TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

———————— * ————————

BÁO CÁO

Linux hệ nhúng theo chuẩn kỹ năng ITSS

ĐỀ TÀI: **Hệ thống thang máy vận chuyển hành lý**

Giảng viên hướng dẫn: THS. **Bành Thị Quỳnh Mai**

Sinh viên thực hiện:

Trần Quang Long – MSSV: 20184142

Nguyễn Mạnh Trường – MSSV: 20184208

Nguyễn Khánh Toàn – MSSV: 20184203

Nguyễn Gia Minh – MSSV: 20184152

Hà Nội, tháng 8 năm 2022

LỜI NÓI ĐẦU

Với sự phát triển nhanh và mạnh trong lĩnh vực công nghệ thông tin, sản phẩm công nghệ ngày càng chịu sự đánh giá khắt khe hơn từ phía những người dùng. Những sản phẩm công nghệ phục vụ cho mục đích kinh doanh ngày càng đa dạng, tốc độ phát triển nhanh. Vì thế, rất nhiều những sản phẩm hay và hấp dẫn đã được ra đời trong thời gian qua.

Với những kiến thức được học trong môn "Linux hệ nhúng theo chuẩn kỹ năng ITSS", nhóm chúng em đã làm báo cáo với đề tài "Hệ thống thang máy vận chuyển hành lý (Luggage vehicular elevator control system)". Đây là một hệ thống với kết cấu đơn giản và phù hợp với việc thực hành để vừa hiểu sâu hơn kiến thức môn học, vừa có thể áp dụng kiến thức đã học, tạo nền tảng áp dụng cho các đề tài lớn hơn sau này.

Trong báo cáo này, chúng em xin được trình bày mô tả về hệ thống thang máy vận chuyển hành lý, các kiến thức – công nghệ đã áp dụng, phân tích thiết kế đề tài. Sau đó là cách thức xây dựng chương trình, minh họa kết quả chạy.

Chúng em xin cảm ơn cô Bành Thị Quỳnh Mai – giảng viên môn học đã giảng dạy và cung cấp các tài liệu cần thiết để hoàn thiện báo cáo này.

Do trong khuôn khổ thời gian ngắn, trình độ chuyên môn, kinh nghiệm và kiến thức của bản thân còn hạn chế, nên chúng em rất mong được sự góp ý của cô và các bạn trong lớp, để đề tài của chúng em ngày càng hoàn thiện hơn và được ứng dụng trong thực tế.

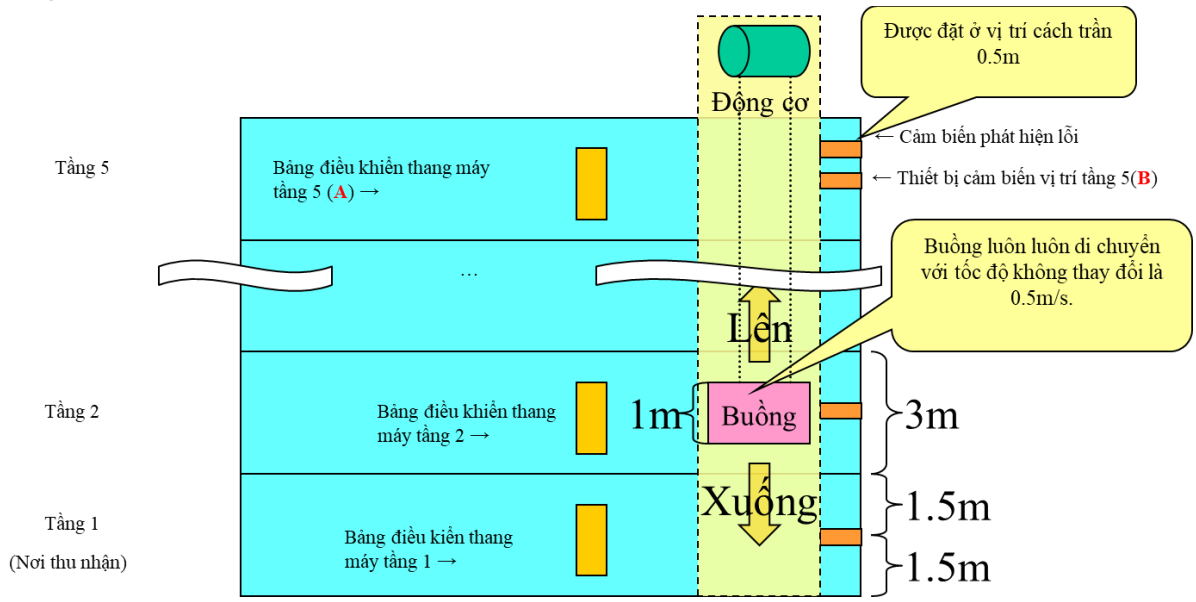
Xin chân thành cảm ơn!

MỤC LỤC

I. GIỚI THIỆU ĐỀ TÀI

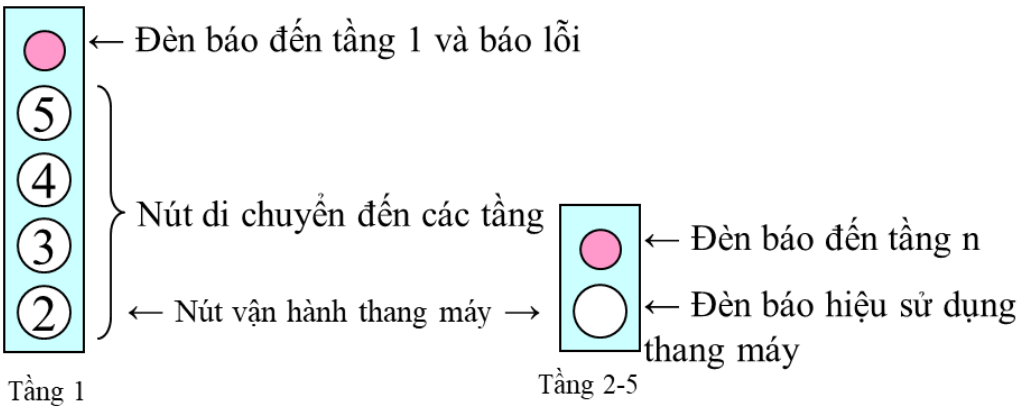
Hệ thống thang máy vận chuyển hành lý (sau đây gọi tắt là thang máy) nằm trong một tòa nhà gồm có 5 tầng.

Nơi thu nhận và vận chuyển hành lý nằm ở tầng 1. Hành lý sau khi được nhận sẽ được chuyển tới một kho cụ thể nằm ở các tầng từ 2 đến 5 bằng hệ thống thang máy. Ngoài ra, chúng ta có thể thực hiện vận chuyển hành lý bằng thang máy từ mỗi tầng từ 2 đến 5 đến nơi thu nhận và vận chuyển hành lý ở tầng 1.



