TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

**VIỆN ĐIỆN TỬ - VIỄN THÔNG**



BÁO CÁO BÀI TẬP LỚN

**HỆ ĐIỀU HÀNH**

**Đề tài:**

**Synchronization :Caltrain Automation**

Sinh viên thực hiện:

**Họ và tên : Trần Thế Rinh**

**Lớp: ĐTVT.09 K62**

**Mã lớp: 119074**

Hà Nội, 12-2020

**LỜI NÓI ĐẦU**

Ngày nay, con người cùng những ứng dụng của khoa học kỹ thuật tiên tiến của thế giới, đã làm chúng ta thay đổi ngày càng văn minh và hiện đại hơn. Sự phát triển của kỹ thuật điện tử đã tạo ra hàng loạt những thiết bị với đặc điểm nổi bật cũng như sự chính xác cao, tốc độ nhanh, gọn nhẹ,… là những yếu tố rất cần thiết góp phần cho hoạt động của con người đạt hiệu quả ngày càng cao hơn.Đặc biết ngày nay sự bùng nổ phát triển IOT các thiết bị ngành càng thông minh hơn, tự động hóa cao hơn.Và điều tiên quyết dẫn đến những sự phát triển lớn mạnh đó phải kể đến vai trò của hệ điều hành nó cái vai trò tiên quyết giúp các thiết bị thông minh ngày nay thông minh hơn mạnh mẽ hơn nhiều lần bên cạnh sự phát triển của phần cứng .Một số hệ điều hành tiêu biểu chúng ta có kể đến như Microsoft , MacOS, Linux, … Đặc biệt là Linux , nó cùng với các bản phối xuất hiện trong rất nhiều hệ thống. Vài năm qua, Linux đã thực sự tạo ra một cuộc cách mạng trong lĩnh vực máy tính. Sự phát triển và những gì chúng mang lại cho máy tính thật đáng kinh ngạc: một hệ điều hành đa nhiệm, đa người dùng. Linux có thể chạy trên nhiều bộ vi xử lý khác nhau như: Intel, Motorola, MC68K, Dec Alpha. Nó tương tác tốt với các hệ điều hành:Apple, Mcrosoft và Novell. Không phải ngẫu nhiên mà ngành công nghê thông tin Việt Nam trong Linux làm hệ điều hành nền cho các chương trình ứng dụng chủ đạo về kinh tế và quốc phòng. Với mã nguồn mở, sử dụng Linux an toàn hơn các ứng dụng Windows. Linux đem đến cho chúng ta lợi ích về kinh tế với rất nhiều phần mềm miễn phí. Mã nguồn mở của hệ điều hành và cac chương trình trên Linux là tài liệu vô giá để chúng ta học hỏi về kỹ thuật lập trình vốn là những tài liệu không được công bố đối với các ứng dụng Windows hay MacOS. Trong dự án lần này chúng ta sẽ tìm hiểu một một trong những vấn đề quan trong của hệ điều hành đó là sysnchronization(đồng bộ hóa) thông qua dự án Caltrain Automation. Một hệ điều hành muốn chạy ổn định và không xảy ra xung đột giữa các process , thread thì bài toán về đồng bộ hóa lại càng tỏ ra quan trọng.

Trong quá trình tìm hiểu giải quyết dự án tuy có nhiều cố gắng nhưng không thể tránh được những sai sót, nhóm em hy vọng nhận được những nhận xét của thầy để đề tài được hoàn thành tốt hơn.

**Mục lục**

1. Cơ sở lý thuyết…………………………………………………………...3

1.Tổng quan về Process và Thread……………………………………………3

1.1 Khái niệm về vệ điều hành……………………………………………...3

1.2 Process ………………………………………………………..…….......4

1.3 Thread …………………………………………………………………..6

2.Các vấn đề về Sysnchronization (đồng bộ hóa)…………………………….7

2.1 Nhu cầu đồng bộ hóa……………………………………………………7

2.2 Các giải pháp đồng bộ hóa……………………………………………...8

2.3 Deadlock……………………………………………………………….11

1. Triển khai thực hiện đề tài………………………………………………12

1. Tổng quan đề tài………………………………………………………..12

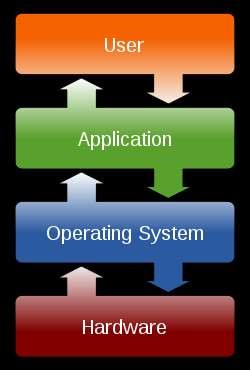
1. Phân tích yêu cầu đề tài………………………………………………..13
2. Triển khai thực hiện……………………………………………………14
3. Kết quả…………………………………………………………………16

