**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN ĐIỆN TỬ - VIỄN THÔNG**

A screenshot of a computer

Description automatically generated with low confidence

**BÁO CÁO BÀI TẬP LỚN**

**HỆ ĐIỀU HÀNH PINTOS**

|  |  |
| --- | --- |
| **Họ và tên:** | **Nguyễn Tiến Thành** |
| **MSSV:** | **20172819** |
| **Lớp:** | **ĐTVT – 01 K62** |

***Hà Nội, tháng 5 năm 2021***

MỤC LỤC:

[CHƯƠNG 1: HỆ ĐIỀU HÀNH PINTOS 3](#_Toc72418131)

[1.1 Tổng quan về Pintos: 3](#_Toc72418132)

[1.2 Hệ thống thư mục của Pintos: 3](#_Toc72418133)

[CHƯƠNG 2: DỰ ÁN 1 – VẤN ĐỀ ALARM CLOCK 4](#_Toc72418134)

[2.1: Cấu trúc Threads: 4](#_Toc72418135)

[2.2 Vấn đề: 6](#_Toc72418136)

[2.3 Phương án giải quyết: 6](#_Toc72418137)

[2.4 Triển khai code: 6](#_Toc72418138)

[2.5 Kết quả: 8](#_Toc72418139)

[CHƯƠNG 3: BỔ SUNG: VẤN ĐỀ SỐ 0 – SYNCHRONIZATION 10](#_Toc72418140)

[3.1 Tổng quan: 10](#_Toc72418141)

[3.2 CalTrain Automation: 10](#_Toc72418142)

[3.3 Chemical Reaction: 11](#_Toc72418143)

[3.4 Kết quả: 12](#_Toc72418144)

[CHƯƠNG 4: KẾT LUẬN 13](#_Toc72418145)

# CHƯƠNG 1: HỆ ĐIỀU HÀNH PINTOS

Ở chương này, ta sẽ tìm hiểu tổng quan về hệ điều hành Pintos cũng như cấu trúc thư mục bên trong.

## 1.1 Tổng quan về Pintos:

Pintos là hệ điều hành đơn giản với kiến trúc 80x86 được thiết kế bởi đại học Stanford nhằm phục vụ cho mục đích học tập và nghiên cứu trong khóa học CS140 của sinh viên. Hệ điều hành này hỗ trợ các cơ chế cơ bản như kernel threads, khởi động và chạy user programs, hệ thống file, tuy nhiên chúng chỉ dừng ở mức cơ bản nhất. Do đó, qua dự án Pintos, sinh viên sẽ tiến hành lập trình và cải tiến các cơ chế trên sao cho chúng đạt được hiệu quả tốt nhất.

Các dự án nhỏ trong Pintos bao gồm:

* Project 1: Threads.
* Project 2: User Programs.
* Project 3: Virtual Memory.
* Project 4: File Systems.

Các dự án trên sẽ được chạy và kiểm tra bằng một hệ thống mô phỏng. Bản chất nó là một chương trình với chức năng mô phỏng lại một hệ vi xử lý kiến trúc 80x86. Báo cáo này sẽ sử dụng trình mô phỏng QEMU.

## 1.2 Hệ thống thư mục của Pintos:

Bên trong mã nguồn pintos, chúng ta có thể thấy cấu trúc thư mục của dự án theo đường dẫn “…/pintos/src”.

* Thư mục “threads/”: là nơi chưa mã nguồn của kernel gốc. Ở dự án 1 ta sẽ chủ yếu triển khai tại thư mục này.
* Thư mục “userpro/”: chưa mã nguồn cho trình tải nạp ứng dụng người dùng (user programs). Tại đây triển khai dự án 2.
* Thư mục “vm/”: Thư mục này hoàn toàn rỗng, ta cần triển khai Virtual memory tại đây (Dự án 3).
* Thư mục “filesys/”: Chứa mã nguồn cho hệ thống file cơ bản. Thư mục này được sử dụng trong dự án 2 và dự án 4.
* Thư mục “devices/”: Đây là thư mục chứa mã nguồn cho việc giao tiếp của hệ thống với các thiết bị vào ra. Chúng ta sẽ cần thay đổi mã nguồn cho bộ định thời timer trong dự án số 1.
* Thư mục “lib/”: Chứa các thư viện chuẩn trong C phục vụ cho các dự án. Ta không cần thay đổi mã nguồn tại đây.
* Thư mục “tests/”: Chứa các kết quả chạy thử của các dự án. Ta có thể thay đổi mã nguồn nếu cần trong quá trình kiểm thử.

