TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN ĐIỆN TỬ- VIỄN THÔNG

BÁO CÁO
BÀI TẬP LỚN HỆ ĐIỀU HÀNH

ĐỀ TÀI: TÌM HIỂU VỀ LINUX KERNEL

GVHD : TS Phạm Văn Tiến

Sinh viên :

Phạm Đức Thức	20153747
Hoàng Văn Toàn	
Nguyễn Anh Tuấn	20154105
Lê Thành Trung	20153965
Trần Quốc Tuân	20154064
Trần Đình Thịnh	

Hà Nội, 05/2019

MỤC LỤC

Chương 1 TỔNG QUAN

1.1. Đặt vấn đề

Như chúng ta đều biết, nằm giữa phần cứng của hệ thống và các chương trình ứng dụng đó chính là hệ điều hành. Nó cung cấp một môi trường để người sử dụng có thể thực hiện các chương trình ứng dụng và làm cho máy tính dễ sử dụng hơn, thuận lợi hơn và hiệu quả hơn. Hệ điều hành có tác dụng chuẩn hóa giao diện người dùng đối với các hệ thống phần cứng khác nhau, giúp cho phần mềm sử dụng hiệu quả tài nguyên phần cứng và khai thác tối đa hiệu suất của phần cứng.

1.2. Mục tiêu bài tập lớn

Trong phạm vi bài tập lớn này, nhóm sẽ đi vào nghiên cứu, tìm hiểu về hệ điều hành linux và triển khai một chỉnh sửa linux kernel thực hiện một số nhiệm vụ cơ bản, để từ đó hiểu hơn về cách hoạt động của hệ điều hành nói chung và linux kernel nói riêng.

1.3. Nội dung chính của các chương

Chương 1: Tổng quan

Đặt vấn đề, mục tiêu của Bài Tập Lớn

Chương 2: Tìm hiểu và lên ý tưởng

Đọc kĩ yêu cầu của đề tài và vạch ra các công việc, nhiệm vụ phải làm:

+ Hiển thị thông tin các thành viên trong nhóm.

+ Sau đó, phát sinh N tiến trình con.

+ Hiển thị RAM và %CPU tiến trình con chiếm dụng theo chu kỳ.

+ Thông báo tỷ lệ % đóng góp của các thành viên.

Các tiến trình con sẽ thực hiện mã hóa video .mp4 (3-5p) theo định dạng WEBM, và thực hiện chức năng:

+ Mã hóa và ghi lại nội dung video.

+ Gửi đề nghị đến 1 tiến trình con khác playback nội dung đã ghi.

+ Hiển thị các thư viện liên kết động mà toàn bộ chương trình đang sử dụng.

+ Bản thân tiến trình con gửi đề nghị cũng playback nội dung do tiến trình con khác gửi đến.

+ Kết thúc playback, tiến trình con hiển thị thông báo dung lương file video, thời lượng playback, rồi thoát tiến trình.

Chương 3: Thiết kế

Đưa ra mô hình tổng quát của chương trình và phương pháp thực hiện các yêu cầu của từng bước.

Chương 4: Kết luận

Kết quả đạt được, hạn chế và hướng phát triển đề tài.

Chương 2 TÌM HIỂU VÀ LÊN Ý TƯỞNG

2.1. Ý tưởng thiết kế

Lên ý tưởng, chúng em sẽ sử dụng lập trình đa tuyến (multi_thread) để viết một chương trình chính gắn với tiến trình cha mẹ thực hiện:

+ Hiển thị thông tin các thành viên trong nhóm.

+ Sau đó, phát sinh N tiến trình con.

+ Hiển thị RAM và %CPU tiến trình con chiếm dụng theo chu kỳ.

+ Thông báo tỷ lệ % đóng góp của các thành viên.

Các tiến trình con sẽ thực hiện mã hóa video .mp4 (3-5p) theo định dạng WEBM, và thực hiện chức năng:

+ Mã hóa và ghi lại nội dung video.

+ Gửi đề nghị đến 1 tiến trình con khác playback nội dung đã ghi.

+ Hiển thị các thư viện liên kết động mà toàn bộ chương trình đang sử dụng.

+ Bản thân tiến trình con gửi đề nghị cũng playback nội dung do tiến trình con khác gửi đến.

