

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

TIỂU LUẬN CUỐI KỲ

HỌC PHẦN: HỆ ĐIỀU HÀNH

ĐỀ TÀI: TÌM HIỂU HỆ ĐIỀU HÀNH ANDROID VÀ MINH
HỌA CÁC GIẢI THUẬT PHÂN PHỐI CPU

Giáo viên hướng dẫn: ThS. Trần Đức Tâm

Nhóm: G08

Danh sách sinh viên thực hiện:

Nguyễn Văn Dũ	49.01.104.017
Diệp Quang Huy	49.01.104.050
Võ Trần Bảo Châu	49.01.104.013

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

TIỂU LUẬN CUỐI KỲ

HỌC PHẦN: HỆ ĐIỀU HÀNH

ĐỀ TÀI: TÌM HIỂU HỆ ĐIỀU HÀNH ANDROID VÀ MINH
HỌA CÁC GIẢI THUẬT PHÂN PHỐI CPU

Giáo viên hướng dẫn: ThS. Trần Đức Tâm
Nhóm: G08

Danh sách sinh viên thực hiện:

Nguyễn Văn Dũ	49.01.104.017
Diệp Quang Huy	49.01.104.050
Võ Trần Bảo Châu	49.01.104.013

Danh Mục Hình Ảnh

Hình 3.1: code class ProcessInfo.	11
Hình 3.2: code class ProcessInfo tiếp theo của hình (3.1).....	12
Hình 3.3: code class ProcessInfo tiếp theo của (hình 3.2).....	12
Hình 3.4: code class ExecutionSegment.	13
Hình 3.5: code class ExecutionSegment tiếp theo của (hình 3.4).	13
Hình 4.1: code thuật toán FCFS.....	15
Hình 4.2: code thuật toán SJF.....	16
Hình 4.3: code thuật toán SRTF.	17
Hình 4.4: code tiếp theo của SRTF.....	18
Hình 4.5: Code thuật toán Round Robin.	19
Hình 4.6: code tiếp theo của thuật toán Round Robin.....	20
Hình 5.1: giao diện khi bắt đầu chạy chương trình.	21
Hình 5.2: giao diện chương trình khi nhấp vào nút “BẮT ĐẦU” trong hình (4.4).	22
Hình 5.3: giao diện khi nhấp vào nút thêm.....	22
Hình 5.4: giao diện khi nhấp vào nút Sửa.....	23
Hình 5.5: hiển thị thông tin tiến trình người dùng đã nhập lên bảng “hiển thị người dùng đã nhập”.	23
Hình 5.6: hiển thị thông tin các đoạn thời gian, thời gian xử lý, thời gian xử lý trung bình, hiệu suất CPU (khi chọn thuật toán và nhấp nút Run).....	24

Mục Lục

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI.....	1
1.1. Tổng quan đề tài	1
1.2. Yêu cầu đề tài	1
1.3. Các chức năng chính.....	1
1.4. Phân công nhiệm vụ.....	1
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT	1
2.1. Lịch sử ra đời hệ điều hành Android	2
2.1.1. Nguồn gốc và sự thành lập	2
2.1.2. Google mua lại Android (2005).....	2
2.2. Các giai đoạn phát triển của Android.	2
2.2.1. Android 1.0 – 1.1 (2008–2009)	2
2.2.2. Android 1.5 đến 2.3 (Cupcake → Gingerbread)	2
2.2.3. Phiên bản cho máy tính bảng – Android 3.x (Honeycomb)	3
2.2.4. Android 4.x – Thống nhất nền tảng	3
2.2.5. Android 5.0 – 6.0 (Lollipop đến Marshmallow)	3
2.2.6. Android 7.0 – 9.0 (Nougat đến Pie).....	3
2.2.7. Android 10 đến Android 13 (2019–2022)	3
2.2.8. Android 14 (2023)	4
2.2.9. Android 15 (2024)	4
2.3. Ý nghĩa của Android trong đời sống công nghệ hiện đại	4
2.4. Lý thuyết định thời CPU.....	4
2.4.1. FCFS (First-Come, First-Served)	4
2.4.2. SJF (Shortest Job First).....	4
2.4.3. SRTF (Shortest Remaining Time First).....	5
2.4.4. Round Robin (RR – Vòng xoay)	5
CHƯƠNG 3. PHÂN TÍCH VÀ THIẾT KẾ	7
3.1. Phân Tích	7
3.1.1. Yêu cầu bài toán:	7
3.1.2. Các chức năng của chương trình:	7

3.1.3. Ý tưởng thuật toán	7
3.2. Thiết kế	9
3.2.1. Class chứa thông tin của từng tiến trình	10
3.2.2. Các file khác:	13
CHƯƠNG 4. CÁC BƯỚC THỰC HIỆN Ý TƯỞNG CỦA CÁC THUẬT TOÁN VÀ CODE MINH HỌA	15
4.1. Thuật toán FCFS	15
4.2. Thuật toán SJF	15
4.3. Thuật toán SRTF.....	16
4.4. Thuật toán Round Robin.....	18
CHƯƠNG 5. SẢN PHẨM	21
5.1. màn hình giao diện mở đầu khi chạy chương trình:	21
5.2. Màn hình giao diện khi nhấp nút “Bắt Đầu”	21
5.2.1. các chức năng chính.....	21
5.2.2. mô tả (Hình 5.2):.....	22
5.3. Hướng dẫn mở chương trình.....	24
5.3.1. Cài đặt Visual Studio	24
5.3.2. Sau khi tải Visual Studio 2022	24
KẾT LUẬN VÀ KIẾN NGHỊ.....	25
HƯỚNG DẪN CÀI ĐẶT VÀ TRIỂN KHAI CHƯƠNG TRÌNH.....	26
TÀI LIỆU THAM KHẢO	27

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1. Tổng quan đề tài

Trong xu hướng phát triển nhanh chóng của công nghệ thông tin, hệ điều hành Android đã trở thành một trong những nền tảng quan trọng nhất trên thiết bị di động. Bên cạnh đó, việc quản lý tiến trình bằng các giải thuật phân phối CPU đóng vai trò then chốt trong việc tối ưu hoá hiệu năng và đảm bảo độ ổn định của hệ thống.

Trong bài tiểu luận này, nhóm chúng em tìm hiểu hai vấn đề:

- Quá trình hình thành và phát triển của hệ điều hành Android.
- Các giải thuật phân phối CPU và chương trình minh họa.

1.2. Yêu cầu đề tài

- Tìm hiểu thông tin lịch sử Android và trình bày có hệ thống.
- Hiểu và mô phỏng hoạt động của các giải thuật phân phối CPU.
- Thiết kế chương trình mô phỏng bằng ngôn ngữ lập trình (Python, C++, C#...).

1.3. Các chức năng chính

- Cho phép người dùng nhập dữ liệu tiến trình.
- Chọn giải thuật CPU và mô phỏng kết quả.
- Vẽ biểu đồ Gant minh họa thời gian xử lý (nhóm thay thế bằng bảng hiển thị các thông tin của từng thời gian).

1.4. Phân công nhiệm vụ

Họ tên	Nội dung thực hiện	Mức độ hoàn thành
Nguyễn Văn Dũ	<ul style="list-style-type: none"> • Thiết kế tạo giao diện, code chính chương trình. • Nội dung word 	100/100
Diệp Quang Huy	<ul style="list-style-type: none"> • Nội dung word • Hỗ trợ thiết kế tạo giao diện. • Powerpoint báo cáo. 	100/100
Võ Trần Bảo Châu	<ul style="list-style-type: none"> • Thiết kế, định dạng, căn chỉnh file word. • Nội dung word • Thiết kế giao diện chương trình. 	100/100

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1. Lịch sử ra đời hệ điều hành Android

2.1.1. Nguồn gốc và sự thành lập

Android Inc. được thành lập vào tháng 10 năm 2003 tại Palo Alto, California bởi Andy Rubin, Rich Miner, Nick Sears và Chris White.

