TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN

NGUYÊN LÝ HỆ ĐIỀU HÀNH

ĐỀ TÀI: LẬP TRÌNH MÔ PHỎNG CÁC PHƯƠNG PHÁP LẬP LỊCH CHO CPU

GVHD: TS. Nguyễn Bá Nghiễn

Thành viên: Nguyễn Minh Đức - 2020605521

Nguyễn Văn Thư - 2020601790

Đỗ Viết Trung - 2020600351

Nguyễn Xuân Trường - 2020601349

Nguyễn Anh Tú - 2020600692

Nhóm: 6

Lớp: IT6025.4 - Khóa: 15

Hà Nội – Năm 2022

MỤC LỤC

DANH MỤC HÌNH ẢNH

LỜI NÓI ĐẦU

Hệ điều hành là phần gắn bó trực tiếp với phần cứng và là môi trường để cho các chương trình ứng dụng khác chạy trên nó. Với chức năng quản lý và phân phối tài nguyên một cách hợp lý, đồng thời giả lập một máy tính mở rộng và tạo giao diện tiện lợi với người sử dụng, hệ điều hành là một thành phần then chốt không thể thiếu được trong mỗi một hệ thống máy tính điện tử.

Một trong những chức năng quan trọng của hệ điều hành là quản lý CPU. Trong môi trường xử lý đa chương, có thể xảy ra tình huống nhiều tiến trình đồng thời sẵn sàng để xử lý. Mục tiêu của các hệ phân chia thời gian(time-sharing) là chuyển đổi CPU qua lại giữa các tiến trình một cách thường xuyên để nhiều người sử dụng có thể tương tác cùng lúc với từng chương trình trong quá trình xử lý.

Để thực hiện được mục tiêu này, hệ điều hành phải lựa chọn tiến trình được xử lý tiếp theo. Bộ điều phối sẽ sử dụng một giải thuật điều phối thích hợp để thực hiện nhiệm vụ này. Một thành phần khác của hệ điều hành cũng tiềm ẩn trong công tác điều phối là bộ điều phối (dispatcher). Bộ phân phối sẽ chịu trách nhiệm chuyển đổi ngữ cảnh và trao CPU cho tiến trình được chọn bởi bộ điều phối để xử lý.

Vì những lợi ích lớn lao mà giải thuật điều phối CPU đem lại và để tìm hiểu kĩ hơn về nguyên tắc hoạt động của chúng, chúng em sẽ cùng nhau tìm hiểu, nghiên cứu đề tài: Lập trình mô phỏng các phương pháp lập lịch cho CPU.

Nhóm 6 sẽ trình bày nội dung chính như sau:

- Tìm hiểu các thuật toán: First Come First Served (FCFS), Round Robin (RR), Shortest Job Firsts(SJF), Shortest Remain Time (SRT).

- Chỉ ra được ưu và nhược điểm cả các thuật toán lập lịch CPU.

Mặc dù nhóm 6 chúng em đã cố gắng tìm hiểu kiến thức, tài liệu về các phương pháp lập lịch CPU, cũng như cố gắng trong quá trình hoàn thiện bài tập. Song do kiến thức, trình độ của chúng em còn nhiều hạn chế nên chắc chắn không tránh khỏi những sai sót. Chúng em mong sự chỉ dẫn của thầy giáo và góp ý của các bạn, để chúng em có thể hoàn thiện hơn bài tập.

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT

1.1. Giới thiệu

1.1.1. Mục tiêu lập lịch

Bộ điều phối không cung cấp cơ chế, mà đưa ra các quyết định. Các hệ điều hành xây dựng nhiều chiến lược khác nhau để thực hiện việc điều phối, nhưng tựu chung cần đạt được các mục tiêu sau:

- Sự công bằng: các tiến trình chia sẻ CPU một cách công bằng không có tiến trình nào phải đợi vô hạn để được cấp phát CPU.

- Tính hiệu quả: Hệ thống phải tận dụng được CPU 100% thời gian.

- Thời gian đáp ứng hợp lý: cực tiểu hóa thời gian hồi đáp cho các tương tác của người sử dụng.

- Thời gian lưu lại trong hệ thống: cực tiểu hóa thời gian hoàn tất các tác vụ xử lý theo lô.

- Thông lượng tối đa: cực đại hóa số công việc được xử lý trong một đơn vị thời gian. Tuy nhiên thường không thể thỏa mãn tất cả các mục tiêu kể trên vì bản thân chúng có sự mâu thuẫn với nhau mà chỉ có thể dung hòa chúng ở mức độ nào đó.

1.1.2. Các đặc điểm của tiến trình

Điều phối hoạt động của các tiến trình là một vấn đề rất phức tạp, đòi hỏi hệ điều hành khi giải quyết phải xem xét nhiều yếu tố khác nhau để có thể đạt được những mục tiêu đề ra. Một số đặc tính của tiến trình cần được quan tâm như tiêu chuẩn điều phối:

- Tính hướng xuất/nhập của tiến trình: Khi một tiến trình được nhận CPU, chủ yếu nó chỉ sử dụng CPU đến khi phát sinh một yêu cầu nhập xuất. Hoạt động của các tiến trình như thế thường bao gồm nhiều lượt sử dụng CPU, mỗi lượt trong một thời gian khá ngắn.

- Tính hướng xử lý của tiến trình: Khi một tiến trình được nhận CPU, nó có khuynh hướng sử dụng CPU đến khi hết thời gian dành cho nó? Hoạt động của các tiến trình như thế thường bao gồm một số ít lượt sử dụng CPU, nhưng mỗi lượt trong một thời gian đủ dài.

- Tiến trình tương tác hay xử lý theo lô: Người sử dụng theo kiểu tương tác thường yêu cầu được hồi đáp tức thời đối với các yêu cầu của họ, trong khi các tiến trình của các tác vụ được xử lý theo lô nói chung có thể trì hoãn trong một thời gian chấp nhận được.

- Độ ưu tiên của tiến trình: Các tiến trình có thể được phân cấp theo một số tiêu chuẩn đánh giá nào đó, một cách hợp lý, các tiến trình quan trọng hơn(có độ ưu tiên cao hơn) cần được ưu tiên cao hơn.

- Thời gian đã sử dụng CPU của tiến trình: một số quan điểm ưu tiên chọn những tiến trình đã sử dụng CPU nhiều thời gian nhất vì hy vọng chúng sẽ cần ít thời gian nhất để hoàn tất và rời khỏi hệ thống. Tuy nhiên cũng có quan điểm cho rằng các tiến trình nhận được CPU trong ít thời gian là những tiến trình đã phải chờ lâu nhất, do vậy ưu tiên chọn chúng.

- Thời gian còn lại tiến trình cần để hoàn tất: Có thể giảm thiểu thời gian chờ trung bình của các tiến trình bằng cách cho các tiến trình cần ít thời gian nhất để hoàn tất được thực hiện trước. Tuy nhiên đáng tiếc là rất hiếm khi biết được tiến trình cần bao nhiêu thời gian nữa để kết thúc xử lý.

1.1.3. Điều phối không độc quyền và điều phối độc quyền

Thuật toán điều phối cần xem xét và quyết định thời điểm chuyển đổi CPU giữa các tiến trình. Hệ điều hành các thể thực hiện cơ chế điều phối theo nguyên lý độc quyền hoặc không độc quyền:

Điều phối độc quyền: Nguyên lý điều phối độc quyền cho phép một tiến trình khi nhận được CPU sẽ có quyền độc chiếm CPU đến khi hoàn tất xử lý hoặc tự nguyện giải phóng CPU. Khi đó quyết định điều phối CPU sẽ xảy ra trong các tình huống sau:

- Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái bị blocked (ví dụ chờ một thao tác nhập xuất hay chờ một tiến trình con kết thúc…).

- Khi tiến trình kết thúc.

Điều phối không độc quyền: Ngược với nguyên lý độc quyền, điều phối theo nguyên lý không độc quyền cho phép tạm dừng hoạt động của một tiến trình sẵn sàng xử lý. Khi một tiến trình nhận được CPU, nó vẫn được sử dụng CPU đến khi hoàn tất

hoặc tự nguyện giải phóng CPU, nhưng khi có một tiến trình khác có độ ưu tiên có thể dành quyền sử dụng CPU của tiến trình ban đầu. Như vậy là tiến trình có thể bị tạm dừng hoạt động bất cứ lúc nào mà không được báo trước, để tiến trình khác xử lý. Các quyết định điều phối xảy ra khi:

- Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái bị blocked.

- Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái ready (vì xảy ra một ngắt).

- Khi tiến trình chuyển từ trạng thái chờ (blocked) sang trạng thái ready (ví dụ một thao tác nhập xuất hoàn tất).

- Khi tiến trình kết thúc.