+ Kết thúc playback, tiến trình con hiển thị thông báo dung lương file video, thời lượng playback, rồi thoát tiến trình.

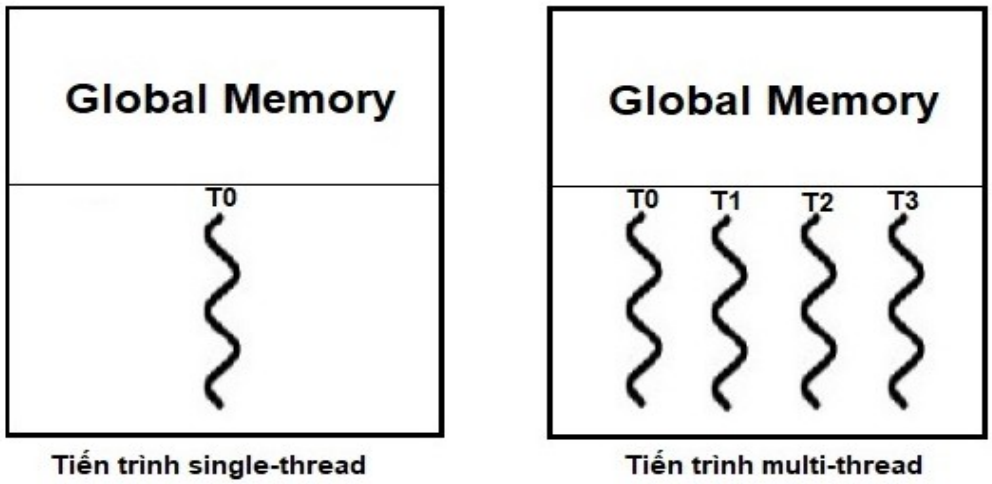
Sau đây là phần trình bày lý thuyết về thread.

2.2. Tổng quan về thread

Thread là một cơ chế cho phép một ứng dụng thực thi đồng thời nhiều công việc (multi-task). Ví dụ một trường hợp đòi hỏi multi-task sau: một tiến trình web server của một trang web giải trí phải phục vụ hàng trăm hoặc hàng nghìn client cùng một lúc. Công việc đầu tiên của tiến trình là lắng nghe xem có client nào yêu cầu dịch vụ không. Giả sử có client A kết nối yêu cầu nghe một bài nhạc, server phải xử lý chạy bài hát client A yêu cầu; nếu trong lúc đó client B kết nối yêu cầu tải một bức ảnh thì server lúc đó không thể phục vụ vì đang bận phục vụ client A. Đây chính là kịch bản yêu cầu một tiến trình cần thực thi multi-task. Qua các bài học về process, ta thấy tiến trình server nói trên có thể giải quyết bài toán này như sau: Server chỉ làm công việc chính là lắng nghe xem có client nào yêu cầu dịch vụ không; khi tiến trình A kết nối, server dùng SYSTEM CALL fork() để tạo ra một tiến trình con chỉ làm công việc client A yêu cầu, trong khi nó lại tiếp tục lắng nghe các yêu cầu từ các client khác. Tương tự, khi client B kết nối, server lại tạo ra một tiến trình con khác phục vụ yêu cầu của client B. Trong trường hợp này thread tỏ ra là thích hợp hơn so với việc sử dụng tiến trình con như đã giải thích ở trên.