Hình 1. Vị trí của buồng và thiết bị cảm biến vị trí

1. Cấu tạo bảng điều khiển thang máy ở tầng 1 và tầng 2-5



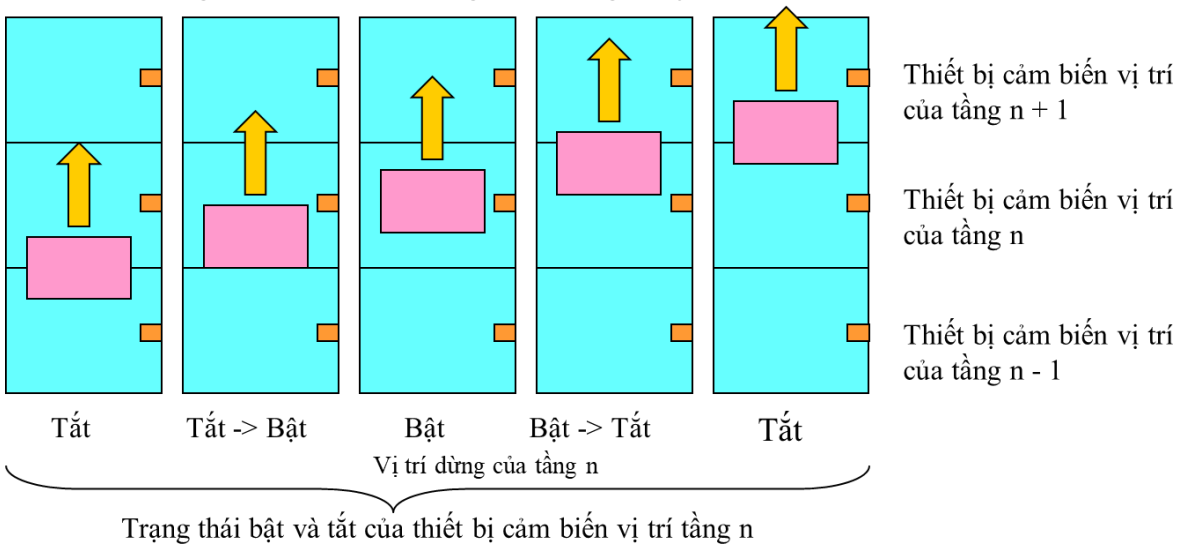
- Nút vận hành thang máy và đèn được thể hiện bằng console.
- Thao tác sau được thực hiện với bảng điều khiển của mỗi tầng:
  - Chọn tầng đích và gọi tầng.

- Hiển thị trạng thái thang máy đang di chuyển, thang máy đến và lỗi.

2. Theo dõi vị trí của thang máy

Thiết bị cảm biến vị trí:

- Thực hiện một chương trình mô phỏng để tạo ra cảm biến bật và tắt tùy theo độ cao của thang máy. Qua đó chúng ta biết được vị trí của thang máy dựa trên trạng thái của cảm biến.
- Thiết bị cảm biến vị trí được bật nếu có một thiết bị cảm biến vị trí nằm giữa phần trên và phần dưới của thang máy.
- Thang máy được tính là dừng ở tầng n nếu thiết bị cảm biến vị trí của tầng đó nằm ở chính giữa thang máy.



Hình 2. Sự thay đổi giá trị trạng thái bật và tắt của thiết bị cảm biến vị trí ở tầng n khi thang máy đi lên

3. Hoạt động của hệ thống điều khiển thang máy

Để có thể điều khiển việc vận chuyển hành lý bằng cách sử dụng thang máy, ta chia thành 2 nhiệm vụ: vận chuyển hành lý từ nơi thu nhận tới mỗi tầng và thu gom hành lý từ mỗi tầng đến nơi thu nhận.

3.1. Vận chuyển hành lý

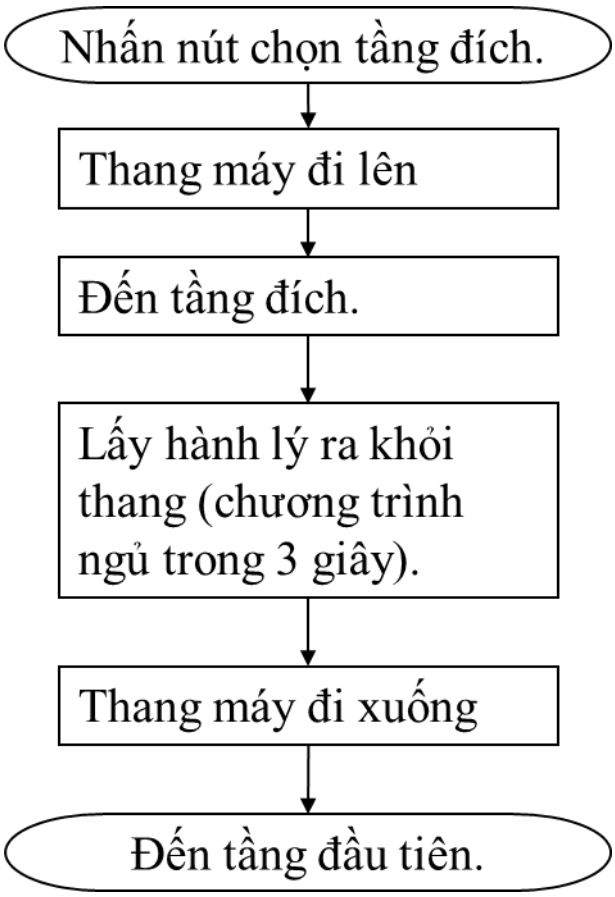
① Cách thức vận hành tại tầng 1 (nơi thu nhận)

Khi hành lý được đặt vào trong thang máy và tầng đích đã được chọn, thang máy sẽ di chuyển đến tầng đích đó.

② Cách thức vận hành tại tầng 2 – 5

Sau khi hoàn thành việc lấy hành lý ra khỏi thang máy khi thang máy đến nơi, thang máy sẽ di chuyển xuống tầng 1 (Quá trình lấy hành lý ra khỏi thang máy được coi như chương trình ngủ 3 giây).

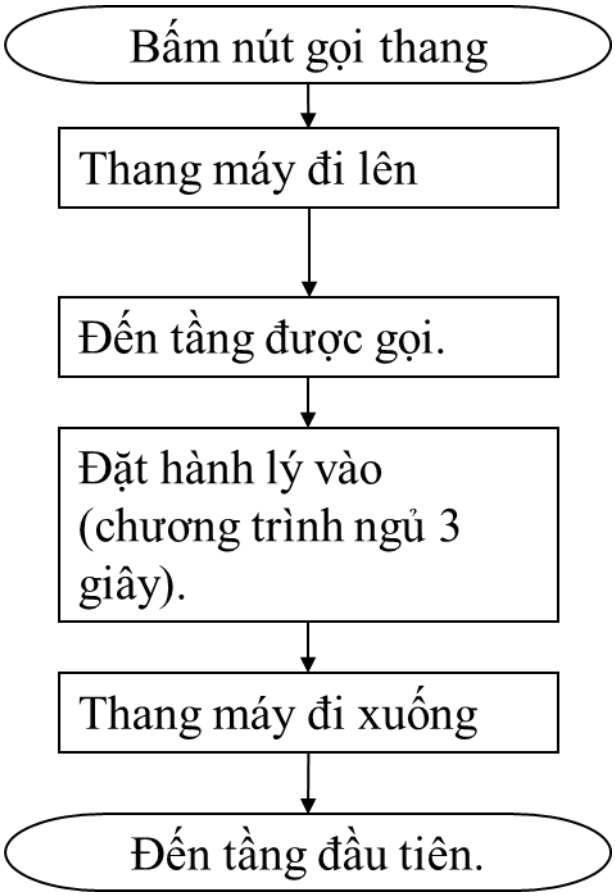
Hoạt động của thang máy khi vận chuyển hành lý:



3.2. Thu gom hành lý

- Ở tầng 2-5, khi ấn nút gọi thang máy để mang hành lý đến địa điểm thu nhận, thang máy sẽ di chuyển đến tầng được gọi.
- Đèn báo hiệu thang máy đến ở tầng đó sẽ sáng lên khi thang máy đến nơi. Sau khi hoàn thành việc đặt hành lý vào trong thang, thang máy sẽ di chuyển xuống tầng đầu tiên. (Quá trình đặt hành lý vào trong thang được coi như chương trình ngủ 3 giây).