# CHƯƠNG 2: DỰ ÁN 1 – VẤN ĐỀ ALARM CLOCK

Ở chương này, ta sẽ tìm hiểu và xử lý vấn đề đầu tiên của Dự án 1 đó là vấn đề về Alarm clock.

## 2.1: Cấu trúc Threads:

Vòng đời của một thread được biểu thị qua các trạng thái từ lúc nó được sinh ra (NEW) cho tới khi bị hệ điều hành tiêu hủy (TERMINATED) được biểu diễn bằng sơ đồ FSM như bên dưới.

Diagram

Hình 2. 2: Sơ đồ FSM của một thread

Trong file “thread.h”, 4 biến trạng thái của một thread được Pintos khai báo sẵn bao gồm:

* THREAD\_RUNNING: Trạng thái mà thread đang chạy và chiếm CPU.
* THREAD\_READY: Trạng thái mà luồng sẵn sàng để chạy nhưng chưa chiếm được CPU. Lúc này, nó được đặt trong hàng đợi ready queue để chờ tới lượt.
* THREAD\_BLOCKED: Thread đang chạy nhưng cần đợi một tín hiệu IO gửi đến hoặc một sự kiện xảy ra thì mới được chạy tiếp. Lúc này nó được đưa vào trạng thái THREAD\_BLOCKED để nhường CPU cho các luồng khác.
* THREAD\_DYING: Trạng thái mà luồng chuyển từ RUNNING sang TERMINATED.

Bên cạnh đó, một biến thread với cấu trúc struct cũng được định nghĩa bao gồm các trường sau:

* tid\_t tid: ID của luồng là một số nguyên, bản chất được định nghĩa từ kiểu dữ liệu int. Các luồng phải có ID khác nhau để phục vụ cho quá trình nhận biết và tránh nhầm lẫn giữa các luồng.
* enum thread\_status status: Trạng thái hiện tại của luồng, gồm 4 trạng thái nêu ở trên.
* char name[16]: Tên của luồng, phục vụ cho quá trình debug.
* uint8\_t \*stack: Vùng nhớ 8 bit để lưu địa chỉ của con trỏ đỉnh ngăn xếp.
* int prioriry: Mức ưu tiên của luồng. Trong Pintos có 64 mức ưu tiên khác nhau từ 0 đến 63.
* struct list\_elem allelem: Trường list\_elem giúp các luồng được liên kết với nhau tạo thành một danh sách có tên là “all\_luồng”. Mỗi luồng sẽ được tự động thêm vào danh sách này ngay khi nó được tạo ra.
* struct list\_elem elem: Trường này có ý nghĩa giúp một luồng có thể được thêm vào 2 danh sách liên kết cùng 1 lúc.
* uint32\_t\* pagedir: Trường này liên quan trực tiếp đến dự án số 2.
* usigned magic: Trường này giúp phát hiện lỗi tràn stack.

## 2.2 Vấn đề:

Trong Pintos, cơ chế sleep của một thread được triển khai trong hàm timer\_sleep() nằm bên trong file “timer.c”. Giải pháp này tuy hoạt động nhưng gây ra vấn đề “busy waiting”. Vấn đề này gây ra một vòng lặp để kiểm tra current time và gọi tới hàm thread\_yield() cho tới khi đạt đủ thời gian yêu cầu. Điều này gây tiêu tốn tài nguyên của hệ thống một cách lãng phí.