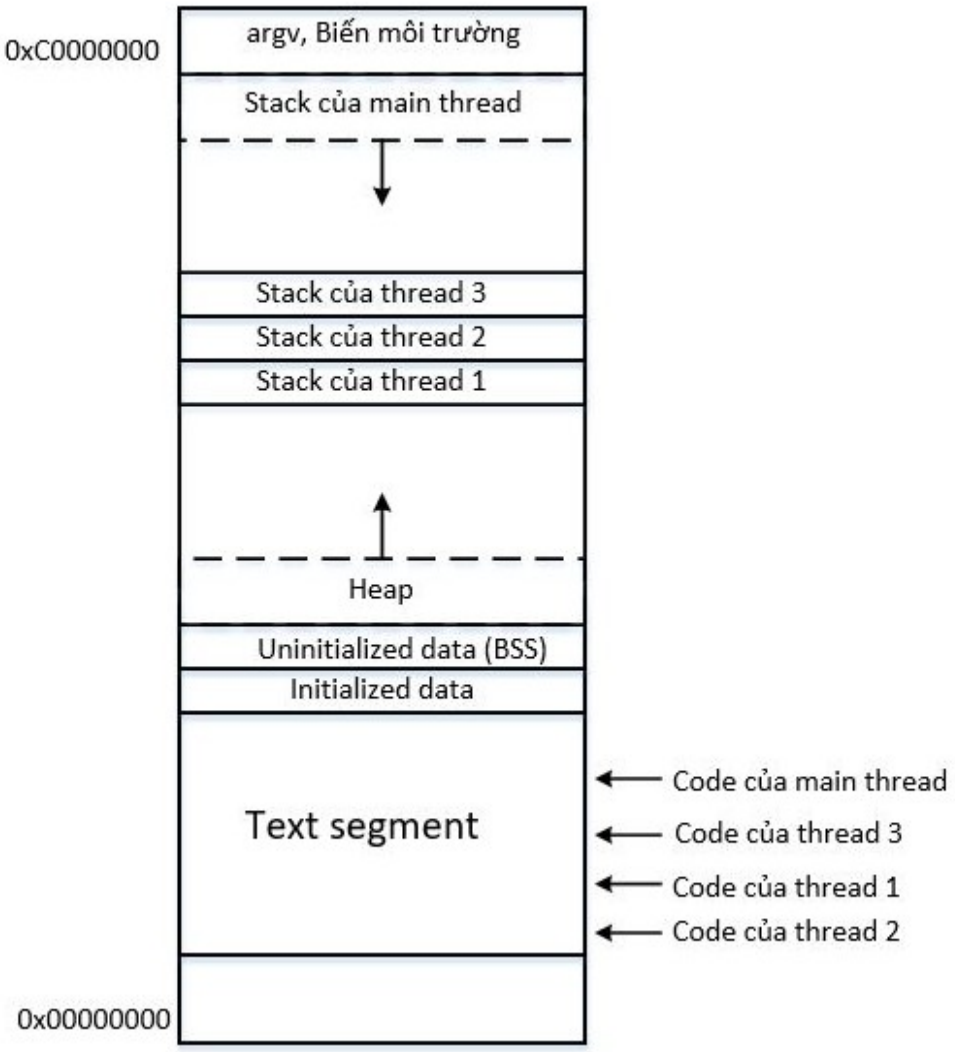
Thread là một thành phần của tiến trình, một tiến trình có thể chứa một hoặc nhiều thread. Hệ điều hành Unix quan niệm rằng mỗi tiến trình khi bắt đầu chạy luôn có một thread chính (main thread); nếu không có thread nào được tạo thêm thì tiến trình đó được gọi là đơn luồng (single-thread), ngược lại nếu có thêm thread thì được gọi là đa luồng (multi-thread). Các thread trong tiến trình chia sẻ các vùng nhớ toàn cục (global memory) của tiến trình bao gồm initialized data, uninitialized data và vùng nhớ heap.

Hình vẽ dưới đây mô tả về một tiến trình đơn luồng (single-thread) và đa luồng (multi-thread):



Hình 2.1 Tiến trình single-thread và multi-thread

Trong hình vẽ trên, một tiến trình có 4 thread, bao gồm 1 main thread (T0) được tạo ra khi tiến trình chạy hàm main(), và 3 thread lần lượt là T1, T2 và T3 được tạo mới trong hàm main(). Bốn thread sử dụng chung vùng nhớ toàn cục (global memory) nhưng mỗi thread có phân vùng stack riêng của mình, cụ thể như hình vẽ dưới đây:



Hình 2.2 Tổ chức bộ nhớ của tiến trình có 4 thread (Linux/x86-32)

Các thread trong tiến trình có thể thực thi đồng thời và độc lập với nhau. Nghĩa là nếu một thread bị block do đang chờ I/O thì các thread khác vẫn được lập lịch và thực thi thay vì cả tiến trình bị block.