Hoạt động của thang máy khi thu gom hành lý:



3.3. Những vấn đề chung đối với việc vận chuyển và thu gom hành lý

- Đèn báo hiệu thang máy đến ở mỗi tầng sẽ sáng khi thang máy dừng ở tầng đó.
- Khi thang máy đang di chuyển, nếu bấm nút gọi thang ở một tầng nào đó, thang máy sẽ đi xuống tầng 1 trước, sau đó mới đi lên tầng được gọi.
- Trong khi thang máy đang di chuyển, nếu bạn bấm nhiều hơn 2 nút gọi thang máy, thang máy sẽ di chuyển theo thứ tự bấm nút của bạn. Ngoài ra, sẽ lưu lại các nút đó vào hoạt động chờ xử lý.

3.4. Phát hiện lỗi của thang máy

Khi cảm biến phát hiện lỗi được bật, ngay lập tức thang máy sẽ dừng và đèn báo hiệu lỗi thang máy ở tầng 1 cũng được bật sáng.

4. Khác

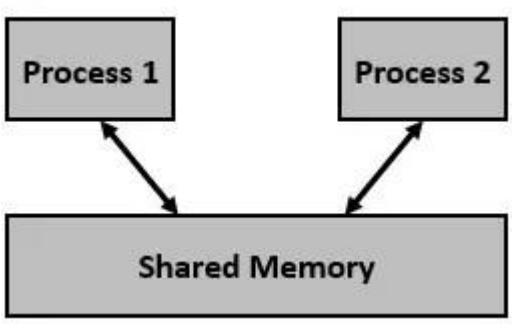
Khi bật nguồn điện hệ thống (khi chương trình bắt đầu), trạng thái ban đầu của thang máy là dừng ở tầng thứ nhất.

II. LÝ THUYẾT SỬ DỤNG

1. Shared memory

1.1. *Khái niệm*

- ***Shared memory (bộ nhớ được chia sẻ)*** là cơ chế giao tiếp liên tiến trình (IPC) có sẵn trong Linux và các hệ thống giống Unix khác, khi một bộ nhớ chung được sử dụng cho hai hoặc nhiều tiến trình khác nhau.
- Trong các cơ chế giao tiếp giữa các tiến trình khác như các pipe (đường ống) hay message queue (hàng đợi tin nhắn), cần thực hiện các bước gửi dữ liệu từ tiến trình này sang trình khác. Tuy nhiên, đối với shared memory, không có bất kỳ hành vi truyền dữ liệu nào cần phải thực hiện ở đây cả, các tiến trình đều có thể truy cập vào bộ nhớ chung. Và giao tiếp được thực hiện thông qua bộ nhớ được chia sẻ này, nơi các thay đổi được thực hiện bởi một tiến trình có thể được xem bởi tiến trình khác.



1.2. *Sử dụng*

Linux cung cấp một số hàm hệ thống được sử dụng để tạo và sử dụng shared memory. Tất cả các lệnh gọi hàm hệ thống dưới đây, nếu xảy ra lỗi và không thành công sẽ trả về cho chúng ta giá trị -1.

❖ **`shmget()`**

```
int shmget(key_t key, size_t size, int shmflg)
```

Giải thích các tham số truyền vào:

- **`key`**: là khóa giúp nhận ra một bộ nhớ được chia sẻ, nó có thể là một giá trị tùy ý hoặc một giá trị được tạo từ hàm `ftok()` - hàm giúp tạo một khóa duy nhất.
- **`size`**: là kích thước của phân đoạn bộ nhớ được chia sẻ.

- **shmflg**: chỉ định một cờ (flag/s) bắt buộc cho bộ nhớ được chia sẻ như IPC_CREAT (tạo phân đoạn bộ nhớ mới) hoặc IPC_EXCL (Được sử dụng với IPC_CREAT để tạo phân đoạn bộ nhớ mới và lời gọi hàm sẽ không thành công, nếu phân đoạn này đã tồn tại). Lưu ý là đối số này còn cần đi kèm với các quyền truy cập vào bộ nhớ chia sẻ, nên cần đặt các quyền phù hợp cho nhu cầu sử dụng.

Sau khi lệnh gọi sẽ hàm này được thực hiện thành công, shmget() trả về một mã định danh cho phân đoạn bộ nhớ được chia sẻ.

❖ **shmat()**

```
void * shmat(int shmid, const void *shmaddr, int shmflg)
```

Trong đó, các đối số:

- **shmid**: là định danh của phân đoạn bộ nhớ chia sẻ và chính là giá trị trả về của lệnh gọi hệ thống shmget().
- **shmadrr**: là chỉ định địa chỉ để gắn phân đoạn bộ nhớ. Thường chúng ta sẽ đặt nó là NULL và khi đó, mặc định hệ thống sẽ chọn địa chỉ phù hợp để gắn phân đoạn.
- **shmflg**: chỉ định một cờ (flag/s) bắt buộc cho bộ nhớ được chia sẻ chẳng hạn như SHM_RND (làm tròn địa chỉ thành SHMLBA) hoặc SHM_EXEC (cho phép nội dung của phân đoạn được thực thi) hoặc SHM_RDONLY (đính kèm phân đoạn với mục đích chỉ đọc, theo mặc định nó là đọc-ghi) hoặc SHM_REMAP (thay thế ánh xạ hiện có trong phạm vi được chỉ định bởi shmaddr và tiếp tục cho đến khi kết thúc phân đoạn).

Sau khi lệnh gọi hàm này được thực hiện thành công, nó sẽ trả về địa chỉ mà phân đoạn bộ nhớ chia sẻ được gắn.

❖ **shmdt()**

```
int shmdt(const void *shmaddr)
```

Sau khi tiến trình của bạn được hoàn thành với việc sử dụng bộ nhớ chia sẻ thì bạn có sẽ cần tách nó ra khỏi bộ nhớ chia sẻ. Điều này được thực hiện bằng cách gọi hàm hệ thống shmdt(). shmaddr là địa chỉ của phân đoạn bộ nhớ chia sẻ được tách ra.

