

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC BÁCH KHOA  
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



BÀI TẬP LỚN  
HỆ ĐIỀU HÀNH (TN) (CO2018)

---

**SIMPLE OPERATING SYSTEM**

---

Nhóm 01 - Lớp L12 - HK232

Giảng viên hướng dẫn: Trần Trương Tuấn Phát  
Sinh viên thực hiện: Hoàng Văn Huy - 2211173  
Lâm Nữ Uyển Nhi - 2212429  
Nguyễn Ngọc Quỳnh Nhi - 2212438  
Lê Phan Bảo Như - 2212466  
Lâm Hoàng Tân - 2213054

*Thành phố Hồ Chí Minh, tháng 4 năm 2024*

## Danh sách thành viên

STT	Họ và tên	MSSV	Nhiệm vụ	Đóng góp
1	Hoàng Văn Huy	2211173	- Hiện thực phần Memory, Paging và TLB	20%
2	Lâm Nữ Uyên Nhi	2212429	- Hiện thực phần Scheduling, Memory và Paging	25%
3	Nguyễn Ngọc Quỳnh Nhi	2212438	- Hiện thực phần Scheduling và Memory	15%
4	Lê Phan Bảo Như	2212466	- Hiện thực phần Scheduling - Làm báo cáo, bài thuyết trình và trả lời câu hỏi	20%
5	Lâm Hoàng Tân	2213054	- Hiện thực phần Scheduling, Paging và Synchronization - Làm báo cáo và bài thuyết trình	20%

# Mục lục

<b>Danh sách hình vẽ</b>	<b>2</b>
<b>1 Mở đầu</b>	<b>4</b>
1.1 Giới thiệu . . . . .	4
1.2 Đối tượng nghiên cứu . . . . .	4
1.3 Mục tiêu nghiên cứu . . . . .	5
<b>2 Nội dung</b>	<b>6</b>
2.1 Scheduler . . . . .	6
2.1.1 Tổng quan . . . . .	6
2.1.2 Trả lời câu hỏi . . . . .	7
2.1.3 Hiện thực . . . . .	7
2.1.4 Kết quả chạy thử . . . . .	9
2.2 Memory Management . . . . .	15
2.2.1 Tổng quan . . . . .	15
2.2.2 Trả lời câu hỏi . . . . .	17
2.2.3 Hiện thực . . . . .	19
2.2.4 Kết quả chạy thử . . . . .	26
2.3 Put it all together (Synchronization) . . . . .	29
2.3.1 Tổng quan . . . . .	29
2.3.2 Trả lời câu hỏi . . . . .	29
2.3.3 Kết quả chạy thử . . . . .	30
<b>3 Tổng kết</b>	<b>42</b>
<b>Tài liệu tham khảo</b>	<b>43</b>

## Danh sách hình vẽ

1	Tổng quan về các module chính trong Bài tập lớn . . . . .	5
2	Ví dụ về việc phân nhóm các hàng đợi trong cơ chế MLQ . . . . .	6
3	Nội dung file input <code>sched</code> . . . . .	9
4	Kết quả định thời của file <code>sched</code> . . . . .	10
5	Gantt chart của quá trình định thời <code>sched</code> . . . . .	10
6	Nội dung file input <code>sched_0</code> . . . . .	11
7	Kết quả định thời từ file <code>sched_0</code> . . . . .	11
8	Gantt chart cho quá trình định thời <code>sched_0</code> . . . . .	11
9	Nội dung file input <code>sched_1</code> . . . . .	12
10	Kết quả định thời từ file <code>sched_1</code> . . . . .	12
11	Gantt chart cho quá trình định thời <code>sched_1</code> . . . . .	12
12	Nội dung file input <code>os_1_singleCPU_mlq</code> . . . . .	13
13	Kết quả định thời từ file <code>os_1_singleCPU_mlq</code> (1) . . . . .	13
14	Kết quả định thời từ file <code>os_1_singleCPU_mlq</code> (2) . . . . .	14
15	Gantt chart cho quá trình định thời <code>os_1_singleCPU_mlq</code> . . . . .	14
16	Cấu trúc entry của bảng phân trang . . . . .	16
17	Nội dung file input <code>os_1_singleCPU_mlq_paging</code> . . . . .	26
18	Output của file <code>os_1_singleCPU_mlq_paging</code> (1) . . . . .	26
19	Output của file <code>os_1_singleCPU_mlq_paging</code> (2) . . . . .	27
20	Gantt chart của quá trình định thời <code>os_1_singleCPU_mlq_paging</code> . . . . .	27
21	Khái quát về Hệ điều hành đơn giản . . . . .	29
22	Nội dung file input <code>os_0_mlq_paging</code> . . . . .	31
23	Output của file <code>os_0_mlq_paging</code> . . . . .	31
24	Gantt chart cho quá trình định thời <code>os_0_mlq_paging</code> . . . . .	31
25	Nội dung file input <code>os_1_mlq_paging</code> . . . . .	32
26	Output của file <code>os_1_mlq_paging</code> (1) . . . . .	32
27	Output của file <code>os_1_mlq_paging</code> (2) . . . . .	33
28	Gantt chart của quá trình định thời <code>os_1_mlq_paging</code> . . . . .	33
29	Nội dung file input <code>os_1_mlq_paging_small_1K</code> . . . . .	34
30	Output của file <code>os_1_mlq_paging_small_1K</code> (1) . . . . .	35
31	Output của file <code>os_1_mlq_paging_small_1K</code> (2) . . . . .	35
32	Gantt chart của quá trình định thời <code>os_1_mlq_paging_small_1K</code> . . . . .	36
33	Nội dung file input <code>os_1_mlq_paging_small_4K</code> . . . . .	37

