|  |  |
| --- | --- |
| TRƯỜNG ĐẠI HỌC SƯ PHẠM  THÀNH PHỐ HỒ CHÍ MINH | KHOA CÔNG NGHỆ THÔNG TIN |
|  |  |

TIỂU LUẬN KẾT THÚC HỌC PHẦN

MÔN HỆ ĐIỀU HÀNH



TP. HCM – 6/2025



|  |  |
| --- | --- |
| tTRƯỜNG ĐẠI HỌC SƯ PHẠM THÀNH PHỐ HỒ CHÍ MINH | KHOA CÔNG NGHỆ THÔNG TIN |

TIỂU LUẬN KẾT THÚC HỌC PHẦN

MÔN HỆ ĐIỀU HÀNH



Đề tài:

Câu 1:

a)Tìm hiểu và so sánh mô hinh quản lý tập tin của hệ điều hành window và Linux

b)Tìm hiểu, phân tích và đánh giá các thuật toán lập lịch sử dụn CPU

c)Tìm hiểu, phân tích bộ nhớ thực, bộ nhớ ảo và các chiến lược cấp phát

Câu 2:

a)Xây dựng chương trình minh hoạ giải thuật banker để phát triển và giải quyến vấn để deallock

b)Xây dựng chương trình mình họa các giải thuật phân phối CPU cho các tiến trinh.

c)Xây dựng chương trình minh hoạ các giải thuật thay thế trang trong quy trình PFSR

d)Xây dựng chương trình minh hoạ giải thuật định thời truy cập đĩa để quản lý hệ thống tập tin.

HỌC PHẦN: COMP1332 – HỆ ĐIỀU HÀNH

Nhóm sinh viên thực hiện:

Trần Đức Anh – 49.01.104.006

Trần Nguyễn Gia Huy – 49.01.104.059

Ngô Dương Đức Thắng – 49.01.104.137

Nguyễn Thị Vinh – 49.01.104.175

Giảng viên hướng dẫn: ThS. Trần Quang Huy

NHIỆM VỤ THÀNH VIÊN NHÓM

|  |  |
| --- | --- |
| Trần Đức Anh | Giải thuật phân phối CPU (câu 2b), hỗ trợ phần 2a, phần 1b và thuyết trình |
| Trần Nguyễn Gia Huy | Giải thuật Banker (câu 2a), hỗ trợ phần 2b, câu 1b, thiết kế giao diện và làm ppt |
| Ngô Dương Đức Thắng | Giải thuật Paging Replacement (câu 2c), hỗ trợ phần 2d, câu 1c và làm word |
| Nguyễn Thị Vinh | Giải thuật Disk Scheduling (câu 2d), hỗ trợ phần 2c, câu 1a và làm word |

MỤC LỤC

[**LỜI NÓI ĐẦU** 6](#_Toc201242818)

[**PHẦN 1: LÝ THUYẾT** 7](#_Toc201242819)

[**1. TÌM HIỂU LỊCH SỬ PHÁT TRIỂN HỆ ĐIỀU HÀNH** 7](#_Toc201242820)

[1.1. Hệ Điều hành là gì 7](#_Toc201242821)

[1.2. Các thành phần của một hệ thống máy tính 7](#_Toc201242822)

[1.3. Các thành phần của hệ điều hành 8](#_Toc201242823)

[1.4. Lịch sử phát triển hệ điều hành 8](#_Toc201242824)

[2. TÌM HIỂU VÀ SO SÁNH MÔ HÌNH QUẢN LÝ TẬP TIN CỦA HỆ ĐIỀU HÀNH WINDOWS VÀ LINUX 11](#_Toc201242825)

[2.1 Khái quát mô hình quản lý tập tin 11](#_Toc201242826)

[2.2. Tổng quan hệ thống quản lý tập tin 11](#_Toc201242827)

[2.3. Cấu trúc hệ thống tập tin Windows 13](#_Toc201242828)

[2.4. Cấu trúc hệ thống tập tin Linux 13](#_Toc201242829)

[2.5. So sánh mô hình quản lý tập tin Windows và Linux 14](#_Toc201242830)

[**3. TÌM HIỂU, PHÂN TÍCH ĐÁNH GIÁ CÁC THUẬT TOÁN LẬP LỊCH SỬ DỤNG CPU** 15](#_Toc201242831)

[3.1. Các mục tiêu của việc lập lịch 15](#_Toc201242832)

[3.2. Mức ưu tiên trong hệ thống 17](#_Toc201242833)

[3.3. Các tiêu chí đánh giá lập lịch 17](#_Toc201242834)

[3.4. Các giải thuật lập lịch 18](#_Toc201242835)

[**4. TÌM HIỂU PHÂN TÍCH BỘ NHỚ THỰC, BỘ NHỚ ẢO, CÁC CHIẾN LƯỢC CẤP PHÁT** 20](#_Toc201242836)

[4.1. Khái niệm cơ bản về Bộ nhớ 20](#_Toc201242837)

[4.2. Mục tiêu và yêu cầu của Quản lý bộ nhớ 21](#_Toc201242838)

[4.3. Các chiến lược Cấp phát Bộ nhớ Liên tục 22](#_Toc201242839)

[4.4. Bộ nhớ Ảo và các Kỹ thuật Cấp phát Bộ nhớ không liên tục (Non-Contiguous Allocation) 23](#_Toc201242840)

[4.5.Các Kỹ thuật Quản lý Bộ nhớ Ảo Nâng cao 27](#_Toc201242841)

[**PHẦN 2: THỰC HÀNH VÀ ỨNG DỤNG** 29](#_Toc201242842)

[**1. GIỚI THIỆU** 29](#_Toc201242843)

[**2. CHƯƠNG TRÌNH MINH HOẠ GIẢI THUẬT BANKER ĐỂ PHÁT TRIỂN VÀ GIẢI QUYẾT DEADLOCK** 31](#_Toc201242844)

[**2.1. Giới thiệu chung** 31](#_Toc201242845)

[**2.2. Các khái niệm cơ bản** 31](#_Toc201242846)

[**2.3. Cách thức thực hiện** 32](#_Toc201242847)

[2.4. Xây dựng chương trình mô phỏng thuật toán Banker 36](#_Toc201242848)

[**3. CHƯƠNG TRÌNH MINH HOẠ CÁC GIẢI THUẬT PHÂN PHỐI CPU CHO CÁC TIẾN TRÌNH** 39](#_Toc201242849)

[3.1. Chương trình CPU 39](#_Toc201242850)

[3.2. FCFS 41](#_Toc201242851)

[3.3. SJF độc quyền 46](#_Toc201242852)

[3.4. SJF không độc quyền 48](#_Toc201242853)

[3.5. Priority độc quyền 49](#_Toc201242854)

[3.6. Priority không độc quyền 52](#_Toc201242855)

[3.7. Round-Robin 56](#_Toc201242856)

[4. CHƯƠNG TRÌNH MINH HOẠ GIẢI THUẬT THAY THẾ TRONG QUY TRÌNH PFSR 59](#_Toc201242857)

[4.1.Khi nào PFSR được kích hoạt? 59](#_Toc201242858)

[4.2.Quy trình xử lý của PFSR 59](#_Toc201242859)

[4.3. Thực hành và giao diện 61](#_Toc201242860)

[**5. CHƯƠNG TRÌNH MINH HOẠ GIẢI THUẬT ĐINH THỜI TRUY CẬP ĐĨA ĐỂ QUẢN LÝ HỆ THỐNG TẬP TIN** 70](#_Toc201242861)

[5.1 Định thời truy cập đĩa: 70](#_Toc201242862)

[5.2 Xây dựng chương trình mô phỏng thuật toán định thời truy cập đĩa: 73](#_Toc201242863)

[**TÀI LIỆU THAM KHẢO** 82](#_Toc201242864)

# **LỜI NÓI ĐẦU**

Đề tài tiểu luận này được thực hiện trong khuôn khổ môn học Điều hành Hệ thống máy tính, nhằm nghiên cứu và giải quyết các vấn đề liên quan đến việc quản lý tài nguyên và xử lý đa nhiệm trong môi trường hệ điều hành. Các bài tập được trình bày trong tiểu luận nhằm thể hiện khả năng ứng dụng các thuật toán điều phối, phân phối tài nguyên và giải quyết các vấn đề như deadlock trong các hệ thống máy tính.

Đề tài sẽ bao gồm các nội dung chính như việc cài đặt và tối ưu hóa các thuật toán trong việc quản lý bộ nhớ, xử lý các tiến trình, và phân phối CPU, đồng thời ứng dụng những kiến thức đã học để xây dựng chương trình giải quyết các vấn đề thực tiễn trong môi trường hệ điều hành.

Thông qua tiểu luận này, chúng ta có thể hiểu rõ hơn về cách thức hoạt động của hệ điều hành trong việc phân chia tài nguyên một cách hợp lý và hiệu quả, từ đó tạo tiền đề vững chắc cho việc nghiên cứu và phát triển các hệ thống máy tính trong tương lai.

.

# **PHẦN 1: LÝ THUYẾT**

## **1. TÌM HIỂU LỊCH SỬ PHÁT TRIỂN HỆ ĐIỀU HÀNH**

### 1.1. Hệ Điều hành là gì

Hệ điều hành (operating systems) là chương trình đóng vai trò trung gian giữa người sử dụng và phần cứng của máy tính. Hệ điều hành che dấu sự phức tạp, đa dạng của phần cứng, giúp việc sử dụng máy tính trở nên đơn giản, hiệu quả. Nhiệm vụ của hệ điều hành là quản lý tài nguyên của máy tính, thực thi các chương trình ứng dụng, hỗ trợ các chức năng mạng.

### 1.2. Các thành phần của một hệ thống máy tính

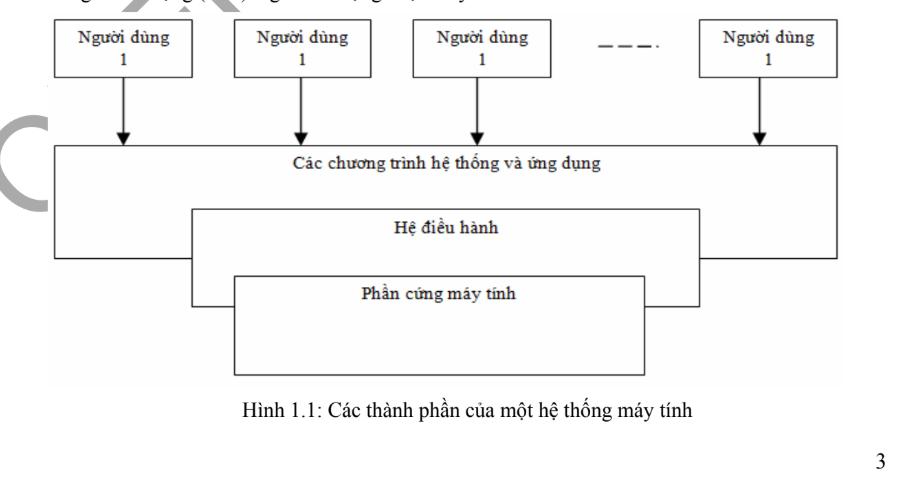
Một hệ thống máy tính được chia thành 4 phần sau :phần cứng, hệ điều hành, chương trình ứng dụng/chương trình hệ thống, người sử dụng.

+ Phần cứng(hardware):CPU,bộ nhớ,các thiết bị nhập/xuất,…

+ Hệ điều hành (operating systems): điều khiển và phối hợp việc sử dụng phần cứng cho nhiều ứng dụng với nhiều người sử dụng khác nhau.

+ Chương trình ứng dụng và chương trình hệ thống (system and applications programs): là các chương trình giải quyết những vấn đề của người sử dụng như chương trình dịch , hệ quản trị cơ sở dữ liệu, chương trình trò chơi, chương trình thương mại,…

+ Người sử dụng (user): người sử dụng hoặc máy tính.



### 1.3. Các thành phần của hệ điều hành

Hệ điều hành gồm có ba phần sau:

+Bộ cấp phát tài nguyên (resource allocator): Quản lý và cấp phát tài nguyên. + + +Chương trình kiểm soát (Control program): Kiểm soát việc thực thi chương trình và kiểm soát hoạt động của các thiết bị nhập/xuất.

+Phần nhân(Kernel): là chương trình “lõi” của hệ điều hành, được thực thi trước tiên và tồn tại trong bộ nhớ cho đến khi tắt máy ( các chương trình khác gọi là chương trình ứng dụng).

### 1.4. Lịch sử phát triển hệ điều hành

Giai đoạn (1945 • 1955): Người dùng viết chương trình chủ yếu trên ngôn ngôn ngữ máy. Các thiết bị ngoại vi còn nghèo cả về số lượng lẫn chủng loại. Phần mềm trợ giúp cho người lập trình chỉ là các thư viện chương trình mẫu và một số chương trình phục vụ. Các ngôn ngữ lập trình đang ở trong giai đoạn hình thành, chưa được đưa ra khai thác rộng rãi.

Giai đoạn (1955 – 1965): Các ngôn ngữ lập trình được sử dụng rộng rãi. Máy tính được trang bị các chương trình dịch. Hệ thống phần mềm được bổ sung các chương trình phục vụ nạp, dịch và thực hiện chương trình ứng dụng, đồng thời hỗ trợ một phần các công việc liên quan tới thiết bị ngoại vi. Những người làm việc với máy tính được phân thành hai lớp: thao tác viên và người lập trình. Vì vậy, ở giai đoạn này xuất hiện ngôn ngữ để thao tác viên đưa yêu cầu vào hệ thống và để người lập trình báo cho thao tác viên biết các công việc cần thực hiện. Đó là ngôn ngữ vận hành hệ thống.

Giai đoạn (1965 – 1980): Với sự ra đời của máy tính thế hệ III, công suất máy tính tăng một cách đáng kể, máy tính được trang bị nhiều thiết bị ngoại vi đa dạng đòi hỏi phải có một hệ thống phục vụ với nhiều tính năng mới. Hệ điều hành với các chức năng như ta thường thấy ngày nay ra đời ở giai đoạn này. Đặc điểm quan trọng của hệ thống là khả năng đảm bảo môi trường đa nhiệm. Hệ điều hành tiêu biểu của giai đoạn này là OS IBM 360/370. Trong hệ điều hành này, các nguyên tắc cơ bản về quản lí thiết bị ngoại vi, quản lí .bộ nhớ, điều độ thực hiện chương trình, ... được hiện thực hoá một cách có hiệu quả. Mạng máy tính và hệ điều hành mạng (trên cơ sở OS IBM 360/370) cũng được ra đời trong những năm bảy mươi của thế kỉ XX. Giai đoạn sau năm 1980: Đây là giai đoạn phát triển bùng nổ của máy tính cá nhân. Những hệ điều hành trang bị cho máy tính cá nhân đầu những năm tám mươi thế kỉ XX như COMMANDOR, APPLE II, IBM PC XT, ... còn đơn giản, về tính năng không khác nhiều với hệ điều hành những nãm sáu mươi của thế kỉ XX. Tuy vậy, trên cơ sở sự phát triển như vũ bão của khoa học công nghệ và được kế thừa kinh nghiệm xây dựng hệ điều hành trước đó, những hệ điều hành cho các máy tính cá nhân nhanh chóng đạt đến chuẩn mực đa nhiệm nhiều người dùng. Từ những năm chín mươi của thế kỉ trước, mạng Internet đã trở nên phổ biến. Hệ điểu hành được mở rộng bổ sung thêm khả năng khai thác có hiệu quả các mạng cục bộ cũng như mạng diện rộng trên cơ sở Internet.

Hệ điều hành Windows

WINDOWS 95. Ra đời tháng 9 - 1995. Đây là hệ điều hành thế hệ mới trong các hệ điểu hành mà hãng Microsoft cung cấp cho thị trường thế giới. Nó được phát triển từ phần mềm Windows 3.11 trước đó. Chế độ đa nhiệm được định hướng cho một người dùng. Các chương trình thực hiện trong Windows 95 có thể đổng thời sử dụng chung các tài nguyên của hệ thống. Biểu tượng

WINDOWS 95 Người dùng cũng được đảm bảo các khả năng làm việc trong môi trường mạng, trao đổi thư điện tử, fax. Hệ thống giao diện đồ hoạ được thiết kế đảm bảo cho người dùng thuận tiện tối đa và nhanh chóng hiểu rõ, làm chủ hệ thống. Chế độ bảo vệ áp dụng trong Windows 95 làm cho hệ thống không bị phá hỏng khi một chương trình ứng dụng gặp lỗi trong quá trình thực hiện, đồng thời làm cho hệ thống an toàn hơn trước tác động của virus.

WINDOWS NT (NT - New Technology) là một hệ điều hành được trang bị nhiều công cụ quản trị mạng, có một hệ thống quản lí tệp hiệu quả và có thể làm việc với đĩa có dung lượng cực lớn. Đây là một hệ thống cho phép khai thác máy tính có hiệu quả trong môi trường mạng cục bộ.

WINDOWS 98 - Được trang bị chương trình duyệt Internet Explorer cho phép làm việc thuận tiện hơn với các trang web. Việc kết nối hệ thống với Internet cũng thuận tiện và đơn giản hơn nhiều. Việc cài đặt Windows 98 cho máy tính để bàn và máy tính xách tay (Notebook) cũng đơn giản như nhau. Đây là một đặc điểm quan trọng, vì ở thời kì này, máy tính xách tay đã trỏ nên rất phổ biến.

WINDOWS 2000/ WINDOWS XP - Hệ thống được nâng cấp để thực hiện chế độ đa nhiệm cho nhiều người dùng. Mỗi người dùng có một môi trường hệ thống riêng, như có một phiên bản hệ điều hành độc lập cho họ, việc thay đổi người dùng được tiến hành đơn giản và không làm gián đoạn hoạt động hệ thống. Windows 2000/XP đảm bảo khả năng “Cắm và chạy” (Plug and Play) vô cùng phong phú cho nhiều thiết bị ngoại vi khác nhau. Các hệ điều hành lớp này có hẳn một thành phần phục vụ quản lí một cách thuận tiện tài nguyên phân tán: cung cấp tài nguyên của mình (màn hình, bàn phím, bộ nhớ ngoài, …) cho các hệ thống khác trong mạng và sử dụng tài nguyên của hệ thống khác. Vì vậy, một hệ điều hành của máy trạm (Client) không khác nhiều so với hệ điều hành quản trị mạng (Server). Windows 2000 Server được trang bị nhiều công cụ để quản trị mạng. Bên cạnh máy tính cá nhân, các siêu máy tính cũng được tiếp tục nghiên cứu phát triển, chế tạo và đưa vào khai thác trong thực tế. Các máy này được lắp ráp với nhiều bộ xử lí. Hệ điều hành được phát triển đảm bảo khả năng tính toán song song: một bài toán có thể được chia thành nhiều phần nhỏ và các phần này được thực hiện đồng thời trên các bộ xử lí, làm giảm đáng kể thời gian thực hiện. Hệ điều hành mạng Sự ra đời và phát triển của các mạng máy tính: Mạng cục bộ (LAN) và mạng diện rộng (WAN) đòi hỏi phải có hệ điểu hành với nhiều chức năng phục vụ quản lí mạng, đảm bảo an toàn và bảo mật. Những hệ điều hành này được gọi là hệ điều hành mạng

+Novell NetWare là hệ điều hành mạng phổ cập rộng rãi đầu tiên trên các máy tính cá nhân, phục vụ cho việc tổ chức mạng cục bộ.

+Windows NT Server là hệ điều hành mạng với các công cụ quản lí tệp hiệu quả cao, sử dụng cho các máy chủ của mạng cục bộ. +Windows 2000 Server là hệ điều hành cho máy chủ của mạng, cung cấp nhiều dịch vụ kết nối với mạng Internet cho mạng cục bộ. Hai hệ điều hành mạng Unix và Linux có đầy đủ các tính năng đa nhiệm, nhiều người dùng và phục vụ truyền thông tốt. Unix cũng như Linux hỗ trợ tốt môi trường lập trình và các ứng dụng mạng.

## 2. TÌM HIỂU VÀ SO SÁNH MÔ HÌNH QUẢN LÝ TẬP TIN CỦA HỆ ĐIỀU HÀNH WINDOWS VÀ LINUX

### 2.1 Khái quát mô hình quản lý tập tin

Trong bất kỳ hệ điều hành nào, quản lý tập tin là một chức năng thiết yếu và đóng vai trò quan trọng trong việc điều phối hoạt động lưu trữ và truy xuất dữ liệu. Tập tin là đơn vị cơ bản để lưu thông tin và là nền tảng cho mọi hoạt động tính toán. Vì vậy, việc tổ chức, kiểm soát và bảo vệ tập tin là nhiệm vụ cốt lõi của hệ điều hành.

Hai hệ điều hành phổ biến hiện nay, Windows và Linux, có cách tiếp cận rất khác nhau trong việc thiết kế mô hình quản lý tập tin. Sự khác biệt này không chỉ đến từ kiến trúc kỹ thuật mà còn phản ánh triết lý phát triển và đối tượng người dùng mà mỗi hệ điều hành hướng đến.