Trong các hệ thống sử dụng nguyên lý điều phối độc quyền có thể xảy ra tình trạng các tác vụ cần thời gian xử lý ngắn phải chờ tác vụ xử lý với thời gian rất dài hoàn tất. Nguyên lý điều phối độc quyền thường chỉ thích hợp với các hệ xử lý theo lô. Đối với các hệ thống tương tác (time sharing), các hệ thời gian thực (real time), cần phải sử dụng nguyên lý điều phối không độc quyền để các tiến trình quan trọng có cơ hội hồi đáp kịp thời. Tuy nhiên thực hiện hiện điều phối theo nguyên lý không độc quyền đòi hỏi nhưng cơ chế phức tạp trong việc phân định độ ưu tiên, và phát sinh thêm chi phí khi chuyển đổi CPU qua lại giữa các tiến trình.

1.2. Các khái niệm cơ bản

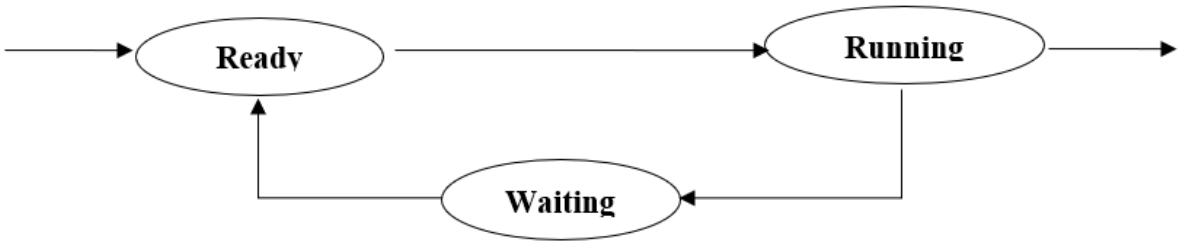
1.2.1. Khái niệm giờ CPU

CPU là một loại tài nguyên quan trọng của máy tính. Mọi tiến trình muốn hoạt động được đều phải có sự phục vụ của CPU(để xử lý, tính toán…). Thời gian mà CPU phục vụ cho tiến trình hoạt động được gọi là giờ CPU.

Tại mỗi thời điểm nhất, chỉ có một tiến trình được phân phối giờ CPU để hoạt động(thực hiện các lệnh của mình).

1.2.2. Các trạng thái của tiến trình liên quan đến giờ CPU

Trong chế độ đa chương trình, có ba trạng thái của tiến trình liên quan mật thiết đến giờ CPU bao gồm:



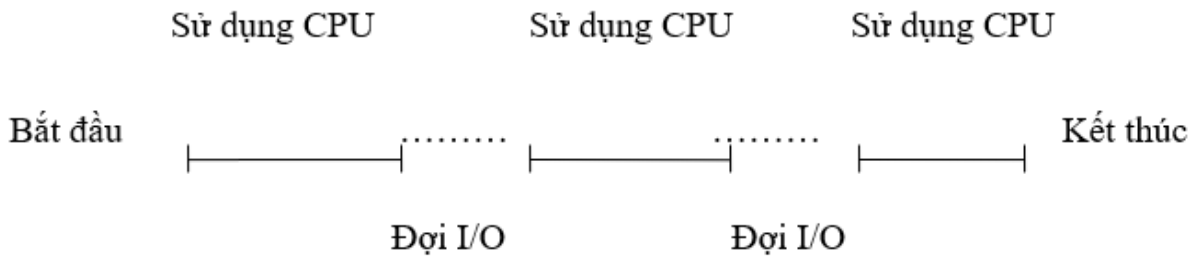
Hình 1.1: Các trạng thái của tiến trình liên quan đến giờ CPU

Sẵn sàng (ready): là trạng thái mà tiến trình được phân phối đầy đủ mọi tài nguyên cần thiết và đang chờ giờ CPU.

Thực hiện (running): là trạng thái mà tiến trình được phân phối đầy đủ mọi tài nguyên cần thiết và giờ CPU.

Đợi (waiting): là trạng thái tiến trình không thực hiện được vì thiếu một vài điều kiện nào đó(đợi dữ liệu vào/ra, đợi tài nguyên bổ sung…). Khi sự kiện mà nó chờ đợi xuất hiện, tiến trình sẽ quay lại trạng thái sẵn sàng.

Như vậy, trong suốt thời gian tồn tại của mình, các tiến trình sẽ tuân thủ theo sơ đồ thực hiện sau:



Hình 1.2: Sơ đồ thực hiện các tiến trình

Một tiến trình đang trong trạng thái thực hiện, nó có thể rời khỏi trạng thái bởi một trong ba lý do:

- Tiến trình đã hoàn thành công việc, khi đó nó trả lại giờ CPU và chuyển sang chờ xử lý kết thúc.

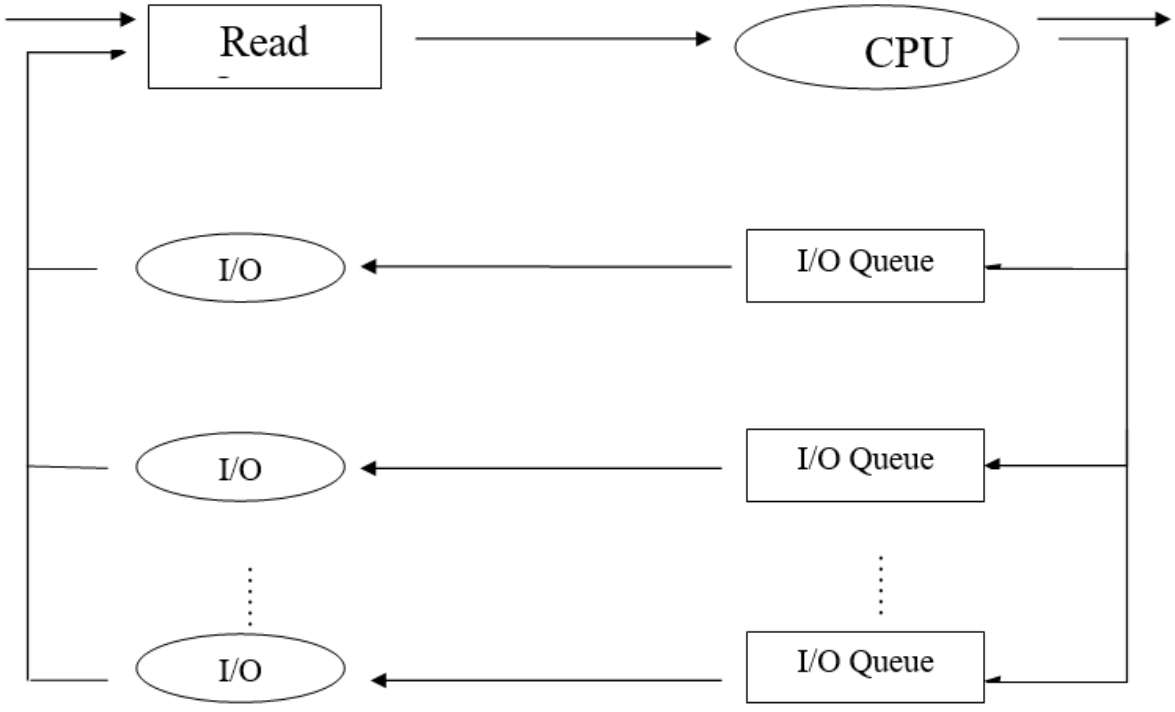
- Tiến trình tự ngắt: Khi tiến trình chờ đợi một sự kiện nào đó, tiến trình sẽ được chuyển sang trạng thái thực hiện khi có xuất hiện sự kiện nó đang chờ.

- Tiến trình sử dụng hết giờ CPU dành cho nó, khi đó nó sẽ được chuyển sang trạng thái sẵn sàng.

Việc chuyển tiến trình sang trạng thái sẵn sàng về bản chất là thực hiện việc phân phối lại giờ CPU.