Cụ thể khi nhìn vào bên trong hàm thread\_yield() ta sẽ thấy nó thực hiện đưa luồng về trạng thái THREAD\_READY, đẩy vào hàng đợi ready\_queue và cuối cùng lập lịch cho nó bằng cách gọi tới hàm schedule().

Do sử dụng vòng while() nên thread sẽ ra vào hàng đợi ready\_queue cho tới khi vòng while() kết thúc. Đây quả thực là một cách làm không hiệu quả vì nếu một thread không có nhu cầu sử dụng CPU, ta cần đưa nó về trạng thái THREAD\_BLOCKED để các thread khác có thể chiếm dụng CPU, thay vì liên tục đưa về trạng thái THREAD\_READY như Pintos đang xử dụng.

## 2.3 Phương án giải quyết:

Mục tiêu đối với vấn đề Alarm clock cơ bản là loại bỏ vòng lặp gây ra “busy waiting” đã nếu ở trên. Nếu thành công, hiệu năng của hệ điều hành sẽ được cải thiện một cách đáng kể.

Để giải quyết vấn đề này, ta sẽ xây dựng một phương án mới với cơ chế sleep. Cụ thể thay vì đưa thread về trạng thái THREAD\_READY như đã nêu ở trên, ta sẽ đưa nó về THREAD\_BLOCKED. Lúc này coi như thread đó đang sleep, nhiều thread cùng trạng thái đó sẽ liên kết với nhau tạo thành một hàng đợi sleep queue. Sau đó ta sẽ lặp lại việc duyệt các phần tử trong hàng đợi này, giảm biến sleep\_ticks đi 1. Nếu sleep\_ticks của một thread trong hàng đợi này về giá trị 0, ta sẽ đánh thức nó và đưa vào hàng đợi ready queue để chờ tới lượt.

## 2.4 Triển khai code:

Với ý tưởng nêu trên, ta sẽ tiến hành hiện thực hóa bằng cách sửa đổi mã nguồn của Pintos. Việc này chủ yếu diễn ra ở các file sau: “devices/timer.c”, “threads/thread.h”, “threads/thread.c”.

Trước khi sửa, ta cần phải tìm hiểu cơ chế của một số hàm quan trọng đã được triển khai sẵn.

Như ta đã biết, khi một ngắt (interrupt) xảy ra thì hệ điều hành sẽ ngưng cuộc việc đang làm để xử lý việc ngắt. Xem xét mã nguồn của 2 hàm có ảnh hưởng trục tiếp tới hàm timer\_sleep() là timer\_interrupt() và thread\_tick(), ta có thể thấy thread\_tick() được gọi sau mỗi một đơn vị tick và nó sẽ kiểu tra xem thời gian luồng chiếm CPU đã vượt quá time slice (= 4 ms) hay chưa để sau đó thực hiện quá trình context switch. Do đó mỗi lần hàm timer\_interrupt(), ta cần update lại số ticks bên trong hàng đợi sleep queue, nếu không thì các threads đang ngủ sẽ không bao giờ được đánh thức.

Ta bắt đầu tiến hành thêm các hàm mới, khai báo các biến,… để phục vụ cho việc giải quyết vấn đề như sau:

Các biến được thêm mới:

* Trạng thái THREAD\_SLEEP: trạng thái ngủ.
* Cấu trúc struct list sleep\_list: được khai báo là biến toàn cục bên trong file “thread.c”. bản chất đây là hàng đợi (queue) chứa các luồng đang ngủ.
* int64\_t sleep\_ticks: thời gian ngủ (nếu bằng 0 thì sẽ được đưa ra khỏi sleep\_list), được thêm vào cấu trúc của thread.