### 2.2. Tổng quan hệ thống quản lý tập tin

Hệ thống quản lý tập tin (File System Management) là thành phần của hệ điều hành chịu trách nhiệm lưu trữ, tổ chức, truy xuất và bảo vệ dữ liệu trên các thiết bị lưu trữ - như ổ cứng, SSD, USB.

Tập tin (File): là đơn vị lưu trữ thông tin cơ bản, có tên, kích thước, và tồn tại độc lập trên bộ nhớ.

Thư mục (Directory): là tập tin đặc biệt chứa nhiều tập tin khác, được tổ chức theo dạng cây thư mục để dễ quản lý.

Hệ điều hành cho phép người dùng tương tác với tập tin thông qua các API hoặc giao diện, trong khi ẩn đi các chi tiết phần cứng như cấu trúc đĩa, sector hoặc block.

2.2.1. Cấu trúc và thuộc tính tập tin

Tập tin có thể được tổ chức theo nhiều dạng cấu trúc:

Dãy byte liên tục: không định dạng (binary).

Tập hợp record cố định: như trong hệ quản trị CSDL.

Cây dữ liệu: record có quan hệ cha – con.

Thuộc tính tập tin thường bao gồm:

Tên file

Kích thước

Ng tạo, ngày chỉnh sửa

Quyền truy cập (read/write/execute)

Kiểu tập tin (binary, text, executable...)

2.2.2. Cấu trúc hệ thống thư mục

Các hệ điều hành có thể tổ chức thư mục theo các cách:

Đơn cấp (single-level): tất cả tập tin nằm chung một thư mục.

Hai cấp (two-level): mỗi người dùng có thư mục riêng.

Đa cấp (tree-structured): phân cấp thư mục lồng nhau, là dạng phổ biến nhất.

Hai dạng đường dẫn được hỗ trợ:

Tuyệt đối (absolute): từ thư mục gốc, ví dụ /home/user/file.txt

Tương đối (relative): từ thư mục hiện hành.

2.2.3. Các thao tác cơ bản trên tập tin và thư mục

Tập tin: tạo (create), mở (open), đọc (read), ghi (write), đổi tên (rename), xóa (delete), cập nhật thuộc tính (chmod).

Thư mục: tạo, xóa, liên kết (hard link, symbolic link), di chuyển.

2.2.4. Quản lý phân vùng và volume

Một ổ đĩa vật lý có thể chia thành nhiều partition (phân vùng) và volume (khối lượng logic).

* Partition chính (Primary Partition) thường chứa hệ điều hành và được thiết lập là active.
* MBR (Master Boot Record) lưu thông tin phân vùng và code khởi động ban đầu.

2.2.5. Các phương pháp cấp phát không gian lưu trữ

* Liên tiếp (Contiguous Allocation): các block lưu trữ nằm liên tục → truy xuất nhanh, nhưng dễ phân mảnh ngoài.
* Liên kết (Linked Allocation): mỗi block chứa con trỏ đến block kế → giảm phân mảnh nhưng chậm hơn.
* Chỉ số (Indexed Allocation): sử dụng bảng chỉ số lưu địa chỉ các block dữ liệu.
* Kết hợp (Hybrid): ví dụ UNIX dùng i-node với nhiều mức con trỏ.

i-node (UNIX/Linux): mỗi tập tin có một inode chứa metadata và địa chỉ dữ liệu.

### 2.3. Cấu trúc hệ thống tập tin Windows

2.3.1 Hệ thống FAT (File Allocation Table)

Là hệ thống tập tin cũ, được sử dụng từ thời MS-DOS.  
 Gồm 3 phần chính: bảng FAT1, FAT2 (sao lưu), thư mục gốc và vùng dữ liệu.  
 Mỗi tập tin có tên giới hạn 8 ký tự và phần mở rộng 3 ký tự (định dạng 8.3).  
 Dữ liệu được lưu thành các khối (clusters), mỗi khối được liên kết bằng bảng FAT (giống danh sách liên kết).  
 Nhược điểm: dễ phân mảnh, giới hạn dung lượng, bảo mật kém.

2.3.2 NTFS (New Technology File System)

Là hệ thống tập tin hiện đại, dùng mặc định trong các phiên bản Windows NT trở lên.  
 Dữ liệu được quản lý bởi bảng Master File Table (MFT).  
 Hỗ trợ:

* Phân quyền chi tiết (ACL).
* Ghi log các giao dịch để phục hồi khi lỗi hệ thống.
* Nén, mã hóa, phân quyền, hỗ trợ tập tin lớn.
* Tên file Unicode, độ dài tên lớn hơn FAT.

Hệ thống mô-đun cho phép mở rộng và cải tiến liên tục.

### 2.4. Cấu trúc hệ thống tập tin Linux

2.4.1 Cấu trúc tổng quát

Linux quản lý tập tin thông qua VFS (Virtual File System) – lớp trung gian giúp hỗ trợ nhiều định dạng hệ thống tập tin như ext2, ext3, ext4, XFS, FAT, NTFS...  
 VFS cung cấp giao diện trừu tượng, làm cho các hệ thống tập tin cụ thể có thể dùng chung cơ chế truy xuất.

2.4.2 EXT2/EXT4 – hệ thống tập tin đặc thù của Linux

Dựa trên cấu trúc inode – mỗi tập tin có một inode chứa metadata và địa chỉ khối dữ liệu.  
 Một phân vùng được chia thành nhiều Block Group, mỗi group chứa inode table, bitmap, block data.  
 EXT4 cải tiến hiệu suất, hỗ trợ journaling, kích thước file lớn, ít phân mảnh hơn.  
 Hỗ trợ mount động, liên kết symbolic/hard link, phân quyền rwx rõ ràng cho user/group/others.

### 2.5. So sánh mô hình quản lý tập tin Windows và Linux

|  |  |  |
| --- | --- | --- |
| Tiêu chí | Windows (NTFS) | Linux (EXT2/EXT4 + VFS) |
| Mô hình quản lý | FAT/NTFS với cấu trúc bảng (FAT table hoặc MFT) | inode + block group, quản lý qua VFS |
| Cấu trúc thư mục | Cấu trúc cây với ổ đĩa riêng (C:, D:...) | Cây thư mục duy nhất bắt đầu từ gốc / |
| Cơ chế phân quyền | ACL chi tiết, mã hóa (EFS) | rwx (read/write/execute) + ACL mở rộng |
| Bảo mật | Tốt, nhưng đóng mã nguồn | Tùy biến cao, tích hợp SELinux/AppArmor (mã nguồn mở) |
| Khả năng mở rộng | Hạn chế định dạng, tích hợp phần mềm dễ | Hỗ trợ nhiều hệ thống tập tin và phần cứng khác nhau |
| Khả năng phục hồi lỗi | Ghi log giao dịch ($LOGFILE) cho phục hồi sau sự cố | Journaling trong ext3/ext4 giúp giữ tính toàn vẹn dữ liệu |
| Hiệu suất | Tốt với SSD và hệ thống lớn, nhưng dễ phân mảnh | Hiệu suất cao hơn với ext4, ít phân mảnh hơn |
| Tương thích | Phù hợp phần mềm phổ biến, văn phòng, game | Phù hợp máy chủ, hệ thống nhúng, người dùng chuyên sâu |

Mô hình quản lý tập tin trong Windows và Linux phản ánh triết lý thiết kế khác nhau:

Windows (NTFS): hướng tới người dùng phổ thông và doanh nghiệp, ưu tiên tính ổn định, phục hồi và bảo mật với cấu trúc MFT hiện đại.

Linux (EXT4 + VFS): hướng đến hiệu năng, sự linh hoạt và khả năng mở rộng, phù hợp cho hệ thống máy chủ và lập trình viên.

Tùy vào mục đích sử dụng – như ổn định và dễ dùng (Windows) hay tùy biến và hiệu năng cao (Linux), người dùng hoặc tổ chức có thể lựa chọn mô hình phù hợp.

## **3. TÌM HIỂU, PHÂN TÍCH ĐÁNH GIÁ CÁC THUẬT TOÁN LẬP LỊCH SỬ DỤNG CPU**

Tiêu chuẩn lập lịch

Các mức lập lịch: Có thể chia thành 3 mức lập lịch khác nhau:

Lập lịch mức cao.

Lập lịch mức giữa.

Lập lịch mức thấp.

Lập lịch mức cao, hay lập lịch cho các task: các công cụ ở mức này xác định bài toán (chương trình) nào được đưa vào hệ thống, nghĩa là tạo ra tiến trình tương ứng với chương trình đó. Lập lịch mức giữa: mức này xác định các tiến trình được sử dụng bộ xử lý. Bộ lập lịch ở mức này phản ứng với các thay đổi của hệ thống. Nó sẽ dừng hoặc kích hoạt các tiến trình để đảm bảo hệ thống hoạt động bình thường, đạt các thông số kỹ thuật đề ra. Lập lịch mức thấp: công cụ ở mức này xác định Ready Process nào tiếp theo sẽ được quyền sử dụng bộ xử lý, do đó thường được gọi là Dispacher.

### 3.1. Các mục tiêu của việc lập lịch

Cơ chế lập lịch cần đạt được các mục tiêu sau

Đúng đắn, nghĩa là cơ chế lập lịch cần phục vụ các tiến trình “công bằng”, tránh tình huống có tiến trình bị rơi vào tình trạng chờ vô hạn.

Đảm bảo khả năng thông qua lớn nhất, tức là tiến tới phục vụ số lượng tiến trình nhiều nhất có thể trong một đơn vị thời gian.

Thời gian phản ứng chấp nhận được với tất cả các tiến trình tối thiểu chi phí, tài nguyên hệ thống.

Cân đối việc sử dụng tài nguyên, cần cố gắng nâng cao hiệu suất sử dụng tài nguyên, theo đó cần ưu tiên tiến trình sử dụng tài nguyên giá thành thấp.

Đảm bảo cân đối giữa thời gian trả lời và hiệu suất sử dụng tài nguyên. Cách tốt nhất để giảm thời gian trả lời là có đủ tài nguyên dự trữ để khi có yêu cầu có thể cấp phát ngay lập tức, nhưng điều đó cũng dẫn tới lãng phí tài nguyên.

Ngăn ngừa tình huống chờ vô hạn.

Cần quan tâm các tiến trình đang sử dụng tài nguyên quan trọng, tránh tình trạng tiến trình có mức ưu tiên thấp chiếm tài nguyên mà tiến trình mức ưu tiên cao hơn cần. Nếu tài nguyên đó là không chia sẻ thì hệ điều hành cần tạo điều kiện để tiến trình giải phóng tài nguyên nhanh nhất.

Chúng ta thấy rằng nhiều yêu cầu, mục tiêu trái ngược nhau, do đó việc lập lịch cho các tiến trình là bài toán phức tạp.

Tiêu chuẩn lập lịch Để đạt được các mục tiêu ở trên, cơ chế lập lịch cần chú ý các yếu tố sau:

Tiến trình có thực hiện yêu cầu thao tác I/O không?

Tiến trình có sử dụng bộ xử lý hết lượng tử thời gian (Quantum) hay không?

Yêu cầu về thời gian trả lời hệ thống cần đạt được.

Mức ưu tiên của từng tiến trình.

Tần suất ngắt Missing Page Fault.

Thời gian tổng cộng tiến trình được sử dụng bộ xử lý.

### 3.2. Mức ưu tiên trong hệ thống

Nói chung các tiến trình có vai trò quan trọng khác nhau. Mức độ quan trọng của tiến trình được thể hiện qua mức ưu tiên (Priority) của nó. Mức ưu tiên của tiến trình được gán bởi hệ điều hành, và phụ thuộc kiến trúc của hệ điều hành mà mức ưu tiên đó có thể là động hoặc tình, có thể được gán theo các tiêu chuẩn xác định hoặc ngẫu nhiên (trong trường hợp hệ điều hành không phân biệt được tiến trình nào cần mức ưu tiên cao hơn). Trong hệ thống sử dụng mức ưu tiên tình, mức ưu tiên của tiến trình được

gần ngay khi nó được tạo ra và không thay đổi trong suốt quá trình tồn tại của tiễn trình. Sơ đồ mức ưu tiên tình dễ dàng thiết kế và cải đặt hơn. Tuy nhiên chúng không có khả năng điều chỉnh để phù hợp với sự thay đổi của môi trường. Ngày nay trong hệ điều hành đều sử dụng sơ đổ mức ưu tiên động. Theo đó mức ưu tiên của tiễn trinh có thể thay đổi khác với mức ưu tiên khởi tạo ban đầu. Cơ chế này cho phép hệ thống thích nghi với sự thay đổi của môi trường để đạt chỉ tiêu tốt hơn, tuy nhiên nó cũng khó khăn hơn trong xây dựng và cải đặt.

Khoảng lượng tử thời gian, ngắt thời gian

Như ta đã biết, tiễn trinh chỉ thực sự hoạt động khi nó sử dụng bộ xử lý. Nếu tiến trinh là tiến trình hệ thống thì lúc đó hệ điều hành thực sự hoạt động. Để tránh tình trạng độc quyền chiếm giữ bộ xử lý, hệ điều hành có các cơ chế cho phép lấy lại quyền kiểm soát bộ xử lý.

Hệ điều hành thiết lập đồng hồ hệ thống, xác định khoảng thời gian gọi là lượng từ thời gian, theo đó sinh ra các tín hiệu ngắt thời gian. Khi đó bộ xử lý chuyển sang phục vụ tiến trinh tiếp theo. Như thế, tiến trinh có thể chiếm bộ xử lý đến khi nó tự giải phóng hoặc khi có ngắt tiếp theo.

Khi bộ xử lý được giải phóng, hệ điều hành sẽ xác định tiến trình nào tiếp theo được chiếm bộ xử lý. Ngắt thời gian giúp hệ thống đảm bảo thời gian trả lời chấp nhận được với tất cả tiến trình, tránh tình trạng chở vô hạn, đồng thời cho phép hệ thống phản ứng với các sự kiện phụ thuộc thời gian.

### 3.3. Các tiêu chí đánh giá lập lịch

Khả năng tận dụng CPU (CPU utilization): Thể hiện qua tải CPU – là một số từ 0% đến 100%. Trong thực tế các hệ thống thường có tải từ 40% (tải thấp) đến 90% (tải cao)

Thông lượng: Nếu CPU bận thực thi các quá trình thì công việc đang được thực hiện. Thước đo của công việc là số lượng quá trình được hoàn thành trên một đơn vị thời gian gọi là thông lượng (throughput).

Thời gian hoàn thành: Khoảng thời gian từ thời điểm gởi quá trình tới khi quá trình hoàn thành.

Thời gian chờ: Là tổng thời gian chờ trong hàng đợi sẵn sàng.

Thời gian đáp ứng: Thời gian từ lúc gởi yêu cầu cho tới khi đáp ứng đầu tiên.

|  |  |  |
| --- | --- | --- |
| Tiêu chí | giá trị thấp | giá trị cao |
| Khả năng tận dụng CPU | xấu | tốt |
| Thông lượng(Throughput) | xấu | tốt |
| Thời gian hoàn thành(turnaround time) | tốt | xấu |
| Thời gian chờ(awaiting time) | tốt | xấu |
| Thời gian đáp ứng(reponse time) | tốt | xấu |

### 3.4. Các giải thuật lập lịch

.3.4.1 Lập lịch theo thời gian kết thúc

Khi áp dụng giải thuật này, hệ thống sử dụng tất cả khả năng hiện có để một ứng dụng nào đó có thể kết thúc trong thời hạn định trước. Ví dụ, trường hợp điều khiển tên lửa, các kết quả tính toán chỉ có ý nghĩa trước thời điểm nào đó. Lập lịch theo cơ chế này vấp phải các khó khăn:

Người dùng cần chỉ rõ các tài nguyên cần thiết phục vụ cho ứng dụng và điều này không phải luôn dễ dàng thực hiện.

Hệ thống một mặt phải thực hiện ứng dụng đúng hạn, mặt khác không được làm ảnh hưởng “quá nhiều” đến các ứng dụng khác. Rất có thể xảy ra việc tranh chấp tài nguyên giữa các ứng dụng.

Nếu đồng thời có nhiều yêu cầu kết thúc các ứng dụng đúng thời hạn thì vấn đề lập lịch có thể rất phức tạp.

Việc phức tạp trong lập lịch thường kéo theo chi phí tài nguyên lớn hơn và làm ảnh hưởng đến cả hệ thống

3.4.2. Lập lịch theo nguyên tắc FIFO (FCFS)

Có lẽ đây là nguyên tắc lập lịch đơn giản nhất. Theo đó bộ xử lý phục vụ các tiến trình theo thứ tự trong danh sách các Ready Process.

A yellow rectangular object with black text

AI-generated content may be incorrect.

Sau khi tiến trình được quyền sử dụng bộ xử lý, nó được thực hiện đến khi kết thúc. Nguyên tắc FIFO là không hoán đổi, nghĩa là bộ xử lý không thực hiện phục vụ quay vòng lần lượt các Ready Process mà phục vụ từng tiến trình đến khi kết thúc. Nguyên tắc FIFO có tính xác định cao, có thể dự đoán tương đối chính xác thời gian thực hiện các bài toán. Tuy nhiên vì nó là không hoán đổi nên dễ xảy ra trường hợp tiến trình quan trọng hơn phải chờ các tiến trình khác đứng trước trong danh sách kết thúc mới được thực hiện

Vì thế hiện nay nguyên tắc này không được áp dụng đơn thuần mà thường kết hợp với các phương pháp khác trong các biện pháp tổ hợp.

3.4.3. Lập lịch theo nguyên tắc SJF (Shortes Job First)

Nguyên tắc SJF là nguyên tắc không hoán đổi, theo đó bài toán có thời gian thực hiện ngắn nhất theo dự đoán sẽ được thực hiện trước.

Khi CPU được tự do, nó sẽ được cấp phát cho tiến trình yêu cầu ít thời gian nhất để kết thúc. Hai giải pháp:

Không ưu tiên trước - Một CPU đã cung cấp cho một tiến trình nào thì phải chờ cho tiến trình đó xử lý xong mới lấy lại.

Có ưu tiên trước– Một tiến trình mới đến nhưng có thời gian yêu cầu ngắn hơn thời gian còn lại của tiến trình đang chạy, khi đó phải dừng hoạt động để chuyển cho tiến trình mới. (Giống mô hình Shortest-Remaining-Time-First (SRTF).

SJF là giải thuật tối ưu – cho phép đạt được thời gian chờ trung bình cực tiểu.

## **4. TÌM HIỂU PHÂN TÍCH BỘ NHỚ THỰC, BỘ NHỚ ẢO, CÁC CHIẾN LƯỢC CẤP PHÁT**

Trong kỷ nguyên công nghệ thông tin hiện đại, khả năng thực thi đồng thời nhiều chương trình (đa chương trình) là một yêu cầu cơ bản đối với hầu hết các hệ thống máy tính. Để đạt được hiệu quả này, việc quản lý tài nguyên, đặc biệt là bộ nhớ chính (RAM), đóng vai trò vô cùng quan trọng và phức tạp trong các hệ điều hành (Operating System – OS). Quản lý bộ nhớ không chỉ đơn thuần là việc phân bổ không gian lưu trữ cho các tiến trình mà còn bao gồm các nhiệm vụ thiết yếu như tối ưu hóa việc sử dụng tài nguyên, đảm bảo tính bảo mật và cho phép chia sẻ dữ liệu một cách hiệu quả giữa các ứng dụng.

### 4.1. Khái niệm cơ bản về Bộ nhớ

Bộ nhớ chính, hay RAM, là thành phần cốt lõi của hệ thống máy tính, nơi lưu trữ tạm thời các chương trình và dữ liệu mà CPU đang cần truy cập để thực thi. Bộ nhớ được tổ chức như một dãy các ô nhớ liên tục, mỗi ô nhớ (thường là một word hoặc byte) có một địa chỉ vật lý duy nhất. Để một chương trình có thể chạy, mã lệnh và dữ liệu của nó phải được nạp vào các ô nhớ vật lý này.

Trong quá trình quản lý và truy cập bộ nhớ, hai loại địa chỉ chính được sử dụng để phân biệt giữa cách chương trình nhìn nhận bộ nhớ và cách phần cứng thực sự tương tác với nó. Địa chỉ Vật lý (Physical Address) là địa chỉ thực tế mà Đơn vị Quản lý Bộ nhớ (MMU - Memory Management Unit) hoặc CPU gửi đến bus bộ nhớ để truy cập một vị trí cụ thể trong RAM. Đây chính là địa chỉ mà phần cứng bộ nhớ hiểu và phản hồi trực tiếp. Ngược lại, Địa chỉ Luận lý (Logical Address), còn được gọi là địa chỉ ảo, là địa chỉ được CPU sinh ra trong quá trình thực thi chương trình. Đây là địa chỉ mà chương trình sử dụng để tham chiếu đến các vị trí trong không gian bộ nhớ riêng của nó, thường là tương đối so với một điểm bắt đầu của chương trình, ví dụ như "12 byte so với địa chỉ gốc”. Đóng vai trò cầu nối thiết yếu giữa hai loại địa chỉ này, MMU đảm nhiệm việc chuyển đổi động các địa chỉ luận lý do CPU tạo ra thành địa chỉ vật lý tương ứng trong bộ nhớ chính, một quá trình nền tảng cho nhiều kỹ thuật quản lý bộ nhớ hiện đại.