1.2.3. Các khái niệm lập lịch cho CPU

Để điều khiển tiến trình ở nhiều trạng thái khác nhau, hệ thống thường tổ chức các từ trạng thái(thực chất là các khối điều khiển tiến trình) để ghi nhận tình trạng sử dụng tài nguyên và trạng thái tiến trình. Các từ trạng thái được tổ chức theo kiểu hàng đợi như sau:



Hình 1.3: Sơ đồ tổ chức hàng đợi các tiến trình

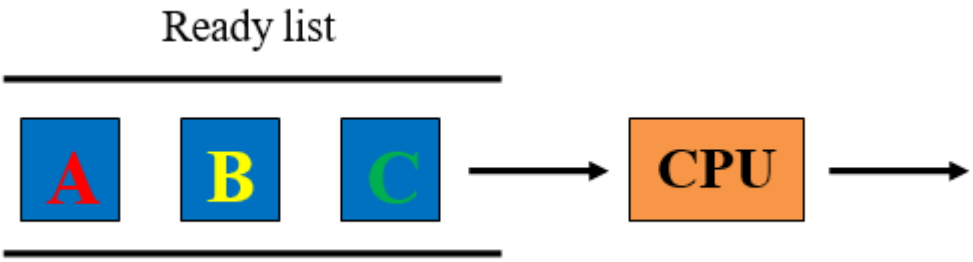
Như vậy lập lịch cho CPU có nghĩa là tổ chức một hàng đợi các tiến trình sẵn sàng để phân phối giờ CPU cho chúng dựa trên độ ưu tiên của các tiến trình; sao cho hiệu suất sử dụng CPU là tối ưu nhất.

Mỗi tiến trình ở trạng thái sẵn sàng sẽ được gắn với một thứ tự ưu tiên. Thứ tự ưu tiên này được xác định dựa vào các yếu tố như: thời điểm hình thành tiến trình, thời gian thực hiện tiến trình, thời gian kết thúc tiến trình.

1.3. Các thuật toán lập lịch

1.3.1. First Come First Served (FCFS)

Trong thuật toán này, độ ưu tiên phục vụ tiến trình căn cứ vào thời điểm hình thành tiến trình. Hàng đợi các tiến trình được tổ chức theo kiểu FIFO (First In First Out – vào trước ra trước, vào sau ra sau). Mọi tiến trình đều được phục vụ theo trình tự xuất hiện cho đến khi kết thúc hoặc bị ngắt.



Hình 1.4: Điều phối FIFO

Ưu điểm: giờ CPU không bị phân phối lại (không bị ngắt) và chi phí thực hiện thấp nhất(vì không phải thay đổi thứ tự ưu tiên phục vụ, thứ tự ưu tiên là thứ tự của tiến trình trong hàng đợi).

Nhược điểm: là thời gian trung bình chờ phục vụ của các tiến trình là như nhau(không kể tiến trình ngắn hay dài), do đó dẫn tới ba điểm sau:

- Thời gian chờ trung bình sẽ tăng vô hạn khi hệ thống tiếp cận tới hạn khả năng phục vụ của mình.

- Nếu độ phát tán thời gian thực hiện tiến trình tăng thì thời gian chờ đợi trung bình cũng tăng theo.

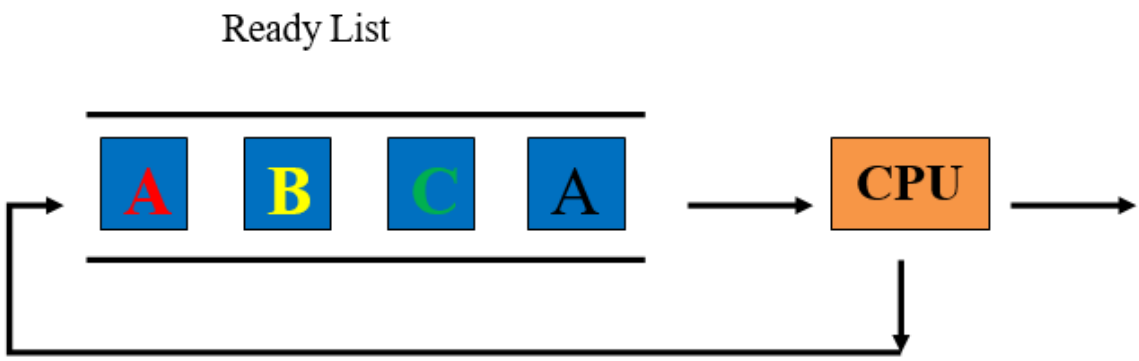
- Khi có tiến trình dài, ít bị ngắt thì các tiến trình khác phải chờ đợi lâu hơn.

1.3.2. Round robin (RR)

Giải thuật định thời luân phiên (round-robin scheduling algorithm-RR) được thiết kế đặc biệt cho hệ thống chia sẻ thời gian. Tương tự như định thời FIFO nhưng sự trưng dụng CPU được thêm vào để chuyển CPU giữa các quá trình. Đơn vị thời gian nhỏ được gọi là định mức thời gian (time quantum) hay phần thời gian (time slice) được định nghĩa. Định mức thời gian thường từ 10 đến 100 mili giây. Hàng đợi sẵn sàng được xem như một hàng đợi vòng. Bộ định thời CPU di chuyển vòng quanh hàng đợi sẵn sàng, cấp phát CPU tới mỗi quá trình có khoảng thời gian tối đa bằng một định mức thời gian.

Để cài đặt định thời RR, chúng ta quản lý hàng đợi sẵn sàng như một hàng đợi FIFO của các quá trình. Các quá trình mới được thêm vào đuôi hàng đợi. Bộ định thời CPU chọn quá trình đầu tiên từ hàng đợi sẵn sàng, đặt bộ đếm thời gian để ngắt sau 1 định mức thời gian và gửi tới quá trình.

Sau đó, một trong hai trường hợp sẽ xảy ra. Quá trình có 1 chu kỳ CPU ít hơn 1 định mức thời gian. Trong trường hợp này, quá trình sẽ tự giải phóng . Sau đó, bộ định thời biểu sẽ xử lý quá trình tiếp theo trong hàng đợi sẵn sàng. Ngược lại, nếu chu kỳ CPU của quá trình đang chạy dài hơn 1 định mức thời gian thì độ đếm thời gian sẽ báo và gây ra một ngắt tới hệ điều hành. Chuyển đổi ngữ cảnh sẽ được thực thi và quá trình được đặt trở lại tại cuối của hàng đợi sẵn sàng. Sau đó, bộ định thời biểu CPU sẽ chọn quá trình tiếp theo trong hàng đợi sẵn sàng.



Hình 1.5: Round Robin

Ưu điểm:

- Các quá trình sẽ được luân phiên cho CPU xử lý nên thời gian chờ đợi sẽ ít.
- Đối với các quá trình liên quan đến nhập/xuất người dùng thì rất hiệu quả.
- Việc cài đặt không quá phức tạp.

Nhược điểm:

- Thời gian chờ đợi trung bình của thuật toán RR thường là quá dài.
- Nếu thời gian định mức cho việc xử lý quá lớn thì RR thành FIFO.
- Nếu thời gian quá ngắn so với thời gian xử lý của một tiến trình trong danh sách hàng đợi thì việc chờ đợi và xử lý luân phiên sẽ nhiều.
- Quy tắc là định mức thời gian nên dài hơn 80% chu kỳ CPU.

1.3.3. Shortest Job First (SJF)

Một tiếp cận khác đối với việc định thời CPU là giải thuật định thời công việc ngắn nhất trước (shortest-job-first-SJF). Giải thuật này gán tới mỗi quá trình chiều dài của chu kỳ CPU tiếp theo cho quá trình sau đó. Khi CPU sẵn dùng, nó được gán tới quá trình có chu kỳ CPU kế tiếp ngắn nhất. Nếu hai quá trình có cùng chiều dài chu kỳ CPU kế tiếp, định thời FIFO được dùng. Chú ý rằng thuật ngữ phù hợp hơn là chu kỳ CPU kế tiếp ngắn nhất (shortest next CPU burst) vì định thời được thực hiện bằng cách xem xét chiều dài của chu kỳ CPU kế tiếp của quá trình hơn là toàn bộ chiều dài của nó.

Ưu điểm:

- Giải thuật được xem là tối ưu, thời gian chờ đợi trung bình giảm.
- Tận dụng hết năng lực của CPU.

Nhược điểm:

- Cài đặt thuật toán phức tạp,tốn nhiều xử lý cho quá trình quản lý.
- Mặc dù SJF là tối ưu nhưng nó không thể được cài đặt tại cấp định thời CPU ngắn vì không có cách nào để biết chiều dài chu kỳ CPU tiếp theo.
- Giải thuật SJF có thể trưng dụng hoặc không trưng dụng CPU, dẫn tới giải thuật này có nhiều dị bản khác nhau và sẽ tối ưu hay không tối ưu phụ thuộc vào trưng dụng CPU.

1.3.4. Shortest Remain Time (SRT)

Tương tự như SJF nhưng trong thuật toán này, độ ưu tiên thực hiện các tiến trình dựa vào thời gian cần thiết để thực hiện nốt tiến trình(bằng tổng thời gian trừ đi thời gian đã thực hiện). Như vậy, trong thuật toán này cần phải thường xuyên cập nhật thông tin về giời gian đã thực hiện của tiến trình. Đồng thời, chế độ phân bổ lại giờ CPU cũng phải được áp dụng nếu không sẽ làm mất tính ưu việc của thuật toán.

