A white rectangular frame with black and blue designs

Description automatically generated**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG THƯƠNG TP. HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**\_\_o0o\_\_**

A logo of a book

Description automatically generated

**TIỂU LUẬN HỌC PHẦN: HỆ ĐIỀU HÀNH**

**ĐỀ TÀI: SEMAPHORE -**

**BÀI TOÁN BỮA ĂN TỐI CỦA CÁC TRIẾT GIA**

**Nhóm: STHG**

**TP.HCM, ngày 5 tháng 10 năm 2024**

A white rectangular frame with black and blue designs

Description automatically generated**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG THƯƠNG TP. HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**\_\_o0o\_\_**

A logo of a book

Description automatically generated

**ĐỀ TÀI 5 : SEMAPHORE – BÀI TOÁN BỮA ĂN TỐI CỦA CÁC TRIẾT GIA**

**NHÓM THỰC HIỆN:** STHG **GVHD:** Phạm Tuần Khiêm

**Trưởng nhóm:** Nguyễn Tấn Sang **Lớp danh nghĩa:** 14DHBM03

**Thành viên: TKB chính thức:** Thứ 7, tiết 4 – 6

1. Tô Trương Trường Thành
2. Vũ Trường Giang
3. Phạm Đình Minh Hiếu

**TP.HCM, Ngày 5 tháng 10 năm 2024**

LỜI CẢM ƠN

Chúng em xin chân thành cảm ơn thầy Phạm Tuấn Khiêm đã tận tình hướng dẫn chúng em trong suốt quá trình thực hiện bài tập nhóm này. Nhờ sự nhiệt tình, tâm huyết và những góp ý quý báu của thầy, chúng em đã hoàn thành bài tiểu luận một cách tốt nhất. Chúng em xin ghi nhận sự quan tâm, động viên và những kiến thức mà thầy đã truyền đạt cho chúng em trong suốt quá trình học tập.

Chúng em xin trân trọng cảm ơn Trường Đại Học Công Thương TP.HCM đã tạo điều kiện cho chúng em được học tập và nghiên cứu trong môi trường giáo dục tiên tiến, hiện đại.

Chúng em cũng xin cảm ơn các cán bộ, nhân viên của trường đã luôn hỗ trợ chúng em trong quá trình học tập. Chúng em xin gửi lời cảm ơn chân thành đến Khoa Công Nghệ thông Tin và các thầy cô trong khoa đã luôn quan tâm, dìu dắt chúng em trong suốt quá trình học tập. Chúng em xin ghi nhận những kiến thức chuyên môn và kỹ năng mà các thầy cô đã truyền đạt cho chúng em.

LỜI CAM ĐOAN

Chúng em xin cam đoan đề tài tiểu luận: Semaphore – Bài Toán Bữa Ăn Tối Của Các Triết gia nhóm STHG nghiên cứu và thực hiện.

Chúng em đã kiểm tra dữ liệu theo quy định hiện hành.

Kết quả bài làm của đề tài Semaphore – Bài Toán Bữa Ăn Tối Của Các Triết Gia là trung thực và không sao chép từ bất kỳ bài tập của nhóm khác.

Các tài liệu được sử dụng trong tiểu luận có nguồn gốc, xuất xứ rõ ràng.

**BẢNG ĐÁNH GIÁ KẾT QUẢ THỰC HIỆN CÔNG VIỆC NHÓM STHG**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| STT | Họ và tên | Công việc đảm nhận | Cá nhân tự đánh giá kết quả | Nhóm đánh giá kết quả | GV đánh giá |
| 1 | Nguyễn Tấn Sang | Thiết kế web, Thuật toán Sleep and Wakeup | Hoàn thành tốt, đúng hạn | Hoàn thành 100% công việc |  |
| 2 | Tô Trường Trường Thành | Code ứng dụng Semaphore – Bài Toán Triết Gia | Hoàn thành tốt, đúng hạn | Hoàn thành 100% công việc |  |
| 3 | Vũ Trường Giang | Semaphore | Hoàn thành tốt, đúng hạn | Hoàn thành 100% công việc |  |
| 4 | Phạm Đình Minh Hiếu | Đồng bộ hoá tiến trình | Hoàn thành tốt, đúng hạn | Hoàn thành 100% công việc |  |