Ban đầu, công ty này định hướng phát triển một hệ điều hành dành cho máy ảnh kỹ thuật số, nhưng sau đó đã chuyển sang phát triển hệ điều hành cho điện thoại di động do nhận thấy tiềm năng lớn của thị trường này.

2.1.2. Google mua lại Android (2005)

Tháng 8 năm 2005, Google mua lại Android Inc. với giá khoảng 50 triệu USD.

Android trở thành dự án phát triển nội bộ bí mật của Google, với đội ngũ đứng đầu vẫn là Andy Rubin.

Năm 2007, Google thành lập Liên minh Thiết bị Cầm tay Mở (Open Handset Alliance - OHA) với nhiều hãng lớn như Samsung, HTC, Qualcomm, Motorola... để phát triển Android như một hệ điều hành mã nguồn mở cho thiết bị di động.

2.2. Các giai đoạn phát triển của Android.

2.2.1. Android 1.0 – 1.1 (2008–2009)

- Android 1.0 (tháng 9 năm 2008): Thiết bị đầu tiên chạy Android là HTC Dream. Các tính năng cơ bản gồm có: trình duyệt web, Google Maps, Gmail, YouTube và kho ứng dụng Android Market.
- Android 1.1 (2009): Cải tiến nhỏ về hiển thị bản đồ và giao diện người dùng.

2.2.2. Android 1.5 đến 2.3 (Cupcake → Gingerbread)

- Android 1.5 – Cupcake: Hỗ trợ bàn phím ảo và tiện ích ngoài màn hình chính.
- Android 1.6 – Donut: Cải tiến giao diện tìm kiếm, hỗ trợ nhiều độ phân giải màn hình.
- Android 2.0/2.1 – Eclair: Tích hợp chỉ đường theo thời gian thực, hỗ trợ đa tài khoản email.
- Android 2.2 – Froyo: Tăng hiệu năng xử lý, hỗ trợ kết nối điểm phát sóng

không dây.

- Android 2.3 – Gingerbread: Giao diện mới, cải thiện quản lý năng lượng.

2.2.3. Phiên bản cho máy tính bảng – Android 3.x (Honeycomb)

- Phát hành năm 2011, tập trung vào máy tính bảng. Giới thiệu giao diện mới phù hợp với màn hình lớn và hỗ trợ xử lý đa nhân.

2.2.4. Android 4.x – Thống nhất nền tảng

- Android 4.0 – Ice Cream Sandwich: Kết hợp giao diện cho cả điện thoại và máy tính bảng, ra mắt năm 2011.
- Android 4.1–4.3 – Jelly Bean: Cải thiện độ mượt (Dự án Butter), thông báo mở rộng.
- Android 4.4 – KitKat: Hoạt động ổn định trên thiết bị có cấu hình thấp, ra mắt năm 2013.

2.2.5. Android 5.0 – 6.0 (Lollipop đến Marshmallow)

- Lollipop (2014): giới thiệu thiết kế giao diện mới (Material Design), cải thiện thời lượng pin.
- Marshmallow (2015): cấp quyền truy cập linh hoạt hơn, hỗ trợ cảm biến vân tay.

2.2.6. Android 7.0 – 9.0 (Nougat đến Pie)

- Nougat (2016): Chia đôi màn hình để dùng song song hai ứng dụng.
- Oreo (2017): Hỗ trợ hình ảnh trong hình, tăng cường bảo mật.
- Pie (2018): Điều khiển bằng cử chỉ, tối ưu hóa pin theo thói quen người dùng.

2.2.7. Android 10 đến Android 13 (2019–2022)

- Android 10: Ra mắt chế độ nền tối, quản lý quyền truy cập tốt hơn.
- Android 11: Cho phép cấp quyền một lần, hiển thị tin nhắn nổi.
- Android 12: Ra mắt kiểu thiết kế mới (Material You), cá nhân hóa giao diện theo màu nền.
- Android 13: Điều chỉnh quyền truy cập ảnh, âm thanh, cải thiện hỗ trợ đa

ngôn ngữ cho từng ứng dụng.

2.2.8. Android 14 (2023)

- Android 14 (2023) tập trung vào tối ưu hóa bảo mật, tiết kiệm pin, và hỗ trợ thiết bị có màn hình gập.

2.2.9. Android 15 (2024)

- Thời điểm ra mắt: Android 15 chính thức được phát hành vào ngày 16 tháng 10 năm 2024, đầu tiên trên các thiết bị Pixel của Google .

2.3. Ý nghĩa của Android trong đời sống công nghệ hiện đại

- Android hiện chiếm trên 70% thị phần hệ điều hành di động toàn cầu.
- Là nền tảng mở, Android tạo điều kiện cho hàng triệu nhà phát triển trên toàn thế giới xây dựng ứng dụng phong phú và đa dạng.
- Android góp phần quan trọng trong việc thúc đẩy chuyển đổi số, tạo ra sự cạnh tranh lành mạnh và giúp người dùng có nhiều lựa chọn phù hợp.

2.4. Lý thuyết định thời CPU

Giải thuật lập lịch (CPU scheduling algorithms) là những thuật toán mà hệ điều hành áp dụng để xác định tiến trình được cấp CPU tiếp theo. Các thuật toán lập lịch bao gồm:

2.4.1. FCFS (First-Come, First-Served)

Nguyên lý hoạt động:

Tiến trình nào đến trước sẽ được phục vụ trước. Đây là thuật toán cơ bản nhất, hoạt động theo hàng đợi FIFO (First-In-First-Out).

Được áp dụng khi cần một hệ thống dễ cài đặt, số lượng tiến trình không nhiều và không cần ưu tiên hay xử lý thời gian thực.

Do sự giản đơn của hệ thống, CPU không yêu cầu thông tin phức tạp về thời gian chạy, nhưng tiến trình ngắn phải chờ lâu nếu đứng sau tiến trình dài. Do đó, tùy vào các tiến trình thời gian chờ trung bình (Average Waiting Time) có thể cao.

2.4.2. SJF (Shortest Job First)

Nguyên lý hoạt động:

Tiến trình có thời gian thực thi (CPU burst) ngắn nhất sẽ được phục vụ trước. Khi một tiến trình đang chạy, nó không bị gián đoạn. (**Non-preemptive**)

Được sử dụng chỉ khi ta có thể ước lượng chính xác thời gian thực thi của tiến trình, thời gian thực thi rất khó xác định trong thực tế. Giải thuật có thể tối ưu hóa thời gian chờ trung bình, chọn tiến trình có thời gian thực thi ngắn nhất trong số các tiến trình đang chờ. Không phù hợp với các hệ thống yêu cầu phản hồi nhanh (real-time).

Các tiến trình dài có thể bị “starvation” (đói tài nguyên), mất rất lâu hoặc không bao giờ được cấp tài nguyên để thực thi, mặc dù đang chờ sẵn.

2.4.3. SRTF (Shortest Remaining Time First)

Nguyên lý hoạt động:

SRTF là phiên bản hoán đổi (preemptive) của SJF. Tiến trình đang chạy có thể bị gián đoạn nếu có tiến trình mới đến với thời gian còn lại ngắn hơn.

Khác với SJF, SRTF liên tục cập nhật thời gian còn lại của tiến trình đang chạy và so sánh với các tiến trình mới đến để đạt được thời gian chờ trung bình thấp nhất. Các tiến trình có thời gian còn lại ngắn nhất tại mọi thời điểm sẽ được chọn và chuyển đổi nếu cần, dẫn đến chi phí hoán đổi tiến trình cao hơn.

SRTF thích hợp cho hệ thống cần phản hồi nhanh, ưu tiên tiến trình nhỏ, nhưng vẫn có thể bị đói tài nguyên (starvation).