Ưu điểm:

- Thời gian chờ đợi, tồn tại trong hệ thống của mỗi tiến trình đều ngắn.
- Thuật toán tối ưu nhất.

Nhược điểm:

- Việc cài đặt thuật toán khá phức tạp.
- Cần quản lý chặt chẽ việc điều phối các tiến trình.
- Quản lý thời gian đến của mỗi tiến trình.

1.3.5. Kết luận

First-come, first-served (FCFS) là thuật toán lập lịch đơn giản nhất nhưng nó khiến các tiến trình ngắn phải chờ các tiến trình rất dài. Lập lịch Shortest-job-first (SJF) có thể là tối ưu nhất, cung cấp thời gian chờ trung bình ngắn nhất. Tuy nhiên việc thực hiện lập lịch SJF rất khó. Thuật toán SJF là một trường hợp đặc biệt của thuật toán lập lịch priority, chỉ đơn giản là phân bổ CPU cho tiến trình có độ ưu tiên cao nhất. Cả lập lịch priority và lập lịch SJF đều có thể bị trì hoãn vô định. Kĩ thuật làm già (tăng độ ưu tiên cho các tiến trình theo thời gian) là kĩ thuật để giải quyết vấn đề này.

Lập lịch Round-robin (RR) thích hợp hơn với các hệ thống chia sẻ thời gian (tương tác). Lập lịch RR chia sẻ CPU cho các tiến trình ở trạng thái sẵn sàng trong hàng đợi cho đơn vị thời gian q, trong đó q là lượng tử thời gian. Nếu sau khoảng thời gian q mà tiến trình không ngắt CPU, nó sẽ được mã hóa và đẩy xuống cuối hàng đợi. Vấn đề chính là việc lựa chọn lượng tử thời gian. Nếu lượng tử thời gian quá lớn lập lịch RR sẽ suy giảm thành lập lịch FCFS . Nếu lượng tử thời gian quá nhỏ sẽ làm cho các tiến trình phải chuyển trạng thái dẫn đến giảm hệ số hữu ích của CPU.

Thuật toán FCFS không mang tính ưu tiên còn thuật toán RR mang tính ưu tiên. Các thuật toán SJF và priority có thể mang tính ưu tiên hoặc không.

Như vậy, tùy thuộc vào từng bài toán cụ thể ta có thể áp dụng chiến lược điều phối nào cho các tiến trình nhằm đưa lại thời gian chờ đợi để được xử lý của các tiến trình một cách nhanh nhất.

CHƯƠNG 2. CÀI ĐẶT THUẬT TOÁN

2.1. Lập trình mô phỏng

2.1.1. Cấu trúc dữ liệu và cài đặt các hàm cần thiết

```
#include <stdio.h >
#include <conio.h >
#include <iostream >
#include <stdlib.h >
#include <string.h >
#include <windows.h >
using namespace std;
int totalTime = 0;
float totalTimePresence=0,timePresenceMedium=0; // Thời gian hiện diện trong CPU
float totalTimeWait=0,timeWaitMedium=0; //Thời gian chờ

struct data
{
   int appearTime; // Thời gian xuất hiện
   int executionTime; // Thời gian thực hiện
   int timeEnd; //Mốc thời gian hoàn thanh công việc
   int presenceTime; //Thời gian hiện diện trong CPU
   int waitTime; // Thời gian chờ CPU cấp phát

   char name[20];
   float ntat; //Thời gian hiện diện/thời gian xử lý
```

```
int remainTime; // Thời gian còn lại để thực hiện nốt tiến trình- Dùng cho SRT
} ;

struct nameProcessGnatt
{
   char name[20];
} ;
void input(struct data inputDataProcess[],int inputNumberProcess);
void FCFS(struct data inputDataProcess[],int inputNumberProcess);
void SJF(struct data inputDataProcess[],int inputNumberProcess);
void displayTable(struct data inputDataProcess[],int inputNumberProcess); // dùng chung cho FCFS, SJF
void displayGantt(struct data inputDataProcess[],int inputNumberProcess); // dùng chung cho FCFS, SJF, PRI
void SRT(struct data inputDataProcess[],int inputNumberProcess); // bao gồm cả table và gnatt

//varibble for round_bind
int timeCount = 0;
int front = 0; //truoc
int rear = 0; // sau
int numberProcessQueue = 0; // Số tiến trình trong hàng đợi
int m = 0, s = 0;
int queue[1000]; // Hàng đợi
int stt;
//end variable for round bind

// function for round bind
```

```
void RR_finding(struct data inputDataProcess[],int inputNumberProcess);
void push(int q) ;
int pop();
void check(struct data inputDataProcess[],int inputNumberProcess);
//end function for round bind


//setup các hàm
//Hàm nhập thông tin tiến trình và sắp xếp lại thứ tự
void input(struct data inputDataProcess[],int inputNumberProcess)
{
   struct data temp;
   cout<<"-----------------------------------\n";
   cout << "Note: Thong Tin Nhap bao gom: [Ten Tien Trinh] + [Thoi Gian Xuat Hien] + [Thoi Gian Thuc Hien] \n";
   cout << "VD: ( p1 0 24 ) ( p2 1 3) (p3 2 3)\n";
   cout << "VD: ( p1 0 11 ) ( p2 3 7) (p3 8 19) (p4 13 4) (p5 17 9)\n";
   cout<<"-----------------------------------\n";
   for(int i=0; i<inputNumberProcess; i++)
   {
      printf("+ Nhap Thong Tin Tien Trinh %d :\n", i+1);
      scanf("%s",&inputDataProcess[i].name);
      scanf("%d",&inputDataProcess[i].appearTime);
      scanf("%d",&inputDataProcess[i].executionTime);
//       inputDataProcess[i].remainTime = inputDataProcess[i].executionTime; // ban dau [thoi gian con lai de lam] bang [thoi gian thuc thi]
      cout<<"-----------------------------------\n";
   }
```

```
//Sắp xếp các tiến trình tăng dần theo thời gian xuất hiện
   for(int i=0; i<inputNumberProcess; i++)
   {
      for(int j=i; j>=1; j--)
      {
         if(inputDataProcess[j].appearTime < inputDataProcess[j-1].appearTime)
         {
            temp=inputDataProcess[j-1];
            inputDataProcess[j-1]=inputDataProcess[j];
            inputDataProcess[j]=temp;
         }
      }
   }
}


//display table ( tinh các chỉ số tiến trình hiển thị dạng bảng)
void displayTable(struct data inputDataProcess[],int inputNumberProcess)
{
   int i,j;
   for( i=0; i<inputNumberProcess; i++)
   {
      inputDataProcess[i].presenceTime = inputDataProcess[i].timeEnd -
inputDataProcess[i].appearTime; // Thời gian hiện diện trên CPU
      inputDataProcess[i].waitTime = inputDataProcess[i].presenceTime -
inputDataProcess[i].executionTime; // Thời gian chờ
      inputDataProcess[i].ntat = (float)inputDataProcess[i].presenceTime /
inputDataProcess[i].executionTime;
      totalTimePresence += inputDataProcess[i].presenceTime;
```

```
        totalTimeWait += inputDataProcess[i].waitTime;
    }

    timePresenceMedium = totalTimePresence/inputNumberProcess; //Thời gian hiện diện trung binh trên CPU
    timeWaitMedium = totalTimeWait/inputNumberProcess; //Thời gian chờ trung binh
    cout << "Note:\t [Name] \t: [Ten Tien Trinh] \n";
    cout << "\t [timeXH] \t: [Moc Thoi Gian Xuat Hien] \n";
    cout << "\t [totalTimeTH] \t: [Thoi Gian Thuc Hien, Xu Ly] \n";
    cout << "\t [timeEnd] \t: [Moc Thoi Gian Hoan Thanh Xong Cong Viec] \n";
    cout << "\t [timeCPU] \t: [Thoi Gian Hien Dien Trong CPU] \n";
    cout << "\t [waitTime] \t: [Thoi Gian Cho Doi CPU Cap Phat] \n";
    cout << "\t [ntat] \t: [Thoi gian hien dien/Thoi gian xu li] \n";
    cout << "-------------------------------------------------------------------------------------------\n";

    printf("Name \t timeXH \t totalTimeTH \t timeEnd \t timeCPU  \t waitTime \t ntat \n");
    for(i=0; i<inputNumberProcess; i++)
    {
        printf("%s \t %d \t\t %d \t\t %d \t\t %d \t\t %d \t\t %f \n",inputDataProcess[i].name,inputDataProcess[i].appearTime,inputDataProcess[i].executionTime,inputDataProcess[i].timeEnd,inputDataProcess[i].presenceTime,inputDataProcess[i].waitTime,inputDataProcess[i].ntat);
    }
    cout << "-------------------------------------------------------------------------------------------\n";

    printf("+ Thoi Gian Hien Dien trong CPU Trung Binh cua cac Tien Trinh : %f \n",timePresenceMedium);
```

```
    printf("+ Thoi Gian Cho Trung Binh cua cac Tien Trinh : %f \n",timeWaitMedium);
}
//display_gantt - FCFS , SJT, PRI
void displayGantt(struct data inputDataProcess[],int inputNumberProcess)
{
    int i,j;
    struct data newDataProcess[inputNumberProcess] ;
    struct data tg;
    for( i = 0; i<inputNumberProcess; i++)
    {
        newDataProcess[i] = inputDataProcess[i]; // Sao chép lại
    }
    // Sắp xếp theo mốc thời gian hoàn thanh công việc tăng dần- sort EndTime
    for (i = 0; i < inputNumberProcess-1; i++)
    {
        int min = i;
        for (j = i+1; j < inputNumberProcess; j++)
        {
            if (newDataProcess[j].timeEnd < newDataProcess[min].timeEnd)  min = j;


        }
        if(min != i)
        {
            tg = newDataProcess[i];
            newDataProcess[i] = newDataProcess[min];
            newDataProcess[min] = tg;
        }
```

```
    }
    // End sort EndTime


    //hiển thị ra màn hình
    for(int i = 0; i < inputNumberProcess; i++)
    {
        if(i == 0) printf("\n\n[Start--(%d)--",newDataProcess[i].appearTime);
        else printf("--(%d)--",newDataProcess[i-1].timeEnd);
        printf("[%s]",newDataProcess[i].name);
    }
    printf("--(%d)--End]\n\n",newDataProcess[inputNumberProcess-1].timeEnd);
}
```