2.3. So sánh Process và Thread

Quay trở lại ví dụ phần 2.1 trên, tiến trình server tạo ra các tiến trình con để phục vụ yêu cầu multi-task. Cách này tuy giải quyết được yêu cầu nhưng tồn tại các hạn chế sau đây:

- Việc chia sẻ dữ liệu giữa các tiến trình khá khó khăn. Vì mỗi tiến trình trong Linux có không gian bộ nhớ riêng biệt nên chúng ta phải sử dụng các phương pháp giao tiếp liên tiến trình (IPC) như shared memory, message queue, socket... để chia sẻ dữ liệu.
- Tạo ra một tiến trình mới bằng system call fork() khá "tốn kém" về mặt tài nguyên cũng như thời gian vì phải tạo ra các vùng nhớ riêng biệt cho tiến trình con. Điều này khá quan trọng trong các hệ thống embedded có phần cứng bị hạn chế.

Thread có thể giải quyết được 2 vấn đề trên vì có các ưu điểm sau:

- Chia sẻ dữ liệu giữa các thread trong tiến trình rất đơn giản vì chúng sử có chung không gian bộ nhớ toàn cục. Do vậy, chỉ cần tạo dữ liệu ở trong các vùng nhớ toàn cục này thì các thread đều có thể truy xuất được.
- Việc tạo ra một thread mới nhanh hơn đáng kể so với việc tạo ra một tiến trình mới vì các thread dùng chung nhiều phần không gian bộ nhớ nên chỉ cần tạo không gian bộ nhớ cho những phần riêng thay vì phải nhân bản toàn bộ các vùng nhớ như khi tạo tiến trình con.

Hiển nhiên thread cũng không phải là chìa khóa vạn năng. Việc sử dụng thread cũng có các nhược điểm sau:

- Vì các thread dùng chung vùng nhớ toàn cục nên việc lập trình trên các thread "nguy hiểm" hơn trên process. Nếu một thread gây ra lỗi trên vùng nhớ toàn cục thì sẽ kéo theo các thread khác cũng bị lỗi theo.
- Các thread cùng chia sẻ vùng nhớ toàn cục của một tiến trình (3 GB với hệ thống 32 bit), cụ thể mỗi thread sẽ được cung cấp một vùng nhớ riêng trong tổng thể bộ nhớ của tiến trình. Bộ nhớ của tiến trình tuy là lớn nhưng cũng là một số hữu hạn. Nên một tiến trình cũng bị giới hạn bởi số lượng thread có thể tạo ra hoặc tạo ra các thread cần bộ nhớ lớn.

Cả hai nhược điểm trên không xảy ra trên tiến trình vì mỗi tiến trình có không gian bộ nhớ riêng.

2.4. Posix Thread

Quay lại thời điểm sơ khai của thread, khi đó mỗi nhà cung cấp phần cứng triển khai thread và cung cấp các API để lập trình thread của riêng mình. Điều này gây khó khăn cho các developer khi phải học nhiều phiên bản thread và viết 1 chương trình thread chạy đa nền tảng phần cứng. Trước nhu cầu xây dựng một giao diện lập trình thread chung, tiêu chuẩn POSIX Thread (pthread) cung cấp các giao diện lập trình thread trên ngôn ngữ C/C++ đã ra đời.

2.4.1. Pthread data type

Dưới đây là một số kiểu dữ liệu pthread định nghĩa riêng:

Kiểu dữ liệu	Mô tả
pthread_t	Số định danh của thread (threadID)
pthread_mutex_t	Mutex
pthread_mutexattr_t	Thuộc tính của mutex
pthread_cond_t	Biến điều kiện
pthread_condattr_t	Thuộc tính của biến điều kiện
pthread_key_t	Khóa cho dữ liệu của thread
pthread_attr_t	Thuộc tính của thread

2.5. Hiển thị RAM và phần trăm CPU

Sử dụng lệnh ‘top’ để thực hiện hiển thị RAM và %CPU các tiến trình còn chiếm dụng.