III. Tổng kết đánh giá………………………………………………………16

IV. Danh mục tài liệu tham khảo…………………………………………..17

1. **Cơ sở lý thuyết**
2. **Tổng quan về Process và Thread**
   1. **Khái niệm về hệ điều hành**

Hệ điều hànhlà một chương trình hay một hệ chương trình hoạt động giữa người sử dụng (user) và phần cứng của máy tính. Mục tiêu của hệ điều hành là cung cấp một môi trường để người sử dụng có thể thi hành các chương trình. Nó làm cho máy tính dể sử dụng hơn, thuận lợi hơn và hiệu quả hơn.



Hệ điều hành là một phần quan trọng của hầu hết các hệ thống máy tính. Một hệ thống máy tính thường được chia làm bốn phần chính : phần cứng, hệ điều hành, các chương trình ứng dụng và người sử dụng.

Mục đích:

* Giúp người sử dụng dễ dàng sử dụng hệ thống
* Quản lý và cấp phát tài nguyên hệ thống 1 cách hiệu quả.
  1. **Process**

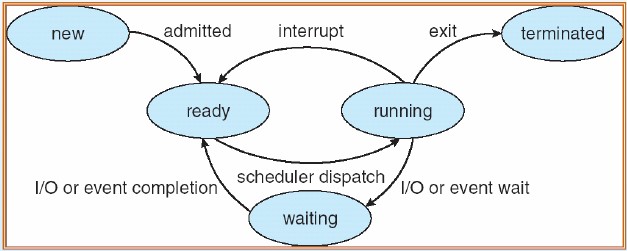
**1.2.1 Khái niệm**

- Process (tiến trình): một chương trình đang thực hiện

* + Mã lệnh chương trình có thể thực thi
  + Dữ liệu của chương trình
  + Ngăn xếp, con trỏ ngăn xếp, các thanh ghi
  + Các thông tin cần thiết cho việc thực thi chương trình

- Tiến trình >< chương trình

* Chương trình: thực thể thụ động, chứa đựng các chỉ thị điều khiển máy tính thực hiện một nhiệm vụ nào đó .
* Tiến trình : trạng thái động của chương trình.
* Hệ thống multiprocess(đa tiến trình) phân chia thời gian:
* HĐH theo chu kỳ :dừng một tiến trình để bắt đầu một tt khác
* Cần lưu thông tin các tt ⇒ Bảng tiến trình (process table)
* Một tiến trình có thể khởi tạo tiến trình khác
* Shell của HĐH kích hoạt một tiến trình thực hiện lệnh; thực hiện xong, hủy tiến trình mới được kích hoạt
* Các tiến trình có thể trao đổi thông tin với nhau
* Một process (tiến trình) có thế bao gồm nhiều thread
* Trạng thái của process
  + Khi thực hiện, tiến trình thay đổi trạng thái
  + Trạng thái của tiến trình là một phần trong hoạt động hiện tại của tiến trình:
    - Khởi tạo (New) Tiến trình đang được khởi tạo
    - Sẵn sàng (Ready) Tiến trình đang đợi sử dụng processor vật lý
    - Thực hiện (Running) Các câu lệnh của tiến trình đang được thực hiện.
    - Chờ đợi (Waiting) Tiến trình đang chờ đợi một sự kiện nào đó xuất hiện (sự hoàn thành thao tác vào/ra) .
    - Kết thúc (Terminated) Tiến trình thực hiện xong.
* Hệ thống có một processor
* Có duy nhất một tiến trình ở trạng thái thực hiện.
* Có thể có nhiều tiến trình ở trạng thái chờ đợi hoặc sẵn sang.
* Lưu đồ thay đổi trạng thái tiến trình



* Khối điều khiển tiến trình(PCB : Process Control Block)
* Mỗi tiến trình được thể hiện trong hệ thống bởi một khối điều khiển tiến trình .
* PCB: cấu trúc thông tin cho phép xác định duy nhất một tiến trình.
  + - Định danh quy trình
    - Trạng thái quy trình
    - Bộ đếm chương trình (PC)
    - Thông tin lập lịch CPU
    - Thông tin quản lý bộ nhớ
    - Thông tin kế toán
    - Thông tin trạng thái I / O
  1. **Thread**