# PHẦN MỤC LỤC

[PHẦN MỤC LỤC v](#_Toc179939619)

[PHẦN MỞ ĐẦU 1](#_Toc179939620)

[1. Lí do chọn đề tài 1](#_Toc179939621)

[2. Mục đích nghiên cứu 1](#_Toc179939622)

[3. Đối tượng nghiên cứu 1](#_Toc179939623)

[4. Phạm vi nghiên cứu 2](#_Toc179939624)

[5. Phương nghiên cứu 2](#_Toc179939625)

[6. Ý nghĩa lý luận và thực tiễn của đề tài 2](#_Toc179939626)

[7. Kết cấu của bài tiểu luận 2](#_Toc179939627)

[PHẦN NỘI DUNG 3](#_Toc179939628)

[1. Bài toán 3](#_Toc179939629)

[1.1. Đề tài 3](#_Toc179939630)

[1.2. Mô tả vấn đề 3](#_Toc179939631)

[1.3. Yêu cầu bài toán 3](#_Toc179939632)

[2. Cơ sở lí thuyết 3](#_Toc179939633)

[2.1. Đồng bộ hoá tiến trình 4](#_Toc179939634)

[*2.1.1. Đồng bộ tiến trình là gì ? 4*](#_Toc179939635)

[*2.1.2. Nhu cầu giao tiếp của bài toán đồng bộ hoá 4*](#_Toc179939636)

[2.2. Tài nguyên găng và đoạn găng 5](#_Toc179939637)

[*2.2.1. Tài nguyên găng (Critical Resource) 5*](#_Toc179939638)

[*2.2.2. Miền găng (Critical Section) 5*](#_Toc179939639)

[*2.2.3. Yêu cầu đối với đoạn găng 5*](#_Toc179939640)

[2.3. Giải thuật Sleep and Wakeup 6](#_Toc179939641)

[*2.3.1. Sleep and Wakeup 6*](#_Toc179939642)

[*2.3.2. Vấn đề của giải thuật Sleep and Wakeup 6*](#_Toc179939643)

[2.4. Semaphore 7](#_Toc179939644)

[*2.4.1. Khái niệm 7*](#_Toc179939645)

[*2.4.2. Cơ chế của Semaphore 7*](#_Toc179939646)

[*2.4.3. Phân loại Semaphore 8*](#_Toc179939647)

[3. Cách giải quyết bài toán 9](#_Toc179939648)

[3.1. Quản lí miền găng 9](#_Toc179939649)

[3.2. Chương trình thực hiện chính 10](#_Toc179939650)

[3.3. Kết quả chương trình 12](#_Toc179939651)

[4. Demo web 13](#_Toc179939652)

[4.1. Bàn ăn 13](#_Toc179939653)

[4.2. Triết Gia 15](#_Toc179939654)

[4.3. Kết quả thực hiện 15](#_Toc179939655)

[PHẦN KẾT LUẬN 17](#_Toc179939656)

[PHẦN TÀI LIỆU THAM KHẢO 18](#_Toc179939657)

# PHẦN MỞ ĐẦU

# Lí do chọn đề tài

Trong thời đại công nghệ hiện nay, khi các hệ thống đa nhiệm và đa luồng ngày càng phát triển, việc quản lý tài nguyên dùng chung giữa các tiến trình trở thành một thách thức quan trọng. Các vấn đề như xung đột tài nguyên, bế tắc hay tình trạng chờ đợi vô tận có thể gây ảnh hưởng nghiêm trọng đến hiệu suất và tính ổn định của hệ thống.

Bài toán triết gia ăn tối là một mô hình lý thuyết nổi bật, minh họa rõ nét các vấn đề này. Nó không chỉ giúp người đọc dễ dàng nhận thấy những thách thức trong việc đồng bộ hóa tài nguyên mà còn làm sáng tỏ vai trò quan trọng của Semaphore, một cơ chế đồng bộ mạnh mẽ. Semaphore cho phép các tiến trình phối hợp với nhau, kiểm soát việc truy cập tài nguyên một cách hiệu quả, đảm bảo không xảy ra bế tắc hay xung đột.

Chính vì vậy, chúng em chọn nghiên cứu đề tài này để khám phá sâu hơn về cách Semaphore được áp dụng trong thực tế để giải quyết các vấn đề đồng bộ hóa, từ đó đóng góp kiến thức vào lĩnh vực phát triển hệ thống đa luồng, giúp đảm bảo sự ổn định và hiệu suất của các hệ thống hiện đại.