2.4.4. Round Robin (RR – Vòng xoay)

Nguyên lý hoạt động:

Mỗi tiến trình được cấp CPU trong một khoảng thời gian cố định gọi là quantum (time slice). Nếu tiến trình chưa xong khi hết quantum, nó bị đẩy về cuối hàng đợi và CPU được cấp cho tiến trình tiếp theo.

Thuật toán được sử dụng trong tình huống hệ thống đa người dùng, đa nhiệm, cần phân chia tài nguyên một cách công bằng, đảm bảo không tiến trình nào bị bỏ rơi (starvation). Thường xuyên được sử dụng trong các hệ điều hành (Linux, Windows).

Đây là thuật toán không chờ đợi (preemptive), công bằng cho mọi tiến trình, thích hợp cho hệ thống chia sẻ thời gian (time-sharing). Mọi tiến trình đều có cơ hội thực thi.

Hiệu năng phụ thuộc vào kích thước quantum. Nếu quá nhỏ có gây ra nhiều lần hoán đổi (context switch), giảm hiệu suất. Nếu quá lớn sẽ giống với FCFS, mất hết lợi ích của Round Robin.

CHƯƠNG 3. PHÂN TÍCH VÀ THIẾT KẾ

3.1. Phân Tích

3.1.1. Yêu cầu bài toán:

Mô phỏng hoạt động của 4 thuật toán định thời CPU:

- First Come First Service (FCFS).
- Shortest Job First (SJF).
- Shortest Remaining Time First (SRTF).
- Round Robin.

Đầu vào: tên tiến trình, thời điểm vào hàng đợi, và thời gian sử dụng Bộ xử lý trung tâm (tên tiếng anh: central processing unit (viết tắt: CPU)), thời điểm, thời gian I/O.

Đầu ra: Biểu đồ Gantt, thời gian đáp ứng, thời gian đáp ứng trung bình, thời gian đợi, thời gian đợi trung bình, thời gian xoay vòng, thời gian xoay vòng trung bình, hiệu suất CPU.

3.1.2. Các chức năng của chương trình:

- Thêm thông tin của tiến trình tiến trình.
- Sửa thông tin của tiến trình.
- Xóa 1 tiến trình đã nhập.
- Lựa chọn thuật toán cần chạy.
- Hiển thị đầu ra:
 - Hiển thị bản các thông tin về từng đoạn thời gian (thay thế cho biểu đồ Gantt).
 - Hiện thị lên bảng các thông tin về thời gian đợi, thời gian đáp ứng, thời gian xoay vòng
 - Hiện thị thời gian đợi trung bình, thời gian đáp ứng trung bình, thời gian xoay vòng trung bình.

3.1.3. Ý tưởng thuật toán

3.1.3.1 thuật toán FCFS

- **Mục đích:** Mô phỏng thuật toán lập lịch CPU theo nguyên tắc "đến trước

phục vụ trước", nơi mỗi tiến trình được xử lý theo thứ tự thời điểm đến mà không có ngắt (non-preemptive).

- **Ý tưởng cốt lõi:**
 - Sắp xếp danh sách tiến trình theo Arrival Time tăng dần
 - Duyệt lần lượt từng tiến trình:
 - Nếu tiến trình đến sau thời gian hiện tại, CPU sẽ đợi đến thời điểm tiến trình đến mới chạy.
 - Nếu tiến trình đến sớm hoặc bằng thời gian hiện tại, CPU sẽ chạy ngay.
 - Tính toán:
 - StartTime: thời điểm bắt đầu thực thi.
 - CompletionTime: thời điểm tiến trình hoàn tất (StartTime + BurstTime).
 - Cập nhật thời gian hiện tại (currentTime) để phục vụ tiến trình kế tiếp.

3.1.3.2 Thuật toán SJF

- **Mục đích:** Thuật toán SJF là một dạng lập lịch không ưu tiên ngắt (non-preemptive), lựa chọn tiến trình có thời gian thực thi ngắn nhất (Burst Time) trong số các tiến trình đã đến để chạy trước.
- **Ý tưởng cốt lõi:**
 - Mỗi thời điểm, CPU sẽ chọn tiến trình có thời gian thực thi (burst time) ngắn nhất trong các tiến trình đã đến (arrival time \leq current Time)
 - Tiến trình được chọn sẽ được chạy đến khi hoàn tất (non-preemptive).
 - Khi hoàn thành, CPU sẽ xét tiếp các tiến trình còn lại, lặp lại quy trình cho đến khi tất cả hoàn thành.

3.1.3.3 Thuật toán SRTF

- **Mục đích:** Thuật toán SRTF là dạng mở rộng của Shortest Job First, nhưng có khả năng ngắt tiến trình đang chạy nếu có một tiến trình mới đến có thời gian còn lại ngắn hơn.

- **Ý tưởng cốt lõi:**

- Tại mỗi đơn vị thời gian, thuật toán duyệt toàn bộ các tiến trình đã đến và chưa hoàn thành.
- Trong các tiến trình đó, chọn tiến trình có thời gian còn lại ngắn nhất để thực thi.
- Nếu đang thực thi một tiến trình và có tiến trình khác đến với thời gian còn lại nhỏ hơn, thì CPU sẽ ngắt tiến trình hiện tại và chuyển sang tiến trình mới.
- Mỗi lần có thay đổi tiến trình đang chạy, ghi lại một đoạn thực thi (ExecutionSegment).
- Cập nhật lại các thông tin như StartTime, CompletionTime và RemainingTime (thời gian thực thi còn lại của tiến trình).

3.1.3.4 Thuật toán Round Robin

- **Mục đích:** Thuật toán Round Robin là một phương pháp lập lịch CPU có ngắt (preemptive), nhằm chia sẻ công bằng thời gian CPU cho các tiến trình. Mỗi tiến trình chỉ được cấp phát CPU trong một khoảng thời gian cố định (quantum time). Nếu chưa hoàn thành sau thời gian này, tiến trình sẽ được đưa về cuối hàng đợi và chờ đến lượt tiếp theo.

- **Ý tưởng cốt lõi:**

- Sử dụng một hàng đợi FIFO (Queue) để lưu trữ các tiến trình sẵn sàng.
- Tại mỗi thời điểm, tiến trình đầu tiên trong hàng đợi sẽ được cấp CPU trong khoảng thời gian quantum hoặc thời gian còn lại nếu nhỏ hơn quantum.
- Nếu chưa hoàn thành → thêm lại vào cuối hàng đợi.
- Nếu trong thời gian tiến trình đang chạy có tiến trình mới đến → thêm vào hàng đợi nếu chưa có.
- Lặp lại quá trình đến khi tất cả các tiến trình hoàn thành.
- Mỗi đoạn thời gian thực thi của tiến trình được lưu vào danh sách execuSegments.

3.2. Thiết kế

Môi trường phát triển: Nhóm dùng Ngôn ngữ C# kết hợp với Windows Forms trên Visual Studio 2022.