---

34	Output của file <code>os_1_mlq_paging_small_4K</code> (1) . . . . .	37
35	Output của file <code>os_1_mlq_paging_small_4K</code> (2) . . . . .	38
36	Gantt chart của quá trình định thời <code>os_1_mlq_paging_small_4K</code> . . . . .	38
37	Nội dung file input <code>os_1_tlbsz_singleCPU_mlq</code> . . . . .	39
38	Output của file <code>os_1_tlbsz_singleCPU_mlq</code> (1) . . . . .	39
39	Output của file <code>os_1_tlbsz_singleCPU_mlq</code> (2) . . . . .	40
40	Gantt chart của quá trình định thời <code>os_1_tlbsz_singleCPU_mlq</code> . . . . .	40

# 1 Mở đầu

## 1.1 Giới thiệu

Hệ điều hành (Operating System hay được viết tắt là OS) là một phần mềm quản lý và điều khiển tài nguyên của máy tính, bao gồm phần cứng và phần mềm, để cung cấp môi trường làm việc cho người dùng và ứng dụng. Hệ điều hành đóng vai trò trung tâm trong việc quản lý các hoạt động của máy tính, bao gồm việc quản lý bộ nhớ, định thời, giao tiếp với phần cứng, và cung cấp giao diện người dùng.

Hệ điều hành bao gồm khả năng tối ưu hóa hiệu suất của máy tính, giúp người dùng dễ dàng tương tác với các ứng dụng và tài nguyên khác nhau. Nó cung cấp một giao diện người dùng thân thiện, cho phép người dùng thực hiện các tác vụ một cách dễ dàng thông qua các ứng dụng và các công cụ quản lý hệ thống. Hơn nữa, hệ điều hành cung cấp các tính năng bảo mật để bảo vệ dữ liệu và thông tin cá nhân của người dùng khỏi mối đe dọa từ bên ngoài.