# Mục đích nghiên cứu

Nghiên cứu này nhằm hiểu rõ cách thức các cơ chế đồng bộ hóa, đặc biệt là Semaphore, ứng dụng nó để giải quyết vấn đề phân chia tài nguyên giữa các tiến trình mà không gây ra xung đột hoặc bế tắc trong bài toán ‘Bữa ăn tối của các triết gia’.

# Đối tượng nghiên cứu

Phương pháp đồng bộ hoá trong lập trình đa luồng với trọng tâm là Semaphore và áp dụng nó để giải quyết bài toán ‘Bữa ăn tối của các triết gia’.

# Phạm vi nghiên cứu

1. Không gian: Trường Đại Học Công Thương TP. Hồ Chí Minh.
2. Thời gian: Tháng 9 năm 2024.

# Phương nghiên cứu

1. Nghiên cứu và đọc hiểu tài liệu
2. Khái quát hóa kiến thức.

# Ý nghĩa lý luận và thực tiễn của đề tài

Về mặt lý luận đề tài giúp hiểu sâu hơn về các cơ chế đồng bộ hoá, đặc biệt là Semaphore, một mô hình quan trọng trong việc quản lý tài nguyên giữa các tiến trình trong lập trình đa luồng. Việc nghiên cứu bài toán ‘Bữa ăn tối của các triết gia’ giúp chúng em nắm vững lý thuyết về quản lý tài nguyên và cách ngăn ngừa các vấn đề như xung đột hay bế tắc, những thách thức khi triển khai các hệ thống, các ứng dụng thời gian thực (Real-time).

Về mặt thực tiễn, việc hiện thực bài toán bằng C/C++ và mô phỏng trực quan bằng Angular đã giúp chúng em kết nối lý thuyết với thực hành. Thông qua việc tự tay lập trình và quan sát quá trình các triết gia ăn và suy nghĩ, chúng em có cái nhìn cụ thể hơn về cách đồng bộ hóa tài nguyên trong các hệ thống đa nhiệm. Điều này không chỉ giúp chúng em hiểu rõ hơn về semaphore mà còn rèn luyện kỹ năng giải quyết các vấn đề đồng bộ hóa trong các dự án thực tế mà chúng em có thể gặp trong công việc sau này.

# Kết cấu của bài tiểu luận

Phần 1: MỤC LỤC

Phần 2: PHẦN MỞ ĐẦU

Phần 3: PHẦN NỘI DUNG

Phần 4: PHẦN KẾT LUẬN

Phần 5: TÀI LIỆU THAM KHẢO

# PHẦN NỘI DUNG

# Bài toán

## Đề tài

Tìm hiểu phương pháp đồng bộ hoá tiến trình Semaphore – Viết chương trình minh hoạ bài toán bữa ăn tối của các triết gia.

## Mô tả vấn đề

Vấn đề này được thiết kế để minh họa những thách thức trong việc tránh tình trạng deadlock, một trạng thái của hệ thống mà trong đó không có tiến triển nào có thể xảy ra. Để thấy rằng một giải pháp thích hợp cho vấn đề này không phải là hiển nhiên, hãy xem xét một đề xuất trong đó mỗi triết gia được hướng dẫn hành động như sau: suy nghĩ trừ khi nĩa bên trái có sẵn; khi có, nhấc nó lên; suy nghĩ trừ khi nĩa bên phải có sẵn; khi có, nhấc nó lên; khi cả hai nĩa đã được giữ, ăn trong một khoảng thời gian cố định; đặt nĩa bên trái xuống; đặt nĩa bên phải xuống; lặp lại từ đầu. Với các hướng dẫn này, tình huống có thể phát sinh khi mỗi triết gia giữ nĩa bên trái của mình; trong tình huống đó, họ sẽ bị kẹt mãi mãi, chờ đợi nĩa khác có sẵn: đó là deadlock.