### 4.2. Mục tiêu và yêu cầu của Quản lý bộ nhớ

Quản lý bộ nhớ là một chức năng thiết yếu của hệ điều hành, nhằm tối ưu hóa việc sử dụng tài nguyên bộ nhớ và nâng cao hiệu suất tổng thể của hệ thống. Mục tiêu chính của việc này là tăng cường tính đa chương trình – tức là khả năng nạp và thực thi nhiều tiến trình cùng lúc trong bộ nhớ chính.

Để đạt được mục tiêu đó, quản lý bộ nhớ phải đáp ứng một số yêu cầu cơ bản. Đầu tiên và quan trọng nhất là cấp phát vùng nhớ cho các process, đảm bảo mỗi tiến trình có đủ không gian cần thiết để chứa mã lệnh và dữ liệu của nó. Bên cạnh đó, hệ điều hành cần có khả năng tái định vị (Relocation) các tiến trình, cho phép chúng được di chuyển trong bộ nhớ chính trong quá trình thực thi (ví dụ, khi thực hiện swapping) mà không làm ảnh hưởng đến hoạt động của tiến trình đó.

Yêu cầu về bảo vệ (Protection) là tối quan trọng, nhằm đảm bảo rằng một tiến trình không thể truy cập hoặc sửa đổi vùng nhớ thuộc về các tiến trình khác hoặc của hệ điều hành một cách trái phép, từ đó duy trì tính ổn định và bảo mật của toàn hệ thống. Để tối ưu hóa việc sử dụng bộ nhớ và tạo điều kiện cho giao tiếp giữa các tiến trình, hệ điều hành còn cần hỗ trợ khả năng chia sẻ (Sharing) các vùng bộ nhớ chung, chẳng hạn như các thư viện dùng chung. Cuối cùng, kết gán địa chỉ (Binding) là quá trình ánh xạ địa chỉ luận lý của tiến trình sang địa chỉ vật lý thực trong bộ nhớ, một hoạt động có thể diễn ra tại các thời điểm khác nhau: thời gian biên dịch, thời gian tải, hoặc thời gian thực thi.

Nếu không có cơ chế quản lý bộ nhớ hiệu quả, hệ thống sẽ phải đối mặt với các vấn đề nghiêm trọng như phân mảnh bộ nhớ (fragmentation), giới hạn thấp về mức độ đa chương, và khó khăn trong việc chạy các chương trình có kích thước lớn, gây suy giảm hiệu suất đáng kể.

Chương 2: Bộ nhớ Thực và các Chiến lược Cấp phát Bộ nhớ Liên tục (Contiguous Allocation)

Bộ nhớ thực đề cập đến dung lượng RAM vật lý có sẵn trong máy tính. Trong các chiến lược cấp phát bộ nhớ liên tục, mỗi tiến trình (hoặc một phần của tiến trình) phải được cấp phát một khối bộ nhớ duy nhất và nằm liên tiếp trong bộ nhớ chính.

### 4.3. Các chiến lược Cấp phát Bộ nhớ Liên tục

Các chiến lược cấp phát liên tục là những phương pháp cơ bản nhất trong quản lý bộ nhớ, trong đó toàn bộ chương trình hoặc một phần của nó được cấp phát một khối bộ nhớ liền kề.

* Mô hình quản lý bộ nhớ đơn giản Đây là phương pháp cơ bản nhất, thường được áp dụng trong các hệ thống đơn nhiệm hoặc hệ thống nhúng. Trong mô hình này, chỉ có một chương trình người dùng được nạp và thực thi trong bộ nhớ cùng lúc với hệ điều hành, tạo nên một môi trường quản lý bộ nhớ tương đối thẳng thắn nhưng kém linh hoạt.
* Fixed Partitioning (Phân vùng cố định) Trong chiến lược phân vùng cố định, bộ nhớ chính được chia thành một số lượng cố định các phân vùng với kích thước được định nghĩa trước khi hệ thống khởi động. Mỗi phân vùng này chỉ có khả năng chứa một tiến trình tại một thời điểm, và các phân vùng có thể có kích thước bằng nhau hoặc khác nhau. Phương pháp này nổi bật nhờ sự đơn giản trong thiết kế và triển khai, cùng với chi phí quản lý thấp. Tuy nhiên, nhược điểm chính của nó là Phân mảnh nội (Internal Fragmentation). Cụ thể, nếu một tiến trình có kích thước nhỏ hơn phân vùng được cấp phát, phần bộ nhớ không sử dụng trong phân vùng đó sẽ bị lãng phí và không thể cấp phát cho tiến trình khác. Hơn nữa, chiến lược này còn giới hạn số lượng tiến trình có thể chạy đồng thời và kích thước chương trình bị giới hạn bởi kích thước của phân vùng lớn nhất có sẵn.
* Dynamic Partitioning (Phân vùng động) Khác với phân vùng cố định, trong chiến lược phân vùng động, bộ nhớ không được chia sẵn mà được cấp phát động theo yêu cầu thực tế của từng tiến trình. Khi một tiến trình cần bộ nhớ, nó sẽ được cấp một khối bộ nhớ liên tục có kích thước vừa đủ nhu cầu. Khi tiến trình kết thúc, vùng nhớ của nó được giải phóng, tạo thành một "lỗ trống" (hole). Phương pháp này sử dụng bộ nhớ hiệu quả hơn Fixed Partitioning, vì nó giảm thiểu phân mảnh nội. Tuy nhiên, vấn đề lớn nhất của phân vùng động là Phân mảnh ngoại (External Fragmentation). Theo thời gian, khi các tiến trình được nạp và giải phóng, bộ nhớ có thể bị chia thành nhiều lỗ trống nhỏ không liên tục. Mặc dù tổng dung lượng của các lỗ trống này có thể đủ để chứa một tiến trình mới, nhưng do chúng không liên tục nên không thể cấp phát được một khối liên tục. Phương pháp này cũng đòi hỏi các thuật toán cấp phát phức tạp hơn như First-Fit, Best-Fit, hoặc Worst-Fit để tìm kiếm và cấp phát lỗ trống phù hợp.
* Swapping (Hoán đổi) Swapping là một kỹ thuật được sử dụng để tăng mức độ đa chương và cho phép các tiến trình có kích thước lớn hơn bộ nhớ chính được thực thi. Đây là quá trình di chuyển toàn bộ một tiến trình từ bộ nhớ chính ra bộ nhớ phụ (thường là ổ đĩa cứng) và ngược lại.

Quá trình swapping bao gồm hai pha chính: Swap-out, đẩy một tiến trình từ bộ nhớ chính ra đĩa nhằm giải phóng không gian, và Swap-in, đưa một tiến trình từ đĩa vào lại bộ nhớ chính để tiếp tục thực thi. Mặc dù swapping cải thiện đáng kể mức độ đa chương và hỗ trợ các chương trình lớn, nó lại là một hoạt động tốn kém về thời gian I/O do tốc độ truy cập đĩa chậm hơn nhiều so với RAM. Bên cạnh đó, nó cũng có thể gây ra hiện tượng phân mảnh bộ nhớ.

* Compaction (Nén bộ nhớ) Để giải quyết vấn đề phân mảnh ngoại vi trong các hệ thống cấp phát động, kỹ thuật nén bộ nhớ được sử dụng. Compaction di chuyển tất cả các khối bộ nhớ đang được sử dụng lại gần nhau, từ đó gộp các lỗ trống nhỏ rải rác thành một hoặc vài khối trống lớn, liên tục. Mặc dù compaction giúp khôi phục các vùng bộ nhớ bị phân mảnh và tạo ra các khối trống đủ lớn để cấp phát cho các tiến trình mới, đây là một hoạt động rất tốn kém về thời gian xử lý và tài nguyên hệ thống, vì nó yêu cầu di chuyển một lượng lớn dữ liệu trong bộ nhớ, làm gián đoạn hoạt động của CPU.

### 4.4. Bộ nhớ Ảo và các Kỹ thuật Cấp phát Bộ nhớ không liên tục (Non-Contiguous Allocation)

Các chiến lược cấp phát bộ nhớ liên tục tuy đơn giản nhưng lại đối mặt với vấn đề phân mảnh, đặc biệt là phân mảnh ngoại vi, làm giảm hiệu quả sử dụng bộ nhớ. Để khắc phục những hạn chế này và nâng cao khả năng quản lý bộ nhớ, khái niệm bộ nhớ ảo và các kỹ thuật cấp phát không liên tục đã được phát triển.

4.4.1. Bộ nhớ Ảo (Virtual Memory)

Bộ nhớ Ảo (Virtual Memory) là một khái niệm cốt lõi trong các hệ điều hành hiện đại, mang lại một cách tiếp cận linh hoạt hơn trong quản lý bộ nhớ. Bộ nhớ ảo cho phép một tiến trình được thực thi ngay cả khi toàn bộ không gian địa chỉ luận lý của nó không nằm trọn vẹn trong bộ nhớ chính cùng một lúc. Thay vào đó, chỉ những phần cần thiết của tiến trình được giữ trong RAM, trong khi phần còn lại được lưu trữ trên bộ nhớ phụ (thường là ổ đĩa cứng) và chỉ được nạp vào khi chúng thực sự được yêu cầu.

Nhu cầu về bộ nhớ ảo phát sinh vì không phải tất cả mã và dữ liệu của một chương trình đều cần thiết trong bộ nhớ chính tại mọi thời điểm. Chẳng hạn, các đoạn mã xử lý lỗi có thể hiếm khi được gọi, hoặc các mảng dữ liệu lớn có thể chỉ được truy cập một phần. Hơn nữa, bộ nhớ ảo cho phép các chương trình có kích thước vượt quá dung lượng bộ nhớ vật lý hiện có được chạy.

Ưu điểm chính của bộ nhớ ảo bao gồm việc tăng mức độ đa chương trình, vì chỉ một phần của tiến trình cần nằm trong RAM, cho phép nhiều tiến trình cùng tồn tại. Nó cũng phá vỡ rào cản về kích thước bộ nhớ vật lý, cho phép các chương trình có kích thước lớn hơn RAM vẫn có thể được chạy. Cuối cùng, việc chỉ nạp các phần cần thiết vào bộ nhớ khi chúng được yêu cầu giúp giảm số lần thao tác I/O không cần thiết.

4.4.2. Phân trang (Paging)

Paging là một kỹ thuật cấp phát bộ nhớ không liên tục phổ biến và hiệu quả nhất, đóng vai trò nền tảng trong việc triển khai bộ nhớ ảo.

Ý tưởng cơ bản: Trong Paging, bộ nhớ vật lý (RAM) được chia thành các khối có kích thước cố định và bằng nhau, được gọi là frame (hoặc page frame). Đồng thời, bộ nhớ luận lý (không gian địa chỉ ảo của mỗi chương trình) cũng được chia thành các khối có kích thước cố định và bằng nhau, gọi là page (hoặc trang nhớ). Điều quan trọng là kích thước của page và frame luôn bằng nhau, thường là lũy thừa của 2 (ví dụ: 4KB). Khi một chương trình được nạp vào bộ nhớ để chạy, các page của nó có thể được nạp vào bất kỳ frame trống nào trong bộ nhớ vật lý, mà không cần phải nằm liên tục. Điều này cho phép hệ điều hành tận dụng tối đa các vùng bộ nhớ trống nhỏ lẻ.

Page Table (Bảng phân trang): Hệ điều hành duy trì một Page Table riêng cho mỗi tiến trình. Bảng này đóng vai trò quan trọng trong việc ánh xạ địa chỉ luận lý của một page thành địa chỉ frame vật lý tương ứng. Mỗi mục trong bảng chứa số frame vật lý nơi page luận lý tương ứng được nạp. Địa chỉ của Page Table trong bộ nhớ được lưu trữ trong thanh ghi Page Table Base Register (PTBR), và kích thước của bảng trang được lưu trong thanh ghi Page-table length register (PTLR).

Cơ chế chuyển đổi địa chỉ (Address Translation): Quá trình chuyển đổi từ địa chỉ luận lý sang địa chỉ vật lý là một phần cốt yếu của Paging. Mỗi địa chỉ luận lý do CPU sinh ra được chia thành hai phần: Page Number (p), xác định số thứ tự của trang trong không gian địa chỉ luận lý, và Page Offset (d), xác định độ lệch (vị trí byte) bên trong trang đó. Để chuyển đổi, hệ thống sử dụng giá trị p làm chỉ mục để tra cứu trong Page Table của tiến trình hiện tại, tìm ra địa chỉ frame vật lý (f). Địa chỉ vật lý cuối cùng được tạo thành bằng cách kết hợp số frame f với độ lệch d (Địa chỉ Vật lý = f \* kích thước frame + d). Tuy nhiên, việc này đòi hỏi hai lần truy cập bộ nhớ cho mỗi lần truy cập dữ liệu (một lần cho Page Table và một lần cho dữ liệu thực). Để tăng tốc độ chuyển đổi địa chỉ, TLB (Translation Look-aside Buffer) được sử dụng. TLB là một bộ nhớ đệm tốc độ cao, nhỏ, chuyên lưu trữ các cặp ánh xạ page-frame được truy cập gần đây nhất, giúp giảm đáng kể thời gian truy cập bộ nhớ.

Các Page Table có thể trở nên rất lớn khi không gian địa chỉ ảo lớn và kích thước page nhỏ, gây tốn bộ nhớ. Để giải quyết vấn đề này, các kỹ thuật như bảng trang phân cấp (Hierarchical paging), bảng trang băm (Hashed page tables), và bảng trang đảo ngược (Inverted page tables) được áp dụng để quản lý và giảm kích thước của bảng trang.

* Ưu điểm của Paging: Ưu điểm nổi bật nhất của Paging là khả năng khắc phục hoàn toàn phân mảnh ngoại vi (External Fragmentation). Vì các page có thể nằm rải rác trong bộ nhớ vật lý, hệ thống có thể tận dụng mọi frame trống mà không bị giới hạn bởi yêu cầu về sự liên tục. Paging cũng đơn giản hóa việc quản lý bộ nhớ trống, khi hệ điều hành chỉ cần theo dõi các frame trống và có thể cấp phát bất kỳ frame nào khi cần một page mới. Đồng thời, nó hỗ trợ hiệu quả Bộ nhớ Ảo, chỉ cần các page cần thiết nằm trong bộ nhớ chính, tối ưu hóa việc sử dụng RAM và cho phép chạy các chương trình lớn hơn bộ nhớ vật lý.
* Nhược điểm của Paging: Paging vẫn tồn tại Phân mảnh nội vi (Internal Fragmentation). Trang cuối cùng của một tiến trình có thể không sử dụng hết toàn bộ một frame, dẫn đến một lượng nhỏ bộ nhớ bị lãng phí trong frame đó. Mặc dù mức độ lãng phí này thường nhỏ (trung bình là một nửa kích thước page trên mỗi process), việc duy trì và quản lý Page Table cũng có thể gây ra chi phí (overhead) đáng kể về bộ nhớ, đặc biệt với các hệ thống 64-bit có không gian địa chỉ lớn.

4.4.3. Phân đoạn (Segmentation)

Khái niệm: Khác với Paging chia chương trình thành các khối cố định, Segmentation chia chương trình thành các đơn vị luận lý có ý nghĩa đối với lập trình viên, được gọi là segment (phân đoạn). Mỗi segment có thể đại diện cho một hàm, một mảng dữ liệu, một đoạn mã, hoặc một ngăn xếp, và chúng có kích thước khác nhau tùy theo nội dung logic của chúng.

Segment Table (Bảng phân đoạn): Tương tự như Page Table, mỗi tiến trình có một Segment Table riêng. Mỗi mục trong Segment Table chứa hai thông tin chính cho một segment: địa chỉ nền (base address) của segment trong bộ nhớ vật lý và giới hạn (limit) kích thước của segment đó. Thanh ghi Segment-table length register (STLR) chỉ ra số segment được sử dụng bởi một chương trình.

Cơ chế chuyển đổi địa chỉ: Một địa chỉ luận lý trong Segmentation bao gồm hai phần: số segment (s) và độ lệch (d) bên trong segment đó. Quá trình chuyển đổi diễn ra bằng cách sử dụng s để tra cứu trong Segment Table, tìm địa chỉ nền và giới hạn của segment đó. Sau đó, hệ thống kiểm tra xem độ lệch d có hợp lệ (tức là d < limit) không; nếu không, sẽ xảy ra lỗi truy cập bộ nhớ. Cuối cùng, địa chỉ vật lý được tính bằng cách cộng địa chỉ nền của segment với độ lệch d.

Kiến trúc phân đoạn: Kiến trúc phân đoạn hỗ trợ tái định vị động thông qua bảng phân đoạn. Các phân đoạn có thể chia sẻ giữa các chương trình bằng cách sử dụng chung chỉ số segment. Trong việc cấp phát, thường sử dụng các thuật toán như first fit/best fit, tuy nhiên, phương pháp này vẫn có hiện tượng phân mảnh ngoại vi. Về bảo vệ, mỗi entry trong Segment Table có thêm một bit "valid bit" để kiểm tra tính hợp lệ của truy cập, và hệ thống hỗ trợ phân quyền theo từng thao tác (read/write/execute).

Ưu điểm của Segmentation: Ưu điểm nổi bật là khả năng phản ánh cấu trúc chương trình một cách logic, giúp lập trình viên dễ dàng hơn trong việc tổ chức, quản lý và gỡ lỗi. Đồng thời, Segmentation cũng mang lại sự linh hoạt trong việc bảo vệ và chia sẻ, vì các segment là các đơn vị logic riêng biệt, việc áp dụng các cơ chế bảo vệ và chia sẻ trở nên tự nhiên hơn.

Nhược điểm của Segmentation: Nhược điểm chính là Phân mảnh ngoại vi, tương tự như Dynamic Partitioning. Do các segment có kích thước khác nhau và được nạp/giải phóng động, bộ nhớ có thể bị phân mảnh ngoại vi. Việc cấp phát bộ nhớ cho các segment cũng có thể phức tạp, đòi hỏi các thuật toán tìm kiếm khoảng trống phù hợp.

4.4.4. Kết hợp Phân trang và Phân đoạn (Segmentation with Paging)

Để kết hợp ưu điểm của cả hai kỹ thuật Paging và Segmentation, đồng thời khắc phục nhược điểm về phân mảnh ngoại vi của Segmentation, nhiều hệ thống hiện đại sử dụng cách tiếp cận kết hợp này.

Ý tưởng và cấu trúc: Mô hình này dựa trên ý tưởng "phân trang trong mỗi phân đoạn). Theo đó, bộ nhớ được coi là tập hợp các phân đoạn, và mỗi phân đoạn lại được chia thành nhiều trang có kích thước cố định. Mục tiêu chính của sự kết hợp này là giải quyết tình trạng phân mảnh ngoại vi, trong khi vẫn giữ được các lợi ích về cấu trúc logic của phân đoạn.

Cách hoạt động và chuyển đổi địa chỉ: Trong mô hình này, địa chỉ luận lý bao gồm ba phần: số phân đoạn (segment number), số trang (page number) trong phân đoạn đó, và độ lệch (offset) trong trang. Hệ thống có một Segment Table chính, nhưng mỗi mục trong Segment Table này không chứa địa chỉ nền vật lý trực tiếp của segment. Thay vào đó, nó chứa địa chỉ nền của Page Table riêng cho segment đó. Quá trình chuyển đổi địa chỉ diễn ra qua hai bước tra cứu chính: đầu tiên, hệ thống dùng số phân đoạn để tra cứu trong Segment Table, lấy ra địa chỉ của Page Table tương ứng với segment đó. Tiếp theo, nó dùng số trang để tra cứu trong Page Table đó, lấy ra địa chỉ của frame vật lý. Cuối cùng, độ lệch được sử dụng để định vị vị trí byte chính xác trong frame.

Ưu điểm: Ưu điểm nổi bật của sự kết hợp này là khả năng giải quyết triệt để phân mảnh ngoại vi, vì các phân đoạn cũng được phân trang, các page của một phân đoạn có thể nằm rải rác trong bộ nhớ vật lý. Nó cũng thành công trong việc kết hợp được tính logic của Segmentation (bảo vệ, chia sẻ theo các đơn vị có ý nghĩa) với hiệu quả quản lý không gian vật lý của Paging.

Nhược điểm: Mặc dù hiệu quả, cấu trúc quản lý trở nên phức tạp hơn do cần hai cấp bảng (Segment Table và nhiều Page Table). Điều này cũng có thể dẫn đến chi phí chuyển đổi địa chỉ cao hơn (do cần nhiều lần truy cập bảng) nếu không có TLB hiệu quả để tăng tốc.

### 4.5.Các Kỹ thuật Quản lý Bộ nhớ Ảo Nâng cao

Khi bộ nhớ ảo đã được triển khai thông qua các chiến lược cấp phát như Paging, hệ điều hành cần các kỹ thuật bổ sung để quản lý hiệu quả các trang bộ nhớ, đặc biệt là khi bộ nhớ vật lý bị đầy.