3.2.1. Class chứa thông tin của từng tiến trình

3.2.1.1 class chứa thông tin tiến trình

- Tên class ProcessInfo.
- Các thuộc tính trong class ProcessInfo:
 - string id (tên của tiến trình).
 - int arrivalTime (thời điểm vào hàng đợi của tiến trình).
 - int burstTime (thời gian sử dụng CPU của tiến trình).
 - int remainingTime (thời gian sử dụng CPU còn lại của tiến trình).
 - int startTime (thời điểm mà tiến trình mới bắt đầu sử dụng CPU).
 - int completionTime (thời điểm mà tiến trình đã chạy xong).
- Các hàm lấy thông tin của thuộc tính, cập nhật thông tin của thuộc tính của class ProcessInfo:
 - Hàm getter, setter Id: lấy, cập nhật tên của tiến trình, cập nhật thông tin của tiến trình.
 - Hàm getter, setter ArrivalTime: lấy, cập nhật thời điểm vào của tiến trình.
 - Hàm getter, setter BurstTime: lấy, cập nhật thời gian sử dụng CPU.
 - Hàm getter, setter RemainingTime: lấy, Cập nhật thời gian sử dụng CPU còn lại.
 - Hàm getter, setter StartTime: lấy, cập nhật thời điểm bắt đầu sử dụng CPU của tiến trình.
 - Hàm getter, setter CompletionTime
 - Hàm tính thời gian đợi của từng tiến trình (tên: WaitingTime): trả về giá trị của thời gian đợi theo công thức tính thời gian đợi (thời gian xoay vòng – thời gian sử dụng CPU).
 - Hàm tính thời gian xoay vòng (tên TurnaroundTime): trả về giá trị của thời gian xoay vòng theo công thức (thời điểm tiến trình xử lý xong – thời điểm vào hàng đợi của tiến trình).
 - Hàm tính thời gian đáp ứng (ResponseTime): trả về giá trị là thời

gian đáp ứng của tiến trình theo công thức (thời điểm tiến trình bắt đầu được sử dụng CPU – thời điểm vào của tiến trình).

```

10 internal class ProcessInfo
11 {
12     private string id;
13     private int arrivalTime;
14     private int burstTime;
15     private int remainingTime;
16     private int startTime;
17     private int completionTime;
18
19     1 reference
20     public ProcessInfo(string id, int arrivalTime, int burstTime)
21     {
22         this.id = id;
23         this.arrivalTime = arrivalTime;
24         this.burstTime = burstTime;
25         this.remainingTime = burstTime;
26         this.startTime = -1;
27         this.completionTime = 0;
28     }
29     11 references
30     public string Id
31     {
32         get { return id; }
33         set { id = value; }
34     }
35     15 references
36     public int ArrivalTime
37     {
38         get { return arrivalTime; }
39         set { arrivalTime = value; }
40     }
41     8 references

```

Hình 3.1: code class ProcessInfo.

```

40     public int BurstTime
41     {
42         get { return burstTime; }
43         set { burstTime = value; }
44     }
45     15 references
46     public int RemainingTime
47     {
48         get { return remainingTime; }
49         set { remainingTime = value; }
50     }
51     9 references
52     public int StartTime
53     {
54         get { return startTime; }
55         set { startTime = value; }
56     }
57     10 references
58     public int CompletionTime
59     {
60         get { return completionTime; }
61         set { completionTime = value; }
62     }
63
64     // Turnaround Time = Completion - Arrival
65     3 references
66     public int TurnaroundTime
67     {
68         get { return completionTime - arrivalTime; }

```



```

8  namespace OS
9  {
10     15 references
11     internal class ExecutionSegment
12     {
13         private string processName; // Tên tiến trình
14         private int startTime; // Thời điểm bắt đầu chạy
15         private int endTime; // Thời điểm kết thúc chạy
16
17         5 references
18         public ExecutionSegment(string _processName, int _startTime, int _endTime)
19         {
20             processName = _processName;
21             startTime = _startTime;
22             endTime = _endTime;
23         }
24
25         1 reference
26         public string ProcessName
27         {
28             get { return processName; }
29             set { processName = value; }
30         }
31
32         4 references
33         public int StartTime
34         {
35             get { return startTime; }
36             set { startTime = value; }
37         }
38     }
39 }

```

Hình 3.4: code class ExecutionSegment.

```

30         get { return startTime; }
31         set { startTime = value; }
32     }
33
34     2 references
35     public int EndTime
36     {
37         get { return endTime; }
38         set { endTime = value; }
39     }
40 }
41

```

Hình 3.5: code class ExecutionSegment tiếp theo của (hình 3.4).

3.2.2. Các file khác:

- IntroInterface.cs [Design]: tạo giao diện phần mở đầu.
- IntroInterface.cs: xử lý sự kiện click vào nút “Bắt Đầu” trong màn hình giao diện mở đầu.
- Form1.cs [Design]: giao diện chọn chức năng thêm sửa xóa tiến trình, hiển thị thông tin tiến trình đã nhập lên bảng, lựa chọn thuật toán để chạy, hiển thị các đoạn thời gian xử lý tiến trình, hiển thị thời gian đợi, thời gian đáp ứng, thời gian xoay vòng, hiển thị thời gian đợi trung bình, đáp ứng trung bình, xoay vòng trung bình, hiệu suất CPU.
- Form1.cs: dùng để, viết mã các thuật toán, xử lý event thêm, sửa, xóa thông

tin tiến trình, chọn thuật toán để chạy, hiển thị thông tin đầu ra sau khi chọn thuật toán.

CHƯƠNG 4. CÁC BƯỚC THỰC HIỆN Ý TƯỞNG CỦA CÁC THUẬT TOÁN VÀ CODE MINH HỌA

4.1. Thuật toán FCFS

- Các bước chính của thuật toán:
 - Bước 1: Kiểm tra nếu không có dữ liệu → hiển thị cảnh báo.
 - Bước 2: Sắp xếp danh sách tiến trình theo ArrivalTime.
 - Bước 3: Khởi tạo thời gian hiện bằng thời điểm vào của tiến trình đầu tiên.
 - Bước 4: Với mỗi tiến trình Nếu thời điểm vào > thời gian hiện: CPU rảnh, tiến trình bắt đầu lúc thời điểm vào của tiến trình đó, kết thúc lúc (thời điểm vào + thời gian sử dụng CPU). Ngược lại tiến trình bắt đầu ngay tại thời gian hiện tại và kết thúc lúc (thời gian hiện tại + thời gian sử dụng CPU của tiến trình).
 - Bước 5: lưu lại đoạn chạy.
 - Bước 6: cập nhật thời gian hiện tại = thời gian kết thúc của đoạn thời chạy.

```

451 |
452 |
453 |
454 |
455 |
456 |
457 |
458 |
459 |
460 |
461 |
462 |
463 |
464 |
465 |
466 |
467 |
468 |
469 |
470 |
471 |
472 |
473 |
474 |

```

```

1
List<ExecutionSegment> execuSegments = new List<ExecutionSegment>(); // danh sách lưu thời gian chạy từng đoạn.
var processSorted = dataInput.OrderBy(p => p.ArrivalTime).ToList();

int currentTime = processSorted[0].ArrivalTime; // Thời gian hiện tại bắt đầu từ thời điểm đến của tiến trình đầu tiên
foreach (var process in processSorted)
{
    if (process.ArrivalTime > currentTime)
    {
        process.StartTime = process.ArrivalTime;
        process.CompletionTime = process.ArrivalTime + process.BurstTime;
    }
    else
    {
        process.StartTime = currentTime;
        process.CompletionTime = currentTime + process.BurstTime;
    }

    execuSegments.Add(new ExecutionSegment(process.Id, process.StartTime, process.CompletionTime));
    currentTime = process.CompletionTime; // Cập nhật thời gian hiện tại
}

```

Hình 4.1: code thuật toán FCFS.

4.2. Thuật toán SJF

- Các bước chính của thuật toán:
 - Bước 1: gán thời gian còn lại bằng thời gian sử dụng CPU cho mỗi tiến trình.