Thread: luồng thực hiện tuần tự bên trong tiến trình

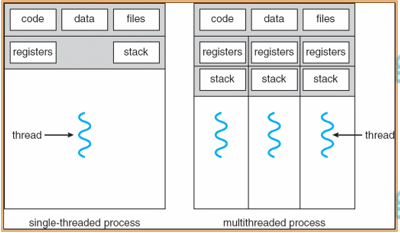
Mã thực thi, dữ liệu

Con trỏ lệnh, ngăn xếp, tập thanh ghi riêng

Tiến trình chỉ chứa một thread (Heavyweight Process)

Tiến trình gồm nhiều thread (Lightweight process)

Mô hình đa luồng (Multi\_Threading): Các luồng h/động song song, chia sẻ biến toàn cục của tiến trình

****

1. **Các vấn đề về Sysnchronization (đồng bộ hóa)**
   1. **Nhu cầu đồng bộ hóa**

Trong một hệ thống cho phép các tiến trình liên lạc với nhau, bao giờ hệ điều hành cũng cần cung cấp kèm theo những cơ chế đồng bộ hóa để bảo đảm hoạt động của các tiến trình đồng hành không tác động sai lệch đến nhau vì các lý do sau đây:

* Yêu cầu độc quyền truy xuất (Mutual exclusion)

Các tài nguyên trong hệ thống được phân thành hai loại: tài nguyên có thể chia sẻ cho phép nhiều tiến trình đồng thời truy xuất, và tài nguyên không thể chia sẻ chỉ chấp nhận một ( hay một số lượng hạn chế ) tiến trình sử dụng tại một thời điểm.

Để giải quyết vấn đề, cần bảo đảm tiến trình độc quyền truy xuất tài nguyên, nghĩa là hệ thống phải kiểm soát sao cho tại một thời điểm, chỉ có một tiến trình được quyền truy xuất một tài nguyên không thể chia sẻ.

* Yêu cầu phối hợp(Sysnchronization)

Nhìn chung, mối tương quan về tốc độ thực hiện của hai tiến trình trong hệ thống là không thể biết trước, vì điều này phụ thuộc vào nhiều yếu tố động như tần suất xảy ra các ngắt của từng tiến trình, thời gian tiến trình được cấp phát bộ xử lý… Có thể nói rằng các tiến trình hoạt động không đồng bộ với nhau. Như ng có những tình huống các tiến trình cần hợp tác trong việc hoàn thành tác vụ, khi đó cần phải đồng bộ hóa hoạt động của các tiến trình , ví dụ một tiến trình chỉ có thể xử lý nếu một tiến trình khác đã kết thúc một công việc nào đó …

* Bài toán về đồng bộ hóa
* Vấn đề tranh đoạt điều khiển (race condition)

Có thể xảy ra khi có nhiều hơn hai tiến trình đọc và ghi dữ liệu trên cùng một vùng nhớ chung, và kết quả phụ thuộc vào sự điều phối tiến trình của hệ thống- được gọi là các tình huống tranh đoạt điều khiển (race condition)

* Miền găng (critical section)

Để ngăn chặn các tình huống lỗi có thể nảy sinh khi các tiến trình truy xuất đồng thời một tài nguyên không thể chia sẻ, cần phải áp đặt một sự truy xuất độc quyền trên tài nguyên đó : khi một tiến trình đang sử dụng tài nguyên, thì những tiến trình khác không được truy xuất đến tài nguyên.

* Tóm tắt

Một số tiến trình trong hệ thống có nhu cầu trao đổi thông tin để phối hợp hoạt động, do mỗi tiến trình có một không gian địa chỉ độc lập nên viêc liên lạc chỉ có thể thực hiện thông qua các cơ chế do hệ điều hành cung cấp.

Một số cơ chế trao đổi thông tin giữa các tiến trình :

* Tín hiệu : thông báo sự xảy ra của một sự kiện
* Pipe : truyền dữ liệu không cấu trúc
* Vùng nhớ chia sẻ : cho phép nhiều tiến trình truy cập đến cùng một vùng nhớ
* Trao đổi thông điệp : truyền dữ liệu có cấu trúc, có thể vận dụng trong các hệ phân tán
* Socket : chuẩn hoán việc liên lạc giữa các hệ thống khác biệt
  1. **Các giải pháp đồng bộ**
  + Giải pháp « busy waiting »
* Các giải pháp phần mềm
* Sử dụng các biến cờ hiệu:

Các tiến trình chia sẻ một biến chung đóng vai trò « chốt cửa » (lock) , biến này được khởi động là 0. Một tiến trình muốn vào miền găng trước tiên phải kiểm tra giá trị của biến lock. Nếu lock = 0, tiến trình đặt lại giá trị cho lock = 1 và đi vào miền găng. Nếu lock đang nhận giá trị 1, tiến trình phải chờ bên ngoài miền găng cho đến khi lock có giá trị 0. Như vậy giá trị 0 của lock mang ý nghĩa là không có tiến trình nào đang ở trong miền găng, và lock=1 khi có một tiến trình đang ở trong miền găng.

while (TRUE)

{ while (lock == 1); // waitlock = 1; critical-section ();lock = 0;Noncritical-section (); }

* Sử dụng việc kiểm tra luân phiên :

Đây là một giải pháp đề nghị cho hai tiến trình. Hai tiến trình này sử dụng chung biến turn (phản ánh phiên tiến trình nào được vào miền găng), được khởi động với giá trị 0. Nếu turn = 0, tiến trình A được vào miền găng. Nếu turn = 1, tiến trình A đi vào một vòng lặp chờ đến khi turn nhận giá trị 0. Khi tiến trình A rời khỏi miền găng, nó đặt giá trị turn về 1 để cho phép tiến trình B đi vào miền găng.

* Giải pháp của Peterson

Petson đưa ra một giải pháp kết hợp ý tưởng của cả hai giải pháp kể trên. Các tiến trình chia sẻ hai biến chung :

Giả sử rằng các hướng dẫn Load và Store là atomic; nghĩa là, không thể bị gián đoạn.

Hai tiến trình chia sẻ hai biến:

int turn;

Boolean flag[2]

biến turn cho biết lượt đi vào critical.

Mảng flag được sử dụng để cho biết liệu một tiến trình đã sẵn sàng để critical section. flag [i] = true ngụ ý rằng xử lý Pi sẵn sàng!

do {

flag[i] = TRUE;

turn = j;

while (flag[j] && turn == j);

critical section flag[i] = FALSE;

remainder section

}while (TRUE);

* Các giải pháp phần cứng
* Cấm ngắt:

Cho phép tiến trình cấm tất cả các ngắt trước khi vào miền găng, và phục hồi ngắt khi ra khỏi miền găng. Khi đó, ngắt đồng hồ cũng không xảy ra, do vậy hệ thống không thể tạm dừng hoạt động của tiến trình đang xử lý để cấp phát CPU cho tiến trình khác, nhờ đó tiến trình hiện hành yên tâm thao tác trên miền găng mà không sợ bị tiến trình nào khác tranh chấp.

* Chỉ thị TSL (Test-and-Set):

Đây là một giải pháp đòi hỏi sự trợ giúp của cơ chế phần cứng. Nhiều máy tính cung cấp một chỉ thị đặc biệt cho phép kiểm tra và cập nhật nội dung một vùng nhớ trong một thao tác không thể phân chia, gọi là chỉ thị Test-and-Set Lock (TSL) và được định nghĩa như sau:

Test-and-Setlock(boolean target)

{

Test-and-Setlock = target;target = TRUE;

}

Nếu có hai chỉ thị TSL xử lý đồng thời (trên hai bộ xử lý khác nhau), chúng sẽ được xử lý tuần tự . Có thể cài đặt giải pháp truy xuất độc quyền với TSL bằng cách sử dụng thêm một biến lock, được khởi gán là FALSE. Tiến trình phải kiểm tra giá trị của biến lock trước khi vào miền găng, nếu lock = FALSE, tiến trình có thể vào miền găng.

* Các giải pháp “Sleep and wakeup”

Các giải pháp "Sleep and wakeup" Để loại bỏ các bất tiện của giải pháp « busy waiting », chúng ta có thể tiếp cận theo hướng cho một tiến trình chưa đủ điều kiện vào miền găng chuyển sang trạng thái blocked, từ bỏ quyền sử dụng CPU. Để thực hiện điều này, cần phải sử dụng các thủ tục do hệ điều hành cung cấp để thay đổi trạng thái tiến trình. Hai thủ tục cơ bản SLEEP và WAKEUP thường được sử dụng để phục vụ mục đích này.

SLEEP là một lời gọi hệ thống có tác dụng tạm dừng hoạt động của tiến trình (blocked) gọi nó và chờ đến khi được một tiến trình khác « đánh thức ». Lời gọi hệ thống WAKEUP nhận một tham số duy nhất : tiến trình sẽ được tái kích hoạt (đặt về trạng thái ready).