❖ **shmctl()**

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf)
```

Lệnh gọi hệ thống trên giúp thực hiện thao tác điều khiển cho một phân đoạn bộ nhớ chia sẻ với:

- **`shmid`**: là ID định danh cho bộ nhớ chia sẻ.
- **`cmd`**: là chỉ định một lệnh được sử dụng trên bộ nhớ được chia sẻ bao gồm:
    - **IPC_STAT**: Sao chép thông tin về các giá trị hiện tại của từng thành phần trong cấu trúc struct shmid_ds vào cấu trúc được chỉ ra bởi con trỏ buf. Lệnh này yêu cầu quyền đọc đối với phân đoạn bộ nhớ được chia sẻ.
    - **IPC_SET:** Đặt ID người dùng, ID nhóm của chủ sở hữu, quyền, v.v. được trỏ đến theo cấu trúc buf.
    - **IPC_RMID:** Đánh dấu phân đoạn sẽ bị hủy. Phân đoạn chỉ bị phá hủy sau khi tiến trình cuối cùng đã tách nó ra.
    - **IPC_INFO**: Trả về thông tin về giới hạn bộ nhớ dùng chung và các tham số trong cấu trúc được trỏ bởi buf.
    - **SHM_INFO**: Trả về cấu trúc shm_info chứa thông tin về tài nguyên hệ thống được tiêu thụ bởi bộ nhớ được chia sẻ.
- **`buf`**: là một con trỏ đến cấu trúc bộ nhớ dùng chung có tên struct shmid_ds. Các giá trị của cấu trúc này sẽ được sử dụng cho cả set hoặc get theo cmd.

1.3. Ưu, nhược điểm

- **Ưu điểm** của việc sử dụng shared memory là việc không cần đến truyền gửi dữ liệu giúp nó tiết kiệm được chi phí và là phương pháp nhanh nhất giúp trao đổi giữa các tiến trình.
- **Nhược điểm** của phương pháp này chính là nó sẽ gây ra những khó khăn nhất định trong việc bảo đảm sự toàn vẹn dữ liệu (coherence), ví dụ: làm sao biết được dữ liệu mà một tiến trình truy xuất là dữ liệu mới nhất mà tiến trình khác đã ghi? Làm thế nào ngăn cản hai tiến trình cùng đồng thời ghi dữ liệu vào vùng nhớ chung?… Vùng nhớ chia sẻ cần được bảo vệ bằng những cơ chế đồng bộ hóa thích hợp. Bên cạnh đó, shared memory cũng không phải là lựa chọn phù hợp trong các hệ phân tán, để trao đổi thông tin giữa các máy tính khác nhau.

2. Xử lý đa tiến trình (Multiprocessing with fork)

2.1. Multiprocessing

- Multiprocessing chỉ đơn giản là việc khởi tạo và chạy song song đồng thời nhiều tiến trình (process) cùng một thời điểm, mỗi tiến trình thực hiện một tác vụ khác nhau.

- Từ một tiến trình chính (main process) ban đầu sinh ra các tiến trình con để làm việc. Tiến trình chính này thường có nhiệm vụ quản lý vào giao việc cho các tiến trình con, đảm bảo không xảy ra lỗi, nếu có thì thực hiện xử lý tương ứng.
- Thứ tự lập trình multiprocessing thường như sau:
  - Tạo ra tiến trình chính, tiến trình chính gọi các tiến trình con
  - Tiến trình con nhận nhiệm vụ (thường là nhận nhiệm vụ thông qua giao tiếp với tiến trình chính), xử lý công việc liên tục
  - Tiến trình chính quản lý tiến trình con, nếu tiến trình con ngừng đột ngột/treo/hoàn thành nhiệm vụ thì thực hiện xử lý tương ứng (ngừng thì tạo mới, treo thì dừng tiến trình, ...)

2.2. Kĩ thuật Fork()

- **fork()** là 1 cách xử lý multi-process trong Unix, hoạt động theo kiểu sinh ra các tiến trình con xử lý nhiều process bằng cách copy chính nó thông qua hàm fork(). Kết quả của tiến trình con là 0, trong khi kết quả của tiến trình cha là PID của tiến trình con.
- Khi một chương trình gọi Fork thì hệ điều hành sẽ copy ra một chương trình nữa có process ID khác với chương trình hiện tại, và hai chương trình sẽ chạy song song.
  - Chương trình đầu gọi là Process Cha.
  - Chương trình sau gọi là Process Con.
  - Cả hai process này sẽ có tất cả mọi thông tin trên Stack giống nhau. Chỉ có hệ điều hành mới thấy sự khác biệt đó là PID của nó khác nhau.
- Cụ thể khi ta gọi lệnh fork thì lệnh này sẽ trả ra một pid, tức process id.