Các hàm được thêm mới:

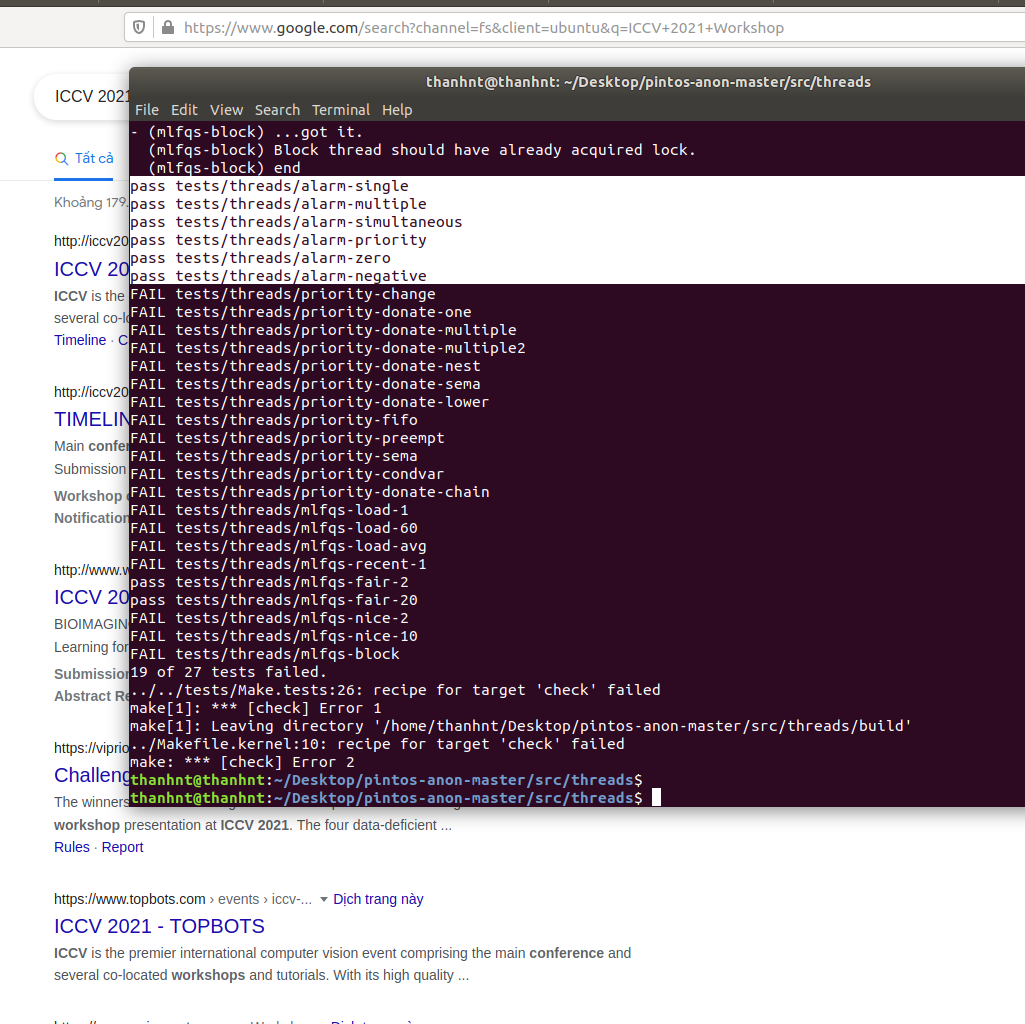
* void thread\_sleep(int64\_t ticks): hàm này sẽ đưa thread hiện tại vào sleep\_list queue, sau đó update thông tin (ticks) của thread và cuối cùng gọi hàm schedule() để tiến hành lập lịch cho thread mới. Hàm này sau đó sẽ được gọi ở bên trong hàm timer\_sleep().
* void thread\_foreach\_sleep(void): hàm này thực hiện việc duyệt (traverse) các thread bên trong sleep\_list. Với mỗi thread, giảm số sleep\_ticks đi 1. Nếu một thread có sleep\_ticks bằng 0, nó sẽ được “đánh thức” bằng cách xóa khỏi sleep\_list và đưa vào hàng đợi ready. Hàm này sẽ được gọi bên trong hàm timer\_interrupt().
* void thread\_less\_priority(): hàm này mục đích để so sánh mức độ ưu tiên (priority) của 2 list threads. Nó được gọi bên trong hàm next\_thread\_to\_run để chọn thread có mức độ ưu tiên hơn trong ready queue.