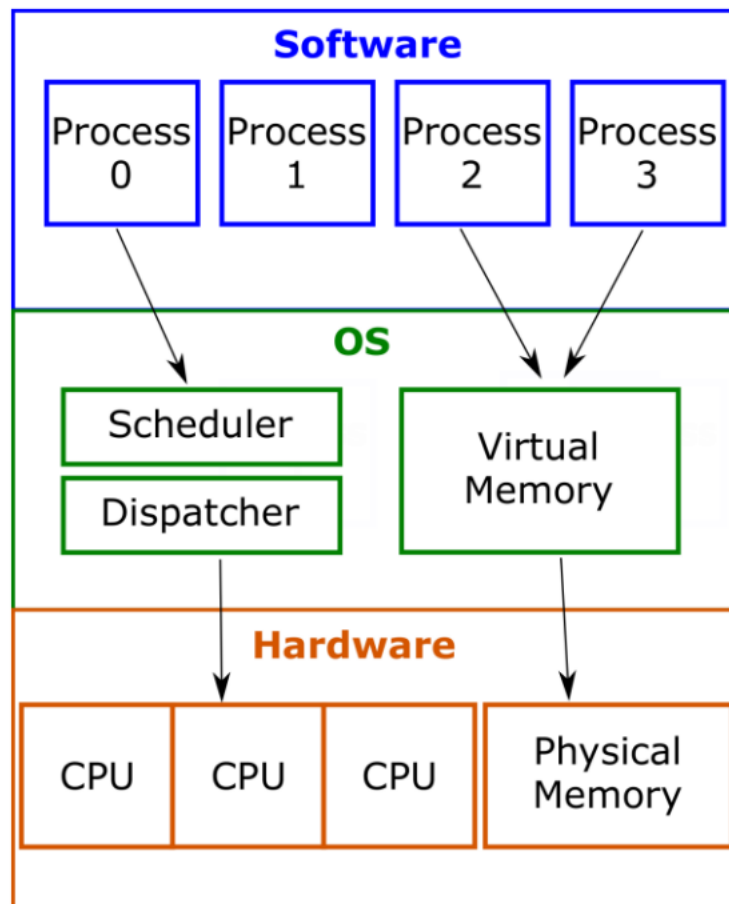
Trong bài tập lớn này, nhóm sẽ tập trung xây dựng một hệ điều hành đơn giản và nghiên cứu về cách hệ điều hành hoạt động.

## 1.2 Đối tượng nghiên cứu

Trong bài tập lớn này, nhóm sẽ tập trung vào hai nội dung chính sau:

- **Scheduler:** Xác định tiến trình (process) nào sẽ được thực thi trên CPU tại một thời điểm nhất định.
- **Virtual Memory engine:** Mỗi tiến trình có một vùng riêng trong bộ nhớ ảo. Nhiệm vụ của virtual memory engine là thực hiện các thao tác ánh xạ và biên dịch địa chỉ của vùng nhớ ảo cung cấp bởi các tiến trình tới địa chỉ vật lý tương ứng trên bộ nhớ vật lý.

Bên cạnh đó, trong suốt tiến trình thực hiện bài tập lớn, nhóm đã sử dụng đến các kiến thức liên quan đến Scheduling, Synchronization và Virtual Memory để hiện thực các module chính được thể hiện ở Hình 1 để thông qua đó, hệ điều hành cho phép nhiều quá trình được tạo ra bởi người dùng chia sẻ và sử dụng các tài nguyên.



**Hình 1:** Tổng quan về các module chính trong Bài tập lớn

### 1.3 Mục tiêu nghiên cứu

Giúp sinh viên có cái nhìn tổng quan hơn về hệ điều hành, đồng thời có thêm kiến thức về Scheduling, Synchronization và Virtual Memory.

Giúp sinh viên hiểu được nguyên lý cũng như cách thức hoạt động của một hệ điều hành đơn giản. Đồng thời hiểu được từng vai trò, ý nghĩa và cách hiện thực của từng module trong hệ điều hành cũng như cách thức hoạt động của nó.

## 2 Nội dung

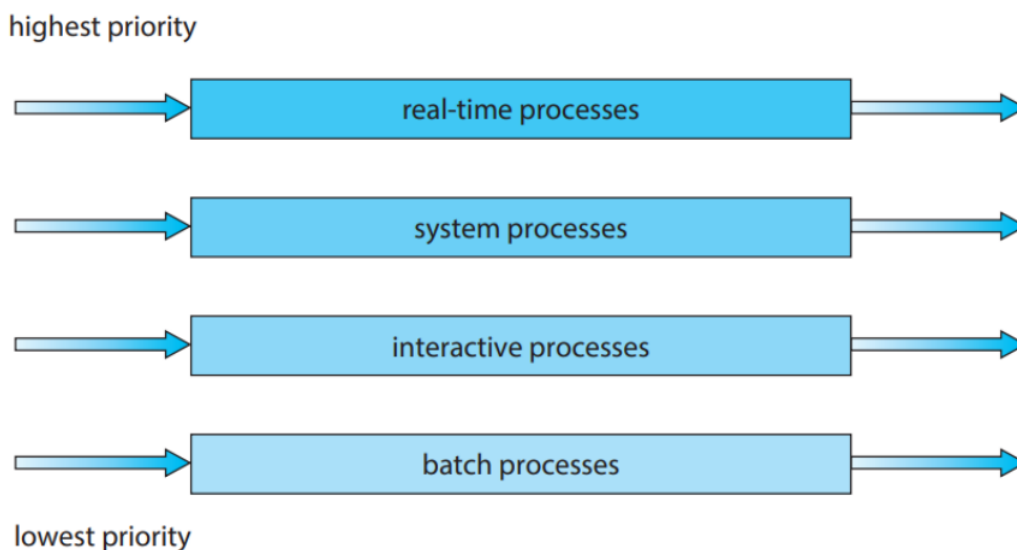
### 2.1 Scheduler

#### 2.1.1 Tổng quan

Chính sách định thời mà hệ điều hành được trang bị là cơ chế Multi Level Queue (MLQ) kết hợp Round Robin.