- Bước 2: khởi tại thời gian hiện tại = 0, số lượng tiến trình đã hoàn thành = 0.
- Bước 3: Lặp lại cho đến khi số lượng tiến trình đã hoàn thành == số lượng tiến trình.
 - Lọc danh sách tiến trình đã đến (thời điểm đến \leq thời gian hiện tại) và chưa xong (thời gian còn lại > 0).
 - Sắp xếp theo BurstTime, chọn tiến trình ngắn nhất.
 - Nếu không có tiến trình đủ điều kiện \rightarrow tăng thời gian hiện tại lên 1 (CPU chờ).
 - Nếu có tiến trình phù hợp \rightarrow tính toán:
 - Thời điểm bắt đầu sử dụng CPU = thời gian hiện tại.
 - Thời gian tiến trình xử lý xong = thời điểm bắt đầu của tiến trình + thời gian còn lại của tiến trình.
 - Lưu đoạn thực thi.
 - Cập nhật thời gian còn lại = 0, thời gian còn lại = thời gian kết thúc của đoạn thực thi.
 - Tăng số lượng tiến trình lên 1.

```

488 List<ExecutionSegment> executionSegments = new List<ExecutionSegment>(); // danh sách lưu thời gian chạy từng đoạn.
489 // Khởi tạo thời gian còn lại bằng burst time
490 foreach (var p in dataInput)
491 {
492     p.RemainingTime = p.BurstTime;
493 }
494
495 int currentTime = 0; // Thời gian hiện tại
496 int completed = 0; // Số tiến trình đã hoàn thành
497 int n = dataInput.Count; // Tổng số tiến trình
498
499 while (completed < n)
500 {
501     // Lấy tất cả tiến trình đến thời gian chạy, sort burst time
502     var validProcces = dataInput
503         .Where(p => p.ArrivalTime <= currentTime && p.RemainingTime > 0)
504         .OrderBy(p => p.BurstTime)
505         .ThenBy(p => p.ArrivalTime)
506         .ToList();
507
508     if (validProcces.Count == 0)
509     {
510         currentTime++; // Không có tiến trình nào đến, thời gian trôi đi
511         continue;
512     }
513
514     var shorstProcces = validProcces.First(); // Lấy tiến trình có thời gian chạy ngắn nhất
515     shorstProcces.StartTime = currentTime; // Ghi lại thời điểm bắt đầu chạy
516     shorstProcces.CompletionTime = currentTime + shorstProcces.RemainingTime; // Tính thời điểm hoàn thành
517     executionSegments.Add(new ExecutionSegment(shorstProcces.Id, shorstProcces.StartTime, shorstProcces.CompletionTime)); // Lưu đ
518     currentTime = shorstProcces.CompletionTime; // Cập nhật thời gian hiện tại
519     shorstProcces.RemainingTime = 0; // Đánh dấu tiến trình đã hoàn thành
520     completed++; // Tăng số tiến trình đã hoàn thành
521 }

```

Hình 4.2: code thuật toán SJF.

4.3. Thuật toán SRTF

- Các bước chính của thuật toán:

- Bước 1: gán thời gian còn lại bằng thời gian sử dụng CPU cho mỗi tiến trình.
- Bước 2: Khởi tạo thời gian hiện tại = 0, số lượng tiến trình đã hoàn thành = 0, tiến trình hiện tại == null (không có giá trị nào được gán).
- Bước 3: Lặp lại đến khi tất cả tiến trình hoàn thành (thời gian hoàn thành \leq số lượng tiến trình)
 - Lọc ra danh sách tiến trình đã đến (thời điểm đến \leq thời gian hiện tại) và chưa xong (thời gian còn lại > 0).
 - Chọn tiến trình có thời gian còn lại nhỏ nhất.
 - Nếu có sự thay đổi tiến trình, lưu lại đoạn thực thi của tiến trình cũ và cập nhật tiến trình hiện tại, thời điểm thay đổi.
 - Nếu tiến trình chạy lần đầu \rightarrow gán thời điểm bắt đầu = thời gian hiện tại.
 - Giảm thời gian còn lại của tiến trình, tăng thời gian hiện tại lên 1.
 - Nếu tiến trình hoàn thành (thời gian còn lại == 0) \rightarrow tăng số lượng tiến trình chạy xong lên 1.

```

537 List<ExecutionSegment> executionSegments = new List<ExecutionSegment>(); // danh sách lưu thời gian chạy từng đoạn.
538
539 // Khởi tạo thời gian còn lại bằng burst time
540 foreach (var p in dataInput)
541 {
542     p.RemainingTime = p.BurstTime;
543 }
544
545 int currentTime = 0;
546 int completed = 0;
547 int n = dataInput.Count;
548 ProcessInfo currentProcess = null;
549 int lastSwitchTime = 0;
550
551 while (completed < n)
552 {
553     // Chọn process còn thời gian chạy nhỏ nhất trong những process đã tới
554     var availableProcesses = dataInput
555         .Where(p => p.ArrivalTime <= currentTime && p.RemainingTime > 0)
556         .OrderBy(p => p.RemainingTime)
557         .ThenBy(p => p.ArrivalTime)
558         .ToList();
559
560     if (availableProcesses.Count == 0)
561     {
562         currentTime++; // Không có process nào đến, time trôi đi
563         continue;
564     }
565
566     var shortestProcess = availableProcesses.First();

```

Hình 4.3: code thuật toán SRTF.

```

567     var shortestProcess = availableProcesses.First();
568
569     // Nếu chuyển process khác, lưu lại thời gian chạy của process cũ
570     if (currentProcess != shortestProcess)
571     {
572         if (currentProcess != null)
573         {
574             // Kết thúc đoạn chạy trước đó
575             excutionSegments.Add(new ExecutionSegment(currentProcess.Id, lastSwitchTime, currentTime));
576         }
577
578         currentProcess = shortestProcess;
579
580         // Nếu process mới chạy lần đầu tiên thì đặt startTime
581         if (currentProcess.StartTime == -1)
582         {
583             currentProcess.StartTime = currentTime;
584         }
585
586         lastSwitchTime = currentTime;
587     }
588
589     // Giảm thời gian còn lại của process đang chạy
590     currentProcess.RemainingTime--;
591     currentTime++;
592
593     // Nếu process hoàn thành
594     if (currentProcess.RemainingTime == 0)
595     {
596         // Kết thúc đoạn chạy cuối cùng
597         excutionSegments.Add(new ExecutionSegment(currentProcess.Id, lastSwitchTime, currentTime));
598
599         currentProcess.CompletionTime = currentTime;
600         completed++;
601         currentProcess = null;
602     }
603 }

```

Hình 4.4: code tiếp theo của SRTF.

4.4. Thuật toán Round Robin

- Các bước chính của thuật toán:
 - Bước 1: gán thời gian còn lại bằng thời gian sử dụng CPU cho mỗi tiến trình.
 - Bước 2: Khởi tạo thời gian hiện tại = 0, số lượng tiến trình đã chạy xong = 0, tạo hàng đợi (queue) rỗng.
 - Bước 3: Lặp lại đến khi tất cả tiến trình hoàn thành (số lượng tiến trình hoàn thành == số lượng tiến trình):
 - Lọc ra các tiến trình đã đến (thời điểm vào ≤ thời gian hiện tại) và còn thời gian chạy.
 - Thêm vào hàng đợi nếu chưa có trong đó.
 - Nếu hàng đợi rỗng → tăng thời gian hiện tại lên 1 và quay lại.
 - Lấy tiến trình đầu tiên trong hàng đợi để xử lý.
 - Nếu đây là lần đầu chạy → gán thời điểm bắt đầu = thời gian hiện tại.
 - Chọn thời gian chạy của đoạn hiện tại = giá trị nhỏ nhất giữa quantum và thời gian còn lại.