Hàm được thay đổi:

* Hàm timer\_sleep(int64\_t ticks): Hoàn toàn không sử dụng lại code có sẵn. Bây giờ ta chỉ cần kiểu tra đầu vào (số ticks) có hợp lệ hay không (>=0) ròi sau đó gọi đến hàm thread\_sleep() đã khai báo.
* Hàm timer\_interrupt(): Khi mỗi ticks tới, thực hiện update sleep\_ticks của các threads bên trong sleep\_list, nếu không thì các thread này sẽ không bao giờ được “đánh thức”.
* Hàm thread\_init(): Cần thực hiện khởi tạo (initialize) sleep\_list cùng mới một thread.
* Hàm next\_thread\_to\_run: Một yêu cầu nữa của vấn đề đó là khi chọn một process tiếp theo để chiếm CPU, cái có mức độ ưu tiên cao hơn sẽ được chọn. Ban đầu hàm này chỉ chọn thread đầu tiên bên trong ready queue, giờ ta sử dụng hàm list\_max() được cung cấp sẵn, với đầu vào là ready queue và thread\_less\_priority() để chọn thread có mức độ ưu tiên cao hơn.

## 2.5 Kết quả:

Sau khi tiến hành triển khai phương án đã nêu và kiểm tra kết quả thông qua lệnh make check, ta thu được kết quả thành công khi cả 6 test case trong vấn đề này đã pass.



# CHƯƠNG 3: BỔ SUNG: VẤN ĐỀ SỐ 0 – SYNCHRONIZATION

Chương này đề cập vấn đề 0 (Project set 0) của dự án Pintos.

## 3.1 Tổng quan:

Bên cạnh 4 vấn đề như đã nêu ở trên, dự án Pintos còn đưa ra một vấn đề số 0 với mục đích để sinh viên làm quen với việc đồng bộ hóa (synchronization) thông qua 2 bài tập là CalTrain Automation và Chemical Reaction.

Thay vì thực hiện sửa code của hệ điều hành pintos, ở trong vấn đề này, ta được tiếp cận các bài toán khá thực tiễn. Qua đó áp dụng các kiến thức về đồng bộ hóa là khóa (locks) và biến điều kiện (condition variables) để giải quyết các vấn đề trên.

Hai bài tập bên trong vấn đề này được yêu cầu thực hiện theo hình thức cá nhân thay vì làm theo nhóm như 4 vấn đề nêu trên. Trong phần này, em sẽ nêu lại bài toán, yêu cầu được đưa ra kèm với kết quả sau khi triển khai giải thuật. Chi tiết về việc triển khai mã nguồn sẽ được em comment đầy đủ trong file caltrain.c và reaction.c tại github repository đính kèm bên dưới.

## 3.2 CalTrain Automation:

**Bài toán**: Công ty tàu lửa CalTrain quyết định thử nghiệm phương pháp cải thiện tính hiệu quả của hệ thống tàu điện bằng cách tự động hóa các con tàu của họ và thậm chí cả khách hàng. Qua đó, mỗi khách hàng sẽ được đại diện bởi một con robot được điều khiển qua một thread. Với tư cách là một kỹ sư Điện tử Viễn Thông, bạn được công ty thuê để lập trình quá trình đồng bộ hóa nhằm đảm bảo việc hoạt động của hệ thống tàu được trơn tru.

Đầu tiên, cần phải định nghĩa một cấu trúc struct station, kèm với các hàm được mô tả như bên dưới.

Khi một con tàu đến ga, nó mở cửa các toa và gọi tới hàm: station\_load\_train(struct station \*station, int count). Trong đó biến count thể hiện số ghế còn trên tàu. Hàm này chỉ được phép return khi tất cả hành khách đều lên tàu và ngồi vào ghế kém với điều kiện là hoặc là số ghế đã đầy hoặc tất cả số hành khách vừa chờ ở bến đều được lên tàu.