Các giá trị đặc trưng khi làm việc với MLQ:

- Fixed priority: độ ưu tiên của hàng đợi. Hệ điều hành phục vụ từ hàng đợi có độ ưu tiên cao đến hàng đợi có độ ưu tiên thấp. Giá trị này càng thấp thì có độ ưu tiên càng cao, và vì giá trị này được cố định nên có thể có tình trạng starvation.
- Time slice: mỗi hàng đợi được nhận một khoảng thời gian chiếm CPU và phân phối cho các process trong hàng đợi khoảng thời gian đó.



**Hình 2:** Ví dụ về việc phân nhóm các hàng đợi trong cơ chế MLQ

Trong hệ thống, mỗi tiến trình có một giá trị `prio` nhất định thể hiện độ ưu tiên của một tiến trình và được dùng để xác định `ready_queue` nào sẽ được dùng để chứa tiến trình đó, giá trị `prio` càng thấp thể hiện cho độ ưu tiên càng cao.

Ban đầu, các hàng đợi có độ ưu tiên khác nhau sẽ có số lần sử dụng CPU khác nhau (thể hiện qua giá trị `slot`, tính bằng công thức  $MAX\_PRIO - prio$ ), hàng đợi có độ ưu tiên càng cao thì càng có nhiều lượt sử dụng CPU.

Hàng đợi có độ ưu tiên cao nhất sẽ được chọn để thi thực trước cho đến khi đã sử dụng hết số slot cho phép. Lúc này, CPU sẽ chuyển sang thực thi các tiến trình trong hàng đợi có độ ưu tiên thấp hơn. Tất cả hàng đợi sẽ được áp dụng cơ chế Round Robin, tức là việc



thực thi một hàng đợi chính xác là thay phiên nhau thực thi các tiến trình trong hàng đợi ấy theo một giá trị **time slice** cho trước, một lần thay phiên được tính là một lần sử dụng CPU.

Đến một thời điểm mà tất cả các hàng đợi đều sử dụng hết số slot ban đầu, hệ thống sẽ cấp phát lại số **slot** cho từng hàng đợi theo công thức ban đầu và sau đó lại tiếp thực hiện công việc định thời.

### 2.1.2 Trả lời câu hỏi

**Question:** *What is the advantage of the scheduling strategy used in this assignment in comparison with other scheduling algorithms you have learned?*

**Answer:**

Trong bài tập lớn này, việc sử dụng **hàng đợi ưu tiên** mang lại nhiều ưu điểm so với các thuật toán định thời khác (FCFS, SJR hay RR). Đầu tiên, **hàng đợi ưu tiên** dùng để quản lý các tiến trình dựa trên độ ưu tiên của chúng. Khi một tiến trình mới được thêm vào hệ thống, nó được đưa vào **hàng đợi ưu tiên** dựa trên độ ưu tiên của nó so với các tiến trình khác đang chờ. Điều này cho phép các tiến trình quan trọng được ưu tiên hơn các tiến trình ít quan trọng hơn, dẫn đến hiệu suất và khả năng đáp ứng của hệ thống tốt hơn.

Bên cạnh đó, việc sử dụng **hàng đợi ưu tiên** cho phép định thời trong môi trường ưu tiên (preemptive scheduling). Nói cách khác, khi một tiến trình mới có độ ưu tiên cao hơn được thêm vào, nó có thể ngắt bỏ hoặc tạm dừng tiến trình có độ ưu tiên thấp hơn đang chạy để định thời tiến trình mới bất cứ lúc nào. Điều này giúp công việc định thời được đảm bảo rằng các tiến trình quan trọng, có độ ưu tiên cao được hoàn thành đúng thời hạn và không lãng phí các tài nguyên cho các tiến trình có độ ưu tiên thấp hơn.