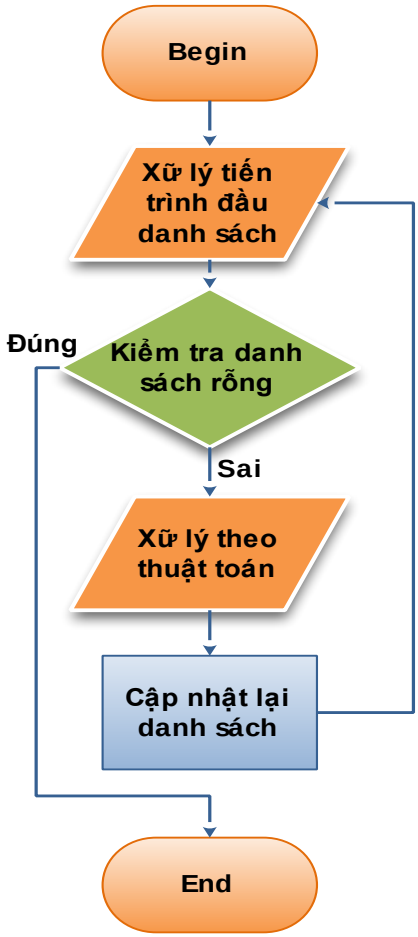
2.1.2. Thuật toán xử lý chung

Việc cài đặt thuật toán được mô phỏng theo cách làm việc của CPU và tất cả các thuật toán con đều theo mô hình thuật toán này.

Tiến trình ở đầu danh sách sẽ được ưu tiên xử lý trước và nó chiếm dụng CPU tại thời điểm đó.

Việc đi kèm theo là xem xét thời gian xử lý các tiến trình đã hết chưa. Nếu đã hết thì nghĩa là hoàn thành việc xử lý, ngược lại thì tiếp tục xử lý theo thuật toán.

Xong mỗi chu kỳ của CPU (1 quantum) thì cập nhật lại danh sách để loại bỏ các tiến trình đã hoàn thành hay sắp xếp hay thêm các tiến trình mới vào.



Hình 2.1: Sơ đồ thuật toán đề xuất chung cho các giải thuật

2.2. Thuật toán

2.2.1. Thuật toán First Come First Served (FCFS)

Code:

```
void FCFS(struct data inputDataProcess[],int inputNumberProcess)
{
   // calc timeEnd
   int stt = inputDataProcess[0].appearTime, tempi;
   inputDataProcess[0].timeEnd = stt + inputDataProcess[0].executionTime;
   stt=inputDataProcess[0].timeEnd;
   for(int i=1; i<inputNumberProcess; i++)
   {
```

```
        tempi=inputDataProcess[i].appearTime; // check thoi gian xuat hien cua tien trinh tiep theo
        if(tempi > stt) stt=tempi; // khi nhap cac tien trinh cung da dc sap xep ngay trong ham input roi
        inputDataProcess[i].timeEnd = stt + inputDataProcess[i].executionTime;
        stt=inputDataProcess[i].timeEnd;
    }
    // End calc timeEnd
    displayTable(inputDataProcess,inputNumberProcess);
}
```

2.2.2. Thuật toán Round Robin (RR)

Code:

```
void push(int stt)
{
    queue[rear++] = stt;
    //rear%=n;
    m++;
}
int pop()
{
    s++;
    int x;
    x = queue[front++];
    return x;
}

void check(struct data inputDataProcess[],int inputNumberProcess)
{
```

```
    //khi tiến trình mới xuất hiện thì đẩy nó vào hàng đợi
    while (inputDataProcess[stt].appearTime <= timeCount && stt < inputNumberProcess) //tất cả các tiến trình phải được kiểm tra, đẩy tiến trình đang thực hiện hiện tại vào hàng đợi
    {
        numberProcessQueue++;
        push(stt++);
    }
}

void RR_finding(struct data inputDataProcess[],int inputNumberProcess)
{
    // nơi lưu lại giá trị để xây dựng biểu đồ gantt
    int timeGnatt[100], countGnatt=0; // lưu lại các mốc time
    int processGnattArr[100]; //STT của các tiến trình
    int saveProcess; ////STT của tiến trình hiện tại

    timeGnatt[countGnatt] = inputDataProcess[0].appearTime;
    processGnattArr[countGnatt] = 0;
    saveProcess = 0;
//    printf("+ %s, Time=%d \n", inputDataProcess[saveProcess].name, timeGnatt[countGnatt]); //check gnatt 1
    countGnatt++;

//  int temp_st[inputNumberProcess];
    int remainTimeArr[inputNumberProcess]; // thời gian thực thi còn lại của các tiến trình
    int countQ = 0; // đếm xem tiến trình mới <= quantum chưa?
    int p_process; //p_process là stt của tiến trình hiện tại
```

```
    int flag = 0; //flag để đánh dấu các tiến trình đó đã được chọn
    stt = 0;
    int quantum;//quantum
    printf("Nhap Gia Tri quantum (q):\n");
    scanf("%d", & quantum);

    for (int i = 0; i < inputNumberProcess; i++)
    {
        remainTimeArr[i] = inputDataProcess[i].executionTime;// Mảng remainTimeArr chứa thời gian thực hiện của các tiến trình
    }

    timeCount = inputDataProcess[0].appearTime; //varibble for round_bind

    numberProcessQueue = 1;
    push(stt++);
//          void push(int q)
//                  {
//                    queue[rear++] = q;
//                    m++;
//                  }
    while (timeCount <= totalTime)
    {

        if (flag == 1 || numberProcessQueue != 0) //flag để đánh dấu tiến trình đã được chọn, 0 là chưa,1 là rồi
        {
            if (flag == 0 && countQ == 0)
```

```
{
   p_process = pop(); //queue[front] – lấy tiến trình đầu hàng đợi
   //       int pop()
   //                    {
   //                     s++;
   //                     int x;
   //                     x = queue[front++];
   //                     return x;
   //                    }
   //p_process=i;
   countQ = 0; // Là tiến trình mới nên reset quantum = 0
   flag = 1; //flag để đánh dấu tiến trình đó đã được chọn, 1 là rồi
}
   remainTimeArr[p_process]--; //Thời gian thực thi còn lại của tiến trình được chọn giảm 1

   //Check xem [[Thời gian thực hiện còn lại của tiến trình]=0 thì tiến trình đã hoan thanh
   if (remainTimeArr[p_process] == 0)
   {
      timeCount++; // timeCount tiếp tục tăng
      countQ = 0; //Reset lại quantum
      inputDataProcess[p_process].timeEnd = timeCount; //Tiến trình kết thúc tại timeCount này
      flag = 0;//flag để đánh dấu tiến trình đã được chọn, 0 là chưa
      numberProcessQueue--;
      check(inputDataProcess,inputNumberProcess);
```

```
            //gantt
            if(saveProcess != p_process)
            {
               timeGnatt[countGnatt] = timeCount; // mốc thời gian
               processGnattArr[countGnatt] = p_process; // tiến trình mới
               saveProcess = processGnattArr[countGnatt];
//                        printf("+ %s, Time=%d \n", inputDataProcess[saveProcess].name,
timeCount); //check gnatt 2
               countGnatt++;
            }
            //end gantt

            continue;
         }
         countQ++;
         //count=count%quantum;

         //check xem countQ da max = quantum
         if (countQ == quantum)
         {
            countQ = 0; //reset
            timeCount++; //timeCount tiep tuc tang
            check(inputDataProcess,inputNumberProcess);
            push(p_process); //cho vao hang doi
            flag = 0; //flag để đánh dấu tiến trình đã được chọn, 0 là chưa
//       printf("+ %s: timeCount=%d, So tien trinh trong hang doi=%d
\n",inputDataProcess[p_process].name, timeCount, numberProcessQueue );
```

```
            //gantt
            if(saveProcess != p_process)
            {
               timeGnatt[countGnatt] = timeCount; // mốc thời gian
               processGnattArr[countGnatt] = p_process; // tiến trình mới
               saveProcess = processGnattArr[countGnatt];
//                  printf("+ %s, Time=%d \n", inputDataProcess[saveProcess].name,
timeCount); //check gnatt 3
               countGnatt++;
            }
            //end gantt
         }
         else
         {
            timeCount++;
            check(inputDataProcess,inputNumberProcess);
         }
      }
      else
      {
         timeCount++;
         check(inputDataProcess,inputNumberProcess);
      }

   } //end of while loop
   cout<<endl<<endl<<"Thong tin lap lich theo thuat toan Round robin: \n\n";
//  printf("+ timeCount=%d, numberProcessQueue=%d \n", timeCount,
numberProcessQueue ); //check
```

```
   //displayTable
   displayTable(inputDataProcess,inputNumberProcess);
   //show gnatt
   timeGnatt[countGnatt]=totalTime; //time ket thuc
   cout << "\n[Start--";
   printf("(%d)", timeGnatt[0] );
   for(int i=0; i<countGnatt; i++)
   {
      printf("--[%s]--(%d)", inputDataProcess[processGnattArr[i]].name,
timeGnatt[i+1] );


   }
   cout << "--End]";
   //end show gantt
```