4.5.1. Khái niệm lỗi trang (Page Fault)

Lỗi trang (Page Fault) xảy ra khi một tiến trình cố gắng truy cập một page (trang bộ nhớ ảo) mà hiện tại không có mặt trong bộ nhớ vật lý (RAM). Khi đó, hệ điều hành sẽ tạm dừng tiến trình và xử lý lỗi trang bằng cách: kiểm tra tính hợp lệ của truy cập, tìm một frame trống trong RAM, đọc page cần thiết từ bộ nhớ phụ (đĩa) vào frame đó, cập nhật Page Table, và sau đó cho phép tiến trình tiếp tục thực thi.

4.5.2. Các thuật toán Thay thế Trang (Page Replacement Algorithms)

Khi bộ nhớ vật lý bị đầy và một page mới cần được nạp vào, hệ điều hành phải lựa chọn một page hiện có trong RAM để đẩy ra bộ nhớ phụ (swap out) nhằm giải phóng frame. Các thuật toán thay thế trang được thiết kế để đưa ra quyết định này nhằm tối ưu hóa hiệu suất (giảm thiểu số lỗi trang).

FIFO (First-In, First-Out): Thuật toán này đơn giản thay thế page nào đã ở trong bộ nhớ lâu nhất. Mặc dù dễ cài đặt, FIFO có thể gặp "hiệu ứng Belady", khi tăng số lượng frame lại làm tăng số lỗi trang trong một số trường hợp.

Optimal Page Replacement (OPT): Đây là thuật toán lý tưởng, thay thế page nào sẽ không được sử dụng trong thời gian dài nhất trong tương lai. OPT mang lại hiệu suất tốt nhất nhưng không thể thực hiện được trong thực tế vì nó yêu cầu biết trước tương lai của các tham chiếu trang.

LRU (Least Recently Used - Ít được sử dụng gần đây nhất): LRU thay thế page nào đã không được sử dụng trong thời gian lâu nhất. Đây là một xấp xỉ tốt cho OPT và thường cho hiệu suất tốt trong thực tế, nhưng việc cài đặt nó (ví dụ: bằng cách sử dụng bộ đếm hoặc ngăn xếp) có thể tốn kém.

Approximate LRU Algorithms (Các thuật toán xấp xỉ LRU): Do chi phí của LRU, các thuật toán xấp xỉ được phát triển như Additional-Reference-Bits Algorithm, Second-Chance Algorithm (còn gọi là Clock Algorithm), và Enhanced Second-Chance Algorithm. Chúng sử dụng các bit tham chiếu (reference bits) và cơ chế đơn giản hơn để mô phỏng hành vi của LRU.

Counting-Based Algorithms: Bao gồm LFU (Least Frequently Used), thay thế page nào được sử dụng ít nhất, và MFU (Most Frequently Used), thay thế page nào được sử dụng nhiều nhất (ít phổ biến hơn).

4.5.3. Phân bổ Frame (Frame Allocation)

Vấn đề phân bổ frame liên quan đến việc quyết định số lượng frame bộ nhớ vật lý sẽ được cấp phát cho mỗi tiến trình. Các phương pháp cơ bản bao gồm phân bổ đều (Equal allocation), nơi mỗi tiến trình nhận được số frame như nhau, và phân bổ tỷ lệ (Proportional allocation), nơi số frame được cấp phát tỷ lệ thuận với kích thước của tiến trình. Ngoài ra, cần xem xét sự khác biệt giữa thay thế trang toàn cục (Global Replacement), nơi một tiến trình có thể chọn bất kỳ page nào trong bộ nhớ để thay thế, và thay thế trang cục bộ (Local Replacement), nơi một tiến trình chỉ có thể thay thế page của chính nó.

4.5.4. Thrashing

Thrashing là một tình trạng nghiêm trọng của hệ thống, nơi hệ điều hành dành quá nhiều thời gian cho việc hoán đổi trang (page swapping) giữa RAM và đĩa, thay vì thực thi công việc hữu ích của các tiến trình.

Nguyên nhân chính của thrashing là việc cấp phát quá ít frame cho các tiến trình, khiến chúng thường xuyên gặp lỗi trang và phải liên tục trao đổi dữ liệu với đĩa. Hậu quả là hiệu suất hệ thống giảm mạnh, CPU hoạt động ít hiệu quả, và thời gian đáp ứng của các chương trình tăng lên đáng kể. Để giải quyết vấn đề này, Working Set Model được đề xuất. Working Set của một tiến trình là tập hợp các page mà tiến trình đã truy cập trong một khoảng thời gian gần đây (working-set window). Ý tưởng là hệ điều hành nên đảm bảo rằng working set của một tiến trình luôn nằm trong bộ nhớ, nếu không, tiến trình đó nên bị tạm dừng để tránh gây ra thrashing.

Kết luận

Quản lý bộ nhớ là một khía cạnh thiết yếu và đầy thách thức trong thiết kế hệ điều hành. Sự tiến hóa từ các chiến lược cấp phát liên tục đơn giản nhưng gặp phải vấn đề phân mảnh (như Fixed Partitioning, Dynamic Partitioning, và Swapping), đến sự ra đời của khái niệm bộ nhớ ảo và các kỹ thuật cấp phát không liên tục như Paging và Segmentation, đã đánh dấu một bước ngoặt quan trọng trong việc tối ưu hóa hiệu suất và khả năng mở rộng của hệ thống.

Paging, với khả năng loại bỏ triệt để phân mảnh ngoại vi và hỗ trợ mạnh mẽ bộ nhớ ảo, đã trở thành nền tảng cho hầu hết các hệ điều hành hiện đại. Sự phức tạp trong việc quản lý bảng trang lớn được giải quyết bằng các cấu trúc đa cấp và TLB, trong khi các thuật toán thay thế trang và cơ chế chống thrashing đảm bảo hiệu suất tối ưu ngay cả khi bộ nhớ vật lý bị hạn chế. Trong khi đó, Segmentation cung cấp một cách nhìn logic hơn về cấu trúc chương trình, tạo điều kiện thuận lợi cho việc bảo vệ và chia sẻ tài nguyên. Cuối cùng, sự kết hợp giữa Paging và Segmentation là một giải pháp mạnh mẽ, tận dụng ưu điểm của cả hai để mang lại sự linh hoạt và hiệu quả cao nhất trong quản lý bộ nhớ, dù phải đánh đổi bằng sự phức tạp trong triển khai.

Việc nắm vững các chiến lược cấp phát bộ nhớ này không chỉ giúp đánh giá được sự phức tạp và tinh vi trong thiết kế của một hệ điều hành mà còn là nền tảng để phát triển các ứng dụng hiệu quả, tận dụng tối đa tài nguyên bộ nhớ có hạn của hệ thống máy tính. Sự phát triển không ngừng của công nghệ bộ nhớ và các kỹ thuật quản lý sẽ tiếp tục là yếu tố then chốt để đáp ứng nhu cầu ngày càng tăng của các ứng dụng và hệ thống trong tương lai.

# **PHẦN 2: THỰC HÀNH VÀ ỨNG DỤNG**

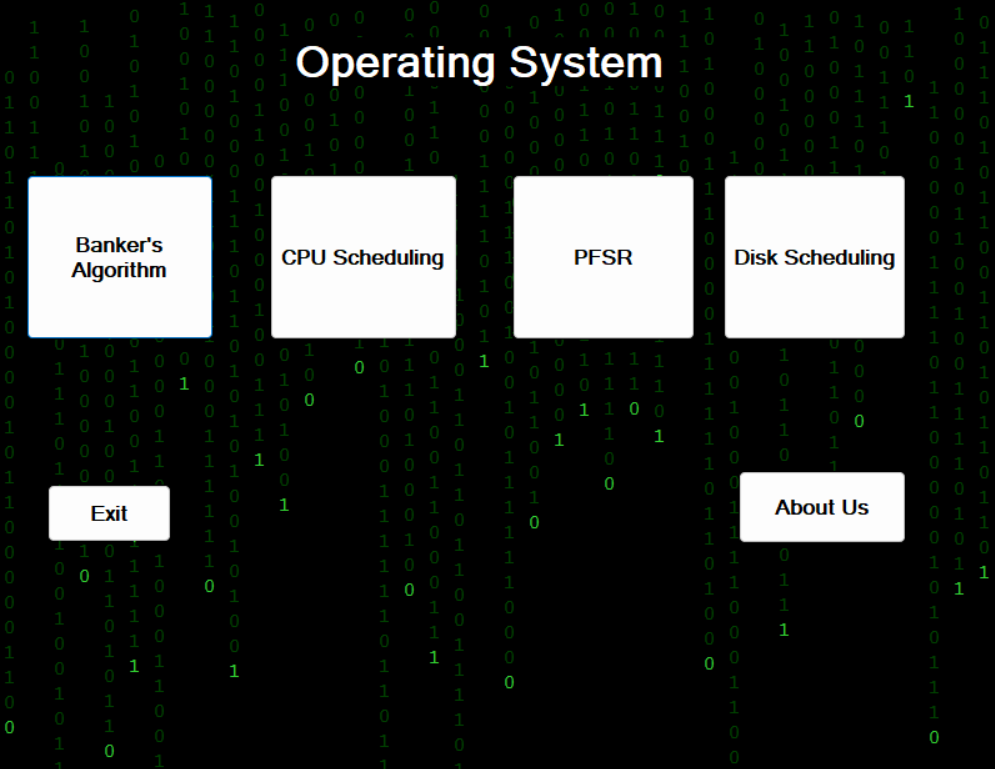
## **1. GIỚI THIỆU**

1.1 Mục đích và ứng dụng của các giải thuật trong hệ điều hành

Các giải thuật quản lý tài nguyên trong hệ điều hành đóng vai trò vô cùng quan trọng trong việc đảm bảo hệ thống hoạt động hiệu quả, công bằng và an toàn. Một hệ điều hành phải có khả năng phân phối tài nguyên như CPU, bộ nhớ, và thiết bị I/O cho các tiến trình đang chạy một cách hợp lý và không để xảy ra các tình trạng lỗi như deadlock hay tràn bộ nhớ. Các giải thuật quản lý tài nguyên giúp tối ưu hóa hiệu suất hệ thống và giảm thiểu xung đột giữa các tiến trình, đồng thời đảm bảo tính ổn định và bảo mật của hệ thống.

1.2. Giới thiệu các Form trong ứng dụng

Thiết kế giao diện người dùng với WinForm.



Các form và điều khiển cơ bản sẽ được thiết kế sao cho người dùng có thể tương tác với các thuật toán và xem kết quả mô phỏng.

Giao diện chính:

* Tiêu đề: "Operating System" (Hệ điều hành)
* Các nút điều khiển: Giao diện có các nút bấm để truy cập vào từng thuật toán cụ thể:
  + Banker's Algorithm
  + CPU Scheduling
  + PFSR
  + Disk Scheduling
  + Exit: Để thoát khỏi ứng dụng.
  + About Us: Thông tin giới thiệu về nhóm phát triển.

## **2. CHƯƠNG TRÌNH MINH HOẠ GIẢI THUẬT BANKER ĐỂ PHÁT TRIỂN VÀ GIẢI QUYẾT DEADLOCK**

### **2.1. Giới thiệu chung**

Thuật toán Banker (Banker's Algorithm) là một phương pháp quản lý tài nguyên trong hệ điều hành, được Edsger W. Dijkstra đề xuất vào năm 1965. Thuật toán này thuộc nhóm phương pháp tránh deadlock (deadlock avoidance), nhằm đảm bảo hệ thống không rơi vào trạng thái bế tắc khi nhiều tiến trình đồng thời yêu cầu tài nguyên có giới hạn. Deadlock xảy ra khi các tiến trình chờ lẫn nhau để truy cập tài nguyên, dẫn đến không tiến trình nào có thể tiếp tục thực thi. Thuật toán Banker thuộc nhóm phương pháp tránh deadlock (deadlock avoidance) mô phỏng cách một ngân hàng quản lý việc cấp phát vốn, chỉ cấp phát tài nguyên khi đảm bảo hệ thống duy trì trạng thái an toàn, tức là có thể hoàn thành tất cả các tiến trình mà không gây bế tắc.

Thuật toán Banker được áp dụng trong các hệ thống đa nhiệm, nơi các tiến trình cạnh tranh để sử dụng tài nguyên như CPU, bộ nhớ, hoặc thiết bị I/O. Nó yêu cầu hệ thống biết trước nhu cầu tối đa của các tiến trình và sử dụng thông tin này để kiểm tra trạng thái an toàn trước khi cấp phát tài nguyên.

### **2.2. Các khái niệm cơ bản**

Thuật toán Banker dựa trên các khái niệm sau:

Tài nguyên (Resources): Hệ thống bao gồm nhiều loại tài nguyên, ký hiệu ( R\_1, R\_2,... R\_m ), ví dụ: CPU, bộ nhớ, thiết bị I/O. Mỗi loại tài nguyên ( R\_i ) có ( W\_i ) thực thể (instances).

Tiến trình (Processes): Các chương trình đang thực thi, ký hiệu ( P\_1, P\_2,... P\_n ). Mỗi tiến trình sử dụng tài nguyên theo chu trình: yêu cầu (request), sử dụng (use), và hoàn trả (release).

Cấu trúc dữ liệu:

Available (Vector tài nguyên sẵn có): Vector độ dài ( m ), biểu diễn số thực thể còn lại của mỗi loại tài nguyên.

Allocation (Ma trận phân bổ): Ma trận ( n x m ), biểu diễn số thực thể của tài nguyên ( R\_j ) đã cấp phát cho tiến trình ( P\_i ).

Max (Ma trận nhu cầu tối đa): Ma trận ( n x m ), biểu diễn số thực thể tối đa của tài nguyên ( R\_j ) mà tiến trình ( P\_i ) có thể yêu cầu.

Need (Ma trận nhu cầu hiện tại): Ma trận ( n x m ), được tính bằng ( Need[i][j] = Max[i][j] - Allocation[i][j] ), biểu diễn số thực thể tài nguyên ( R\_j ) mà tiến trình ( P\_i ) còn cần để hoàn thành.

Request (Ma trận yêu cầu): Ma trận ( n x m ), biểu diễn số thực thể tài nguyên ( R\_j ) mà tiến trình ( P\_i ) yêu cầu tại một thời điểm.

### **2.3. Cách thức thực hiện**

Thuật toán Banker hoạt động dựa trên hai thành phần chính: giải thuật cấp phát tài nguyên và giải thuật kiểm tra trạng thái an toàn. Các giải thuật này đảm bảo rằng việc cấp phát tài nguyên không dẫn hệ thống đến trạng thái không an toàn, nơi không thể tìm được chuỗi thực thi để hoàn thành tất cả các tiến trình. Ngoài ra còn có giải thuật phát hiện Deadlock bằng Wait-For Graph khi mỗi tài nguyên chỉ có 1 thể hiện, và giải thuật Banker (để phát hiện) khi mỗi tài nguyên có nhiều thể hiện

**2.3.1. Giải thuật cấp phát tài nguyên**

Khi một tiến trình ( P\_i ) yêu cầu tài nguyên, được biểu diễn bởi vector ( Request\_i ), hệ thống thực hiện các bước sau:

1. Kiểm tra tính hợp lệ của yêu cầu:

So sánh ( Request\_i <= Need\_i ). Nếu yêu cầu vượt quá nhu cầu tối đa của tiến trình, báo lỗi và từ chối yêu cầu.

So sánh ( Request\_i <= Available ). Nếu không đủ tài nguyên sẵn có, tiến trình ( P\_i ) phải chờ.

1. Giả lập cấp phát tài nguyên:

Cập nhật trạng thái hệ thống như sau:

( Available = Available} - Request\_i )

( Allocation\_i = Allocation\_i + Request\_i )

( Need}\_i = Need\_i - Request\_i )

1. Kiểm tra trạng thái an toàn:

Chạy giải thuật kiểm tra trạng thái an toàn để xác định xem trạng thái mới có an toàn hay không.

Nếu an toàn, tài nguyên được cấp phát thực sự cho ( P\_i ).

Nếu không an toàn, hủy cấp phát, khôi phục trạng thái ban đầu, và tiến trình ( P\_i ) phải chờ.

**2.3.2. Giải thuật kiểm tra trạng thái an toàn**

Giải thuật này xác định xem hệ thống có tồn tại một chuỗi an toàn (safe sequence) của các tiến trình, trong đó mỗi tiến trình có thể nhận đủ tài nguyên để hoàn thành và giải phóng tài nguyên cho các tiến trình khác. Các bước thực hiện:

1. Khởi tạo:

Tạo vector ( Work = Available} ), biểu diễn tài nguyên hiện có.

Tạo vector ( Finish ) với ( n ) phần tử. Đặt Finish[i] = false với i = 1….n

1. Tìm tiến trình khả thi:

Tìm một tiến trình ( P\_i ) thỏa mãn:

* + - Finish[i] = false.
    - Need\_i <= Work

Nếu không tìm thấy ( P\_i ), chuyển sang bước 4.

1. Giả lập hoàn thành tiến trình:

Cập nhật:

* + - Work} = Work + Allocation\_i
    - Finish[i] = true

Quay lại bước 2.

1. Kết luận:

Nếu tất cả Finish}[i] = true, hệ thống ở trạng thái an toàn và chuỗi các tiến trình được chọn là chuỗi an toàn.

Nếu tồn tại Finish}[i] = false, hệ thống không an toàn, và việc cấp phát tài nguyên bị từ chối.

Độ phức tạp của giải thuật kiểm tra trạng thái an toàn là ( O(m x n^2) ), với ( m ) là số loại tài nguyên và ( n ) là số tiến trình.

Ưu điểm và hạn chế

Ưu điểm:

Ngăn chặn deadlock hiệu quả bằng cách kiểm tra trạng thái an toàn trước khi cấp phát tài nguyên.

Tối ưu hóa sử dụng tài nguyên so với các phương pháp ngăn chặn deadlock.

Phù hợp với các hệ thống đa nhiệm có nhu cầu tài nguyên động.

Hạn chế:

Yêu cầu thông tin trước về nhu cầu tối đa của các tiến trình, điều này không phải lúc nào cũng khả thi trong thực tế.

Độ phức tạp tính toán cao, đặc biệt khi số lượng tiến trình và tài nguyên lớn.

Không hiệu quả trong các hệ thống có nhu cầu tài nguyên thay đổi liên tục hoặc không thể dự đoán.

Ứng dụng thực tế

Thuật toán Banker được sử dụng trong:

Hệ điều hành: Quản lý tài nguyên như CPU, bộ nhớ, hoặc thiết bị I/O trong các hệ điều hành như UNIX, Windows.

Hệ thống nhúng: Phân bổ tài nguyên trong các thiết bị có tài nguyên hạn chế.

Cơ sở dữ liệu: Ngăn chặn bế tắc trong các giao dịch đồng thời.

Mô phỏng hệ thống: Phân tích hiệu suất và quản lý tài nguyên trong các công cụ mô phỏng.

**2.3.4 Phát hiện Deadlock**

Phát hiện deadlock là một phương pháp khác với tránh deadlock, được sử dụng khi hệ thống chấp nhận khả năng xảy ra deadlock và cần kiểm tra để xác định trạng thái này.

Giải thuật phát hiện Deadlock

Trường hợp mỗi loại tài nguyên có một thực thể:

* + Sử dụng Wait-for Graph, dẫn xuất từ Resource Allocation Graph (RAG) bằng cách loại bỏ nút tài nguyên và ghép cạnh. Một cạnh từ ( P\_i ) đến ( P\_j ) có nghĩa là ( P\_i ) chờ tài nguyên mà ( P\_j ) giữ.
  + Kiểm tra chu trình trong Wait-for Graph. Nếu có chu trình, hệ thống bị deadlock. Độ phức tạp: ( O(n^2) ).

Trường hợp mỗi loại tài nguyên có nhiều thực thể:

* + Sử dụng giải thuật tương tự kiểm tra trạng thái an toàn, nhưng với ma trận Request thay vì Need:
    1. Khởi tạo: Work = Available, Finish[i] = false nếu Allocation\_i != 0 , ngược lại true
    2. Tìm tiến trình: Tìm ( P\_i ) thỏa mãn Finish[i] = false và Request\_i <= Work
    3. Cập nhật: Work = Work + Allocation, Finish[i] = true. Quay lại bước 2.
    4. Kết luận: Nếu tồn tại Finish[i] = false hệ thống bị deadlock; các tiến trình với Finish[i] = false thuộc tập deadlock.
  + Độ phức tạp: ( O(m x n^2) ).

**Ưu điểm và hạn chế**

**Ưu điểm:**

Thuật toán Banker ngăn chặn deadlock hiệu quả bằng cách kiểm tra trạng thái an toàn.

Tối ưu hóa sử dụng tài nguyên so với phương pháp ngăn chặn deadlock.

Giải thuật phát hiện deadlock hỗ trợ xác định và xử lý bế tắc khi cần.

**Hạn chế:**

Yêu cầu thông tin trước về nhu cầu tối đa, khó áp dụng trong hệ thống động.

Độ phức tạp tính toán cao, không phù hợp cho hệ thống lớn.

Phát hiện deadlock chỉ xác định vấn đề mà không ngăn chặn.

**Ứng dụng thực tế**

Hệ điều hành: Quản lý CPU, bộ nhớ, thiết bị I/O.

Hệ thống nhúng: Phân bổ tài nguyên hạn chế.