```
pid_t pid = fork();
```

- Bên trong chương trình của mình, mình có thể kiểm tra giá trị pid này để biết được mình đang ở trong process cha hay trong process con.
  - Nếu giá trị pid mà khác 0, tức là ta đang ở trong process cha.
  - Nếu giá trị pid mà bằng 0, tức là ta đang ở trong process con.
  - Nếu giá trị pid mà bằng -1, tức là thực hiện fork thất bại.

2.3. Ưu điểm

- fork() là kỹ thuật kiểu pipeline nên sẽ không bị ảnh hưởng bởi việc hệ thống có thread-safe hay không. Trong 1 pipeline, chỉ xử lý 1 process, và hoàn toàn độc lập, không share thông tin với pipeline khac, vì thế sẽ không có khả năng gặp deadlock như multi-thread.

- fork() Sử dụng kỹ thuật pipeline, nên khi fork() 1 tiến trình, phần bộ nhớ được cung cấp cho nó là cố định, và sẽ do nó sở hữu cho tới khi pipeline kết thúc. Vì thế không sợ tiến trình sẽ chết giữa chừng do ảnh hưởng 1 sự kiện nào đó phía ngoài pipeline.

III. PHÂN TÍCH THIẾT KẾ

1. Phân tích

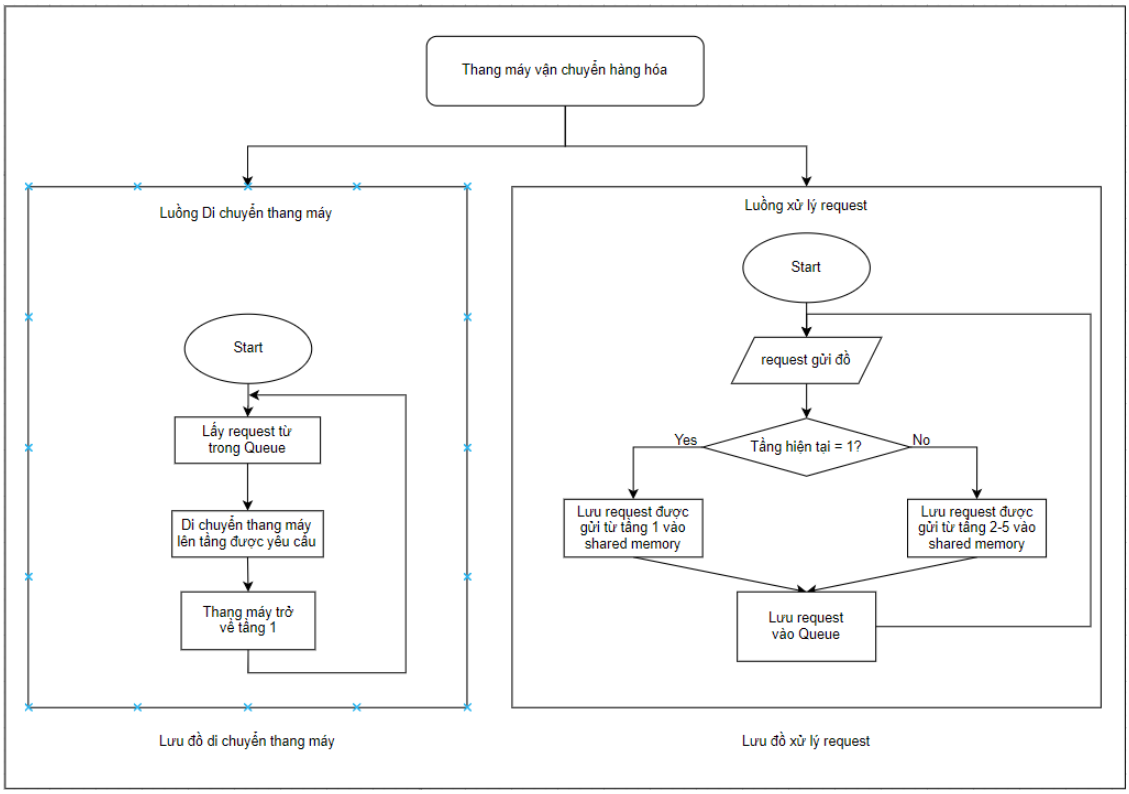
- Có thể coi hệ thống gồm 2 đối tượng chính:
  - Tầng 1: thực hiện việc vận chuyển hàng lên trên.
  - Nhóm tầng từ tầng 2 đến tầng 5: thực việc việc gửi hàng xuống tầng 1.

1.1. Ta sẽ tách 2 đối tượng này thành 2 file code do có nhiều xử lý logic và giao diện khác nhau giữa tầng 1 và các tầng khác.

- Tuy nhiên, vị trí và trạng thái hoạt động của thang máy cần được chia sẻ chung ở mọi tầng, ta sẽ lưu những thông tin này dưới dạng toàn cục.
- Tầng 1 sẽ mặc định là trạng thái khởi tạo của thang máy, khi hàng hóa được chuyển từ tầng 1 lên các tầng khác mà nhận được request từ tầng 2->5, thang máy vẫn cần được tiếp tục hoạt động chứ không thể dừng lại để xử lý những request đó.

1.2. Xử lý đa luồng cho tầng 2->5 để thang máy luôn được hoạt động.

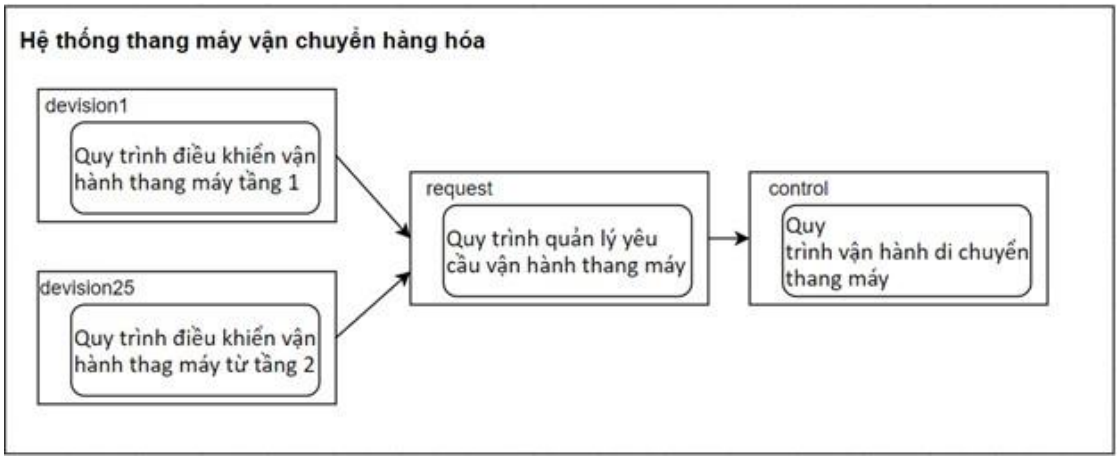
- Hệ thống sẽ gồm 2 luồng logic chính như sau:
  - Logic di chuyển thang máy khi có request.
  - Logic nhận và xử lý 1 request mới.



**Lưu ý: Đây là 2 luồng logic chính và song song nhau, thực tế hệ thống sẽ hoạt động theo luồng riêng của các tầng.*

2. Thiết kế

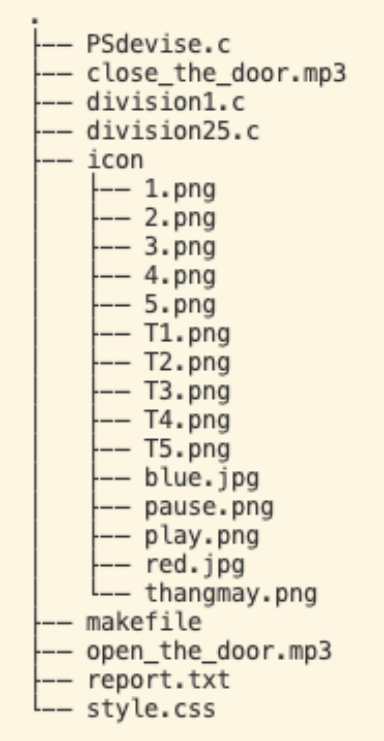
- Theo như thiết kế, ta sẽ tách Tầng 1 và Tầng 2-5 và xử lý riêng logic cũng như giao diện như sau:
  - devision1: Quy trình điều khiển vận hành thang máy tầng 1.
  - devision25: Quy trình điều khiển ban vận hành thang từ máy tầng 2 trở lên.
- Các yêu cầu cũng được chia tương tự:
  - request1: Quy trình quản lý yêu cầu vận hành thang máy từ tầng 1.