Ý tưởng sử dụng SLEEP và WAKEUP như sau : khi một tiến trình chưa đủ điều kiện vào miền găng, nó gọi SLEEP để tự khóa đến khi có một tiến trình khác gọi WAKEUP để giải phóng cho nó. Một tiến trình gọi WAKEUP khi ra khỏi miền găng để đánh thức một tiến trình đang chờ, tạo cơ hội cho tiến trình này vào miền găng

* Semaphore
* Nó không yêu cầu busy waiting
* Gồm 2 phép là :wait() và signal() trước kia gọi là P() và V()
* Ít phức tạp hơn
* Chỉ có thể truy cập qua 2 phép không thể phân chia (atomic)

wait (S) {

while S <= 0

; // no-op

S--;

}

signal (S)

{ S++; }

Couting semaphore - giá trị số nguyên có thể nằm trong phạm vi miền không hạn chế

Binary Semaphore - giá trị số nguyên chỉ có thể nằm trong khoảng giữa 0

và 1; có thể đơn giản hơn để thực hiện : còn được gọi là khóa mutex

Có thể triển khai một couting semaphore S như một binary semaphore

Cung cấp sự loại trừ lẫn nhau

Semaphore mutex; // initialized to 1

do { wait (mutex);

// Critical Section

signal (mutex); // remainder section

} while (TRUE);

* Monitors

Để có thể dễ viết đúng các chương trình đồng bộ hóa hơn, Hoare(1974) và Brinch & Hansen (1975) đã đề nghị một cơ chế cao hơn được cung cấp bởi ngôn ngữ lập trình , là monitor. Monitor là một cấu trúc đặc biệt bao gồm các thủ tục, các biến và cấu trúc dữ liệu có các thuộc tính sau :

Các biến và cấu trúc dữ liệu bên trong monitor chỉ có thể được thao tác bởi các thủ tục định nghĩa bên trong monitor đó. (encapsulation).

Tại một thời điểm, chỉ có một tiến trình duy nhất được hoạt động bên trong một monitor (mutualexclusive).

Trong một monitor, có thể định nghĩa các biến điều kiện và hai thao tác kèm theo là Wait và Signal như sau : gọi c là biến điều kiện được định nghĩa trong monitor:

Wait(c): chuyển trạng thái tiến trình gọi sang blocked , và đặt tiến trình này vào hàng đợi trên biến điều kiện c.

Signal(c): nếu có một tiến trình đang bị khóa trong hàng đợi của c, tái kích hoạt tiến trình đó, và tiến trình gọi sẽ rời khỏi monitor.

* 1. **Deadlock**
* Định nghĩa:

Một tập hợp các tiến trình được định nghĩa ở trong tình trạng tắc nghẽn khi mỗi tiến trình trong tập hợp đều chờ đợi một sự kiện mà chỉ có một tiến trình khác trong tập hợp mới có thể phát sinh được.

Nói cách khác, mỗi tiến trình trong tập hợp đều chờ được cấp phát một tài nguyên hiện đang bị một tiến trình khác cũng ở trạng thái blocked chiếm giữ. Như vậy không có tiến trình nào có thể tiếp tục xử lý , cũng như giải phóng tài nguyên cho tiến trình khác sử dụng, tất cả các tiến trình trong tập hợp đều bị khóa vĩnh viễn !