### 2.1.3 Hiện thực

Đầu tiên, ta định nghĩa cấu trúc hàng đợi `queue_t` trong file `queue.h`. Cấu trúc này gồm hai thuộc tính, bao gồm mảng lưu trữ các tiến trình và số tiến trình hiện tại.

```
1 struct queue_t {  
2     struct pcb_t * proc[MAX_QUEUE_SIZE];  
3     int size;  
4 };
```

Ta sử dụng hàm `enqueue()` để thêm một tiến trình (PCB) mới vào hàng đợi và hàm `dequeue()` để lấy ra tiến trình tiếp theo từ hàng đợi. Hai hàm trên được hiện thực trong file `queue.c`.

```

1 void enqueue(struct queue_t *q, struct pcb_t *proc)
2 {
3     if (q->size >= MAX_QUEUE_SIZE)
4         return;
5     q->size++;
6     q->proc[q->size - 1] = proc;
7 }

```

```

1 struct pcb_t *dequeue(struct queue_t *q)
2 {
3     /* TODO: return a pcb whose priority is the highest
4      * in the queue [q] and remember to remove it from q
5      */
6     if (empty(q))
7         return NULL; // Return NULL if the queue is empty
8
9     struct pcb_t *head = q->proc[0]; // Get the head of the queue
10
11     // Shift all the remained processes to the left
12     for (int i = 0; i < (q->size - 1); i++)
13         q->proc[i] = q->proc[i + 1];
14
15     q->proc[q->size - 1] = NULL; // Assign NULL value for the tail of the
        ↪ queue
16     q->size--; // Decrease the size of the queue
17     return head; // Return the head of the queue
18 }

```

Ta sử dụng hàm `get_mlq_proc()` để lấy PCB của một tiến trình đang đợi trong hàng đợi ưu tiên `PRIORITY[ready_queue]` và sử dụng hàm `get_proc()` để lấy PCB của một tiến trình đang đợi trong hàng đợi `[ready_queue]`. Hai hàm trên được hiện thực trong file `sched.c`.

```

1 struct pcb_t * get_mlq_proc(void) {
2     struct pcb_t * proc = NULL;
3     /*TODO: get a process from PRIORITY [ready_queue].
4      * Remember to use lock to protect the queue.

```

```

5      * */
6      pthread_mutex_lock(&queue_lock);
7      if(slot_left == 0 || empty(&mlq_ready_queue[curr_prio])) {
8          curr_prio = 0;
9          slot_left = MAX_PRIO - curr_prio;
10     }
11     for (int i = 0; i < MAX_PRIO; i++) {
12         if (slot_left == 0 || empty(&mlq_ready_queue[curr_prio])) {
13             curr_prio = (curr_prio + 1) % MAX_PRIO;
14             slot_left = MAX_PRIO - curr_prio;
15         }
16         else {
17             proc = dequeue(&mlq_ready_queue[curr_prio]);
18             slot_left--;
19             break;
20         }
21     }
22     pthread_mutex_unlock(&queue_lock);
23     return proc;
24 }

```

```

1 struct pcb_t * get_proc(void) {
2     return get_mlq_proc();
3 }

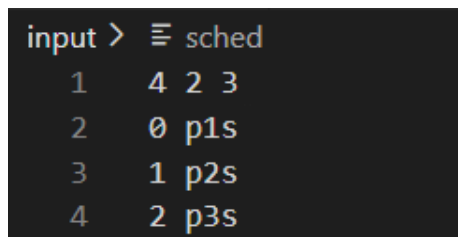
```

### 2.1.4 Kết quả chạy thử

Nhóm đã tiến hành chạy thử một số testcase nhằm kiểm tra việc hiện thực cơ chế Multilevel Queue (MLQ) cho bộ định thời.

#### 2.1.4.1 File `sched`

Nội dung file `sched`:



```

input > ≡ sched
1      4 2 3
2      0 p1s
3      1 p2s
4      2 p3s