  - request25: Quy trình quản lý yêu cầu vận hành thang máy từ tầng 2 trở lên.
- mainProcess: Quy trình vận hành di chuyển thang máy.
- main: Các xử lý về yêu cầu cũng như di chuyển sẽ được nhúng vào main.



- Do thiết kế yêu cầu xử lý đa luồng, hệ thống sẽ áp dụng kỹ thuật Fork tách luồng cho các tầng.
- Những thông tin có tính toàn cục (vị trí, trạng thái) sẽ cần được chia sẻ chung giữa các file trong hệ thống, ở đây ta dùng shared memory để lưu những biến như:
  - Vị trí thang máy hiện tại.
  - Vị trí thang máy được yêu cầu từ tầng 1.
  - Vị trí thang máy được yêu cầu từ tầng 2 trở lên.
- Áp dụng nguyên lý "First come first served", ta sẽ dùng cấu trúc dữ liệu Queue để lưu yêu cầu được gọi từ các tầng.

IV. GIẢI THÍCH CODE

1. Cấu trúc project



2. Chức năng từng file

division1.c	Giao diện thang máy ở tầng 1	
division25.c	Giao diện thang máy từ tầng 2 đến tầng 5	
PSdevise.c	Chương trình chính lưu yêu cầu di chuyển giữa các tầng và in ra trạng thái thang máy (độ cao, đóng mở)	
style.css	Stylesheet chứa config style cho button, label, trạng thái thang máy,...	
open_the_door.mp3 **close_the_door.mp3**	Nhạc phát khi thang máy mở/đóng	
makefile	File script để compile nhanh toàn bộ chương trình	
report.txt	File text giải thích toàn bộ thông tin về project	
icon/	Folder chứa ảnh sử dụng cho giao diện gtk	
	{1-5}.png	Nút số tầng

	background.jpg	Ảnh nền
	thangmay.png	Ảnh thang máy
	{play/pause}.png	Nút di chuyển/ngừng thang máy
	{blue/red}.jpg	Báo hiệu mở/đóng thang máy

3. Giải thích các hàm trong các file chính

division1.c	
nofication()	hàm này được gọi mỗi 0.5s thông qua hàm activate, để cập nhật màu đèn thang máy (button1) và vị trí tầng hiện tại
on_clicked_division{2-5}()	các hàm callback được gọi ở hàm activate để cập nhật trạng thái shared memory, khi nhận được tín hiệu click vào nút tương ứng của thang máy trên giao diện
activate()	hàm khởi tạo giao diện (border, grid, button, style...) và kích hoạt các chức năng tương ứng khi click button, được gọi từ app chính trong main
main()	tạo app giao diện, kích hoạt hàm activate
division25.c	
nofication()	được gọi mỗi 0.5s, để update màu đèn thang máy và chuyển nút stop thành play khi thang máy chạy xong
on_clicked_button()	đổi icon play thành stop tương ứng ở button được click, và lưu thông tin button đó (vị trí tầng muốn đến) vào shared memory (shm[2])
activate()	khởi tạo và hiển thị giao diện các tầng từ 2-5, gọi đến on_click_button khi có sự kiện click xảy ra, và gọi hàm nofication để update giao diện mỗi 0.5s