## Yêu cầu bài toán

**Tiến trình:** Số triết gia trên bàn ăn

**Hành động:** ăn, suy nghĩ

**Tài nguyên cần:** Mỗi triết gia cần 2 chiếc đũa

**Tắc nghẽn:** Các triết gia phải đợi lẫn nhau dẫn đến không ai được ăn

**Đói:** Một triết gia có thể chết đói nếu ông ta không có cách nào ăn được

# Cơ sở lí thuyết

## Đồng bộ hoá tiến trình

### Đồng bộ tiến trình là gì ?

Đồng bộ tiến trình là phương pháp đảm bảo nhiều tiến trình hoặc luồng (thread) hoạt động cùng nhau một cách hợp lí mà không gây xung đột hoặc lỗi. Và việc đồng bộ hoá này cần thực hiện hai yêu cầu:

* **Yêu cầu độc quyền truy xuất(Mutal exlution):** Tại một thời điểm, chỉ có một tiến trình được truy xuất một tài nguyên. Điều này có nghĩa là một tài nguyên không thể được chia sẻ đồng thời bởi nhiều tiến trình để tránh xung đột hoặc lỗi.
* **Yêu cầu phối hợp(synchronization):** các tiến trình cần hợp tác với nhau để hoàn thành công việc một cách hiệu quả. Đồng bộ hoá giúp các tiến trình phối hợp mượt mà hơn, đặc biệt là khi nhiều tiến trình cần chia sẻ tài nguyên chung như bộ nhớ hoặc dữ liệu.

### Nhu cầu giao tiếp của bài toán đồng bộ hoá

* **Mutual exclusion** (độc quyền truy xuất):

Tài nguyên hệ thống có hai loại:

**Tài nguyên có thể chia sẻ được:** Những tài nguyên mà nhiều tiến trình có thể truy cập và sử dụng đồng thời mà không gây ra xung đột hay lỗi, ví dụ như tập tin chỉ đọc.

**Tài nguyên không thể chia sẻ được:** Những tài nguyên chỉ có thể được sử dụng bởi một tiến trình tại một thời điểm, ví dụ như máy in.

* **Coordination** (Yêu cầu phối hợp):

Có một vài tiến trình cần phải thực hiện theo một trình tự nhất định. Ví dụ, khi một tiến trình cần dữ liệu một tiến trình khác thì chúng cần phối hợp với nhau để dữ liệu được truyền đúng cách và đúng thời điểm.

## Tài nguyên găng và đoạn găng

### Tài nguyên găng (Critical Resource)

Tài nguyên găng là những tài nguyên không thể truy cập đông thời bởi nhiều tiến trình mà không gây ra lỗi hay xung đột. Ví dụ: Số dư trong tài khoản ngân hàng, điểm trung bình tích luỹ của sinh viên trong một kì,...

### Miền găng (Critical Section)

Miền găng là đoạn mã mà khi có một tiến trình thực thi thì không tiến trình nào khác được phép truy cập vào phần mã đó cùng lúc để tránh xung đột. Ví dụ đoạn mã sau:

A screenshot of a computer

Description automatically generated

Hình 2.2‑1 Minh hoạ cho bài toán rút tiền

Đoạn mã từ (3) tới (6) đang nằm trong miền găng, tính toán, cập nhật cho tiền\_trong\_tài\_khoản(tài nguyên găng), nếu không xử lí đồng bộ thì sẽ vi phạm một trong các tiêu chí, yêu cầu đối với miền găng sau đây.

### Yêu cầu đối với đoạn găng

Có 4 tiêu chí:

Không có hai tiến trình hoặc nhiều hơn nữa tiến trình mà có thể tham gia vào miền găng cùng một lúc.

Dù cho hệ thống có tốc độ CPU nhanh hay số lượng core nhiều thì vẫn không ảnh hưởng tới việc thực thi.

Một tiến trình ở bên ngoài Miền Găng không được phép chặn các tiến trình khác truy xuất vào Miền Găng.

Không có tiến trình nào đợi vô hạn.

## Giải thuật Sleep and Wakeup

### Sleep and Wakeup

Một trong những cách để giải quyết bài toán đồng bộ hoá đó chính là giải thuật ‘*Sleep and Wakeup’.* Giải thuật này hoạt động bằng cách cho phép các tiến trình tạm dừng (Sleep) khi không thể tiếp tục thực thi do thiếu tài nguyên hoặc điều kiện chưa được đáp ứng, và sau đó đánh thức (Wakeup) các tiến trình này khi điều kiện được thoả mãn.