Cơ sở dữ liệu: Ngăn chặn hoặc phát hiện bế tắc trong giao dịch.

### 2.4. Xây dựng chương trình mô phỏng thuật toán Banker

A screenshot of a computer

AI-generated content may be incorrect.

**Chức năng chính**

**Nhập liệu:**

Total processes: Số lượng tiến trình

Total resource types: Số loại tài nguyên

Allocation matrix: Ma trận hiển thị số tài nguyên đã cấp cho mỗi tiến trình (ví dụ: P0 = [0, 1, 0], P1 = [2, 0, 0], ...).

Max matrix: Ma trận thể hiện nhu cầu tối đa của mỗi tiến trình (ví dụ: P0 = [7, 5, 3], P1 = [3, 2, 2], ...).

Available matrix: Số tài nguyên hiện có (3, 3, 2 trong cả hai hình).

Need Matrix: Ma trận nhu cầu hiện tại, được tính bằng Max - Allocation (ví dụ: P0 = [7, 4, 3], P1 = [1, 2, 2], ...).

Request : Nhập yêu cầu tài nguyên (Request process = 1, Request resources = [1, 0, 2]).

**Chức năng nút:**

Continue: Bắt đầu cấu hình sau khi nhập số tiến trình và tài nguyên.

Detect Deadlock: Chuyển sang chế độ phát hiện deadlock.

Check: Thực hiện kiểm tra trạng thái an toàn hoặc xử lý yêu cầu.

Reset: Làm mới toàn bộ dữ liệu nhập vào.

Back: Quay lại màn hình trước.

Chế độ chọn:

Safe sequence: Chế độ kiểm tra trạng thái an toàn

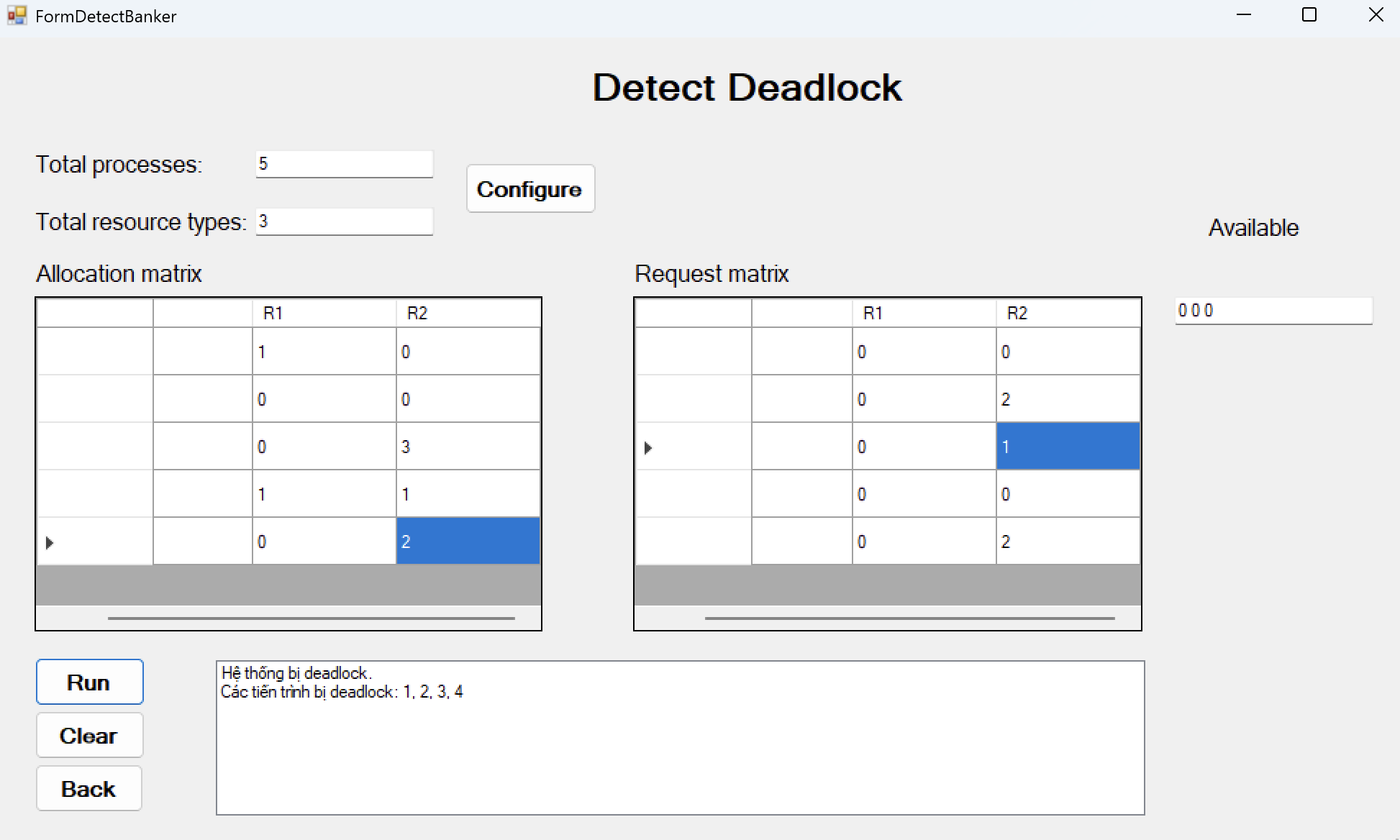
Request: Chế độ cấp phát tài nguyên. Nhập yêu cầu tài nguyên (Request process = 1, Request resources = [1, 0, 2]).

**2.4.1. Ý nghĩa kết quả**

Safe state: Hệ thống có thể cấp phát tài nguyên theo chuỗi an toàn, tránh deadlock.

Chuỗi an toàn: Thứ tự thực thi các tiến trình để hoàn thành mà không gây bế tắc (P1 -> P3 -> P4 -> P0 -> P2), với P1req trong trường hợp chế độ request.

Request xử lý: Xác nhận việc cấp phát tài nguyên cho P1 không làm hệ thống mất an toàn.



**1. Chức năng chính**

* Nhập liệu:
  + Total processes: Số lượng tiến trình.
  + Total resource types: Số loại tài nguyên.
  + Allocation matrix: Ma trận hiển thị tài nguyên đã cấp (ví dụ: P0 = [1, 0, 0], P1 = [0, 0, 3], ...).
  + Request matrix: Ma trận hiển thị yêu cầu tài nguyên (ví dụ: P0 = [0, 0, 0], P1 = [0, 0, 2], P2 = [0, 1, 0], ...).
  + Available: Số tài nguyên hiện có (0, 0, 0).
* Chức năng nút:
  + Configure: Cấu hình lưới dựa trên số tiến trình và tài nguyên.
  + Run: Thực hiện giải thuật phát hiện deadlock.
  + Clear: Xóa toàn bộ dữ liệu nhập vào.
  + Back: Quay lại màn hình chính.

**2. Ý nghĩa kết quả**

Phát hiện Deadlock: Hệ thống không thể tìm được chuỗi an toàn, dẫn đến các tiến trình P1, P2, P3, P4 bị mắc kẹt do không có đủ tài nguyên để hoàn thành (Available = [0, 0, 0] và Request vượt quá khả năng cấp phát).

Trường hợp không phát hiện Deadlock, chương trình sẽ xuất ra chuỗi tiến trình an toàn tương ứng



## **3. CHƯƠNG TRÌNH MINH HOẠ CÁC GIẢI THUẬT PHÂN PHỐI CPU CHO CÁC TIẾN TRÌNH**

### 3.1. Chương trình CPU

3.1.1. Giới thiệu

CPU - I/O Burst Cycle: bao gồm CPU burst thực hiện việc tính toán và I/O Burst thực hiện thao tác nhập xuất.

Thủ tục lập lịch:

+Scheduler (Bộ định thời): là thành phần của OS thực hiện các hoạt động điều phối. Bao gồm:  
+Long term scheduler (Job scheduler): chọn các công việc từ Job pool và tải chúng vào ready queue.  
+Short term scheduler (CPU scheduler): chọn 1 process để thực thi từ ready queue.  
+Medium term scheduler: quản lý lựa chọn tiến trình nào được giao tài nguyên trong waiting Queue, việc vào ra WQ của các tiến trình.  
+Dispatcher (Bộ điều phối): cung cấp CPU cho process được chọn bởi Short term scheduler.  
+Tiêu chí của lập lịch: Công bằng và hiệu quả, Tối đa hoá sử dụng CPU, Cực đại hoá thông lượng (số lượng tiến trình được hoàn thành trên 1 đơn vị thời gian), Giảm đi thời gian lưu trú (Turnaround), đợi (waiting: ở trong ready queue) và hồi đáp (response: từ lúc vào đến phản hồi đầu tiên).

Khi nào cần lập lịch: Tạo mới tiến trình (cha hay con được thực thi), Kết thúc tiến trình, Tiến trình bị blocked để đợi 1 thao tác I/O hay sự kiện, Ngắt (Interrupt) xảy ra hoàn thành I/O hoặc hết time slice.

Các loại điều phối:

Điều phối độc quyền/Không thương lượng (Nonpreemptive): tiến trình đang chạy sẽ giữ CPU cho đến khi nó kết thúc hoặc chuyển đổi sang trạng thái waiting.

Điều phối không độc quyền/Có thương lượng (Preemptive): tiến trình đang chạy sẽ bị ngắt bởi OS khi có tiến trình có độ ưu tiên cao hơn đến hoặc hết time slice.

Các thuật toán điều phối

* First-Come, First-Served (FCFS): độc quyền, chọn tiến trình đứng đầu tiên trong ready queue. Đơn giản nhưng hiệu năng thấp và không xem xét đến các tiến trình ưu tiên.
* Round-Robin (RR): Không độc quyền, được dùng trong HĐH chia sẻ thời gian. Chọn tiến trình đầu tiên trong ready queue để thực thi trong 1 khoảng thời gian nhỏ q (time slice). Sau khoảng thời gian đó nếu tiến trình chưa thực hiện xong thì sẽ được chuyển cuối ready queue. Công bằng, giảm thời gian đợi cho các tiến trình ngắn nhưng không xem đến sự ưu tiên và khó trong việc chọn q cho hợp lý.
* Shortest-Job-First (SJF): độc quyền.
* Shortest-Remaining-Time-Next (SRTN): không độc quyền. Chọn tiến trình có CPU-Burst thấp nhất. Khi có 1 tiến trình có CPU burst thấp hơn vào ready queue thì SJF sẽ tiếp tục chạy tiến trình hiện tại trong khi SRTN sẽ đổi cho tiến trình mới. Đơn giản, giảm Turnaround (TT) và Waiting Time (WT) cho tiến trình ngắn, tăng thông lượng nhưng khó trong việc ước lượng CPU-burst cho tất cả process. Vấn đề đói CPU khi process có CPU\_burst cao hơn luôn không được thực thi bởi các tiến trình nhỏ hơn.
* Priority: có 2 phiên bản độc quyền và không độc quyền, cũng được dùng trong HĐH chia sẻ thời gian. Mỗi process được gắn với 1 độ ưu tiên. Chọn process có độ ưu tiên cao nhất trong ready queue. Tùy vào độc quyền hay không thì tiến trình có độ ưu tiên cao hơn đến sau có được thực thi liền hay không. Vấn đề đói CPU với các tiến trình có độ ưu tiên thấp -> giải pháp là thêm cơ chế tăng độ ưu tiên với các tiến trình ở lâu trong ready queue.
* Multilevel Priority Queue: Điều phối với hàng đợi đa cấp, chia ready queue thành nhiều hàng đợi với độ ưu tiên khác nhau. Mỗi queue sẽ sử dụng 1 thuật toán phù hợp. Mỗi tiến trình được gắn với 1 queue. Chọn process ở hàng đợi có độ ưu tiên cao hơn trước và việc chọn process nào trong queue phụ thuộc vào thuật toán của queue.
* Khác: Guaranteed scheduling, Lottery scheduling, Fair-Share scheduling.

3.1.2. Chương trinh CPU

A screenshot of a computer

AI-generated content may be incorrect.

### 3.2. FCFS

A screenshot of a computer

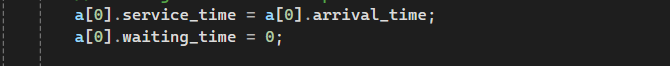
AI-generated content may be incorrect.

### 

FCFS (First Come First Serve) và arrangePriority, bao gồm việc tính toán thời gian chờ (waiting\_time) và thời gian quay vòng (turnaround\_time) cho các tiến trình, sắp xếp các tiến trình theo mức độ ưu tiên và hiển thị kết quả.

1. Hàm FCFS (First Come First Serve)

Hàm này thực hiện thuật toán FCFS, là thuật toán lập lịch đơn giản nhất. Trong thuật toán FCFS, tiến trình đến trước sẽ được xử lý trước. Các bước chính trong hàm này như sau:

a. Sắp xếp các tiến trình theo thời gian đến (arrangeArrival)

arrangeArrival(a, n);

Trước khi bắt đầu tính toán thời gian chờ và thời gian quay vòng, ta cần sắp xếp các tiến trình theo thời gian đến (arrival\_time).

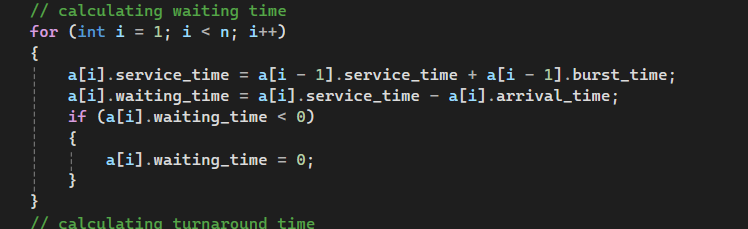
arrangeArrival là hàm sắp xếp các tiến trình sao cho tiến trình có arrival\_time nhỏ nhất sẽ được xử lý đầu tiên. (Hàm này chưa được định nghĩa trong mã bạn gửi, nhưng có thể sắp xếp bằng cách sử dụng thuật toán sắp xếp đơn giản như Bubble Sort, Selection Sort, hoặc Quick Sort).

b. Tính toán thời gian chờ (waiting\_time) cho từng tiến trình

// Thời gian chờ của tiến trình đầu tiên là 0

* Tiến trình đầu tiên bắt đầu ngay khi nó đến (service\_time = arrival\_time), và thời gian chờ là 0 vì không có tiến trình nào chạy trước nó.

Tính toán thời gian chờ cho các tiến trình còn lại



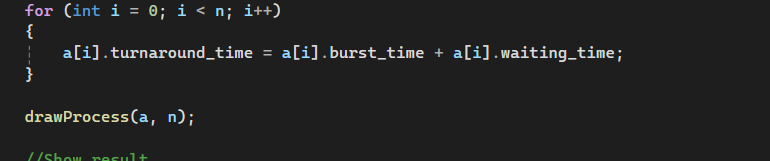
Đối với mỗi tiến trình, ta tính thời gian dịch vụ (service\_time) là thời gian kết thúc của tiến trình trước cộng với thời gian thực thi (burst\_time) của tiến trình đó.

Thời gian chờ (waiting\_time) của tiến trình là sự khác biệt giữa thời gian dịch vụ và thời gian đến (arrival\_time).

Nếu thời gian chờ có giá trị âm (nghĩa là tiến trình đến trước khi có tiến trình nào chạy), ta gán thời gian chờ là 0.

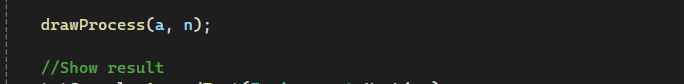
c. Tính toán thời gian quay vòng (turnaround\_time)

// Tính toán thời gian quay vòng (turnaround time)



* Thời gian quay vòng (turnaround\_time) của mỗi tiến trình được tính bằng tổng thời gian burst\_time (thời gian xử lý của tiến trình) và waiting\_time (thời gian chờ của tiến trình).

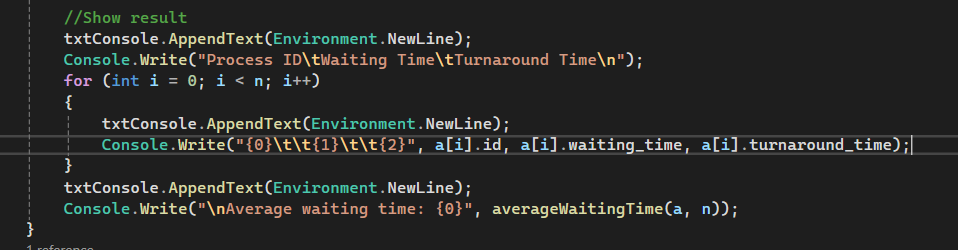
d. Vẽ biểu đồ Gantt (bỏ trong trường hợp này)



Hàm drawProcess được gọi để vẽ biểu đồ Gantt cho các tiến trình, nhưng vì bạn yêu cầu loại bỏ phần liên quan đến màu sắc và bảng vẽ, bạn có thể bỏ qua hàm này trong trường hợp không cần vẽ Gantt.

e. Hiển thị kết quả và tính toán thời gian trung bình

Hiển thị kết quả

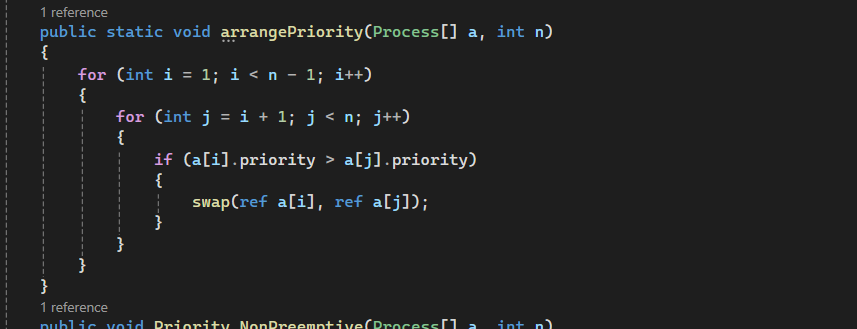


In ra bảng thông tin cho từng tiến trình, bao gồm ID tiến trình, thời gian chờ và thời gian quay vòng.

Tính toán và hiển thị thời gian chờ trung bình bằng cách sử dụng hàm averageWaitingTime (hàm này tính tổng thời gian chờ và chia cho số tiến trình).

2. Hàm arrangePriority

Hàm này thực hiện việc sắp xếp các tiến trình theo mức độ ưu tiên. Trong thuật toán Priority Non-Preemptive (hoặc Preemptive), các tiến trình có mức độ ưu tiên cao (giá trị priority nhỏ nhất) sẽ được xử lý trước.



* arrangePriority sắp xếp các tiến trình sao cho tiến trình có priority nhỏ nhất (ưu tiên cao nhất) sẽ đứng trước trong mảng `a[].
* swap là hàm dùng để hoán đổi vị trí giữa hai tiến trình trong mảng.

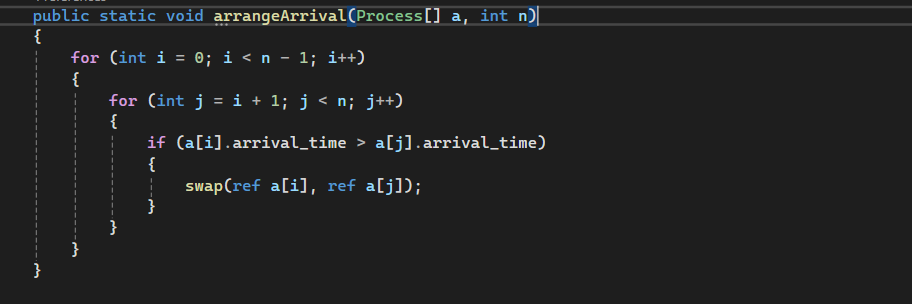
### 3.3. SJF độc quyền

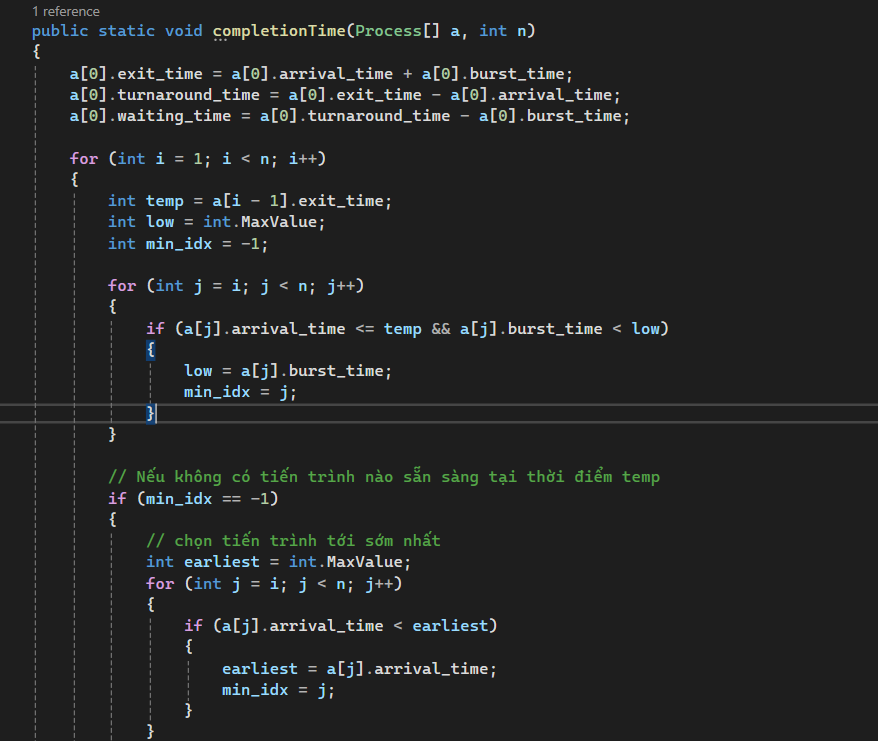
A screenshot of a computer

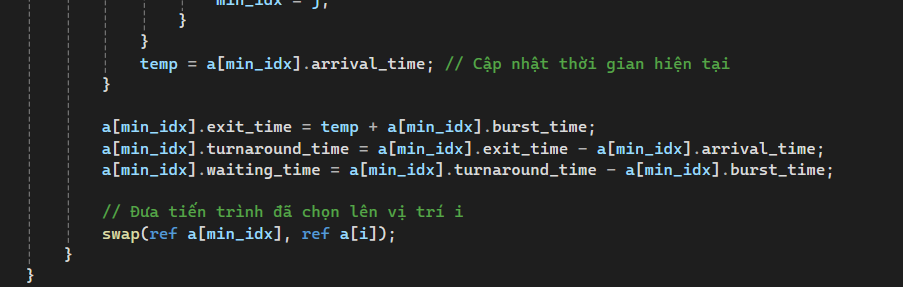
AI-generated content may be incorrect.