2.3.3. Thuật toán Shortest Job First (SJF)

Code:

```
void SJF(struct data inputDataProcess[],int inputNumberProcess)
{
   int timeCount=0,c; //c, timeCount là bộ đếm tăng dần,quá trình lưu trữ các tiến trình
đã xuất hiện
   int i =0, maxProgressOccurred = 0; // Tiến trình gần nhất đã xuất hiện sẽ lưu vào đây
//       cout << "+ Test Thuc Thi Thuat Toan SJF";
         printf("+ timeCount=%d , totalTime=%d \n",timeCount, totalTime); // check
tổng thời gian tiến trình đã đúng chưa?
   while(timeCount < totalTime)
   {
//                printf("+ timeCount=%d < totalTime=%d \n",timeCount, totalTime);
//check timeCount
```

```
c=0;
maxProgressOccurred = 0;
for(i=0; i < inputNumberProcess; i++)
{
   //Tìm ra tiến trình lớn nhất đã xuất hiện
   if(inputDataProcess[i].appearTime <= timeCount)
   {
      maxProgressOccurred=i+1;
   }
}
// lúc này maxProgressOccurred chính là tiến trình gần nhất đã xuất hiện
if(maxProgressOccurred > 0)
{
   int sttProcessMin=0; // Số thứ tự của tiến trình có time thực hiện ngắn nhất sẽ lưu vào đây
   int minExecutionTime = inputDataProcess[0].executionTime; //time thực hiện ngắn nhất đó

   // Tìm ra tiến trình có thời gian thực hiện ngắn nhất trong các tiến trình đã xuất hiện
   for(i=1; i<maxProgressOccurred; i++)
   {
      if(minExecutionTime > inputDataProcess[i].executionTime)
      {
         sttProcessMin=i;
         minExecutionTime = inputDataProcess[i].executionTime;
      }
   }
```

```
        printf("+ sttProcessMin=%d, minExecutionTime=%d \n", sttProcessMin, minExecutionTime);
        //end vòng lặp – lúc này đã tìm được tiến trình có thời gian thực hiện ngắn nhất trong các tiến trình đã xuất hiện

        timeCount += inputDataProcess[sttProcessMin].executionTime; //sttProcessMin là quá trình với thời gian phục vụ tối thiểu
        inputDataProcess[sttProcessMin].timeEnd = timeCount;
        inputDataProcess[sttProcessMin].executionTime += 999; //thiết lập thời gian thực thi lớn để nó không được chọn để so sánh nứa
      }
      else
      {
        timeCount++;
      }
   }
   // khôi phục lại thời gian thực hiện ban đầu của các tiến trình
   for(i=0; i<inputNumberProcess; i++)
   {
      inputDataProcess[i].executionTime -= 999;
   }
   displayTable(inputDataProcess,inputNumberProcess);
```

2.3.4. Thuật toán Shortest Remain Time (SRT)

Code:

```
void SRT(struct data inputDataProcess[],int inputNumberProcess)
{
   // nơi lưu lại giá trị để xây dựng biểu đồ gantt
   int timeGnatt[100], countGnatt=0; // lưu lại các mốc time
   int processGnattArr[100]; //STT của các tiến trình
```

```
    int saveProcess; ////STT của tiến trình hiện tại


    timeGnatt[countGnatt] = inputDataProcess[0].appearTime;
    processGnattArr[countGnatt] = 0;
    saveProcess = 0;
//    printf("+ %s, Time=%d \n", inputDataProcess[saveProcess].name, timeGnatt[countGnatt]); //check gnatt 1
    countGnatt++;


    //srt
    int remainTimeArr[inputNumberProcess]; // thời gian còn lại của các tiến trình
    int TimeLandmarkArr[inputNumberProcess + 1]; // mốc thời gian để check lại[time thực thi còn lại] của các tiến trình
    for(int i = 0; i<inputNumberProcess; i++)
    {
        remainTimeArr[i]=inputDataProcess[i].executionTime;
        TimeLandmarkArr[i]=inputDataProcess[i].appearTime;
//                printf("remainTime[%d]=%d ; TimeLandmarkArr[%d]=%d\n",i+1, inputDataProcess[i].executionTime,i+1, inputDataProcess[i].appearTime);
    }
    TimeLandmarkArr[inputNumberProcess] = totalTime; // mốc time lớn nhất
    int saveSttTimeLandmark=0; //Ban dau moc tai vi tri dau tien
//        int timeLandmark = TimeLandmarkArr[saveSttTimeLandmark]; // mốc thời gian xét


    int timeCount=0,c; //c,timeCount là bộ đếm tăng dần, quá trình lưu trữ các tiến trình đã xuất hiện
```

```
   int i =0, maxProgressOccurred = 0; // tiến trình gần nhất xuất hiện sẽ được lưu vào đây
//        cout << "+ Test Thuc Thi Thuat Toan SRT\n";


   while(timeCount < totalTime && timeCount <= TimeLandmarkArr[saveSttTimeLandmark])
   {
//                printf("\n------ timeCount=%d <= timeLandmark=%d -----\n",timeCount, TimeLandmarkArr[saveSttTimeLandmark]); //check timeCount
      c=0;
      maxProgressOccurred = 0;
      for(i=0; i < inputNumberProcess; i++)
      {
         //Tìm ra tiến trình lớn nhất đã xuất hiện
         if(inputDataProcess[i].appearTime <= timeCount)
         {
            maxProgressOccurred=i+1;
         }
      }
      // lúc này maxProgressOccurred chính là tiến trình gần nhất đã xuất hiện
      if(maxProgressOccurred > 0)
      {
         int sttProcessMin=0; // Số thứ tự của tiến trình có thời gian thực hiện còn lại ngắn nhất sẽ được lưu vào đây
         int minRemainTime = remainTimeArr[0]; //thời gian thực hiện còn lại ngắn nhất đó


         // tìm ra tiến trình có thời gian thực hiện còn lại ngắn nhất trong tất cả các tiến trình đã xuất hiện
```

```
        for(i=1; i<maxProgressOccurred; i++)
        {
           if(minRemainTime > remainTimeArr[i])
           {
              sttProcessMin=i;
              minRemainTime = remainTimeArr[i];
           }
        }
//          printf("+ %s, minRemainTime=%d \n", inputDataProcess[sttProcessMin].name, minRemainTime);
        //end vòng lặp – tìm được tiến trình có thời gian thực hiện ngắn nhất trong các tiến trình đã xuất hiện


        //gantt
        if(saveProcess != sttProcessMin)
        {
           timeGnatt[countGnatt] = timeCount; // mốc thời gian
           processGnattArr[countGnatt] = sttProcessMin; // tiến trình mới
           saveProcess = processGnattArr[countGnatt];
//                      printf("+ %s, Time=%d \n", inputDataProcess[saveProcess].name, timeCount); //check gnatt 2
           countGnatt++;
        }
        //end gantt


        remainTimeArr[sttProcessMin]--; // chạy được 1s nên time còn lại giảm
//          printf("+ remainTime[%d]=%d\n", sttProcessMin, remainTimeArr[sttProcessMin]);
        timeCount++;
```

```
            //kiểm tra tiến trình đã chạy xong chưa
            if(remainTimeArr[sttProcessMin] == 0)
            {
               inputDataProcess[sttProcessMin].timeEnd = timeCount;
//                inputDataProcess[sttProcessMin].executionTime += 999; //thiết lập thời gian thực thi lớn nhất để nó không được chọn để so sánh tiếp
               remainTimeArr[sttProcessMin]=999; ////thiết lập thời gian thực thi còn lại lớn nhất để nó không được chọn để so sánh tiếp
            }

        }
        else
        {
           timeCount++;
        }
        //kiểm tra mốc time
        if(timeCount > TimeLandmarkArr[saveSttTimeLandmark] )
        {
           saveSttTimeLandmark++; // tang len moc time tiep theo
//         printf("+ saveSttTimeLandmark=%d\n", saveSttTimeLandmark);
        }
    }
    //show table
    displayTable(inputDataProcess,inputNumberProcess);

    //show gnatt
    timeGnatt[countGnatt]=totalTime; //time ket thuc
    cout << "\n[Start--";
```

```
    printf("(%d)", timeGnatt[0] );
    for(int i=0; i<countGnatt; i++)
    {
        printf("--[%s]--(%d)", inputDataProcess[processGnattArr[i]].name,
timeGnatt[i+1] );
    }
    cout << "--End]";
    //end show gantt
}
```