main()	mỗi tầng từ 2-5 tương ứng với một child process chứa giao diện được khởi tạo và chạy đồng thời, dùng chung shared memory
PSdevise.c (chương trình chính, không chứa giao diện)	
openthedoor() closethedoor()	phát nhạc mỗi khi cửa mở và đóng, nhờ thư viện vlc
putQueue() getQueue()	đưa vào/lấy ra các yêu cầu di chuyển, gửi đồ từ hàng đợi
updateRequest()	gọi đến hàm putQueue để đưa thông tin shared memory vào hàng đợi

main()	chạy vĩnh viễn, liên tục đưa thông tin shared memory vào hàng đợi thông qua hàm updateRequest, lần lượt lấy từng request của hàng đợi ra để xử lý bên trong hàm mainProcess
mainProcess()	đối với mỗi lần di chuyển, in ra trạng thái thang máy (đang đi lên, đi xuống, dừng, đóng mở, độ cao hiện tại), độ cao = (số tầng - 1) * 3 mét, mỗi giây đi được 0.5m

V. HƯỚNG DẪN VÀ DEMO

1. Thư viện cần cài đặt

- **lvlc-dev(lvlc.h)**:
  ```
  sudo apt-get install libvlc-dev
  ```
- **gtk3+**:
  ```
  sudo apt-get install libgtk-3-dev
  ```
- **vlc**:
  ```
  sudo apt-get install vlc
  ```

2. Hướng dẫn compile

- Cách 1: `make`
- Cách 2: Chạy lần lượt
  - `gcc `pkg-config --cflags gtk+-3.0` -o 1 division1.c `pkg-config --libs gtk+-3.0` -w`
  - `gcc `pkg-config --cflags gtk+-3.0` -o 25 division25.c `pkg-config --libs gtk+-3.0` -w`
  - `gcc -o PSdevise PSdevise.c -lvlc`

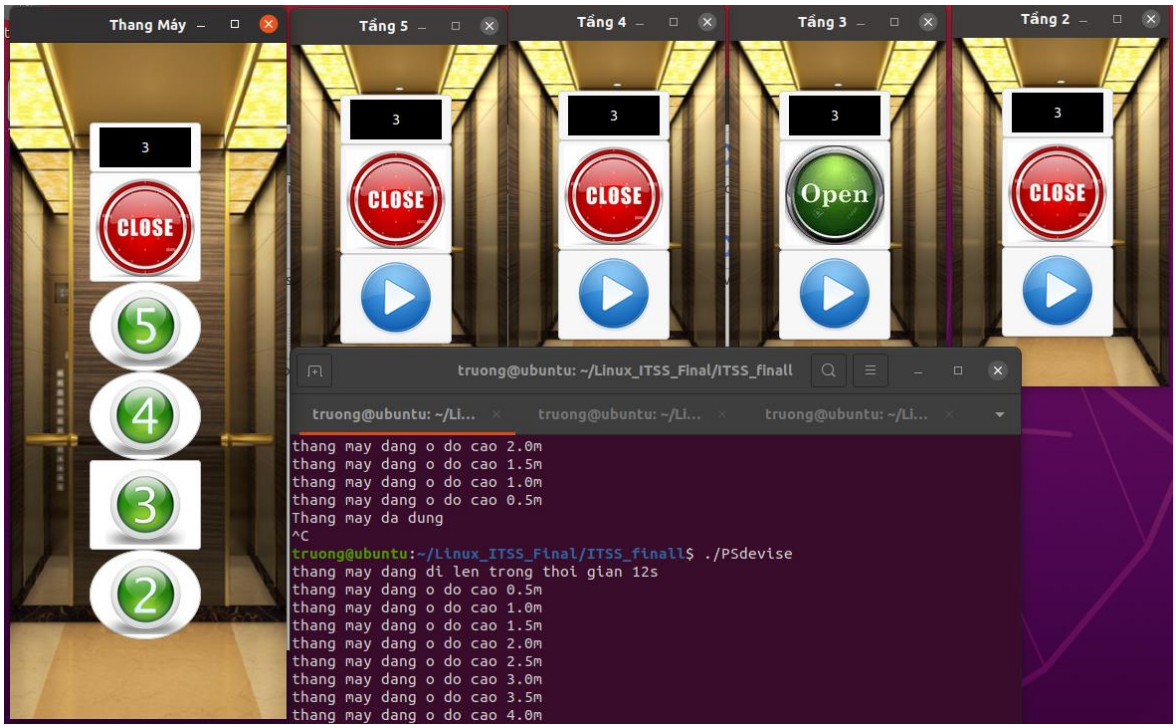
3. Hướng dẫn chạy

Mở lần lượt các file đã compiled:

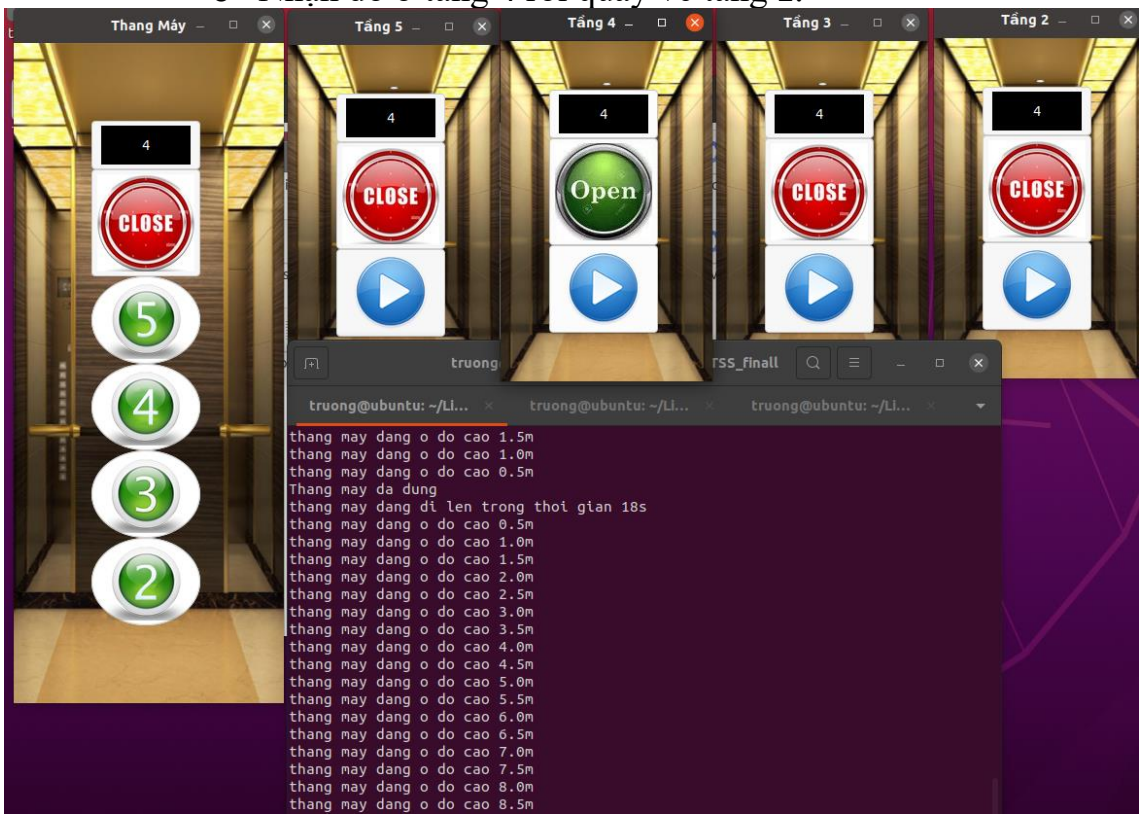
- `./PSdevise`
- `./1`
- `./25`
- Vị trí ban đầu luôn là tầng 1. Di chuyển đến tầng bất kỳ bằng 2 cách:
  - Từ giao diện Elevator, chọn số tầng mong muốn.
  - Từ giao diện Division{2-5} tương ứng số tầng muốn lên, ấn play.
- Theo dõi trạng thái của thang máy thông qua Terminal của PSdevice.

4. Demo

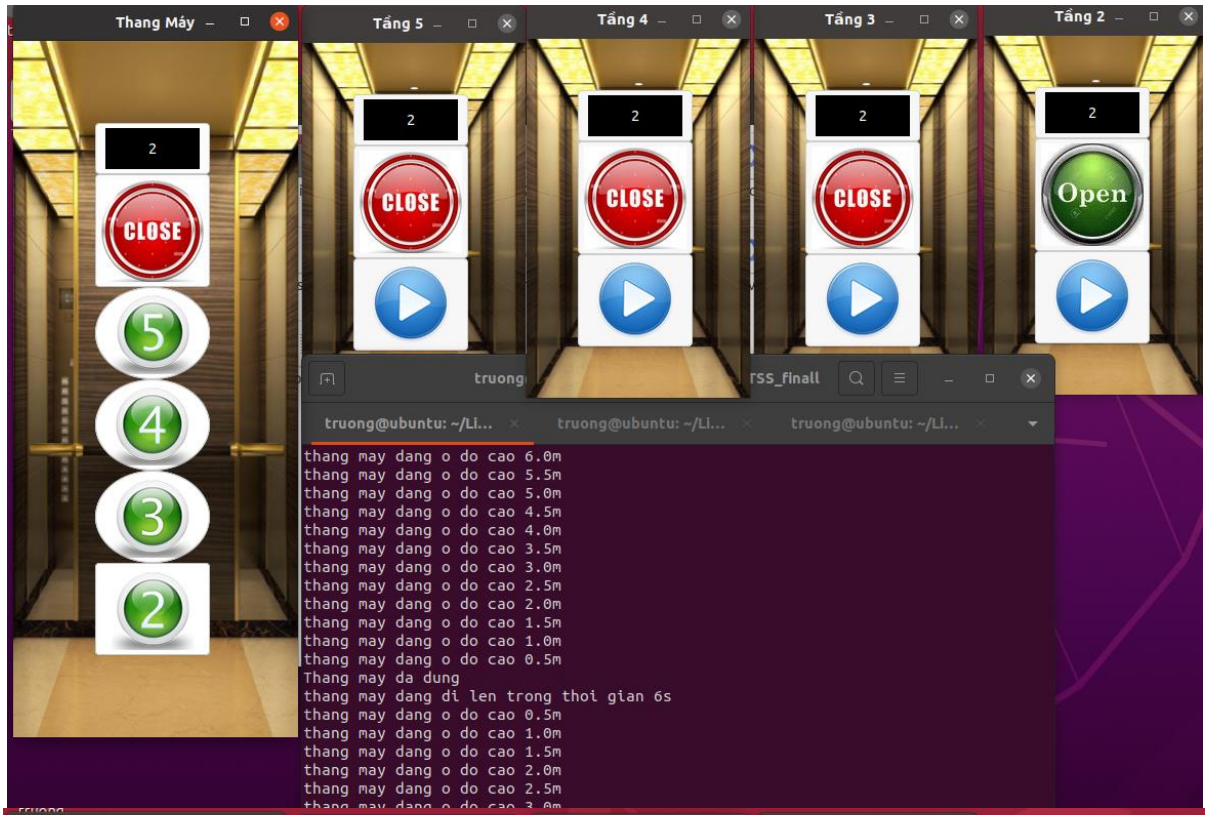
- Chọn **gửi đồ lên tầng 3** từ giao diện **Elevator** ở tầng 1.
  - Đèn xanh **Division3** có nghĩa là vị trí thang máy đang ở tầng 3.
  - Đồng thời vị trí thang máy cũng được cập nhật ở giao diện các tầng.

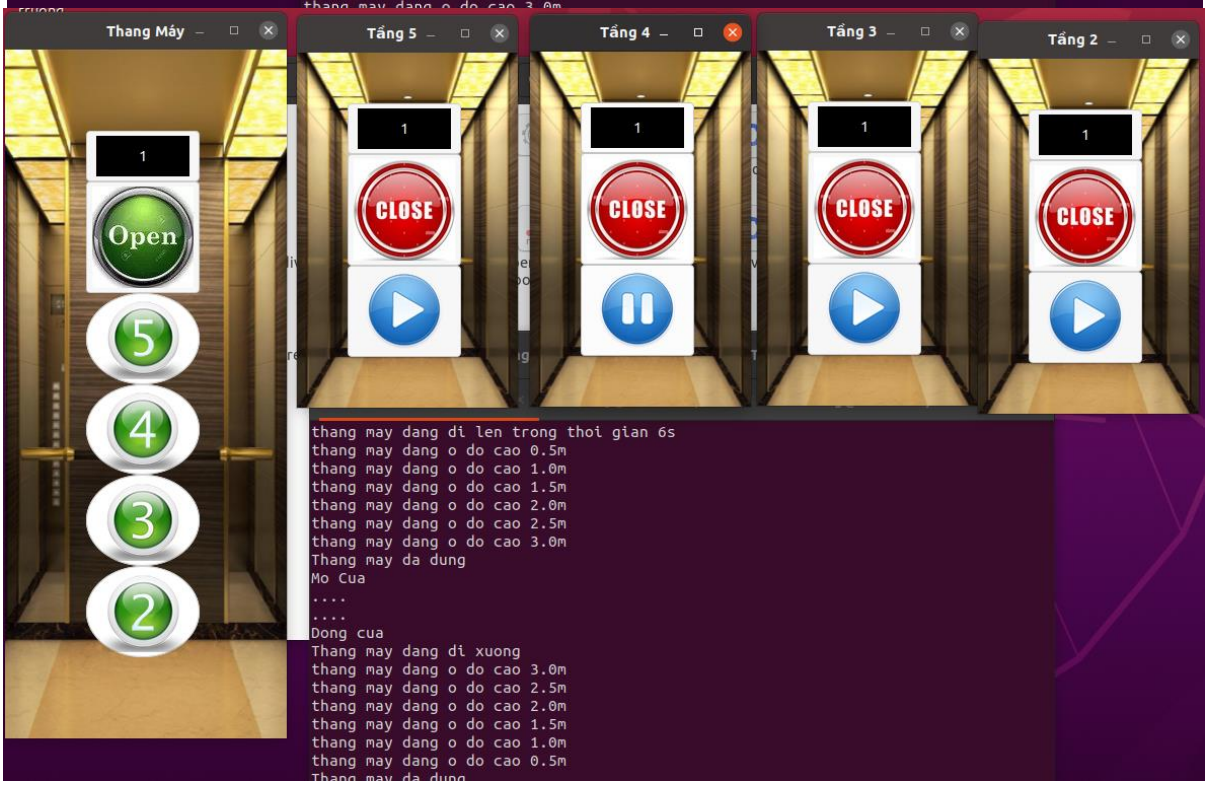


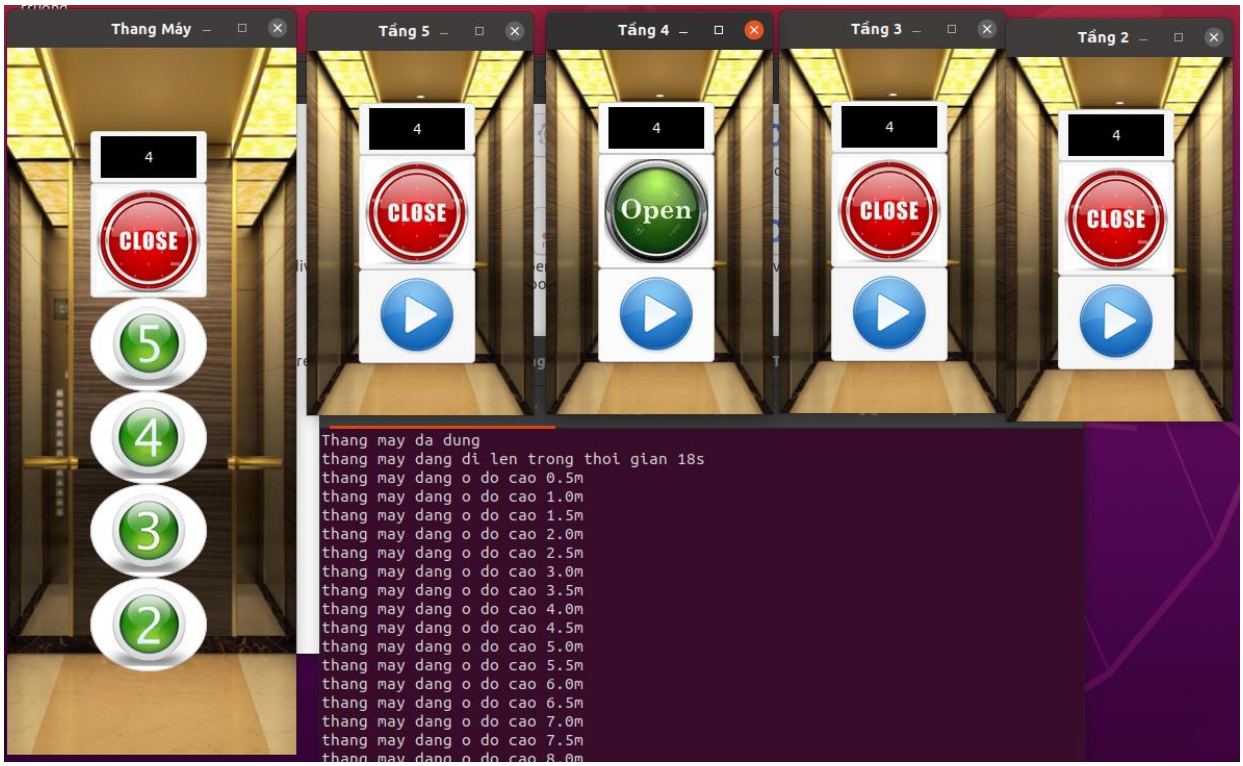
- Chọn **gửi đồ xuống tầng 1** từ giao diện **Division4** ở tầng 4 (nút Play).
  - Thang máy đang ở tầng 1 (Open) bắt đầu đi lên.
  - Nhận đồ ở tầng 4 rồi quay về tầng 1.



- Chọn **đồng thời** gửi đồ lên tầng 2 và nhận đồ từ tầng 4.
  - Các yêu cầu sẽ được thực hiện theo thứ tự FIFO của Queue.