Khi một khách hàng (robot) tới ga, nó gọi tới hàm: station\_wait\_for\_train(struc station \*station). Hàm này không được phép return cho đến khi có một con tàu về tới ga (hàm station\_load\_train đang được thực thi) và trên con tàu đó có đủ số ghế thừa cho các khách đang chờ. Sau khi hàm này được return, khách hàng sẽ lên tàu và ngồi vào ghế. Sau đó nó sẽ gọi tới hàm station\_on\_board(struct station \*station). Hàm này nhằm mục đích cho phép con tàu rời bến.

Tiếp theo, tạo file caltrain.c chứa việc khởi tạo các hàm nếu trên cùng với cấu trúc struct station, kèm theo hàm station\_init() để khởi tạo đối tượng station khi bắt đầu thực thi hệ thống.

Một số yêu cầu cần lưu ý:

* Các hàm được viết bằng ngôn ngữ C, CHỈ sử dụng các hàm khóa và điều kiện của Pintos dưới đây:
  + lock\_init()
  + lock\_acquire()
  + lock\_release()
  + cond\_init()
  + cond\_wait()
  + cond\_signal()
  + cond\_broadcast()
* Không được sử dụng nhiều hơn một lock ở mỗi struct station.
* Có thể coi như mỗi lần chỉ có 1 tàu về bến và mọi tàu đều đi chung 1 hướng.
* Code cần cho phép nhiều khách hàng lên tàu lần lượt.
* Code không được bị busy-waiting.

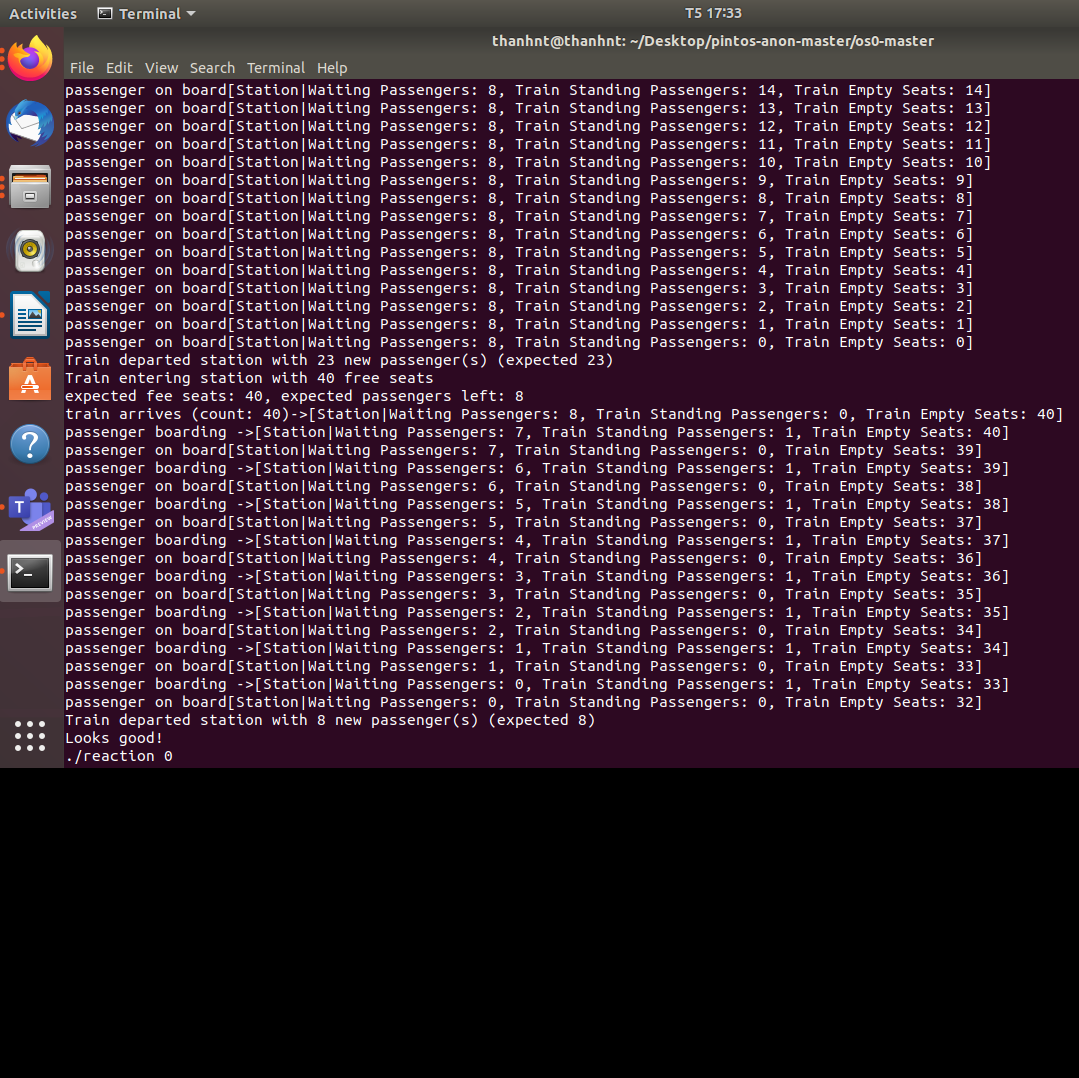
## 3.3 Chemical Reaction:

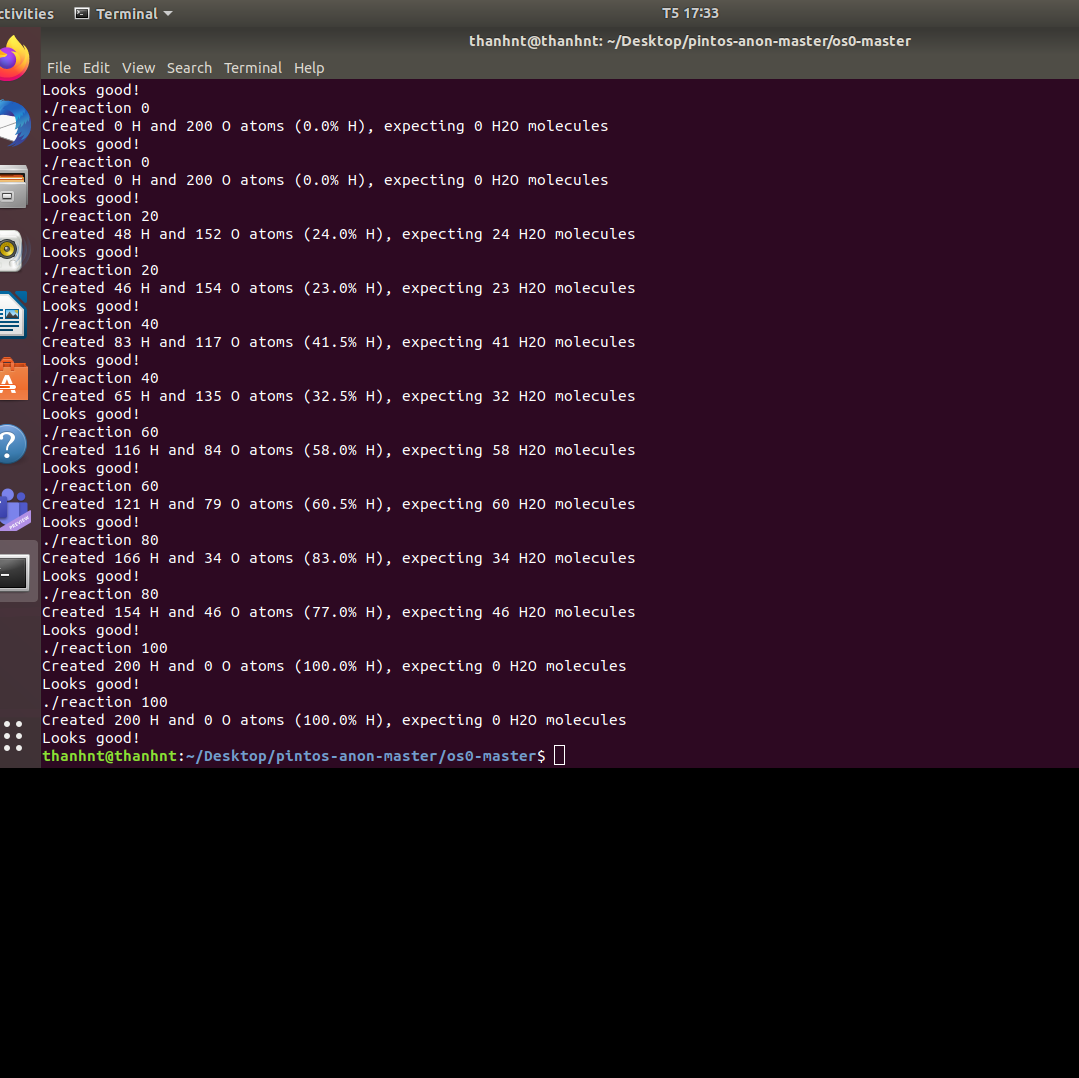
**Bài toán:** Bạn được Mẹ thiên nhiên thuê để giúp về vấn đề phản ứng hóa học để tạo thành nước. Mẹo là dùng 2 nguyên tử H và 1 nguyên tử O cùng một lúc. Ở đây mỗi nguyên tử được đại diện bởi một thread. Mỗi nguyên tử H gọi tới hàm void reaction\_h(struct reaction \*r), khi sẵn sàng phản ứng, mỗi nguyên tử O gọi tới hàm void reaction\_o(struct reaction \*r).