2.3. Minh họa qua một bài toán lập lịch CPU

Process	$T_{t/hiện}$	$T_{x/hiện}$
P1	11	0
P2	7	3
P3	19	8
P4	4	13
P5	9	17

a) Vẽ sơ đồ Gantt

b) Tính thời gian chờ đợi trung bình của các tiến trình.

2.3.1. First Come First Served (FCFS)

a. Sơ đồ Gantt

P1	P2	P3	P4	P5	
0	11	18	37	41	50

b. Tính thời gian chờ trung bình

Thời gian chờ của P1 = 0

Thời gian chờ của P2 = 8

Thời gian chờ của P3 = 10

Thời gian chờ của P4 = 24

Thời gian chờ của P5 = 24

Thời gian chờ đợi trung bình của các tiến trình:

Ttb = (0 + 8 + 10 + 24 + 24)/5 = 13,2.

2.3.2. Shortest Job First (SJF)

a. Sơ đồ Gantt

P1	P2	P1	P4	P1	P5	P3	
0	3	10	13	17	22	31	50

b. Tính thời gian chờ trung bình

Thời gian chờ của P1 = 0

Thời gian chờ của P2 = 8

Thời gian chờ của P3 = 23

Thời gian chờ của P4 = 5

Thời gian chờ của P5 = 5

Thời gian chờ đợi trung bình của các tiến trình:

Ttb = (0 + 8 + 23+ 5+ 5)/5 = 8,2.

2.3.3. Shortest Remain Time (SRT)

a. Sơ đồ Gantt

P1	P2	P1	P4	P1	P5	P3	
0	3	10	13	17	22	31	50

b. Tính thời gian chờ trung bình

Thời gian chờ của P1 = 11

Thời gian chờ của P2 = 0

Thời gian chờ của P3 = 23

Thời gian chờ của P4 = 0

Thời gian chờ của P5 = 5

Thời gian chờ đợi trung bình của các tiến trình:

Ttb = (11 + 0 + 23+ 0 + 5)/5 = 7,8.

2.3.4. Round robin (RR)

a. Sơ đồ Gantt

P1	P1	P2	P1	P3	P2	P1	P4	P3	P5	P2	P4	P3	P5	P3	P5	P3	
0	3	6	9	12	15	18	20	23	26	29	30	31	34	37	40	43	50

b. Tính thời gian chờ trung bình

Thời gian chờ của P1 = 9

Thời gian chờ của P2 = 20

Thời gian chờ của P3 = 23

Thời gian chờ của P4 = 14

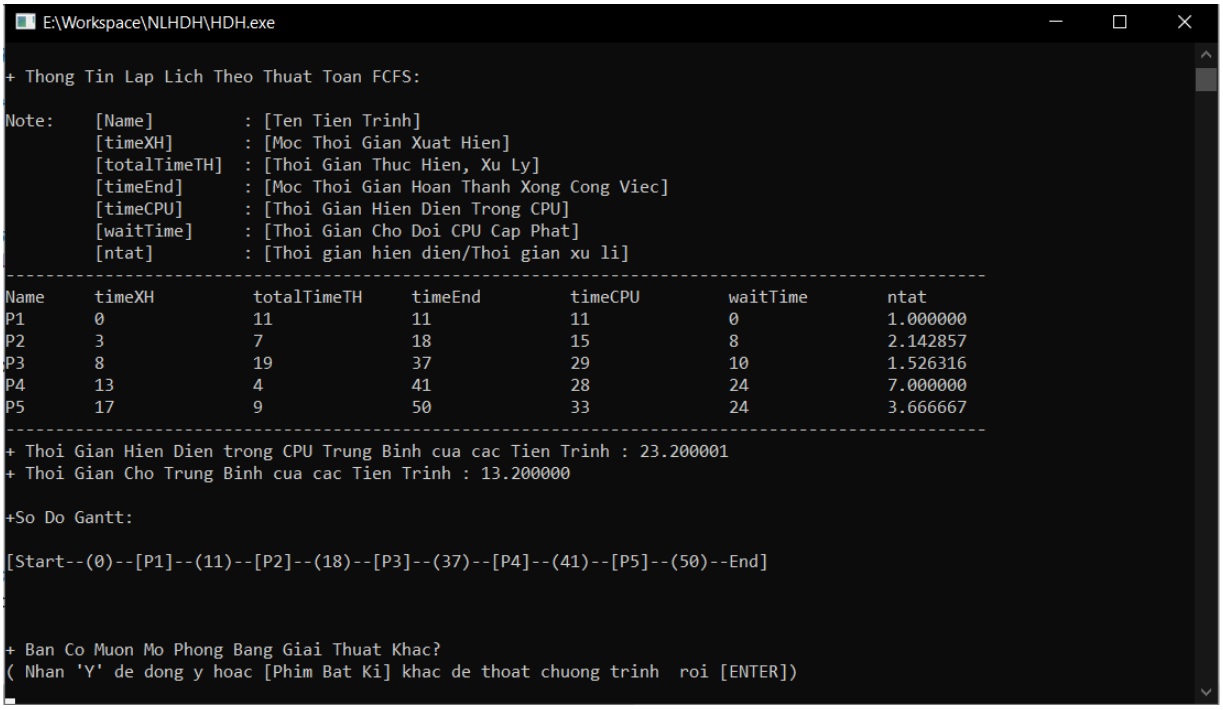
Thời gian chờ của P5 = 17

Thời gian chờ đợi trung bình của các tiến trình:

Ttb = (9 + 20 + 23+ 14+ 17)/5 = 16,4.