* Điều kiện xảy deadlock
* Có sử dụng tài nguyên không thể chia sẻ (Mutual exclusion): Mỗi thời điểm, một tài nguyên không thể chia sẻ được hệ thống cấp phát chỉ cho một tiến trình , khi tiến trình sử dụng xong tài nguyên này, hệ thống mới thu hồi và cấp phát tài nguyên cho tiến trình khác.
* Sự chiếm giữ và yêu cầu thêm tài nguyên (Wait for): Các tiến trình tiếp tục chiếm giữ các tài nguyên đã cấp phát cho nó trong khi chờ được cấp phát thêm một số tài nguyên mới.
* Không thu hồi tài nguyên từ tiến trình đang giữ chúng (No preemption): Tài nguyên không thể được thu hồi từ tiến trình đang chiếm giữ chúng trước khi tiến trình này sủ dụng chúng xong.
* Tồn tại một chu kỳ trong đồ thị cấp phát tài nguyên ( Circular wait): có ít nhất hai tiến trình chờ đợi lẫn nhau : tiến trình này chờ được cấp phát tài nguyên đang bị tiến trình kia chiếm giữ và ngược lại.
* Giải pháp
* Giải pháp # 1: Phát hiện deadlock
  + Xác định thời điểm hệ thống bị deadlock
  + Phá vỡ deadlock bằng cách kết thúc một trong các thread
  + Thường không thực tế trong các hệ điều hành, nhưng thường được sử dụng trong các hệ thống cơ sở dữ liệu nơi các giao dịch có thể bị hủy bỏ và thử lại
* Giải pháp # 2: Ngăn chặn deadlock: loại bỏ một trong những điều kiện cần thiết cho deadlock
  + Không cho phép truy cập độc quyền? Không hợp lý cho hầu hết các ứng dụng.
  + Tạo đủ tài nguyên để chúng không bao giờ cạn kiệt? Có thể hoạt động với những thứ như dung lượng ổ đĩa, nhưng các ổ khóa để đồng bộ hóa bị giới hạn về số lượng.
  + Cho phép ưu tiên? Hoạt động cho một số tài nguyên nhưng không hoạt động cho một số tài nguyên khác (ví dụ: không thể chặn trước một khóa).
  + Yêu cầu các luồng yêu cầu tất cả các tài nguyên cùng một lúc; hoặc lấy tất cả hoặc đợi tất cả.
    - Khó thực hiện: phải đợi nhiều thứ mà không khóa bất kỳ thứ nào trong số chúng.
    - Bất tiện cho chủ đề: khó đoán trước nhu cầu. Có thể yêu cầu luồng phân bổ quá mức chỉ để an toàn.
  + Ngăn chặn sự tuần hoàn: tất cả các luồng yêu cầu tài nguyên theo cùng một thứ tự (ví dụ: luôn khóa l1 trước l2 ). Đây là cách tiếp cận phổ biến nhất được sử dụng trong hệ điều hành.

1. **Triển khai thức hiện đề tài**
2. **Tổng quan đề tài**

Caltrain Automation là 1 trong các đề tài trong trong phần Sysnchrozation(đồng bộ hóa ) trong môn hệ điều hành của đại học Standford .Bài toán được đạt ra để giúp sinh viên củng cố những kiến thức đã học và áp dụng giải quyết các vấn đề về đồng bộ hóa.

1. **Phân tích yêu cầu đề tài**

CalTrain đã quyết định nâng cao hiệu quả của mình bằng cách tự động hóa không chỉ các chuyến tàu mà còn cả hành khách của nó. Từ bây giờ, hành khách sẽ là robot. Mỗi robot và mỗi đoàn tàu được điều khiển bởi một sợi chỉ. Chúng ta đã được thuê để viết các chức năng đồng bộ hóa để đảm bảo việc tải các đoàn tàu có trật tự.

Trước tiên ta phải xây dựng struct về Station và một số chức năng như:

* Khi một đoàn tàu đến ga và đã mở cửa, nó sẽ gọi hàm

station\_load\_train (struct station \* station, int count)

trong đó count cho biết có bao nhiêu chỗ ngồi trên tàu. Chức năng này không được hoạt động trở lại cho đến khi tàu được xếp đầy đủ (tất cả hành khách đã ngồi trên ghế của mình và tàu đã đầy hoặc tất cả hành khách đang chờ đã lên tàu).

* Khi tàu chở khách đến nhà ga, đầu tiên nó sẽ gọi chức năng

station\_wait\_for\_train (struct station \* station)

Chức năng này không được hoạt động trở lại cho đến khi có tàu trong ga (tức là đang thực hiện một cuộc gọi đến station\_load\_train) và có đủ ghế trống trên tàu cho hành khách này ngồi xuống. Khi chức năng này hoạt động trở lại, robot chở khách sẽ di chuyển hành khách trên tàu và vào một chỗ ngồi .Khi hành khách đã ngồi vào chỗ, nó sẽ gọi hàm

station\_on\_board (struct station \* station)

để cho tàu biết rằng hành khách đang ở trên tàu.