- Lưu kết quả đoạn chạy.
- Cập nhật thời gian còn lại và thời gian hiện tại.
- Nếu tiến trình hoàn thành → lưu thời điểm kết thúc của tiến trình, tăng số lượng tiến trình hoàn thành lên 1.
- Nếu chưa hoàn thành → kiểm tra có tiến trình mới đến trong khoảng vừa chạy và thêm vào hàng đợi (nếu chưa có), rồi đưa lại tiến trình chưa xong vào cuối hàng đợi.

```

617 |
618 | ✓
619 |
620 |
621 |
622 |
623 |
624 |
625 |
626 |
627 |
628 |
629 |
630 | ✓
631 |
632 |
633 |
634 |
635 |
636 |
637 |
638 |
639 | ✓
640 |
641 | ✓
642 |
643 |
644 |
645 |
646 |
647 |
648 | ✓
649 |
650 |
651 |
652 |
653 |

```

```

foreach (var process in dataInput)
{
    process.RemainingTime = process.BurstTime;
}

List<ExecutionSegment> execuSegments = new List<ExecutionSegment>(); // lưu kết quả từng đoạn time
Queue<ProcessInfo> queue = new Queue<ProcessInfo>(); // hàng đợi tiến trình

int currentTime = 0; // thời gian hiện tại
int completed = 0; // số tiến trình đã hoàn thành
int n = dataInput.Count; // tổng số tiến trình

while (completed < n)
{
    // Lấy tất cả các tiến trình đã đến và còn thời gian chạy
    var arrived = dataInput
        .Where(p => p.ArrivalTime <= currentTime && p.RemainingTime > 0)
        .OrderBy(p => p.ArrivalTime)
        .ToList();

    // Thêm vào hàng đợi nếu chưa có
    foreach (var p in arrived)
    {
        if (!queue.Contains(p))
        {
            queue.Enqueue(p);
        }
    }

    // Nếu hàng đợi rỗng, thời gian trôi đi
    if (queue.Count == 0)
    {
        currentTime++;
        continue;
    }
}

```

Hình 4.5: Code thuật toán Round Robin.

```

653 |
654 |
655 |
656 | ✓
657 |
658 |
659 |
660 |
661 |
662 |
663 |
664 |
665 |
666 |
667 |
668 | ✓
669 |
670 |
671 |
672 |
673 | ✓
674 |
675 |
676 |
677 |
678 |
679 |
680 |
681 |
682 | ✓
683 |
684 |
685 |
686 |
687 |
688 |
689 |

```

```

var currentProcess = queue.Dequeue();

if (currentProcess.StartTime == -1)
{
    currentProcess.StartTime = currentTime;
}

int executeTime = Math.Min(quantum, currentProcess.RemainingTime);
execuSegments.Add(new ExecutionSegment(currentProcess.Id, currentTime, currentTime + executeTime));

currentTime += executeTime;
currentProcess.RemainingTime -= executeTime;

// Sau khi chạy, kiểm tra xem process đã hoàn thành chưa
if (currentProcess.RemainingTime == 0)
{
    currentProcess.CompletionTime = currentTime;
    completed++;
}
else
{
    // Nếu chưa xong thì thêm lại vào cuối hàng đợi
    // cập nhật tiến trình mới đến trong khoảng thời gian vừa chạy
    var newArrived = dataInput
        .Where(p => p.ArrivalTime > currentTime - executeTime && p.ArrivalTime <= currentTime && p.RemainingTime > 0)
        .OrderBy(p => p.ArrivalTime)
        .ToList();

    foreach (var p in newArrived)
    {
        if (!queue.Contains(p))
        {
            queue.Enqueue(p);
        }
    }

    queue.Enqueue(currentProcess); // thêm lại tiến trình chưa xong vào cuối hàng đợi
}

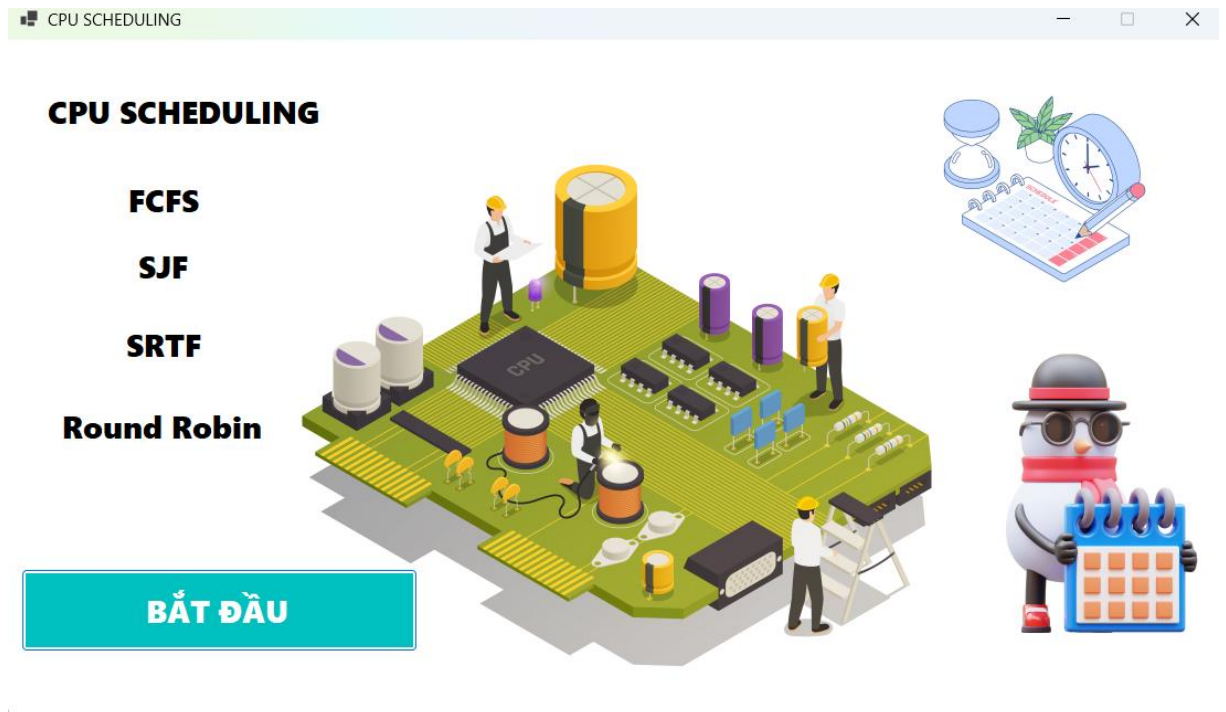
```


Hình 4.6: code tiếp theo của thuật toán Round Robin.

CHƯƠNG 5. SẢN PHẨM

5.1. màn hình giao diện mở đầu khi chạy chương trình:

- Hiện thị giao diện mở đầu và có nút bắt đầu, khi nhấp vào nút bắt đầu sẽ hiển thị ra giao diện thực hiện chức năng chính của chương trình.

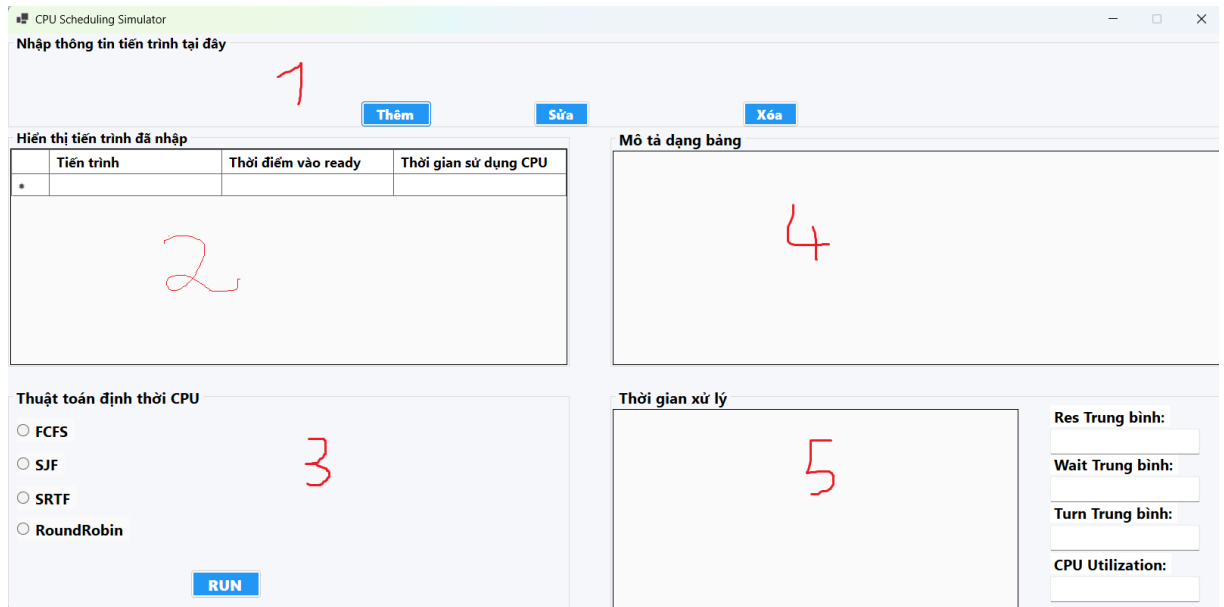


Hình 5.1: giao diện khi bắt đầu chạy chương trình.