```

**Hình 3:** Nội dung file input `sched`

## Kết quả định thời:

```

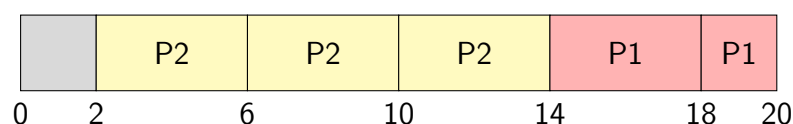
Time slot 0
ld_routine
  Loaded a process at input/proc/p1s, PID: 1 PROC PRIORITY: 1
da add
  CPU 1: Dispatched process 1
Time slot 1
  Loaded a process at input/proc/p2s, PID: 2 PROC PRIORITY: 20
da add
Time slot 2
  CPU 0: Dispatched process 2
  Loaded a process at input/proc/p3s, PID: 3 PROC PRIORITY: 7
da add
Time slot 3
Time slot 4
  CPU 1: Put process 1 to run queue
  CPU 1: Dispatched process 1
Time slot 5
Time slot 6
  CPU 0: Put process 2 to run queue
  CPU 0: Dispatched process 3
Time slot 7
Time slot 8
  CPU 1: Put process 1 to run queue
  CPU 1: Dispatched process 1
Time slot 9
Time slot 10
  CPU 0: Put process 3 to run queue
  CPU 0: Dispatched process 3
  CPU 1: Processed 1 has finished
Time slot 11
  CPU 1: Dispatched process 2
Time slot 12
Time slot 13
Time slot 14
  CPU 0: Put process 3 to run queue
  CPU 0: Dispatched process 3
  CPU 1: Put process 2 to run queue
  CPU 1: Dispatched process 2
Time slot 15
Time slot 16
Time slot 17
  CPU 0: Processed 3 has finished
  CPU 0 stopped
Time slot 18
Time slot 19
  CPU 1: Processed 2 has finished
  CPU 1 stopped

```

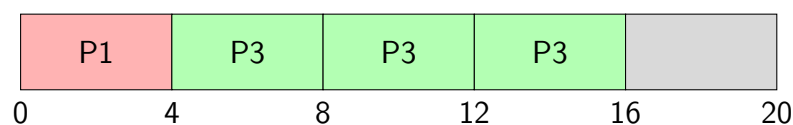
Hình 4: Kết quả định thời của file `sched`

## Gantt chart:

### CPU 0



### CPU 1



Hình 5: Gantt chart của quá trình định thời `sched`

### 2.1.4.2 File `sched_0`

Nội dung file `sched_0`:

```
input > ≡ sched_0
1      2 1 2
2      0 s0
3      4 s1
```

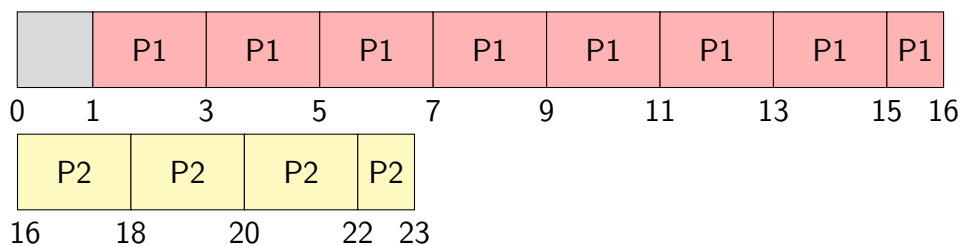
Hình 6: Nội dung file input `sched_0`

Kết quả định thời:

```
Time slot 0      Time slot 15
ld_routine      CPU 0: Put process 1 to run queue
                  CPU 0: Dispatched process 1
                  Loaded a process at input/proc/s0, PID: 1 PROC PRIORITY: 12
da add          Time slot 16
                  CPU 0: Processed 1 has finished
                  CPU 0: Dispatched process 2
Time slot 1      CPU 0: Dispatched process 1
Time slot 2      Time slot 17
Time slot 3      Time slot 18
                  CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 2
                  CPU 0: Put process 1 to run queue
                  CPU 0: Dispatched process 1
Time slot 4      Time slot 19
                  Loaded a process at input/proc/s1, PID: 2 PROC PRIORITY: 20
da add          Time slot 20
                  CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 2
Time slot 5      Time slot 21
                  CPU 0: Put process 1 to run queue
                  CPU 0: Dispatched process 1
Time slot 6      Time slot 22
                  CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 2
Time slot 7      Time slot 23
                  CPU 0: Processed 2 has finished
                  CPU 0: Dispatched process 1
                  CPU 0: Put process 1 to run queue
                  CPU 0: Dispatched process 1
Time slot 8      CPU 0: Put process 1 to run queue
                  CPU 0: Dispatched process 1
Time slot 9      CPU 0: Put process 1 to run queue
                  CPU 0: Dispatched process 1
Time slot 10     CPU 0: Put process 1 to run queue
                  CPU 0: Dispatched process 1
Time slot 11     CPU 0: Put process 1 to run queue
                  CPU 0: Dispatched process 1
Time slot 12     CPU 0: Put process 1 to run queue
                  CPU 0: Dispatched process 1
Time slot 13     CPU 0: Put process 1 to run queue
                  CPU 0: Dispatched process 1
Time slot 14     CPU 0: Put process 1 to run queue
                  CPU 0: Dispatched process 1
```