Theo yêu cầu ta phải tạo một tệp caltrain.c có chứa khai báo struct stationvà định nghĩa ba hàm trên, cùng với hàm station\_init, hàm này sẽ được gọi để khởi tạo đối tượng trạm khi CalTrain khởi động. Ngoài ra:

Chúng ta phải viết giải pháp của mình bằng C bằng cách sử dụng các hàm Pintos cho các biến khóa và biến điều kiện:

* + - lock\_init (struct lock \* lock)
    - lock\_acquire (struct lock \* lock)
    - lock\_release (struct lock \* lock)
    - cond\_init (struct condition \* cond)
    - cond\_wait (struct condition \* cond, struct lock \* lock)
    - cond\_signal (struct condition \* cond, struct lock \* lock)
    - cond\_broadcast ( struct condition\* cond, struct lock \* lock)

Chỉ sử dụng các chức năng này (ví dụ: không có semaphores hoặc các nguyên thủy đồng bộ hóa khác).

* **Yêu cầu**:

Chúng không thể sử dụng nhiều hơn single lock trong mỗi struct station.

Chúng ta có thể giả định rằng không bao giờ có nhiều hơn một chuyến tàu trong nhà ga cùng một lúc và tất cả các chuyến tàu (và tất cả hành khách) đều đi đến cùng một điểm đến (nghĩa là mọi hành khách có thể lên bất kỳ chuyến tàu nào).

Code của chúng ta phải cho phép nhiều hành khách lên tàu đồng thời (phải có thể có nhiều hành khách đã gọi station\_wait\_for\_train và chức năng đó phải hoạt động trở lại đối với từng hành khách, trước khi bất kỳ hành khách nào gọi đến station\_on\_board).

Code của chúng ta không được dẫn đến việc chờ đợi.

Chúng ta có thể cho rằng không có đánh thức giả trên các biến điều kiện.

* **Mục tiêu**
* Chương trình chạy đúng các yêu cầu của đề bài đặt ra
* Tối ưu hóa các dòng code
* Không xảy ra các lỗi

1. **Triển khai thực hiện**

* Phương pháp thực hiện
* Để thuận tiện cho việc thực hiện lần demo kết quả . Chúng ta xây dựng file thư viện caltrain.h chứa định nghĩa về station gồm các thuộc tính như Fseats(chỗ trống) , passengers\_W(hành khách chờ), passengers\_L(hành khách rời đi) và khai báo khóa mutex và 2 biến điều kiện là passengers\_OB(khách trên tàu) và Fseats\_available (ghế trống có sẵn) và khai báo các chức năng đề bài yêu cầu như : station\_init(struct station \*station) , station\_load\_train(struct station \*station, int count), station\_wait\_for\_train(struct station \*station), station\_on\_board(struct station \*station).
* Ta xây dựng file caltrain.c để triển khai toàn bộ các chức năng đã được khai báo ở file thư viện caltrain.h
* station\_init(struct station \*station)

Ta khởi tạo giá trị mặc định cho các thuộc tính

Ở cả 3 hàm tiếp theo đây ta đều sử dụng hóa mutex để đảm bảo cho việc thực thi được tốt ra tránh các tranh chấp điều khiển và tài nguyên với các luồng truy cập đến nó

* station\_load\_train(struct station \*station,int count){

pthread\_mutex\_lock(&(station->mutex));

if (!count || !station->passengers\_W) {

pthread\_mutex\_unlock(&(station->mutex));

return;

}

station->Fseats = count;

pthread\_cond\_broadcast(&(station->Fseats\_available));

pthread\_cond\_wait(&(station->passengers\_OB), &(station->mutex));

station->Fseats = 0;

pthread\_mutex\_unlock(&(station->mutex));}

Ở đây ta dùng khóa mutex để tránh xảy ra tranh chấp, như chúng ta đã phân tích ở phần phân tích . Nếu số ghế hoặc khách chờ bằng 0 thì mở hóa và return thoát hàm. Ngược lại ta có được số lượng ghế trống và đánh thức luồng với biến điều kiện Fseats\_available, sau đó ta đưa vào luồng ngủ không làm gì cho đến khi tàu xếp đầy đủ bởi biến điều kiện pasengers\_OB và ta mở khóa mutex để kết thúc phiên làm việc và cho các luồng mới đang đợi vào.

* station\_wait\_for\_train(struct station \* station)

{pthread\_mutex\_lock(&(station->mutex));

station->passengers\_W++;

while (!station->Fseats)

pthread\_cond\_wait(&(station->Fseats\_available), &(station->mutex));

station->passengers\_W--;

station->passengers\_L++;

station->Fseats--;

pthread\_mutex\_unlock(&(station->mutex));}

Như đã phân tích hàm này có chức năng xếp chỗ cho hành khách trên tàu và nó sẽ không hoạt động trở lại cho đến khi có tàu đến ga và có chỗ trống khi đó pthread\_cond\_wait (&(station->Fseats\_available), &(station->mutex)) sẽ thực hiện điều đó.