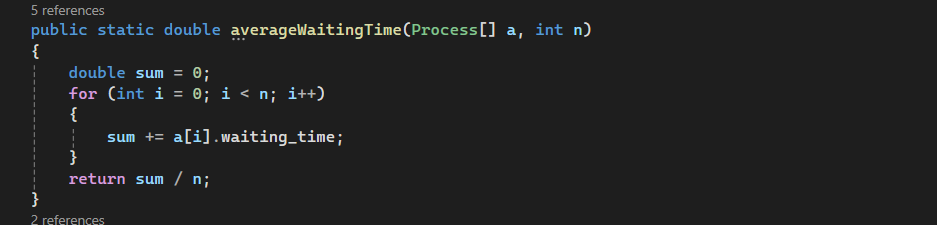
Các Hàm Liên Quan (Không Thay Đổi):

1. Hàm arrangeArrival – Sắp xếp tiến trình theo thời gian đến.



1. Hàm completionTime – Tính toán thời gian hoàn thành, thời gian quay vòng và thời gian chờ cho các tiến trình.  
   



1. Hàm averageWaitingTime – Tính toán thời gian chờ trung bình.  
   

### 3.4. SJF không độc quyền

A screenshot of a computer

AI-generated content may be incorrect.

1. Hàm averageWaitingTime – Tính toán thời gian chờ trung bình cho các tiến trình:

public static double averageWaitingTime(Process[] a, int n)

{

double sum = 0;

for (int i = 0; i < n; i++)

{

sum += a[i].waiting\_time;

}

return sum / n;

}

### 3.5. Priority độc quyền

A screenshot of a computer

AI-generated content may be incorrect.

1. Sắp xếp các tiến trình theo ưu tiên (arrangePriority)

Hàm arrangePriority được sử dụng để sắp xếp các tiến trình theo mức độ ưu tiên của chúng, giúp đảm bảo rằng các tiến trình có mức độ ưu tiên cao sẽ được xử lý trước. Cách thức sắp xếp có thể là sắp xếp giảm dần hoặc tăng dần, tùy thuộc vào yêu cầu của thuật toán.

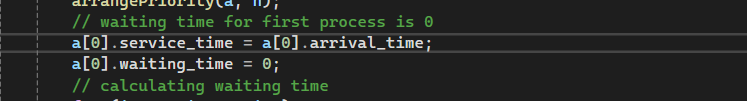
Giải thích:

Mỗi tiến trình có một thuộc tính priority. Trong thuật toán này, tiến trình có ưu tiên cao nhất (priority nhỏ nhất) sẽ được xử lý trước.

Sắp xếp giúp đảm bảo rằng các tiến trình có mức độ ưu tiên cao sẽ được xử lý ngay khi nó đến, mà không bị trì hoãn bởi tiến trình có ưu tiên thấp hơn.

2. Thời gian dịch vụ (service\_time) và thời gian chờ (waiting\_time)

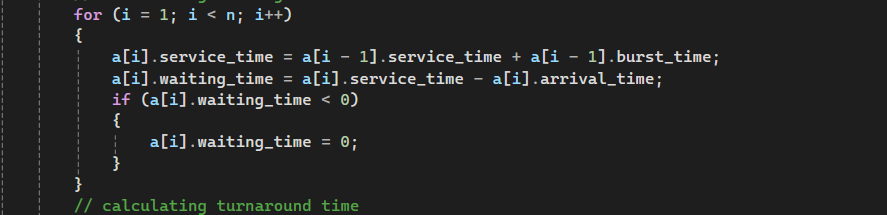
a. Tính toán thời gian chờ cho tiến trình đầu tiên



Thời gian dịch vụ của tiến trình đầu tiên (tiến trình có thời gian đến nhỏ nhất) sẽ bằng thời gian đến của nó (arrival\_time).

Thời gian chờ của tiến trình đầu tiên là 0, vì nó bắt đầu ngay khi đến.

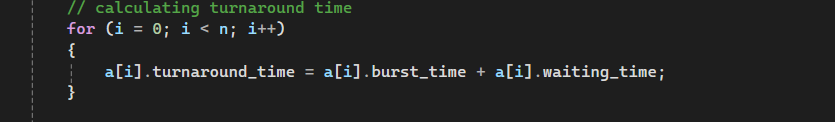
b. Tính toán thời gian chờ cho các tiến trình còn lại



Thời gian dịch vụ (service\_time) của mỗi tiến trình sau được tính là thời gian dịch vụ của tiến trình trước đó cộng với thời gian burst (thời gian thực thi) của tiến trình trước đó.

Thời gian chờ (waiting\_time) của mỗi tiến trình là sự chênh lệch giữa thời gian dịch vụ và thời gian đến của tiến trình đó. Nếu tiến trình đến sau thời gian dịch vụ của tiến trình trước, thời gian chờ sẽ là 0 (nếu có giá trị âm).

3. Tính toán thời gian quay vòng (turnaround\_time)

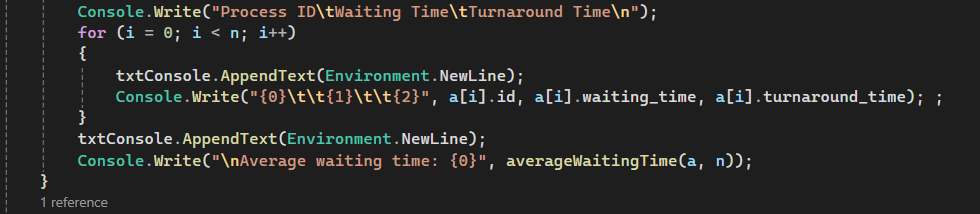


Thời gian quay vòng của một tiến trình là tổng của thời gian burst (thời gian thực thi) và thời gian chờ. Nó đại diện cho tổng thời gian từ khi tiến trình đến cho đến khi nó hoàn thành.

turnaround\_time = burst\_time + waiting\_time.

4. Tính toán và hiển thị kết quả

float totalWaitingTime = 0, totalTurnaroundTime = 0;



Tính tổng thời gian chờ (totalWaitingTime) và tổng thời gian quay vòng (totalTurnaroundTime).

Hiển thị thông tin về từng tiến trình (ID, thời gian chờ, thời gian quay vòng).

Cuối cùng, tính toán và hiển thị thời gian chờ trung bình và thời gian quay vòng trung bình cho tất cả các tiến trình.

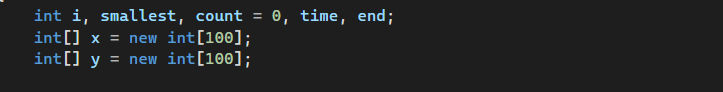
### 3.6. Priority không độc quyền

A screenshot of a computer

AI-generated content may be incorrect.

Priority Preemptive Scheduling, trong đó các tiến trình được xử lý theo mức độ ưu tiên, và tiến trình có ưu tiên cao hơn có thể "ngắt" tiến trình có ưu tiên thấp hơn. Thuật toán này thực hiện bằng cách chọn tiến trình có ưu tiên cao nhất (giá trị priority nhỏ nhất) và cho phép nó chạy đến khi hoàn thành hoặc đến khi có một tiến trình có ưu tiên cao hơn.

1. Khởi tạo các biến cần thiết:



x[]: Mảng này lưu trữ thời gian còn lại (remaining\_time) của mỗi tiến trình.

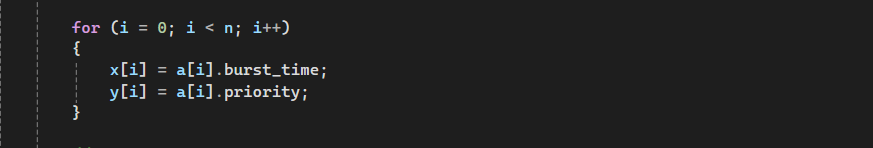
y[]: Mảng này lưu trữ mức độ ưu tiên của các tiến trình. Các tiến trình có mức độ ưu tiên nhỏ nhất sẽ được xử lý trước.

count: Đếm số lượng tiến trình đã hoàn thành.

time: Thời gian hiện tại trong quá trình thực thi thuật toán.

end: Lưu trữ thời gian kết thúc của tiến trình.

2. Khởi tạo thời gian còn lại và mức độ ưu tiên:



Mỗi tiến trình có một thời gian burst (tức là thời gian cần để tiến trình hoàn thành) và một mức độ ưu tiên. Thuật toán này sử dụng mảng x[] để theo dõi thời gian còn lại của mỗi tiến trình và mảng y[] để lưu mức độ ưu tiên.

3. Vòng lặp chính của thuật toán:

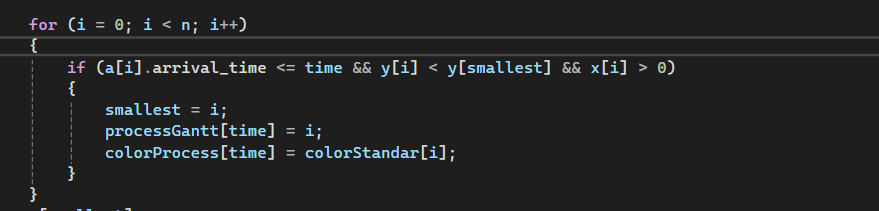
for (time = 0; count != n; time++)

{

smallest = 9; // Khởi tạo với một chỉ số lớn để tìm ra tiến trình có ưu tiên cao nhất

Thuật toán chạy cho đến khi tất cả các tiến trình hoàn thành. time đại diện cho thời gian hiện tại trong quá trình xử lý.

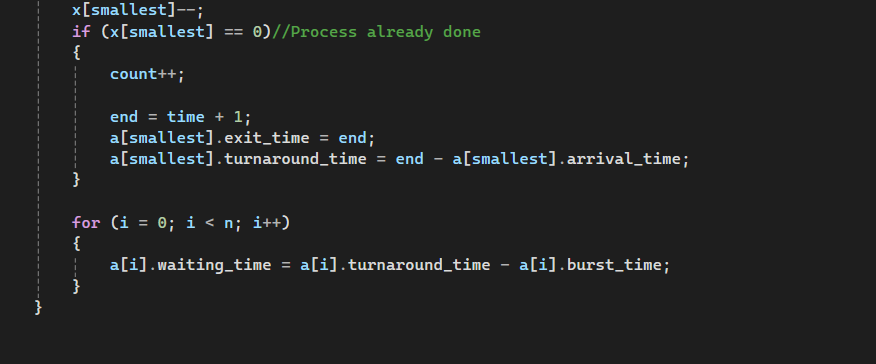
a. Tìm tiến trình có ưu tiên cao nhất:



Thuật toán duyệt qua tất cả các tiến trình. Tiến trình có thời gian đến (arrival\_time) nhỏ hơn hoặc bằng thời gian hiện tại (time) và mức độ ưu tiên nhỏ nhất sẽ được chọn để chạy.

Chỉ chọn tiến trình có thời gian còn lại (x[i] > 0) vì chỉ có tiến trình chưa hoàn thành mới được chọn.

b. Giảm thời gian còn lại của tiến trình đang xử lý:



Giảm thời gian còn lại (x[smallest]--) của tiến trình đã được chọn.

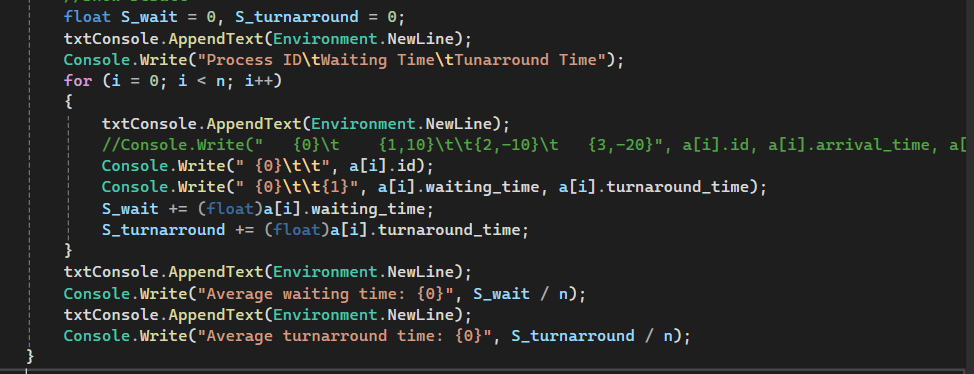
Nếu thời gian còn lại của tiến trình bằng 0, tiến trình đó hoàn thành và count được tăng lên. Tiến trình được tính toán thời gian quay vòng (turnaround\_time) và thời gian kết thúc (exit\_time).

Tính toán thời gian chờ (waiting\_time) cho các tiến trình:



Thời gian chờ (waiting\_time) của mỗi tiến trình được tính là thời gian quay vòng trừ đi thời gian burst.

Tính toán và hiển thị kết quả:



Sau khi tất cả các tiến trình hoàn thành, thuật toán tính tổng thời gian chờ và tổng thời gian quay vòng cho tất cả các tiến trình.

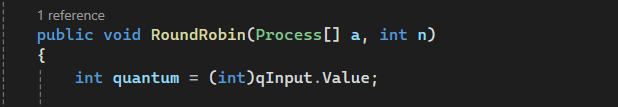
Sau đó, tính toán và hiển thị thời gian chờ trung bình và thời gian quay vòng trung bình.

### 3.7. Round-Robin

A screenshot of a computer

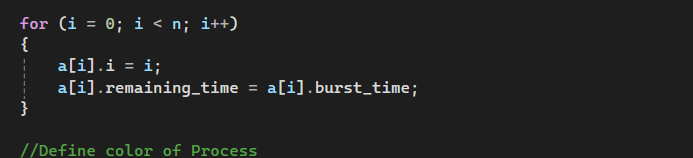
AI-generated content may be incorrect.

1. Hàm RoundRobin – Thuật toán Round Robin (Sử dụng Queue)

 int i, time;

double avg = 0, tt = 0;

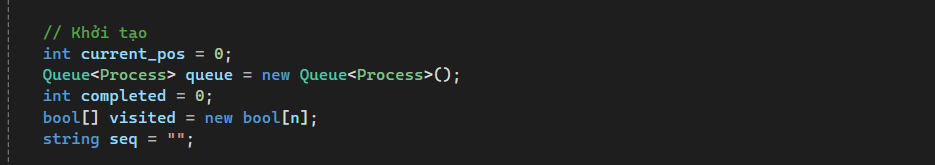
Khởi tạo các tiến trình



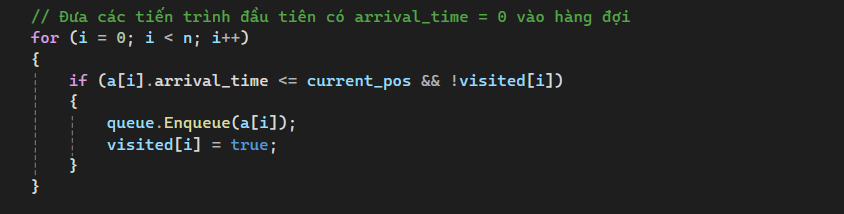
Sắp xếp theo thời gian đến của tiến trình

arrangeArrival(a, n);

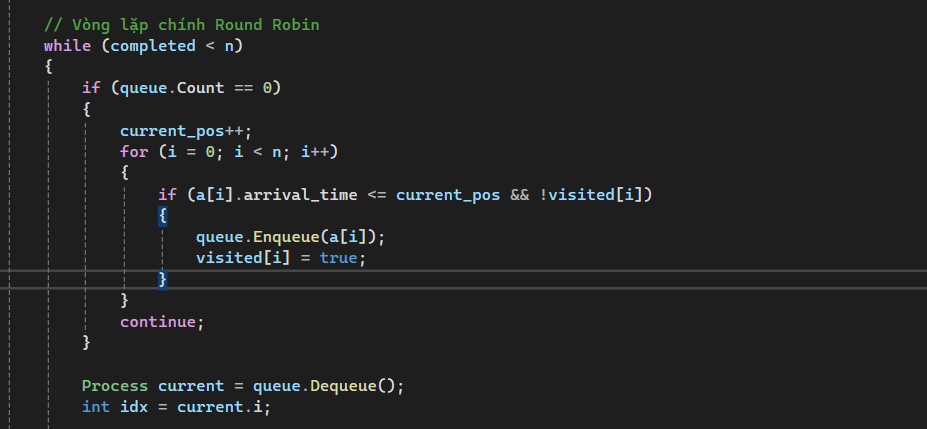
Khởi tạo hàng đợi và biến kiểm tra tiến trình



Đưa các tiến trình đầu tiên có arrival\_time <= current\_pos vào hàng đợi



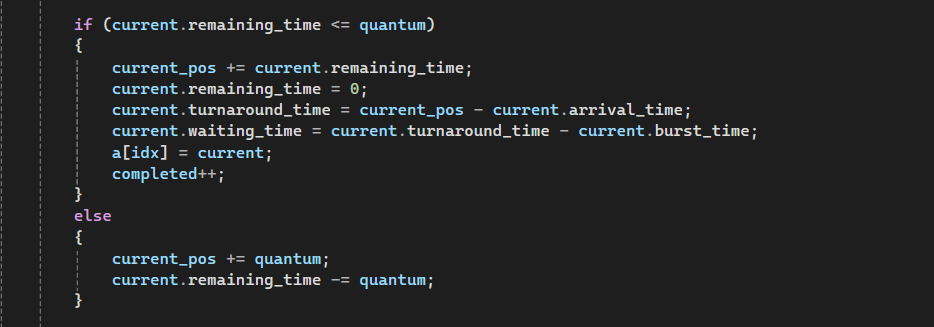
Vòng lặp chính của thuật toán Round Robin



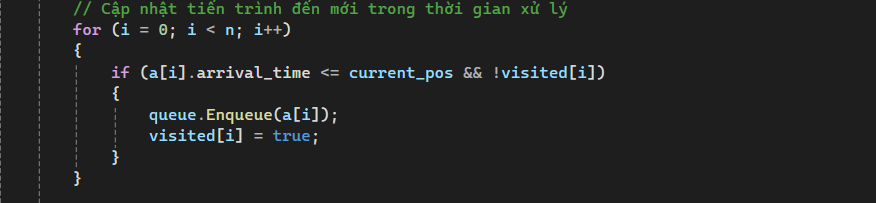
Cập nhật Gantt Chart Sequence

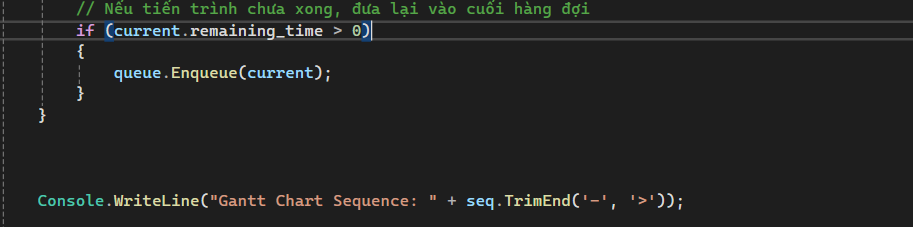


Tính toán thời gian quay vòng và thời gian chờ

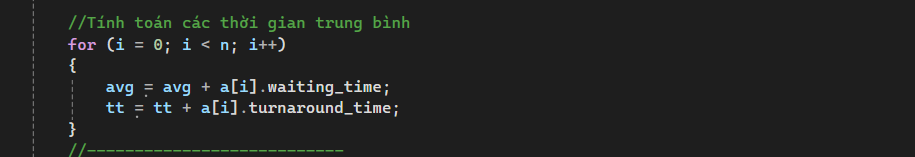


Cập nhật tiến trình mới trong thời gian xử lý

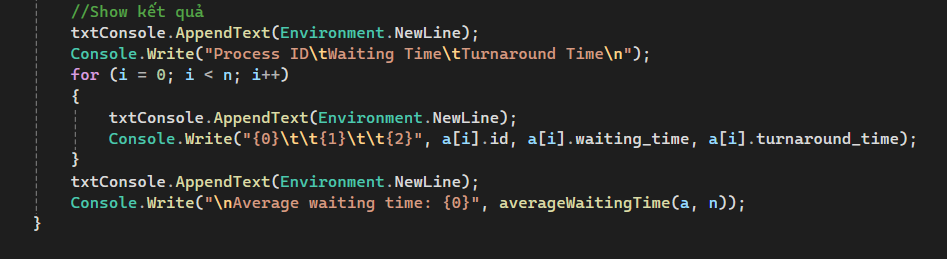


Nếu tiến trình chưa xong, đưa lại vào cuối hàng đợ 

Tính toán các thời gian trung bình



Hiển thị kết quả trong Console



## 4. CHƯƠNG TRÌNH MINH HOẠ GIẢI THUẬT THAY THẾ TRONG QUY TRÌNH PFSR

PFSR, hay "Page-Fault Service Routine" (Thủ tục phục vụ lỗi trang), là một thuật toán cốt lõi trong hệ điều hành, đặc biệt quan trọng trong cơ chế quản lý bộ nhớ ảo. Chức năng chính của PFSR là xử lý các tình huống khi một chương trình cố gắng truy cập vào một trang dữ liệu hoặc mã lệnh mà trang đó hiện không nằm trong bộ nhớ vật lý (RAM) mà thay vào đó đang được lưu trữ trên bộ nhớ phụ (thường là đĩa cứng).