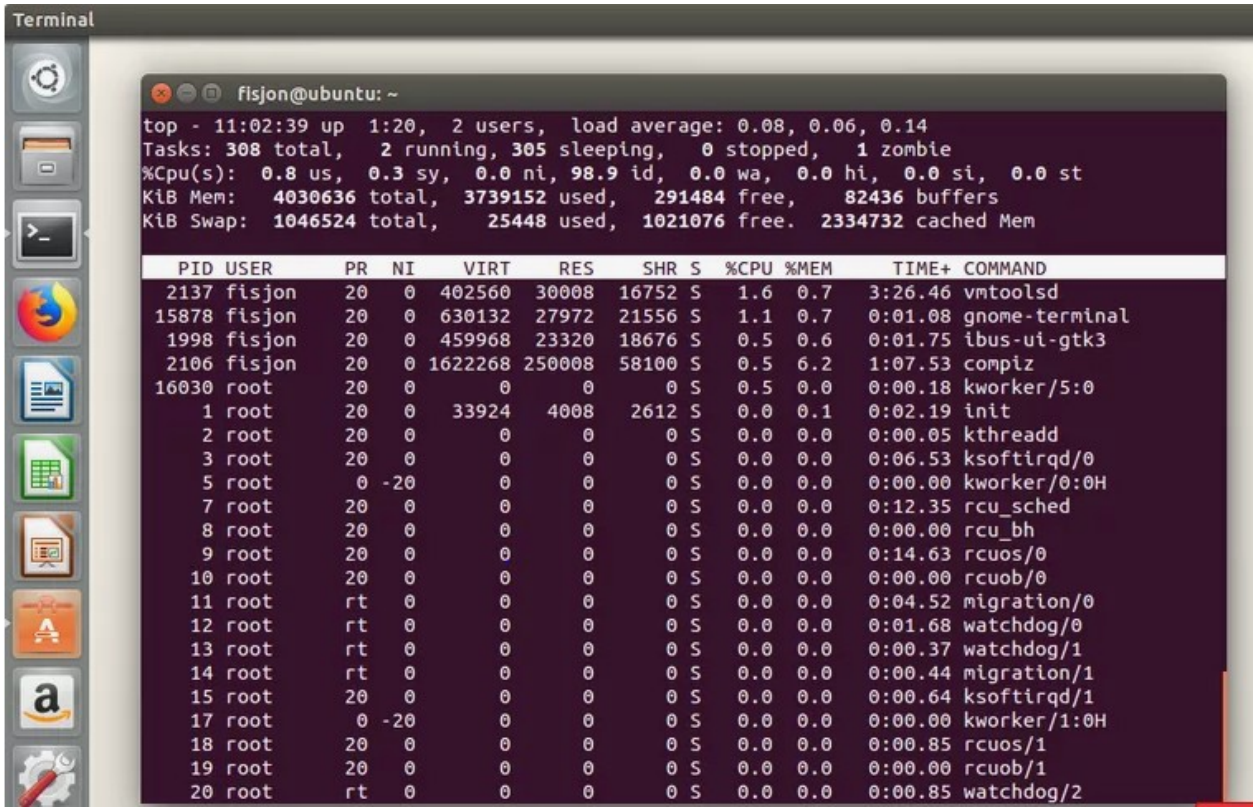
Có thể ví dụ qua một số câu lệnh:

+ ‘ top -d 5’ : làm mới sau mỗi 5s.

+ ‘top -o %CPU’ : hiển thị sắp xếp theo %CPU.

+ ‘top -u gary’ : hiển thị các tiến trình đang chạy.

+ .v..v…



Hình 2.3. Minh họa câu lệnh "top"

Trong ảnh minh họa trên, ta có thể thấy:

+ Dòng 1 hiển thị thời gian, máy tính chạy được bao lâu, số lượng users, tải trung bình.

+ Dòng 2 hiển thị tổng số Task, số lượng Task đang chạy, Task ngủ, Task đang dừng, và các Task zombie.

+ Dòng 3 hiển thị tỉ lệ %CPU sử dụng của user, system, low priority, idle process IO wait, hardware interrupts, softwarre interrupts, steal time.