Hình 7: Kết quả định thời từ file `sched_0`

Gantt Chart:



Hình 8: Gantt chart cho quá trình định thời `sched_0`

#### 2.1.4.3 File `sched_1`

Nội dung file sched\_1 :

```
input > ≡ sched_1
  1    2 1 4
  2    0 s0
  3    4 s1
  4    6 s2
  5    7 s3
```

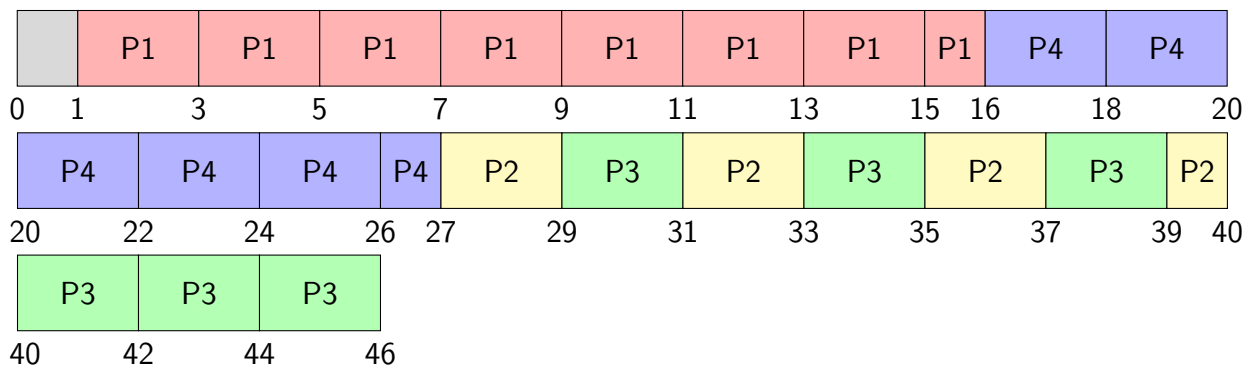
**Hình 9:** Nội dung file input sched\_1

Kết quả định thời:

Time slot 0	ld_routine	Time slot 17		Time slot 36	
	Loaded a process at input/proc/s0, PID: 1 PROC PRIORITY: 12	Time slot 18	CPU 0: Put process 4 to run queue	Time slot 37	CPU 0: Put process 2 to run queue
Time slot 1	CPU 0: Dispatched process 1		CPU 0: Dispatched process 4		CPU 0: Dispatched process 3
Time slot 2		Time slot 19		Time slot 38	
Time slot 3	CPU 0: Put process 1 to run queue	Time slot 20	CPU 0: Put process 4 to run queue	Time slot 39	CPU 0: Put process 3 to run queue
	CPU 0: Dispatched process 1		CPU 0: Dispatched process 4		CPU 0: Dispatched process 2
Time slot 4	Loaded a process at input/proc/s1, PID: 2 PROC PRIORITY: 20	Time slot 21		Time slot 40	CPU 0: Processed 2 has finished
Time slot 5	CPU 0: Put process 1 to run queue	Time slot 22	CPU 0: Put process 4 to run queue		CPU 0: Dispatched process 3
	CPU 0: Dispatched process 1		CPU 0: Dispatched process 4	Time slot 41	
Time slot 6	Loaded a process at input/proc/s2, PID: 3 PROC PRIORITY: 20	Time slot 23		Time slot 42	CPU 0: Put process 3 to run queue
		Time slot 24	CPU 0: Put process 4 to run queue		CPU 0: Dispatched process 3
Time slot 7	CPU 0: Put process 1 to run queue		CPU 0: Dispatched process 4	Time slot 43	
	CPU 0: Dispatched process 1	Time slot 25		Time slot 44	CPU 0: Put process 3 to run queue
	Loaded a process at input/proc/s3, PID: 4 PROC PRIORITY: 7	Time slot 26	CPU 0: Put process 4 to run queue		CPU 0: Dispatched process 3
Time slot 8			CPU 0: Dispatched process 4	Time slot 45	
Time slot 9	CPU 0: Put process 1 to run queue	Time slot 27	CPU 0: Processed 4 has finished	Time slot 46	CPU 0: Processed 3 has finished
	CPU 0: Dispatched process 1		CPU 0: Dispatched process 2		CPU 0 stopped
Time slot 10		Time slot 28			
Time slot 11	CPU 0: Put process 1 to run queue	Time slot 29	CPU 0: Put process 2 to run queue		
	CPU 0: Dispatched process 1		CPU 0: Dispatched process 3		
Time slot 12		Time slot 30			
Time slot 13	CPU 0: Put process 1 to run queue	Time slot 31	CPU 0: Put process 3 to run queue		
	CPU 0: Dispatched process 1		CPU 0: Dispatched process 2		
Time slot 14		Time slot 32			
Time slot 15	CPU 0: Put process 1 to run queue	Time slot 33	CPU 0: Put process 2 to run queue		
	CPU 0: Dispatched process 1		CPU 0: Dispatched process 3		
Time slot 16	CPU 0: Processed 1 has finished	Time slot 34			
	CPU 0: Dispatched process 4	Time slot 35	CPU 0: Put process 3 to run queue		
			CPU 0: Dispatched process 2		