### 4.1.Khi nào PFSR được kích hoạt?

Lỗi trang (page fault) xảy ra khi Bộ quản lý bộ nhớ (MMU) phát hiện ra rằng bit "valid" của một mục bảng trang là 0, cho thấy trang đó không hợp lệ hoặc không có trong bộ nhớ chính. Khi đó, phần cứng sẽ tạo ra một "page-fault trap" và kích hoạt PFSR của hệ điều hành.

### 4.2.Quy trình xử lý của PFSR

PFSR hoạt động theo một chuỗi các bước để đưa trang bị thiếu vào bộ nhớ chính và cho phép tiến trình tiếp tục hoạt động. Ban đầu, trong cơ chế demand paging (nạp trang theo yêu cầu), PFSR có các bước cơ bản sau:

Chuyển tiến trình sang trạng thái chờ (blocked): Tiến trình gây ra lỗi trang sẽ bị tạm dừng để hệ điều hành có thể xử lý việc nạp trang. Trong thời gian này, CPU có thể được cấp cho một tiến trình khác để thực thi, tối ưu hóa việc sử dụng tài nguyên.

Phát yêu cầu đọc đĩa: PFSR xác định vị trí của trang cần thiết trên đĩa và gửi yêu cầu đọc để nạp trang đó vào một khung trống (frame) trong bộ nhớ chính.

Hoàn tất I/O và ngắt: Khi thao tác đọc đĩa hoàn tất, một ngắt (interrupt) sẽ được gửi đến hệ điều hành để báo hiệu rằng trang đã được nạp.

Cập nhật bảng trang và tiếp tục tiến trình: Hệ điều hành cập nhật bảng trang của tiến trình để phản ánh vị trí mới của trang trong RAM (đặt bit "valid" thành 1) và chuyển tiến trình trở lại trạng thái sẵn sàng (ready) để nó có thể tiếp tục thực thi lệnh đã bị gián đoạn.

PFSR với cơ chế thay thế trang:

Một trường hợp phức tạp hơn xảy ra khi không còn khung bộ nhớ vật lý (frame) trống để nạp trang mới. Khi đó, PFSR phải bao gồm thêm bước thay thế trang:

Xác định vị trí trang cần thiết trên đĩa: Tương tự như bước đầu tiên.

Tìm một frame trống:

Nếu có frame trống, PFSR sẽ sử dụng ngay.

Nếu không có frame trống, hệ điều hành phải chọn một trang "nạn nhân" (victim page) để loại bỏ khỏi bộ nhớ chính, giải phóng một frame. Việc lựa chọn trang nạn nhân này dựa trên các thuật toán thay thế trang:

OPT (Optimal - Tối ưu): Thay thế trang sẽ được tham chiếu trong tương lai xa nhất. Đây là thuật toán lý tưởng nhưng không thể hiện thực được trong thực tế vì không thể biết trước tương lai.

LRU (Least Recently Used - Ít được sử dụng gần đây nhất): Thay thế trang đã không được tham chiếu trong khoảng thời gian lâu nhất.

FIFO (First-In, First-Out - Vào trước ra trước): Thay thế trang đã được nạp vào bộ nhớ lâu nhất. Thuật toán này có thể gặp hiện tượng "Belady’s anomaly", khi số lỗi trang tăng lên mặc dù số lượng frame được cấp cho tiến trình nhiều hơn.

Clock (Đồng hồ): Các frame được cấp cho tiến trình được xem như một bộ đệm xoay vòng. Mỗi frame có một "use bit" được đặt bằng 1 khi trang được nạp vào hoặc được tham chiếu. Khi cần thay thế, hệ thống tìm trang đầu tiên có "use bit" bằng 0 để thay thế, đồng thời đặt lại các "use bit" gặp trên đường tìm kiếm về 0. Thuật toán Clock thường có hiệu suất gần với LRU và được sử dụng trong các hệ thống thực tế (ví dụ: Windows 2000 sử dụng FIFO, có thể kết hợp với các biến thể như Clock).

Ghi trang nạn nhân lên đĩa (nếu cần) và cập nhật bảng trang/bảng frame: Nếu trang nạn nhân đã bị sửa đổi (được chỉ ra bởi "modified bit"), nó cần được ghi lại vào đĩa trước khi frame đó được tái sử dụng. Sau đó, bảng trang và bảng frame được cập nhật.

Đọc trang cần thiết vào frame trống: Trang mới được nạp vào frame đã được giải phóng.

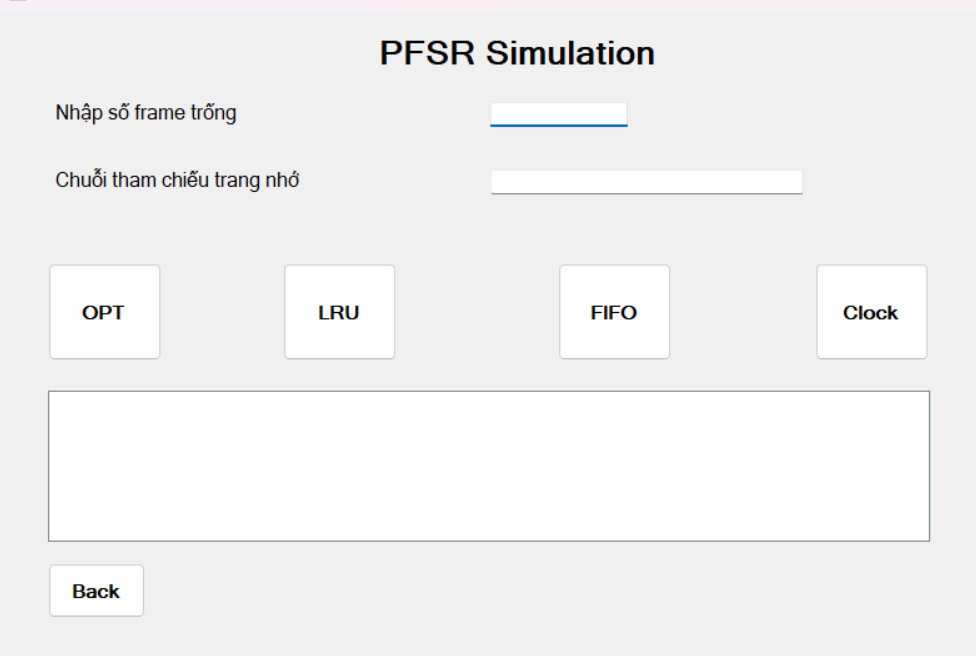
Cập nhật bảng trang và bảng frame: Đảm bảo thông tin về trang mới và frame tương ứng là chính xác.

Tầm quan trọng của PFSR:

PFSR là nền tảng cho việc hiện thực bộ nhớ ảo, cho phép các tiến trình sử dụng một không gian địa chỉ luận lý lớn hơn nhiều so với bộ nhớ vật lý thực tế có sẵn. Nó giúp tối ưu hóa việc sử dụng RAM bằng cách chỉ giữ các trang cần thiết trong bộ nhớ và quản lý việc hoán đổi dữ liệu giữa RAM và đĩa. Các giải pháp như Working Set Model và Page-Fault Frequency (PFF) cũng được sử dụng để điều khiển tỷ lệ lỗi trang và hạn chế hiện tượng "thrashing" (tình trạng các trang bị hoán đổi liên tục vào/ra bộ nhớ, gây giảm hiệu suất hệ thống).

### 4.3. Thực hành và giao diện

4.3.1. Giao diện chính



Giao diện ứng dụng có thiết kế đơn giản, tập trung vào chức năng mô phỏng các thuật toán thay thế trang. Nền của ứng dụng màu xám nhạt, với các thành phần điều khiển được sắp xếp rõ ràng. Tiêu đề "PFSR Simulation" nằm ở giữa phía trên cùng, cho thấy đây là một công cụ mô phỏng.

Các thành phần đầu vào:

"Nhập số frame trống": Đây là một nhãn văn bản, bên cạnh là một ô nhập liệu (textbox) trống. Ô này dùng để người dùng nhập vào số lượng frame bộ nhớ vật lý (RAM) có sẵn cho quá trình mô phỏng. Đây là một thông số quan trọng ảnh hưởng đến hoạt động của các thuật toán thay thế trang.

"Chuỗi tham chiếu trang nhớ": Tương tự, đây là một nhãn văn bản, theo sau là một ô nhập liệu trống. Ô này dùng để người dùng nhập vào chuỗi các tham chiếu trang nhớ. Chuỗi này đại diện cho thứ tự các trang mà CPU yêu cầu truy cập theo thời gian, là đầu vào chính để các thuật toán lập lịch xử lý.

Các nút chọn thuật toán (Lập lịch Thay thế Trang): Phía dưới các ô nhập liệu là một hàng gồm bốn nút bấm, mỗi nút đại diện cho một thuật toán thay thế trang khác nhau mà ứng dụng có thể mô phỏng:

OPT: Nút này có lẽ để chạy mô phỏng bằng thuật toán Optimal Page Replacement (Tối ưu).

LRU: Nút này để chạy mô phỏng bằng thuật toán Least Recently Used (Ít được sử dụng gần đây nhất).

FIFO: Nút này để chạy mô phỏng bằng thuật toán First-In, First-Out (Vào trước, ra trước).

Clock: Nút này để chạy mô phỏng bằng thuật toán Clock (một dạng xấp xỉ của LRU, còn gọi là Second-Chance).

Các nút này có kích thước tương đối lớn, hình vuông với góc bo tròn, màu trắng và chữ đen, giúp dễ dàng nhận diện và thao tác.

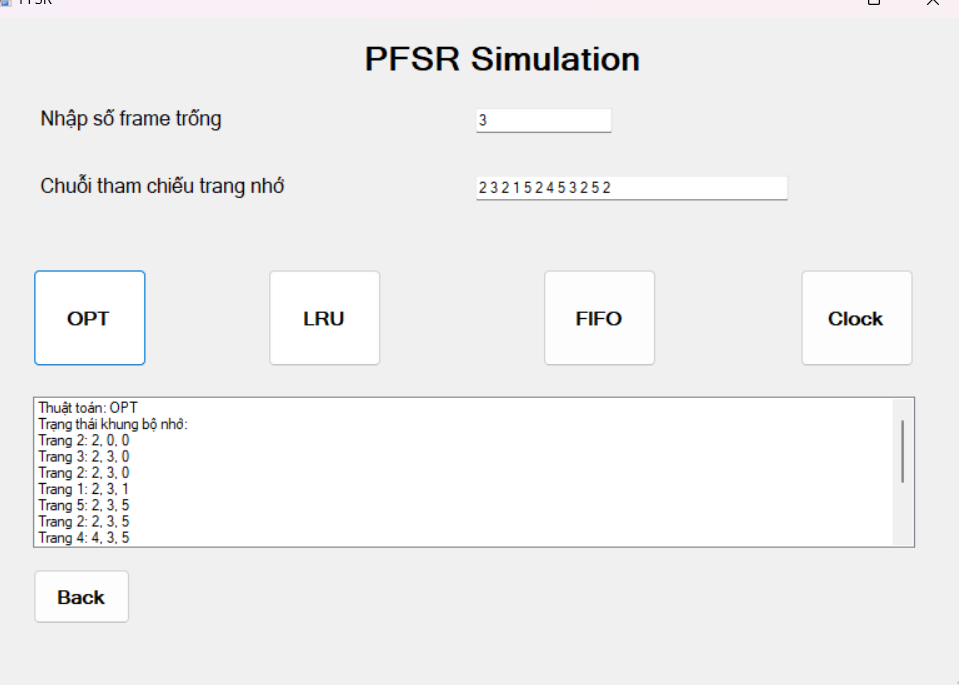
Khu vực hiển thị kết quả/log: Phía dưới các nút thuật toán là một khung văn bản lớn, trống. Khu vực này có thể được sử dụng để hiển thị kết quả của quá trình mô phỏng, chẳng hạn như số lỗi trang (page faults) xảy ra, trạng thái của các frame bộ nhớ theo từng bước, hoặc các thông báo khác liên quan đến quá trình chạy thuật toán.

Nút điều hướng: Ở góc dưới cùng bên trái, có một nút nhỏ hơn mang nhãn "Back". Nút này có thể dùng để quay lại màn hình trước đó hoặc đóng ứng dụng.

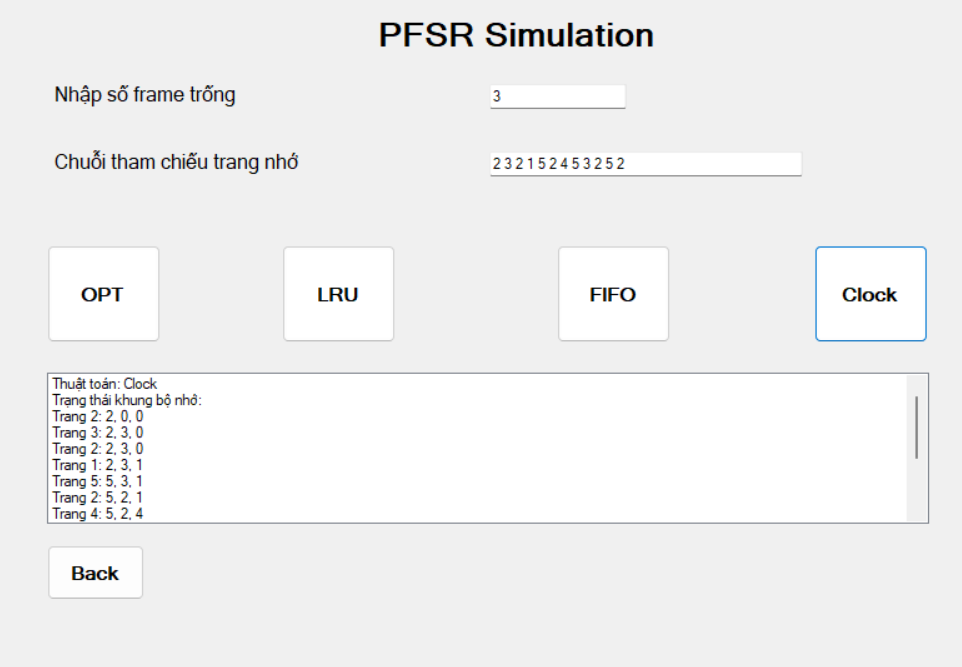
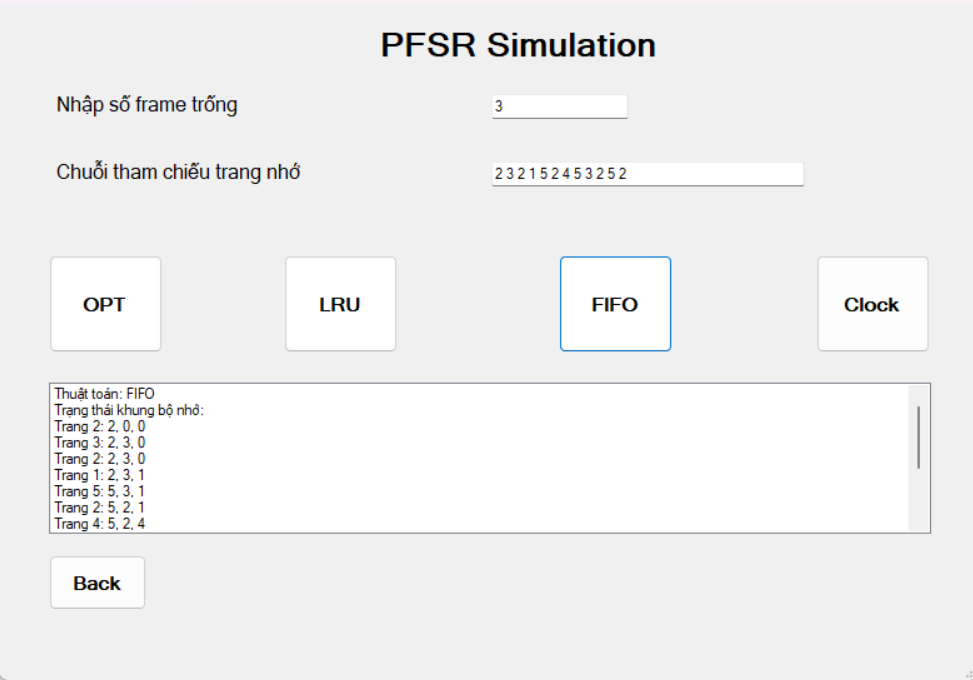
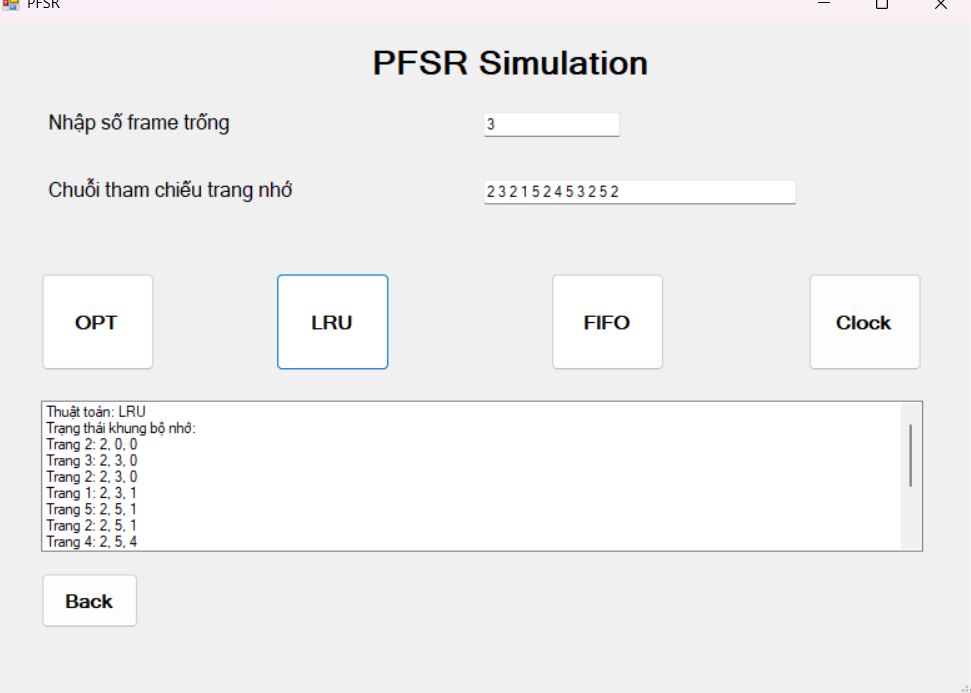
Mục đích sử dụng: Giao diện này cho thấy ứng dụng được thiết kế để phục vụ mục đích giáo dục hoặc nghiên cứu, giúp người dùng dễ dàng nhập các tham số và quan sát cách các thuật toán thay thế trang khác nhau hoạt động và so sánh hiệu quả của chúng dựa trên số lỗi trang.