5.2. Màn hình giao diện khi nhấp nút “Bắt Đầu”

5.2.1. các chức năng chính

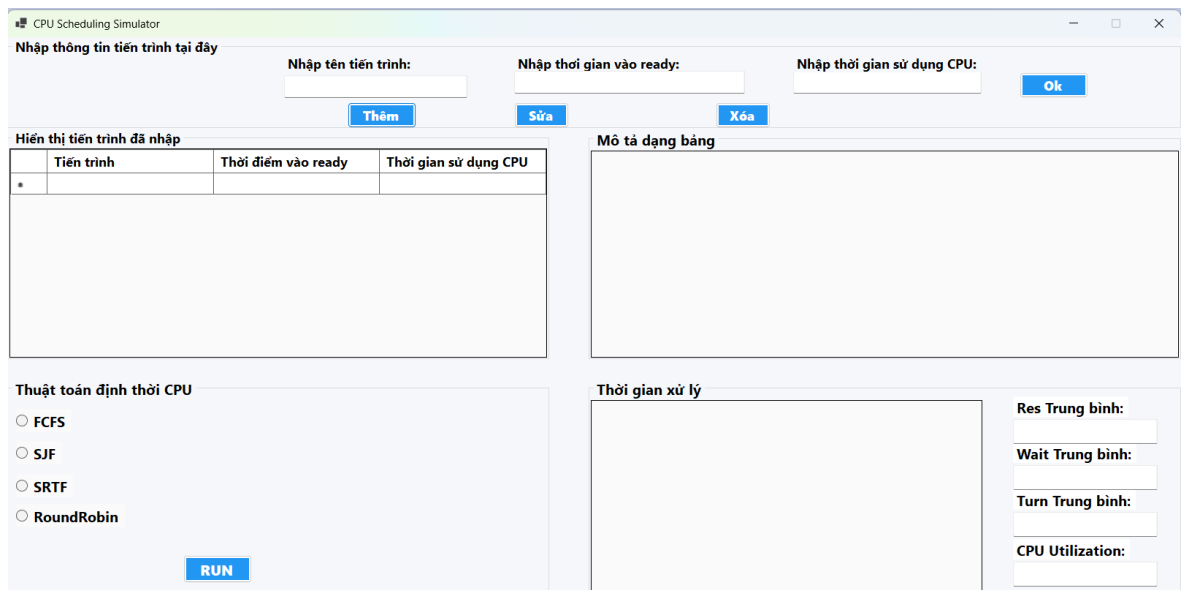
- Hiện thị ra giao diện cho người dùng thêm, sửa xóa thông tin tiến trình.
- Hiện thị thông tin đã nhập dưới dạng bảng.
- Cho người dùng chọn thuật toán muốn chạy.
- Hiện thị các đoạn thời gian (thời điểm vào và ra) khi tiến trình sử dụng CPU.
- Hiện thị bảng các thời gian đợi, thời gian xoay vòng, thời đáp ứng,
- Hiện thị thời gian sử dụng trung bình, thời gian đáp ứng trung bình, thời gian xoay vòng trung bình, hiệu suất CPU.



Hình 5.2: giao diện chương trình khi nhấp vào nút “BẮT ĐẦU” trong hình (4.4).

5.2.2. mô tả (Hình 5.2):

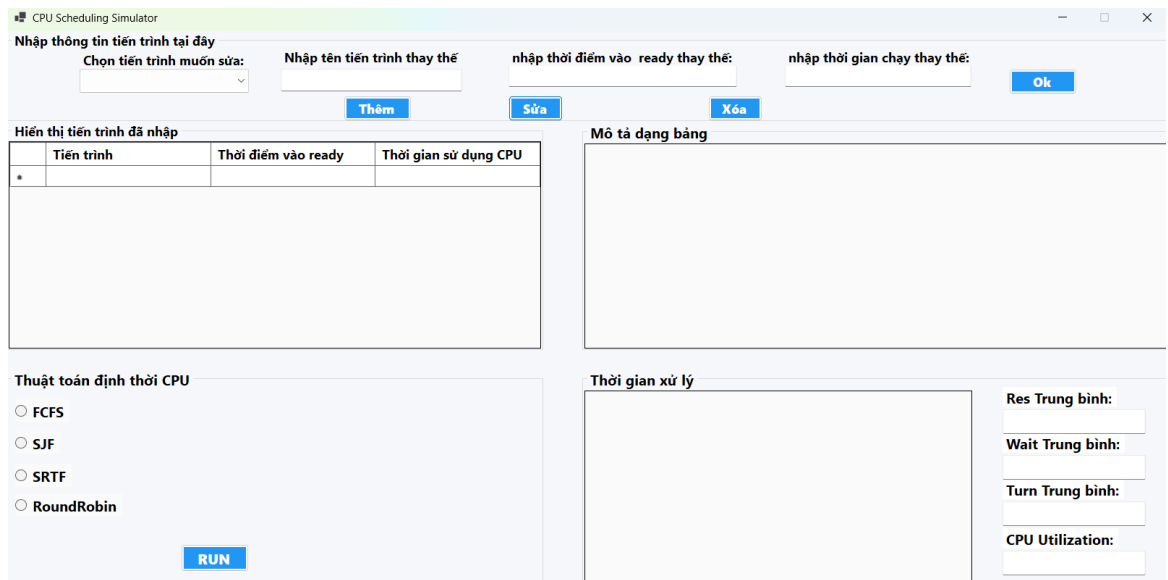
- (1) Dùng để thêm thông tin của tiến trình như tên tiến trình, thời gian vào hàng đợi, thời gian thực thi CPU (khi nhấp vào nút thêm), thay đổi thông tin của tiến trình đã nhập (khi nhấp nút sửa), xóa thông tin nút khi chọn vào góc trái của bảng hiển thị thông tin đã nhập.



Hình 5.3: giao diện khi nhấp vào nút thêm.

- Trong (Hình 5.3) khi nhập đầy đủ thông tin và thỏa ràng buộc (thời gian vào hàng đợi ≥ 0 , thời gian thực thi CPU > 0) rồi nhấp nút “Ok” thì mới

thêm thông tin tiến trình đó vào bảng hiển thị thông tin đã nhập.