  - Thang máy sẽ thực hiện yêu cầu gửi đồ trước, di chuyển lên tầng 2 và quay lại tầng 1.
  - Sau đó thực hiện tiếp yêu cầu nhận đồ, từ tầng 1 đi lên tầng 4.


Thang Máy
Tầng 5
Tầng 4
Tầng 3
Tầng 2
2
CLOSE
Open
5
4
3
2
truong@ubuntu: ~/Li...
thang may dang o do cao 6.0m
thang may dang o do cao 5.5m
thang may dang o do cao 5.0m
thang may dang o do cao 4.5m
thang may dang o do cao 4.0m
thang may dang o do cao 3.5m
thang may dang o do cao 3.0m
thang may dang o do cao 2.5m
thang may dang o do cao 2.0m
thang may dang o do cao 1.5m
thang may dang o do cao 1.0m
thang may dang o do cao 0.5m
Thang may da dung
thang may dang di len trong thoi gian 6s
thang may dang o do cao 0.5m
thang may dang o do cao 1.0m
thang may dang o do cao 1.5m
thang may dang o do cao 2.0m
thang may dang o do cao 2.5m


Thang Máy
Tầng 5
Tầng 4
Tầng 3
Tầng 2
1
CLOSE
Open
5
4
3
2
thang may dang di len trong thoi gian 6s
thang may dang o do cao 0.5m
thang may dang o do cao 1.0m
thang may dang o do cao 1.5m
thang may dang o do cao 2.0m
thang may dang o do cao 2.5m
thang may dang o do cao 3.0m
Thang may da dung
Mo Cua
....
....
Dong cua
Thang may dang di xuong
thang may dang o do cao 3.0m
thang may dang o do cao 2.5m
thang may dang o do cao 2.0m
thang may dang o do cao 1.5m
thang may dang o do cao 1.0m
thang may dang o do cao 0.5m


Thang Máy
Tầng 5
Tầng 4
Tầng 3
Tầng 2
4
CLOSE
Open
5
4
3
2
Thang may da dung
thang may dang di len trong thoi gian 18s
thang may dang o do cao 0.5m
thang may dang o do cao 1.0m
thang may dang o do cao 1.5m
thang may dang o do cao 2.0m
thang may dang o do cao 2.5m
thang may dang o do cao 3.0m
thang may dang o do cao 3.5m
thang may dang o do cao 4.0m
thang may dang o do cao 4.5m
thang may dang o do cao 5.0m
thang may dang o do cao 5.5m
thang may dang o do cao 6.0m
thang may dang o do cao 6.5m
thang may dang o do cao 7.0m
thang may dang o do cao 7.5m

PHÂN CHIA CÔNG VIỆC

Thành viên	Tỷ lệ đóng góp	Nhiệm vụ
Trần Quang Long	25%	- Phân tích yêu cầu và chức năng đề bài. - Tìm hiểu và code shared memory.
Nguyễn Mạnh Trường	25%	- Phân tích thiết kế, cấu trúc. - Code tổng quan project.
Nguyễn Khánh Toàn	25%	- Phân tích chức năng nhận đồ tầng 2 – 5. - Tìm hiểu gtk và code phần xử lý đa tiến trình cho giao diện tầng 2 – 5.
Nguyễn Gia Minh	25%	- Phân tích chức năng gửi đồ tầng 1. - Tìm hiểu gtk và code phần xử lý giao diện tầng 1.

TÀI LIỆU THAM KHẢO

[1]. Embedded Linux Slides 2020
[2]. https://docs.w3cub.com/gtk~3.24/ch01s02