### Vấn đề của giải thuật Sleep and Wakeup

Mặc dù giải thuật ‘*Sleep and Wakeup’* có thể giải quyết một số bài toán đồng bộ hoá, nhưng nó cũng gặp phải những vấn đề nhất định. Một trong những vấn đề lớn nhất là race conditions(điều kiện tranh đua). Khi các tiến trình không được đồng bộ hoá tốt, chúng có thể truy cập và thay đổi tài nguyên đồng thời, dẫn đến các lỗi không mong muốn. Ngoài ra, giải thuật này cũng có thể gây ra tình trạng deadlock khi các tiến trình chờ nhau để được đánh thức, dẫn đến việc hệ thống không thể tiếp tục hoạt động.

Do những hạn chế này, người ta đã phát triển Semaphore để khắc phực các vấn đề của giải thuật ‘*Sleep and Wakeup’.* Semaphore là một cơ chế đồng bộ hoá mạnh mẽ hơn, giúp kiểm soát truy cập tới tài nguyên bằng cách sử dụng các tín hiệu (signals). Nó đảm bảo rằng chỉ có một số lượng nhất định tiến trình có thể truy cập vào tài nguyên động thời ngăn ngừa tình trạng race conditions và deadlock.

## Semaphore

### Khái niệm

1. **Định nghĩa:**

Semaphore là một biến hoặc một cấu trúc dữ liệu được sử dụng để điều khiển việc truy cập vào tài nguyên dùng chung trong môi trường có nhiều luồng (threads) hoặc tiến trình (processes). Nó là một trong những phương pháp đồng bộ hóa phổ biến nhất, giúp ngăn chặn các vấn đề như race condition (xung đột luồng) và deadlock (tắc nghẽn).

Semaphore có thể được hiểu như một bộ đếm (counter) cho tài nguyên. Nếu tài nguyên sẵn có, giá trị của semaphore sẽ lớn hơn 0. Nếu tài nguyên đã hết (không còn sẵn sàng), giá trị của semaphore bằng 0 và các tiến trình yêu cầu tài nguyên sẽ phải chờ.

1. **Lịch sử:**

Semaphore được phát minh bởi nhà khoa học máy tính Edsger W. Dijkstra vào những năm 1960 để quản lý các vấn đề đồng bộ hóa trong hệ điều hành. Cụ thể, Dijkstra sử dụng semaphore để giải quyết vấn đề đồng bộ hóa giữa các tiến trình trong hệ điều hành MULTICS. Ý tưởng về semaphore sau đó đã trở thành nền tảng cho nhiều cơ chế đồng bộ hóa trong lập trình hệ thống và đa luồng.

### Cơ chế của Semaphore

Semaphore cung cấp hai phép toán cơ bản:

1. **Wait** (hoặc P operation):

Khi một tiến trình muốn truy cập tài nguyên, nó sẽ thực hiện phép toán wait để kiểm tra giá trị của semaphore.

Nếu giá trị của Semaphore lớn hơn 0, Semaphore sẽ giảm đi 1, và tiến trình có thể truy cập tài nguyên.

Nếu giá trị của Semaphore bằng 0, tiến trình sẽ phải chờ cho đến khi tài nguyên được giải phóng.

1. **Signal** (hoặc V operation):

Khi một tiến trình hoàn thành việc sử dụng tài nguyên, nó sẽ thực hiện phép toán signal để tăng giá trị semaphore lên 1, báo hiệu rằng tài nguyên đã sẵn sàng cho tiến trình khác sử dụng.

### Phân loại Semaphore

Semaphore có hai loại chính:

1. **Binary Semaphore** (Semaphore nhị phân):

**Giá trị:** 0 hoặc 1.

**Mục đích:** Dùng để kiểm soát việc truy cập các tài nguyên chỉ có một đơn vị, tương tự như một khóa mutex (mutual exclusion).

**Hoạt động:** Nếu giá trị của semaphore là 1, tài nguyên sẵn sàng; nếu là 0, tài nguyên đang bị chiếm dụng.

**Ví dụ:** Điều khiển việc truy cập vào một máy in, khi chỉ có một máy in duy nhất.

1. **Counting Semaphore** (Semaphore đếm):

**Giá trị:** Có thể lớn hơn 1.

**Mục đích:** Dùng để kiểm soát các tài nguyên có nhiều đơn vị. Ví dụ, khi có nhiều tài nguyên giống nhau (như số lượng ghế trong một nhà hàng), semaphore đếm sẽ giúp quản lý số lượng tài nguyên còn lại.