CPU Scheduling Simulator

Nhập thông tin tiến trình tại đây

Chọn tiến trình muốn sửa: Nhập tên tiến trình thay thế: nhập thời điểm vào ready thay thế: nhập thời gian chạy thay thế: **Ok**

Thêm **Sửa** **Xóa**

Hiển thị tiến trình đã nhập

Tiến trình	Thời điểm vào ready	Thời gian sử dụng CPU
*		

Mô tả dạng bảng

Thuật toán định thời CPU

☐ FCFS
☐ SJF
☐ SRTF
☐ RoundRobin

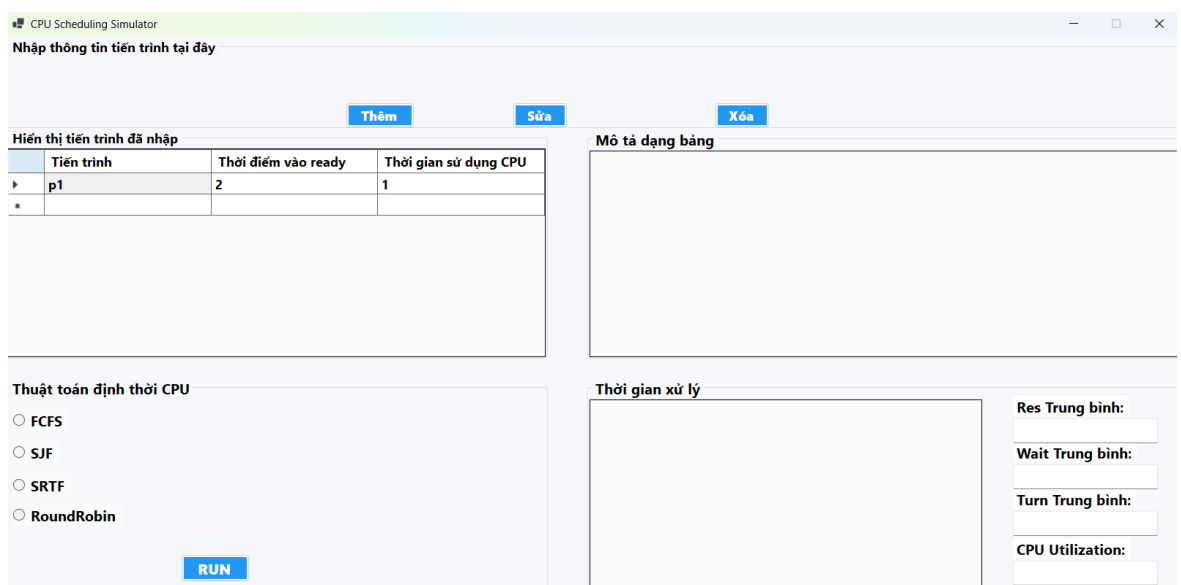
RUN

Thời gian xử lý

Res Trung bình:
 Wait Trung bình:
 Turn Trung bình:
 CPU Utilization:

Hình 5.4: giao diện khi nhấp vào nút Sửa.

- Trong (hình 5.4) khi nhấp vào sửa thì sẽ hiển thị ra giao diện cho người dùng lựa chọn tên tiến trình cần sửa và nhập đầy đủ thông tin của tiến trình, thỏa ràng buộc (thời gian vào hàng đợi ≥ 0 , thời gian thực thi CPU > 0) rồi nhấp nút “Ok” thì mới sửa thông tin của tiến trình.
- (2) dùng để hiển thị thông tin các tiến đã nhập.



CPU Scheduling Simulator

Nhập thông tin tiến trình tại đây

Thêm **Sửa** **Xóa**

Hiển thị tiến trình đã nhập

Tiến trình	Thời điểm vào ready	Thời gian sử dụng CPU
p1	2	1
*		

Mô tả dạng bảng

Thuật toán định thời CPU

☐ FCFS
☐ SJF
☐ SRTF
☐ RoundRobin

RUN

Thời gian xử lý

Res Trung bình:
 Wait Trung bình:
 Turn Trung bình:
 CPU Utilization:

Hình 5.5: hiển thị thông tin tiến trình người dùng đã nhập lên bảng “hiển thị người dùng đã nhập”.

- (3) cho phép người dùng chọn thuật toán vào nhấp vào nút “Run” thì mới

chạy

- (4) hiển thị các đoạn thời gian tiến trình chạy lên bảng “mô tả dạng bảng”,
- (5) hiển thị thời gian đợi, thời gian xoay vòng, thời gian đáp ứng lên bảng Thời gian xử lý, và thời gian đợi trung bình, thời gian xoay vòng trung bình, thời gian đáp ứng trung bình, hiệu suất sử dụng CPU (khi người dùng đã chọn thuật toán và nhấn nút “Run”).

The screenshot shows the 'CPU Scheduling Simulator' window. It has a title bar with standard window controls. Below the title bar is a text input field labeled 'Nhập thông tin tiến trình tại đây' with buttons 'Thêm', 'Sửa', and 'Xóa' above it. The main area is divided into several sections:

- Hiển thị tiến trình đã nhập:** A table with 3 columns: 'Tiến trình', 'Thời điểm vào ready', and 'Thời gian sử dụng CPU'. It contains 4 rows of data.
- Mô tả dạng bảng:** A table with 3 columns: 'Tiến trình', 'Thời gian bắt đầu', and 'Thời gian vào lại ready'. It contains 6 rows of data.
- Thuật toán định thời CPU:** Radio buttons for FCFS, SJF, SRTF, and RoundRobin (selected). A 'Quantum' input field is set to 2. A 'RUN' button is at the bottom.
- Thời gian xử lý:** A table with 4 columns: 'Tiến trình', 'Response Time', 'Turnaround Time', and 'Waiting Time'. It contains 4 rows of data.
- Summary Metrics:** A box on the right showing 'Res Trung bình: 2', 'Wait Trung bình: 12.75', 'Turn Trung bình: 19.25', and 'CPU Utilization: 100%'.

Hình 5.6: hiển thị thông tin các đoạn thời gian, thời gian xử lý, thời gian xử lý trung bình, hiệu suất CPU (khi chọn thuật toán và nhấn nút Run).

5.3. Hướng dẫn mở chương trình

5.3.1. Cài đặt Visual Studio

- Truy cập: [link tải Visual Studio](#).
- Chọn Visual Studio 2022.

5.3.2. Sau khi tải Visual Studio 2022

- Khi cài đặt Visual Studio, bạn cần chọn workload sau: .NET desktop development.
- Sau khi đã cài .NET desktop development chọn dự án và chạy.

KẾT LUẬN VÀ KIẾN NGHỊ

Mục tiêu thực hiện được:

Trình bày được các quá trình hình thành lịch sử của hệ điều hành Android, viết được chương mô tả được các thuật toán định thời CPU, hiển thị được từng đoạn xử lý tiến trình dưới dạng (thay thế cho giản đồ Gant), thời gian xử lý như thời gian đợi, đợi trung bình, thời gian đáp ứng, đáp ứng trung bình, thời gian xoay vòng, xoay vòng trung bình, hiệu Suất sử dụng CPU

Mục tiêu chưa thực hiện được

- Chương trình mô tả chưa xử lý được đầu vào có thời điểm, thời gian I/O, chưa hiển thị được Giản đồ Gant (nhưng thay th bằng bảng đoạn thời gian).
- **Lí do:** đây là khoảng thời gian thi khá bận rộn và không có nhiều thời gian nên nhóm đã làm ra sản phẩm chưa chuẩn lắm, nhóm mong thầy thông cảm.

Hướng phát triển của ứng dụng

- Thêm chức năng xử lý đầu vào có thời gian điểm, thời gian I/O.
- Hiển thị Giản đồ Gant để trực quan hơn.
- Làm giao diện dễ nhìn, bắt mắt hơn.

HƯỚNG DẪN CÀI ĐẶT VÀ TRIỂN KHAI CHƯƠNG TRÌNH

Cài đặt Visual Studio

- Truy cập: [link tải Visual Studio](#).
- Chọn Visual Studio 2022.

Sau khi tải Visual Studio 2022

- Khi cài đặt Visual Studio, bạn cần chọn workload sau: .NET desktop development.
- Sau khi đã cài .NET desktop development chọn dự án và chạy.

TÀI LIỆU THAM KHẢO

Website: [1] OpenAI. “” Introductin ChatGPt.”. Truy cập tại openai.com.