2.1.1 Thuật toán OPT, LRU, FIFO, Clock:

Sau khi nhập đầy đủ fame trống và chuỗi tham chiếu. Ta click vào box có ký tự “OPT” để tiến hành chạy thuật toán OPT. Kết quả trả lại sẽ như sau:

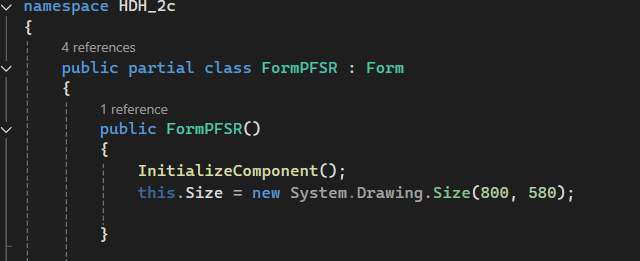


và tương tự như vậy cho các thuật toán còn lại



2.3 Cách thức hoạt động

Khai báo và và khởi tạo form



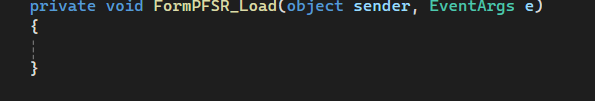
Chức năng:

Tạo form FormPFSR.

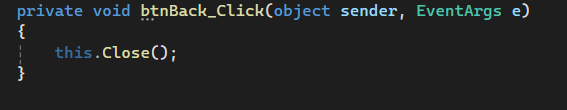
InitializeComponent() để khởi tạo giao diện từ file designer.

Đặt kích thước cửa sổ form là 800x580 px.

1. Form load

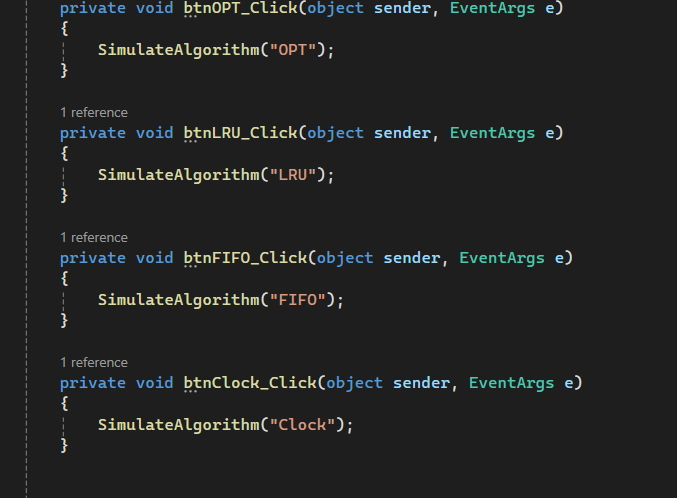
Sự kiện khi form vừa mở lên.  
Hiện tại bạn chưa thêm code vào sự kiện này.

Form Back



đóng form, quay về màn hình chính

3.Các nút chọn thuật toán



Khi nhấn vào box “OPT” , “LRU” , “FIFO”, “Clock”, gọi hàm mô phỏng tương ứng và truyền tham số

iện

IV. Thuật toán thay thế trang ;

| 1 Thuật toán OPT (Optimal Page Replacement)

private int SimulateOPT(int[] pages, int capacity, List<int> frames, ListBox listBox)

Chức năng:

• Tính toán số lần xảy ra page fault theo thuật toán OPT.

• Logic:

• Nếu còn frame trống – nạp trang vào.

• Nếu hết frame:

• Xét các frame hiện có xem trang nào xuất hiện trễ nhất hoặc không xuất hiện nữa trong tương lai.

• Thay thế trang đó bằng trang mới.

• Mỗi lần thay trang thì cập nhật lại listBox kết quả.

2 Thuật toán LRU (Least Recently Used)

csharp

private int SimulateLRU(int[] pages, int capacity, List<int> frames, ListBox listBox)

Chức năng:

• Thay thế trang ít được sử dụng gần đây nhất.

• Sử dụng Dictionary pageUsage để lưu lần cuối mỗi trang được sử dụng.

• Mỗi lần có page fault:

• Nếu có frame trống — nạp.

• Nếu hết frame → tìm trang có chỉ số pageUsage nhỏ nhất để thay thế.

3 Thuật toán FIFO (First In First Out)

private int SimulateFIFO(int[] pages, int capacity, List<int> frames, ListBox listBox)

Chức năng:

• Thay thế trang vào đầu tiên.

• Sử dụng Queue để lưu thứ tự trang được nạp vào.

• Khi đầy frame, trang đầu tiên trong queue sẽ bị loại bỏ.

4 Thuật toán Clock

private int SimulateClock(int[] pages, int capacity, List<int> frames, ListBox listBox)

Chức năng:

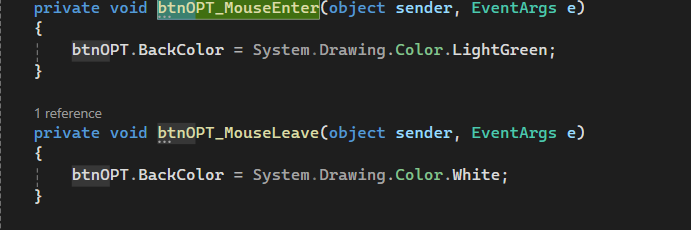
• Giả lập thuật toán đồng hồ:

• Mỗi trang có thêm bit tham chiếu (reference bit).

• Con trỏ pointer quét tuần tự vòng tròn.

• Khi gặp trang có bit 0 thì thay thế, nếu bit 1 thì đặt lại 0 rồi tiếp tục.

V. Hiệu ứng giao d (Hover Button màu sắc)



Chức năng:

* Khi rê chuột vào các nút, đổi màu nền từng nút:  
  + OPT: LightGreen
  + LRU: LightBlue
  + FIFO: LightYellow
  + Clock: Red
* Tạo hiệu ứng trực quan dễ nhận diện các thuật toán khi thao tác.

## **5. CHƯƠNG TRÌNH MINH HOẠ GIẢI THUẬT ĐINH THỜI TRUY CẬP ĐĨA ĐỂ QUẢN LÝ HỆ THỐNG TẬP TIN**

### 5.1 Định thời truy cập đĩa:

Để truy xuất các khỏi trên đĩa, trước tiên phải di chuyển đầu đọc đến track thích hợp, thao tác này gọi là seek và thời gian để hoàn tất thao tác này gọi là seek time. Một khi đã đến đúng track, còn phải chờ cho đến khi khỏi cần thiết đến dưới đầu đọc, thời gian chờ này gọi là latency time. Cuối cũng là chuyển dữ liệu từ đĩa vào bộ nhớ chỉnh, thời gian này gọi là transfer time. Tổng thời gian cho dịch vụ đĩa chính là tổng của ba khoảng thời gian trên (seek time latency time transfer time). Trong dò seek time và latency time là mất nhiều thời gian nhất, do đó để giảm thiểu thời gian truy xuất, hệ điều hành cần đưa ra các thuật toán lập lịch đời đầu đọc sao cho tối ưu.

Các giải thuật định thời truy cập đĩa

First Come, First Served (FCFS)

Shortest-Seek-Time First (SSTF, SSF)

SCAN

C-SCAN (Circular SCAN)

5.1.1 First Come, First Served (FCFS)

Thuật toán First Come First Serve (FCFS) là một trong những thuật toán đơn giản và cơ bản nhất được sử dụng để lập lịch truy cập đĩa trong hệ điều hành. Nguyên lý hoạt động của FCFS rất dễ hiểu: các yêu cầu truy cập đĩa (IO requests) được xử lý theo thứ tự đến, tức là yêu cầu nào đến trước thì được phục vụ trước, không có sự phân biệt ưu tiên giữa các yêu cầu.

Cách thức hoạt động:

1. Xếp hàng theo thứ tự đến: Khi một yêu cầu truy cập đĩa (ví dụ, đọc hoặc ghi dữ liệu) được gửi đến, nó sẽ được đưa vào một hàng đợi.

2. Xử lý lần lượt: Hệ thống sẽ phục vụ các yêu cầu theo thứ tự trong hàng đợi, tức là yêu cầu đầu tiên đến sẽ được xử lý trước, yêu cầu thứ hai đến sẽ được xử lý tiếp theo, và cứ như vậy cho đến khi tất cả các yêu cầu được hoàn thành.

Ưu điểm:

Đơn giản và dễ triển khai: FCFS rất dễ cài đặt trong hệ điều hành và không yêu cầu các thuật toán phức tạp.

Không cần phân tích ưu tiên: Không cần tính toán hay đưa ra quyết định về ưu tiên giữa các yêu cầu, giúp giảm độ phức tạp trong việc quản lý hàng đợi.

Nhược điểm:

Không tối ưu thời gian tìm kiếm đĩa (seek time): Trong nhiều trường hợp, các yêu cầu đĩa đến từ những vị trí khác nhau trên đĩa. Nếu yêu cầu đến quá xa so với vị trí hiện tại của đầu đọc, việc di chuyển đầu đọc đĩa sẽ tốn thời gian rất lớn, dẫn đến hiệu suất kém. Điều này có thể gây ra convoy effect, tức là một số yêu cầu phải chờ đợi rất lâu nếu các yêu cầu trước đó ở vị trí xa.

Hiệu suất kém trong môi trường nhiều yêu cầu: Khi có quá nhiều yêu cầu đĩa đến gần nhau, FCFS có thể gây ra sự không công bằng, làm cho các yêu cầu sau phải chờ đợi lâu, thậm chí có thể làm tắc nghẽn hệ thống.

5.1.2 Shortest-Seek-Time First (SSTF, SSF)

Thuật toán lập lịch đĩa Shortest Seek Time First (SSTF) là một phương pháp quản lý truy cập đĩa trong hệ điều hành, nhằm tối ưu hóa thời gian truy cập bằng cách chọn yêu cầu truy cập gần nhất với vị trí hiện tại của đầu đọc. SSTF hoạt động dựa trên nguyên tắc rằng việc phục vụ các yêu cầu gần nhất sẽ giảm thiểu thời gian tìm kiếm.

Nguyên lý hoạt động:

Khi một yêu cầu truy cập đĩa được gửi, SSTF sẽ xem xét tất cả các yêu cầu đang chờ và xác định yêu cầu nào có vị trí gần nhất với vị trí hiện tại của đầu đọc. Thuật toán sẽ phục vụ yêu cầu đó trước, sau đó lặp lại quy trình cho các yêu cầu còn lại.

Ưu điểm

Tối ưu hóa thời gian tìm kiếm: SSTF thường giảm thiểu thời gian truy cập so với các thuật toán đơn giản như FCFS, vì nó ưu tiên các yêu cầu gần hơn.

Cải thiện hiệu suất: Thuật toán này thường cho phép hệ thống hoàn thành các yêu cầu nhanh hơn, dẫn đến hiệu suất tổng thể tốt hơn.

Nhược điểm

Hiện tượng "starvation": Một số yêu cầu có thể bị bỏ qua trong thời gian dài nếu chúng nằm ở vị trí xa, dẫn đến tình trạng không được phục vụ kịp thời.

Khó dự đoán: Thời gian phục vụ có thể biến đổi lớn do phụ thuộc vào vị trí của các yêu cầu.

5.1.3 Scan:

Thuật toán lập lịch đĩa Elevator, còn được gọi là thuật toán SCAN, là một phương pháp quản lý truy cập đĩa trong hệ điều hành, nhằm tối ưu hóa hiệu suất truy cập đĩa bằng cách di chuyển đầu đọc theo một hướng nhất định và phục vụ các yêu cầu trong khi di chuyển.

Nguyên lý hoạt động:

Thuật toán SCAN hoạt động theo cách mà đầu đọc di chuyển qua tất cả các yêu cầu trên đĩa trong một hướng (từ ngoài vào trong hoặc ngược lại). Khi đầu đọc gặp yêu cầu, nó phục vụ yêu cầu đó. Khi không còn yêu cầu nào trong hướng di chuyển hiện tại, đầu đọc sẽ thay đổi hướng và tiếp tục phục vụ các yêu cầu trong hướng ngược lại.

Ưu điểm

Giảm thiểu thời gian chờ: SCAN giúp giảm thiểu thời gian tìm kiếm, vì nó phục vụ nhiều yêu cầu liên tiếp mà không phải quay lại một cách không cần thiết.

Đều đặn và công bằng: Tất cả các yêu cầu đều có cơ hội được phục vụ, giúp giảm thiểu hiện tượng "starvation".

Nhược điểm

Thời gian chờ không đồng đều: Mặc dù SCAN cải thiện độ công bằng, nhưng thời gian chờ vẫn có thể không đồng đều, đặc biệt với các yêu cầu ở hai đầu của đĩa.

Tăng thời gian di chuyển: Đầu đọc có thể phải di chuyển quãng đường dài hơn so với các thuật toán khác như SSTF.

5.1.4 C-SCAN (Circular SCAN):

Thuật toán lập lịch đĩa Circular SCAN (C-SCAN) là một chiến lược quản lý truy cập đĩa trong hệ điều hành, được thiết kế nhằm cung cấp sự công bằng hơn giữa các yêu cầu truy cập, đồng thời đảm bảo hiệu suất ổn định trong quá trình vận hành. C-SCAN hoạt động bằng cách di chuyển đầu đọc đĩa theo một hướng cố định và phục vụ các yêu cầu theo thứ tự cylinder tăng dần (hoặc giảm dần), sau đó quay trở lại điểm đầu mà không phục vụ yêu cầu nào trong quá trình quay lại.

Nguyên lý hoạt động:

Khi đầu đọc bắt đầu di chuyển, thuật toán sẽ chỉ phục vụ các yêu cầu theo một hướng duy nhất (ví dụ: từ trong ra ngoài). Khi đến giới hạn ngoài cùng của đĩa (cylinder lớn nhất), đầu đọc được đưa về đầu kia của đĩa (cylinder nhỏ nhất) mà không xử lý bất kỳ yêu cầu nào trong hành trình quay lại. Quá trình này lặp lại theo chu kỳ, do đó thuật toán có tên là Circular SCAN.

Ưu điểm

Đảm bảo thời gian phục vụ trung bình đồng đều hơn cho các yêu cầu ở đầu và cuối đĩa.

Tránh tình trạng “starvation” (đói tài nguyên) cho các yêu cầu ở các vùng xa vị trí hiện tại của đầu đọc.

Giữ nguyên tốc độ xử lý theo chiều ưu tiên đã chọn, giúp dễ dàng dự đoán hiệu suất.

Nhược điểm:

Có thể làm tăng quãng đường di chuyển của đầu đọc đĩa do phải quay lại đầu đĩa sau mỗi lượt xử lý.

Một số yêu cầu có thể phải chờ lâu hơn nếu nằm ở phía ngược chiều di chuyển của đầu đọc.

### 5.2 Xây dựng chương trình mô phỏng thuật toán định thời truy cập đĩa:

5.2.1. Chương trình Disk Scheduling Algorithms:

A screenshot of a computer

AI-generated content may be incorrect.

Nhận dữ liệu đầu vào từ người dùng:

Cho phép người dùng nhập chuỗi các yêu cầu truy cập đĩa (ví dụ: 98 183 37 122 14 124 65 67).

Cho phép nhập vị trí ban đầu của đầu đọc đĩa (ví dụ: 53).

Lựa chọn thuật toán mô phỏng: Giao diện hiển thị 4 nút để người dùng chọn một trong các thuật toán mô phỏng:

FCFS (First Come First Serve)

SSTF (Shortest Seek Time First)

SCAN (Elevator)

C-SCAN (Circular SCAN)

Chức năng Back để người dùng đóng form khi không muốn thực hiện mô phỏng.

5.1.1 First Come, First Served (FCFS)

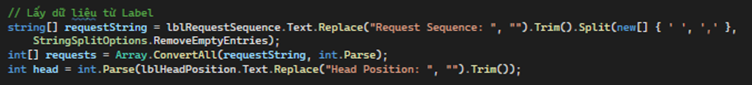
Giao diện (ví dụ, kết quả):

A screenshot of a computer

AI-generated content may be incorrect.

Xây dựng chức năng:

Khi người dùng nhấn nút "Simulate", sự kiện btnSimulate\_Click sẽ được kích hoạt và tiến hành các bước xử lý sau để mô phỏng thuật toán First Come First Serve (FCFS):



Xóa kết quả cũ trong danh sách hiển thị và đặt lại tổng số bước seek về 0 trước khi bắt đầu mô phỏng mới.



Thực hiện mô phỏng thuật toán First Come, First Served (FCFS)

A screen shot of a computer program

AI-generated content may be incorrect.

5.1.2 SSTF (Shortest Seek Time First):

Giao diện (ví dụ, kết quả):

A screenshot of a computer

AI-generated content may be incorrect.

Xây dựng chức năng:

A screen shot of a computer program

AI-generated content may be incorrect.

Mô phỏng SSTF

Hàm khởi tạo để thực hiện mô phỏng giải thuật SSTF với phân tử được duyệt qua tất cả yêu cầu còn lại và tìm phần tử có khoảng cách ngắn nhất đến đầu đọc hiện tại (currentHead). Vị trí yêu cầu gần nhất được lưu ở nearestIndex.

Di chuyển đầu đọc đến vị trí gần nhất nextPosition. Tính khoảng cách seek và cộng vào tổng totalSeeks. Hiển thị từng bước di chuyển trong danh sách listBoxResult.

Cập nhật kết quả cuối cùng tổng seek totalSeeks.

5.1.3. SCAN (Elevator):

Giao diện (ví dụ, kết quả):

A screenshot of a computer

AI-generated content may be incorrect.

Xây dựng chức năng:

A computer screen shot of a program

AI-generated content may be incorrect.

Tìm chỉ số trong mảng đã sắp xếp, nơi đầu đọc hiện tại nên bắt đầu.

Nếu không tìm thấy chính xác, dùng toán tử ~startIndex để xác định vị trí gần nhất lớn hơn.

Thực hiện tất cả các yêu cầu nằm ở phía lớn hơn hoặc bằng vị trí đầu đọc ban đầu.

Cập nhật số bước di chuyển seek và vị trí hiện tại currentHead.

maxCylinder nếu đầu đọc chưa đến biên phải, di chuyển thêm để chạm biên (giải sử biên 199). Điều này đúng với cơ chế SCAN: luôn chạm biên trước khi quay đầu.

Sau khi chạm biên, đầu đọc quay lại và phục vụ các yêu cầu nhỏ hơn head.

Duyệt theo chiều giảm từ chỉ số startIndex - 1 trở về đầu mảng.

lblTotalSeeks hiển thị tổng seek totalSeeks.

5.1.4. C-SCAN (Circular SCAN):

Giao diện (ví dụ, kết quả):

A screenshot of a computer

AI-generated content may be incorrect.

Xây dựng chức năng:

A black background with white text

AI-generated content may be incorrect.

Biến totalSeeks lưu tổng số bước di chuyển của đầu đọc.

Sao chép mảng yêu cầu để tránh ảnh hưởng dữ liệu gốc.

Sắp xếp mảng tăng dần để xử lý đúng hướng hoạt động của thuật toán.



Xác định vị trí đầu tiên trong mảng đã sắp xếp mà đầu đọc sẽ bắt đầu xử lý.

Nếu currentHead không trùng khớp yêu cầu nào, lấy chỉ số gần nhất lớn hơn để bắt đầu.

A computer code on a black background

AI-generated content may be incorrect.Di chuyển lên trên (tăng dần) đến biên

A computer code on a black background

AI-generated content may be incorrect.

Đảm bảo đầu đọc di chuyển đến biên ngoài cùng (ví dụ: 199).

A black background with colorful text

AI-generated content may be incorrect.

Quay lại vị trí 0 (đầu đĩa).

A computer screen with text on it

AI-generated content may be incorrect.

Di chuyển tăng dần từ 0 đến các yêu cầu còn lại (nếu có) và cập nhật tổng seeks.

# **TÀI LIỆU THAM KHẢO**

**HDH-Buoi05-MemoryManagement.pdf** Khoa Công nghệ Thông tin, Đại học Sư phạm TP.HCM. (2019). *Chương 6: Bộ nhớ thực*

**HDH-Buoi05-Page Replacement.pdf** Khoa Công nghệ Thông tin, Đại học Sư phạm TP.HCM. (2019). *Chương 8: Thay thế trang*

**HDH-Buoi05-Paging\_Segmenting.pdf** Khoa Công nghệ Thông tin, Đại học Sư phạm TP.HCM. (2019). *Chương 7: Kỹ thuật phân trang và phân đoạn*

**HDH-Buoi05-QuanlyBonho.pdf** Khoa Công nghệ Thông tin,Đại học Sư phạm TP.HCM. (2019) *Quản lý bộ nhớ*

**HDH-Buoi07-File System Management.pdf** Khoa Công nghệ Thông tin,Đại học Sư phạm TP.HCM. (2019). *Hệ thống quản lý tập tin*

Tài Liệu Sinh Viên. (2018, 17 tháng 1). *Bài giảng Hệ điều hành – Chương 8: Bộ nhớ ảo*. Truy cập ngày 19 tháng 6 năm 2025, từ<https://tailieusinhvien.net/bai-giang-he-dieu-hanh-chuong-8-bo-nho-ao-truong-dai-hoc-cong-nghe-thong-tin-10943/>

ThuThuatPhanMem.vn. (n.d.). *OS (Operating System) là gì? Tổng quan về hệ điều hành OS*. Truy cập ngày 19 tháng 6 năm 2025, từ<https://thuthuatphanmem.vn/os-operating-system-la-gi-tong-quan-ve-he-dieu-hanh-os/>

[Link sản phầm](https://github.com/CornBoy2k5/HDH_2421COMP1332_THAIHOTPOT)