* Station\_on\_board(struct station \*station)

pthread\_mutex\_lock(&(station->mutex));

station->passengers\_L--;

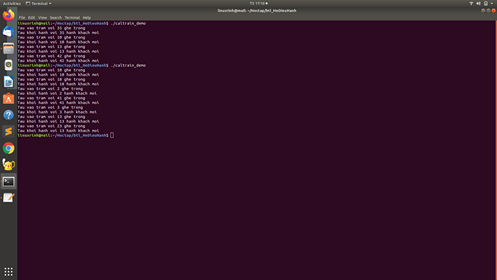
if (!station->passengers\_L && !(station->passengers\_W && station->Fseats))

pthread\_cond\_signal(&(station->passengers\_OB));

pthread\_mutex\_unlock(&(station->mutex));

Chức năng này báo hiệu khi khách đã ở trên tàu hết ghế trống hoặc không còn hành khách nào đợi , sau đó nó sẽ đánh thức luồng đang thực thi trên hàm station\_load\_train đang ngủ bởi biến điều kiện passengers\_OB

* Ta xây dựng file caltrain\_demo.c để thực hiện thử nghiệm các hàm ta đã triển khai đã thực sự đạt kết quả như yêu cầu chưa. Ta lần dựng xây các hàm alarm để báo hiệu các lỗi có thể xảy ra với caltrain và các hàm passengers\_thread và load\_train\_thread để phục vụ cho việc kiểm thử ở đây ta có 2 giá trị load\_train\_returned và thread\_completed để xác định tạo luồng để chạy các hàm trên đã thành công hay chưa. Ta giả sử có 100 hành khách cùng gọi hàm station\_wait\_for\_train bằng cách tạo mỗi luồng mới cho mỗi hành khách. Tiếp theo ta thực hiện khởi tạo các luồng để chạy station\_load\_train với số ghế trống là ngẫu nhiên với số ghế trống tối đa là 50 ghế để đón dựa trên số ghế của tàu vào ga và số hành khách chờ ta xếp hết khách vào ghế trống hoặc hết khách chờ khi các tàu khi mỗi luồng trên khởi tạo thành công được xác định bởi thread\_completed ta thực hiện gọi hàm station\_on\_board và vòng lập kết thúc khi tất cả hành khách đã lên hết các chuyến tàu .

1. **Kết quả**
   * Kết quả thực hiện đạt được như yêu cầu đề ra
2. **Đánh giá nhận xét**

Bài tập lớn Caltrain đã đạt được những yêu cầu cơ bản của đề bài đạt ra.

Qua công việc thiết kế tìm hiểu project lần này, em đã thu được những kinh nghiệm và kiến thức vô cùng hữu ích. Hệ điều hành vừa giúp em nâng cao kiến thức về cấu trúc máy tính, các kiến thức sâu rộng của hệ điều hành giúp em này giúp nâng cao khả năng lập trình, khả năng tư duy thiết kế hệ thống, khả năng học tập ,tự học được nâng cao.

Thông qua việc thực hiện project, em đã tính lũy rất nhiều kiến thức thực tế về các về vấn trên hệ điều hành , thầy Phạm Văn Tiến đã tạo cho chúng em niềm say mê học tập, tìm tòi kiến thức mới. Ngoài những kiến thức về bộ môn, chúng em cũng học được từ thầy những kinh nghiệm, kỹ năng thiết thực giúp ích cho chúng em thực hiện tốt hơn những nhiệm vụ, bài tập của mình …...

Trong quá trình thực hiện project này, dù rất cố gắng nhưng do vốn kiến thức hạn hẹp nên không thể tránh khỏi những sai sót. Em rất mong nhận được những đóng góp, phê bình, chia sẻ của thầy để các bài tập tiếp theo của em sẽ hoàn thiện hơn.

Em xin chân thành cảm ơn thầy!

1. **Danh mục tài liệu tham khảo**

* Slide hệ điều hành của thầy Phạm Văn Tiến
* Giáo trình hệ điều hành voer.edu.vn
* Sử dụng thư viện thread trên <https://timoday.edu.vn/su-dung-thu-vien-pthead/>
* <https://gcc.gnu.org/onlinedocs/gcc/_005f_005fsync-Builtins.html>