**Hoạt động:** Giá trị của semaphore đếm thể hiện số lượng tài nguyên sẵn có. Mỗi khi một tài nguyên được sử dụng, giá trị semaphore sẽ giảm đi, và khi một tài nguyên được giải phóng, giá trị semaphore sẽ tăng lên.

# Cách giải quyết bài toán

## Quản lí miền găng

Chương trình xem mỗi *philosophers* là một tiến trình, *spoons* là tài nguyên dùng chung cần được quản lí.

* Hàm **sem\_wait** (tương đương với Wait):

Được dùng để quản lý các tài nguyên dùng chung.

Khi một tiến trình muốn truy cập tài nguyên, nó sẽ gọi sem\_wait. Nếu tài nguyên có sẵn, tiến trình sẽ được phép truy cập. Nếu không, tiến trình sẽ bị đưa vào hàng đợi hay dễ hiểu là bị chặn cho đến khi tài nguyên có sẵn.

* Hàm **sem\_post** (tương đương với Signal):

Được dùng để báo hiệu rằng tài nguyên sẽ được giải phóng và có sẵn để tiến trình khác truy cập

Trạng thái của *spoons*

* + Mỗi spoons có hai trang thái:

*+ spoons* = 1: tài nguyên đang có sẵn

*+ spoons* = 0: tài nguyên đang bận hoặc đã có *philosophers* khác sử dụng

void\* philosophers(void\* number) {

int id = \*(int\*)number;

printf("triet gia %d dang nghi...\n", id);

sleep(1);

sem\_wait(&spoons[id]);

sem\_wait(&spoons[(id + 1) % N]);

printf("-- --triet gia %d dang an...\n", id);

sleep(2);

sem\_post(&spoons[id]);

sem\_post(&spoons[(id + 1) % N]);

printf("-> Triet gia % da an xong va quay lai nghi...\n", id);

return NULL;

}

Khi một *philosphers* muốn sử dụng một *spoons,* họ sẽ gọi hàm **sem\_wait.** Nếu *spoons* đang có sẵn (*spoons* = 1), họ sẽ lấy *spoons* và trạng thái của nó sẽ chuyển sang bân (*spoons = 0).* Khi *philosophers* hoàn thành công việc và giải phóng *spoons,* nó sẽ gọi hàm sem\_post để báo hiệu rằng *spoons* đã có sẵn trở lại (*spoons* = 1)

## Chương trình thực hiện chính

int main() {

printf("Nhap so luong Triet gia: ");

scanf("%d", &N);

pthread\_t threads[N]; // Mỗi thread đại diện cho một triết gia

int philosophers\_ids[N]; // Mảng chứa ID của các triết gia

for (int i = 0; i < N; i++) {

sem\_init(&spoons[i], 0, 1); // Khởi tạo từng semaphore với

// giá trị ban đầu là 1

}

for (int i = 0; i < N; i++) {

philosophers\_ids[i] = i;

pthread\_create(&threads[i], NULL, philosophers, &philosophers\_ids[i]);

}

for (int i = 0; i < N; i++) {

pthread\_join(threads[i], NULL); // Chờ mỗi thread kết thúc

}

for (int i = 0; i < N; i++) {

sem\_destroy(&spoons[i]); // Hủy semaphore giải phóng

// tài nguyên

}

return 0;

}

**Pseudocode:**

1. Khởi tạo:

* Semaphore cho các *spoons* được khởi tạo với giá trị 1 (*spoons* có sẵn).
* Các threads được tạo, mỗi thread đại diện cho 1 *philosophers.*

1. Hành động của Triết Gia:

* Mỗi thread của triết gia bắt đầu và chạy hàm *philosophers.*
* Triết gia đầu tiên ‘nghĩ’ (được mô phỏng bằng hàm **Sleep( )**)
* *philosophers* cố gắng lấy hai cái *spoons* bằng cách gọi hàm **sem\_wait( ).**
* Nếu thành công, triết gia sẽ vào trạng thái ‘ăn’
* Sau khi ăn, triết gia sẽ buông và đặt lại giá trị *spoons* bằng cách gọi hàm **sem\_post( ).**
* Quay lại trạng thái ‘nghĩ’.