Bạn cần viết code cho 2 hàm này. Các hàm cần trì hoãn tới khi ít nhất 2 H và 1 O xuất hiện, và sau đó chỉ một hàm được gọi để tạo make\_water() – Hàm này đã được Mẹ thiên nhiên viết sẵn. Sau mỗi lần make\_wake() được gọi, 2 instances của reaction\_h và 1 của reaction\_o được return.

Tạo file reaction.c chứa các hàm nêu trên kèm với khai báo struct reaction (chưa tất cả các biến cần được đồng bộ). Các yêu cầu còn lại giống với vấn đề CalTrain đã nêu.

## 3.4 Kết quả:





# CHƯƠNG 4: KẾT LUẬN

Như vậy, ở báo cáo này em đã thực hiện triển khai giải quyết vấn đề Alarm clock tại Project 1 kèm với thực hiện Project 0 gồm 2 bài là CalTrain Automation và Chemical Reaction.

Ở vấn đề Alarm clock, giải pháp em đưa ra đã giúp hệ điều hành tiết kiệm được tài nguyên đáng kể so với giải pháp ban đầu của Pintos. Còn với Project 0, em thu được rất nhiều kiến thức, cách sử dụng các phương pháp như lock và condition variable để thực hiện việc đồng bộ, cũng như một hướng khá hay để áp dụng vào bài toán thực tiễn.

Tuy nhiên bên cạnh các phương án triển khai của em, còn có rất nhiều cách khác với số dòng code ngắn hơn hoặc hiệu quả hơn. Hy vọng em sẽ được thầy và các bạn bổ sung nhận xét và đóng góp thêm ý kiến.

Em xin chân thành cảm ơn thầy Phạm Văn Tiến đã cung cấp rất nhiều kiến thức bổ ích qua các tiết học cũng như giới thiệu hệ điều hành Pintos để sinh viên chúng em có một trải nghiệm rất thực tiễn trong môn học Hệ Điều Hành.

Link github repository: https://github.com/paty0504/NguyenTienThanh-2819-pintOs