (Python) Biểu đồ minh họa tiến trình thực hiện tasks với RM*

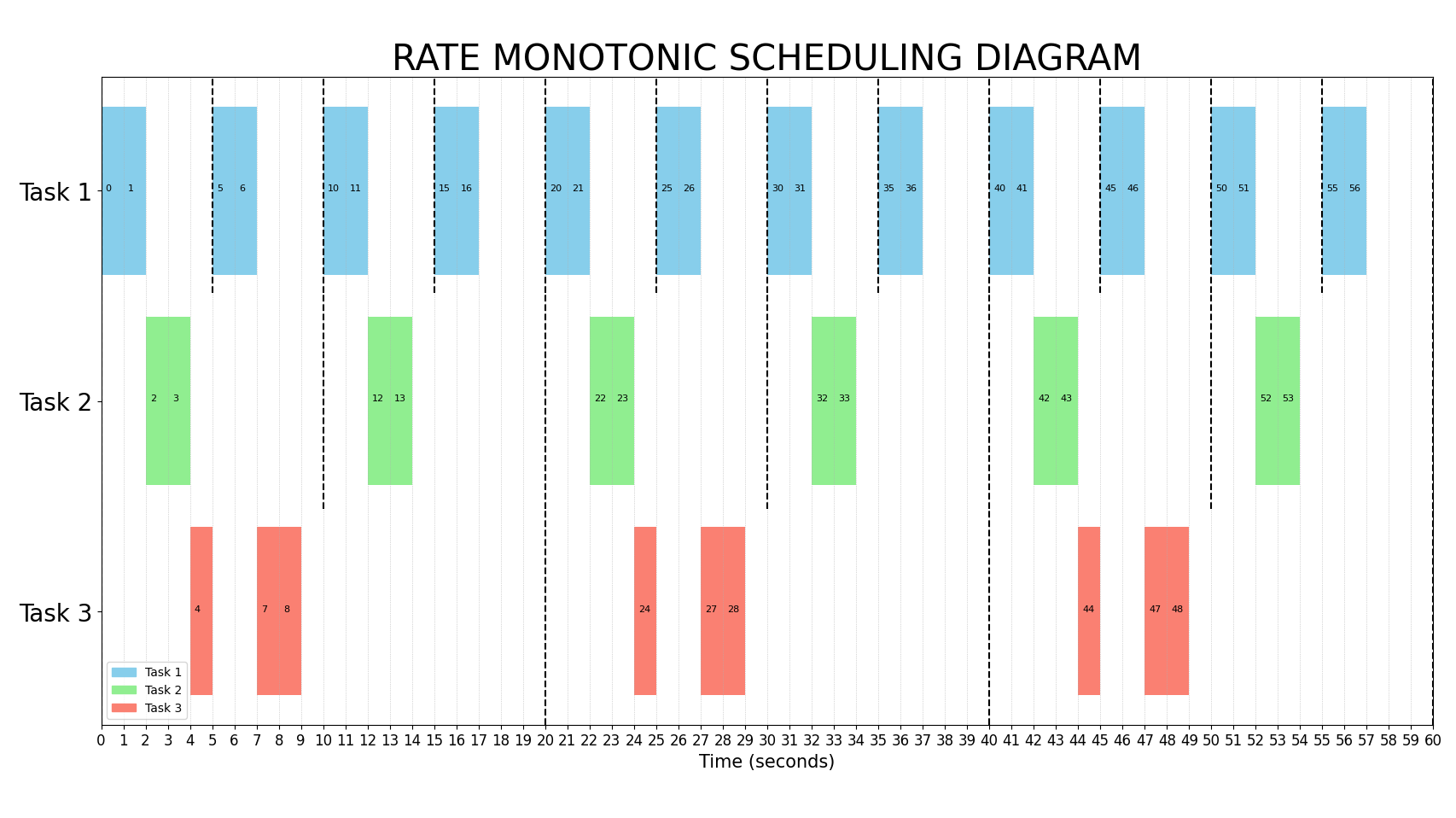
Nhận xét: Hạn định trễ ở vị trí số 7 (mũi tên trong hình) do điều kiện [1] không được thỏa mãn.

***Ví dụ 2:*** Xét 03 tác vụ được xác định theo bảng sau:

|  |  |  |  |
| --- | --- | --- | --- |
| Tác vụ | Thời gian thực hiện | Chu kỳ | Hạn định |
| T1 | 3 | 20 | 20 |
| T2 | 2 | 5 | 5 |
| T3 | 2 | 10 | 10 |

Ta có: U = 3/20 + 2/5 + 2/10 = 0.75 < 0.7977 =

Dựa vào độ ưu tiên về thời gian của chu kỳ, T1 → Task 3, T2 → Task 1, T3 → Task 2.



*Hình 4.3 – (Python) Biểu đồ minh họa tiến trình thực hiện tasks với RM*

Nhận xét: Không có hạn định trễ, do điều kiện [1] đã được thỏa mãn.

### **4.3. Nhận xét thuật toán**

**Ưu điểm**

* Cài đặt tương đối dễ dàng.
* Nếu tồn tại bất kỳ thuật toán gán ưu tiên tĩnh nào có thể đáp ứng thời hạn thì RM cũng có thể làm điều tương tự. Thuật toán này là tối ưu.

**Nhược điểm**

* Rất khó để hỗ trợ các tasks rời rạc và không định kỳ bằng RM.
* RM không tối ưu khi thời gian thực hiện và thời hạn khác nhau.

# **C. Mã nguồn (source code)**

Github: [Scheduling Real-time Tasks](https://github.com/adachankawaii/Scheduling-Real-time-Tasks)

# **D. Tài liệu tham khảo**

[1] – <https://microcontrollerslab.com/least-laxity-first-llf/>

[2] – <https://www.amd.e-technik.uni-rostock.de/veroeff/scopell.pdf>

[3] – International Journal of Computer Applications – Volume 158 – No 6, January 2017, *“Comparison of Real Time Task Scheduling Algorithms”.*

[4] – GeeksForGeeks – *“Rate-monotonic scheduling”*

<https://www.geeksforgeeks.org/rate-monotonic-scheduling/>

Truy cập lần cuối: 19/05/2025.