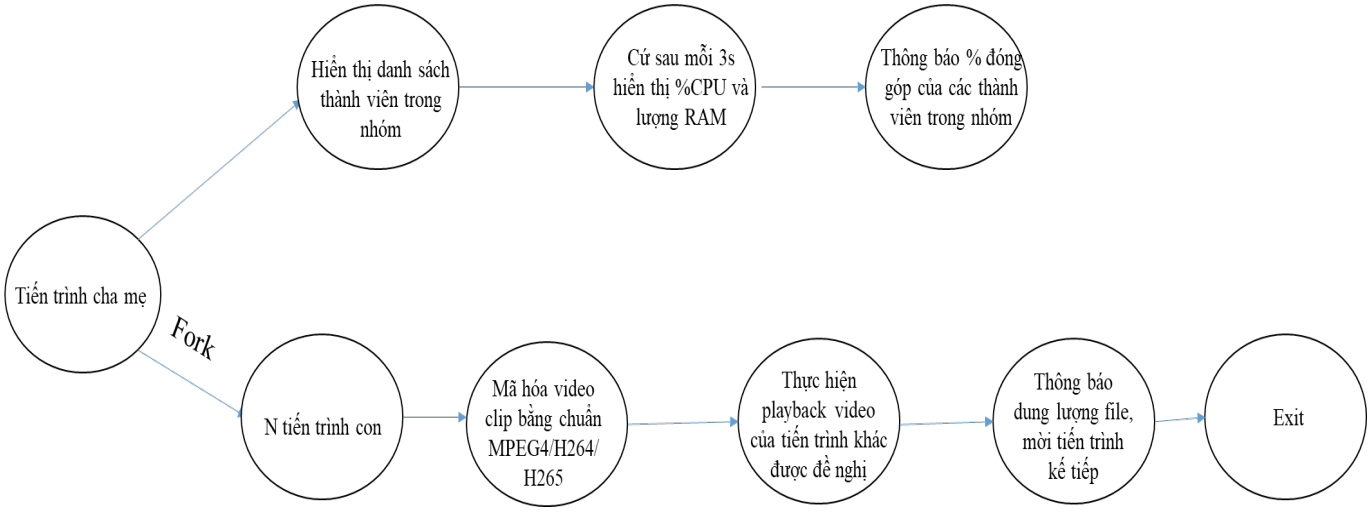
+ Dòng 4 hiển thị total system memory, free memory, memory used, buffer cache.

+ Dòng 5 hiển thị total swap : available, free, used, memory.

+ Bảng chính có: Process ID, User, Priority, Nice level, Virtual memory used by process, Resident memory used by a process, Shareable memory, %CPU, %MEM, Time.

Chương 3 THIẾT KẾ

3.1. Mô hình tổng quát thiết kế chương trình



Hình 3.1. Mô hình tổng quát chương trinh

3.2. Tiến trình cha mẹ

Tiến trình cha mẹ là tiến trình được sinh ra ngay sau khi chương trình được chạy (là hàm main() trong lập trình c). Trong thiết kế của chương trình tiến trình cha mẹ sau khi được sinh ra sẽ thực hiện lần lượt các công việc:

- Thực hiện hiển thị tên của các thành viên trong nhóm
- Sinh ra 5 tiến trình con
- Cứ sau mỗi 3s in ra màn hình %CPU và dung lượng RAM mà tiến trình chiếm đóng
- Chờ các tiến trình con thực hiện xong thì thông báo ra màn hình phần trăm đóng góp của các thành viên trong nhóm
- Kết thúc chương trình

3.2.1. Hiển thị danh sách các thành viên trong nhóm

Chương trình được viết trong bài tập lớn sử dụng ngôn ngữ lập trình c. Phần chương trình thực hiện n ra màn hình danh sách các thành viên trong nhóm:

```
printf("Thanh vien trong nhom \n");
printf("1. Hoang Van Toan - 20153825 \n");
printf("2. Pham Duc Thuc – 20153747 \n");
printf("3.  \n");
printf("4.  \n");
printf("5.  \n");
```

Phần chương trình sử dụng lệnh printf trong c được cung cấp bởi thư viện stdio.h trong c.

3.2.2. Sinh ra 5 tiến trình con

Ở phần này, có hai công việc cần thực hiện là tạo ra một tiến trình trong c sau đó dùng lập trình đa luồng trong C thực hiện công việc cho 5 tiến trình con này được chạy gần như là cùng một lúc (song song với nhau). Để sinh ra một tiến trình con trong c, ta sử dụng lệnh fork(). Khi ta gọi lệnh fork() thì chương trình sẽ tạo ra một tiến trình con. Phần code c thực hiện sinh ra một tiến trình con:

```
fork();   // khởi tạo tiến trình con
wait(NULL);  // tiến trình chính không chờ đợi gì ở tiến trình con
   */ do something */    // Các công việc mà tiến trình con thực hiện
exit(0);  // Kết thúc tiến trình con
```

Khi lệnh fork() được gọi, chương trình chính sẽ yêu cầu hệ điều hành tạo ra một tiến trình con có tất cả các trạng thái có thể có như các tiến trình khác.