1. Hoàn thành chương trình:

* Thread chính chờ tất cả các thread của triết gia hoàn thành bằng cách dùng hàm **pthread\_join( ).**
* Khi tất cả các threads hoàn thành, các semaphores bị huỷ để giải phóng tài nguyên.

## Kết quả chương trình

A screenshot of a computer program

Description automatically generated

Hình 3.3‑1 Kết quả Console

|  |  |  |
| --- | --- | --- |
| Triết gia | Trạng thái ban đầu | Hành động đã thực hiện |
| 0 | Ăn | Ăn đầu tiên, ăn xong sau đó quay lại nghĩ |
| 1 | Chờ | bắt đầu Ăn sau khi 0 đã ăn xong |
| 2 | Ăn | bắt đầu Ăn sau 0 một lúc |
| 3 | Chờ | bắt đầu Ăn sau khi 4 Ăn xong |
| 4 | Chờ | bắt đầu Ăn sau khi 2 Ăn xong |

# Demo web

Tại đây người dùng sẽ bấm ‘Get started’ để vào giao diện mô phỏng bàn ăn của 5 Triết Gia.

A group of men sitting at a table

Description automatically generated

Hình 4‑3.3‑1 Giao diện trang Home

## Bàn ăn

Trên bàn ăn sẽ bao gồm 5 chiếc dĩađại diện cho 5 *philosophers*, 5 chiếc đũa đại diện cho 5 *spoons.* Mỗi khi tới lượt ai vào trạng thái *Ăn* chiếc dĩa sẽ đổi sang màu xanh.

A circular object with a number in the center

Description automatically generated

Hình 4.1‑1 Minh hoạ bàn ăn lúc đầu

A screenshot of a computer generated image

Description automatically generated

Hình 4.1‑2 Minh hoạ bàn ăn lúc chạy thuật toán

## Triết Gia

Giao diện của một *philosophers* bao gồm **State** dùng để diễn tả trạng thái của Triết Gia thời điểm hiện tại, **Time** dùng để tính thời gian từ lúc *philosophers* bắt đầu nghĩ cho tới lúc Ăn xong tức cầm được chiếc *spoons* thứ 2.

A person with a beard

Description automatically generated

Hình 4.2‑1 Minh hoạ Triết Gia

## Kết quả thực hiện

A screenshot of a computer

Description automatically generated

Hình 4.3‑1 Kết quả khi chạy mô phỏng thuật toán

# PHẦN KẾT LUẬN

Trong bài tiểu luận này, chúng em đã giải quyết bài toán triết gia ăn tối, một trong những vấn đề tiêu biểu về đồng bộ hóa trong lập trình đa luồng. Dựa trên cơ sở lý thuyết về đồng bộ hóa, tài nguyên găng, sleep and wakeup, chúng em đã áp dụng giải pháp semaphore để đảm bảo việc phân phối tài nguyên một cách an toàn và hiệu quả.

Chúng em đã hiện thực bài toán này trên môi trường C/C++, giúp mô phỏng rõ nét cách mà semaphore quản lý truy cập tài nguyên trong một hệ thống thực tế. Đồng thời, việc sử dụng Angular để mô phỏng trực quan quá trình của từng triết gia ăn và suy nghĩ đã giúp chúng em minh họa sinh động các khái niệm lý thuyết.

Thông qua việc thực hiện bài toán, chúng em nhận ra rằng semaphore không chỉ là một công cụ lý thuyết mạnh mẽ, mà còn là một mô hình đồng bộ hóa có thể được ứng dụng rộng rãi trong các hệ thống phức tạp. Nó giúp đảm bảo tính toàn vẹn của tài nguyên và ngăn chặn các vấn đề như bế tắc, từ đó đóng vai trò thiết yếu trong việc xây dựng các hệ thống đa luồng ổn định và hiệu quả.

# PHẦN TÀI LIỆU THAM KHẢO

[1]Wikipedia.Dining\_philosophers\_problem.<https://en.wikipedia.org/wiki/Dining_philosophers_problem>

[2] Abraham Silberschatz.2018.Operating System Concepts.10th

[3] Andrew S. Tanenbaum, Albert S. Woodhull.2006.Operating Systems Design And Implementation.3rd

[4] Giáo trình Hệ điều hành HUFI