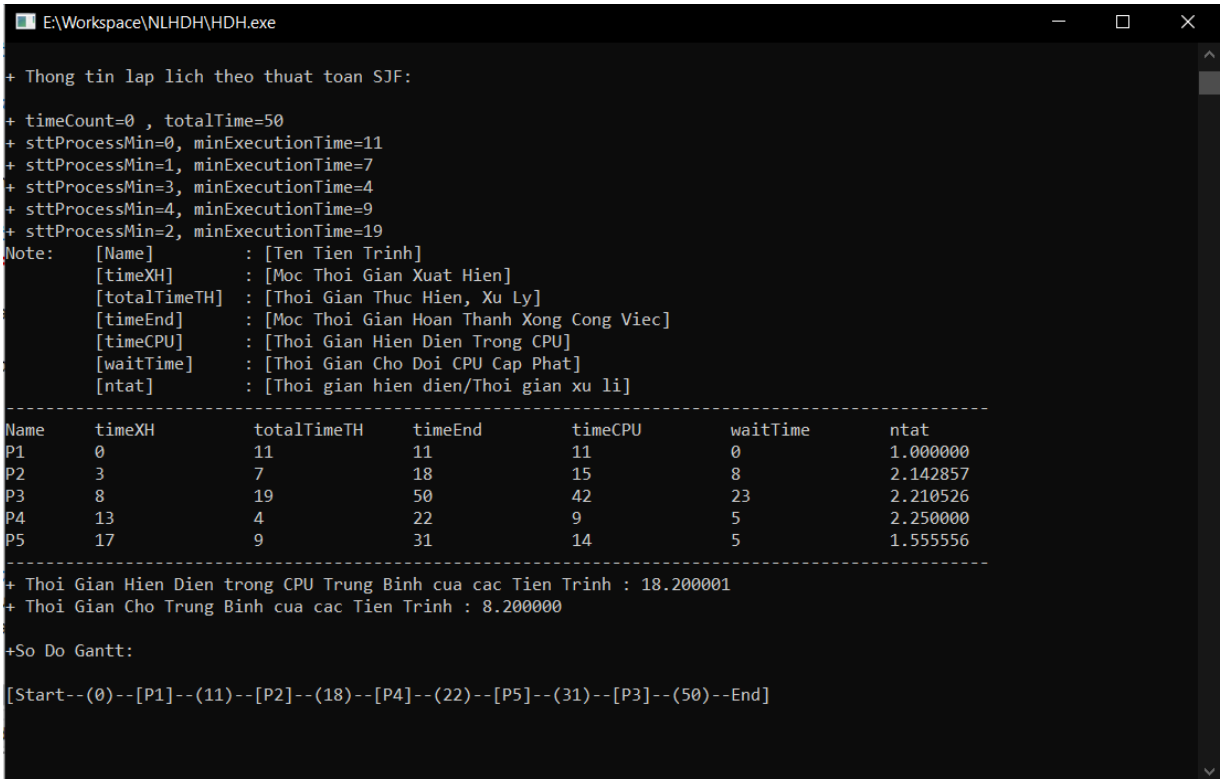
2.4. Kết quả chạy code

2.4.1. First Come First Served (FCFS)



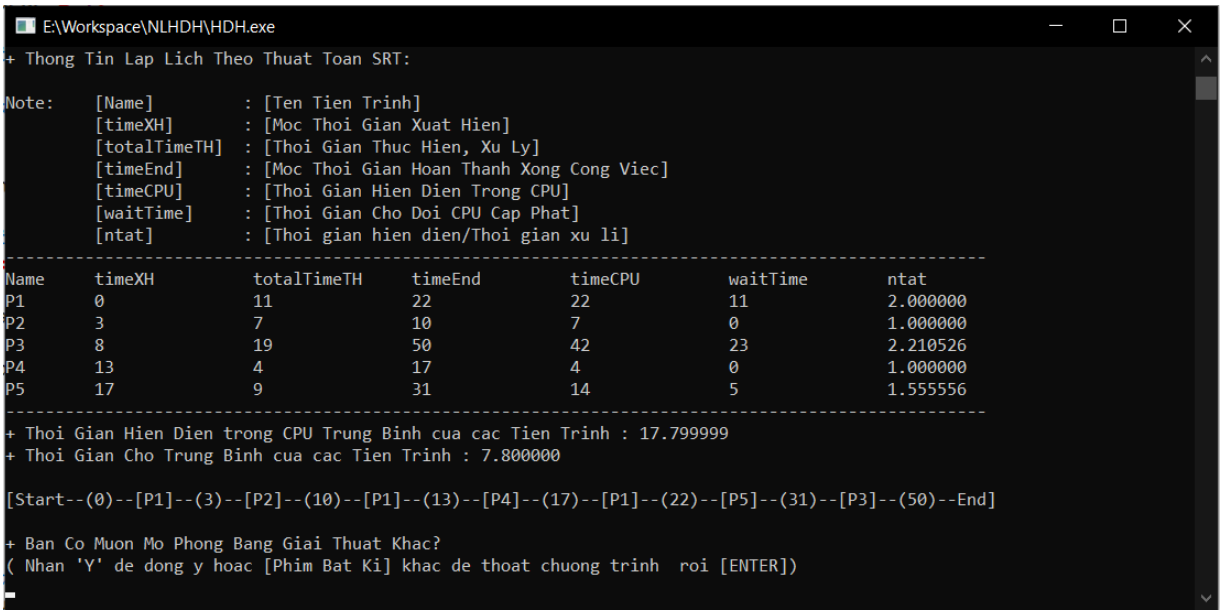
Hình 2.2: Kết quả chạy thuật toán FCFS

2.4.2. Shortest Job First (SJF)



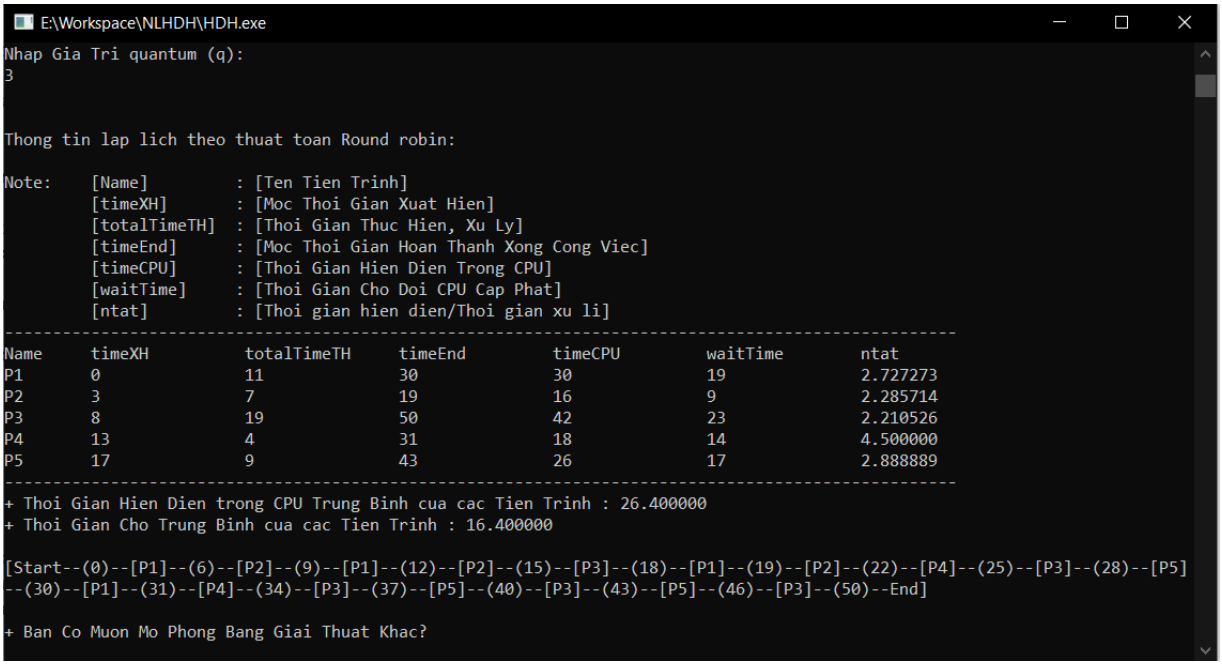
Hình 2.3: Kết quả chạy thuật toán SJF

2.4.3. Shortest Remain Time (SRT)



Hình 2.4: Kết quả chạy thuật toán SRT

2.4.4. Round Robin (RR)



Hình 2.5: Kết quả chạy thuật toán RR

KẾT LUẬN

Qua quá trình tìm hiểu và nghiên cứu về các phương pháp lập lịch cho CPU. Nhóm 6 chúng em đã củng cố thêm được kiến thức và hiểu biết thêm về CPU, hiểu được tầm quan trọng và vai trò của CPU trong việc điều khiển và quản lý các tiến trình.

Nhưng điều quan trọng hơn là qua bài tập lớn lần này chúng em đã học tập được nhiều kinh nghiệm để làm việc theo nhóm. Học hỏi được nhiều phương pháp tìm kiếm, tra cứu thông tin. Chúng em đã tìm kiếm được nhiều thông tin bổ ích không những cho bài tập lớn của mình mà còn nhiều thông tin bổ ích khác liên quan đến ngành học của mình thông qua các kênh thông tin khác nhau.

Tuy nhiên, chúng em vẫn chưa có kinh nghiệm và kiến thức còn hạn chế không thể tránh khỏi những sai sót, khiếm khuyết mong thầy giáo và các bạn góp ý để chúng em chỉnh sửa.

Chúng em xin chân thành cảm ơn!

TÀI LIỆU THAM KHẢO

Tài liệu tiếng Việt

1. Nguyễn Thanh Hải, Giáo trình Nguyên lý hệ điều hành, 2016.
2. Michael Tischer (1992), Cẩm nang lập trình hệ thống, NXB Giáo dục.

Tài liệu tiếng Anh

1. Abraham Silberschatz, Peter B. Galvin, Greg Gagne (2009), Operating System Concepts 8th edition.