Sau khi thực hiện tạo ra các chương trình con bằng lệnh fork(), chúng ra cần cho các tiến trình này chạy song song với nhau bằng lập trình đa tuyến trong c.

3.2.3. Cứ sau mỗi 3s in ra màn hình phần tram CPU và dung lượng RAM

3.2.4. In ra màn hình phần trăm đóng góp của các thành viên trong nhóm

Tương tự như phần in ra màn hình danh sách các thành viên trong nhóm, ta cũng sử dụng lệnh printf() trong c để in dữ liệu ra màn hình.

3.2.5. Kết thúc chương trình

Để kết thúc tiến trình cha mẹ trong c, trong phần chương trình main ta sử dụng lện return 0 để kết thục tiến trình trong c. Khi gặp lênh return thì chương trình sẽ được kết thúc và giải phóng các tài nguyên cho hệ điều hành.

3.3 Tiến trình con

Chương trình chính khi chạy sinh ra 5 tiến trình con, mỗi tiến trình con thực hiện các công việc:

- Mã hóa video clip bằng chuẩn MPEG4/H264/H265
- Thực hiện phát lại video của tiến trình khác khi nhận được yêu cầu của tiến trình đó
- Thông báo ra màn hình dung lượng file, chiều dài thời gian video
- Mời tiến trình con kế tiếp thực hiện
- Kết thúc tiến trình

3.3.1. Mã hóa video clip bằng chuẩn MPEG4/H264/H265

Để thực hiện mã hóa video clip trong c, chúng ta gọi lệnh system(“lệnh trong Ubuntu”). Sauk hi gọi lệnh system thì chương trình sẽ thực hiện gọi lệnh trong Ubuntu và thực hiện câu lệnh đó. Phần code thực hiện mã hóa video clip:

```
system("ffmpeg –i ‘path video input’ ‘path video output’");
```

Trong câu lệnh trên sử dụng lệnh trong Ubuntu ffmpeg –i ‘path video input’ ‘path video output’ thực hiện mã hóa video với đường link là path video input với chuẩn của video input thành video output với đường link cuả video output được viết trong path video output với chuẩn mà mình quy định.

3.3.2. Phát video clip

Tương tự như phần mã hóa video clip ở trên, ở phần này cũng sử dụng lệnh của Ubuntu và sử dụng một chương trinh để phát lại video là Vlc. Phần chương trình phát video trong chương trình c:

```
system("vlc ‘path video’");
```

3.3.3. Thông báo ra màn hình dung lượng file

3.3.4. Mời chương trình kế tiếp thực hiện

Ở đây, với các chương trình con của bài tập lớn đều được viêt bằng một hàm con trong c, nên khi mời chương trình con tiếp theo chạy thì trong chương trình con hiện tại chỉ cần gọi lại chương trình con tiếp theo thực hiện.

3.3.5. Kết thúc chương trình con

Tương tự như kết thúc tiến trình cha mẹ, khi gặp lệnh return trong c thì chương trình con cũng kết thúc và giải phóng tài nguyên lại cho hệ điều hành quản lí.

Chương 4 TỔNG KẾT

4.1. Kết quả đạt được

- Về lí thuyết

  - Nắm được các kiến thức căn bản về hệ điều hành và linux kernel.

  - Thấy được cách thức hoạt động của một kernel.

- Về thực nghiệm

  Triển khai được hệ thống kernel gần đáp ứng được các các yêu cầu của đề tài.

4.2. Hạn chế

- Chưa thực hiện được chính xác các yêu cầu vì thời gian tìm hiểu còn ngắn.

4.3. Hướng phát triển

- Sau khi kết thúc môn học sẽ tiếp tục tìm hiểu về kernel để hoàn thiện bài tập. Hướng tới có thể xây dựng một module driver cho hệ điều hành.