**Hình 10:** Kết quả định thời từ file sched\_1

### Gantt Chart:



**Hình 11:** Gantt chart cho quá trình định thời sched\_1

#### 2.1.4.4 File `os_1_singleCPU_m1q`

```
input > os_1_singleCPU_m1q
1 2 1 8
2 1 s4 4
3 2 s3 3
4 4 m1s 2
5 6 s2 3
6 7 m0s 3
7 9 p1s 2
8 11 s0 1
9 16 s1 0
```

Hình 12: Nội dung file input `os_1_singleCPU_m1q`

#### Kết quả định thời:

```
Time slot 0      ld routine
Time slot 1      Loaded a process at input/proc/s4, PID: 1 PRIO: 4
                  CPU 0: Dispatched process 1
Time slot 2      Loaded a process at input/proc/s3, PID: 2 PRIO: 3
                  CPU 0: Dispatched process 1
Time slot 3
Time slot 4      Loaded a process at input/proc/m1s, PID: 3 PRIO: 2
                  CPU 0: Put process 1 to run queue
                  CPU 0: Dispatched process 1
Time slot 5
Time slot 6      CPU 0: Put process 1 to run queue
                  CPU 0: Dispatched process 1
                  Loaded a process at input/proc/s2, PID: 4 PRIO: 3
Time slot 7      Loaded a process at input/proc/m0s, PID: 5 PRIO: 3
Time slot 8      CPU 0: Put process 1 to run queue
                  CPU 0: Dispatched process 1
Time slot 9      Loaded a process at input/proc/p1s, PID: 6 PRIO: 2
                  CPU 0: Processed 1 has finished
                  CPU 0: Dispatched process 3
Time slot 10
Time slot 11      CPU 0: Put process 3 to run queue
                  CPU 0: Dispatched process 6
                  Loaded a process at input/proc/s0, PID: 7 PRIO: 1
Time slot 12
Time slot 13      CPU 0: Put process 6 to run queue
                  CPU 0: Dispatched process 3
                  Freeing...
Time slot 14
Time slot 15      CPU 0: Put process 3 to run queue
                  CPU 0: Dispatched process 6
Time slot 16      Loaded a process at input/proc/s1, PID: 8 PRIO: 1
Time slot 17      CPU 0: Put process 6 to run queue
                  CPU 0: Dispatched process 3
                  Freeing...
Time slot 18      Freeing...
Time slot 19      CPU 0: Put process 3 to run queue
                  CPU 0: Dispatched process 6
Time slot 20
Time slot 21      CPU 0: Put process 6 to run queue
                  CPU 0: Dispatched process 3
                  Freeing...
Time slot 22      Freeing...
Time slot 23      CPU 0: Processed 3 has finished
                  CPU 0: Dispatched process 6
Time slot 24
Time slot 25      CPU 0: Put process 6 to run queue
                  CPU 0: Dispatched process 6
Time slot 26
Time slot 27      CPU 0: Processed 6 has finished
                  CPU 0: Dispatched process 8
Time slot 28
Time slot 29      CPU 0: Put process 8 to run queue
                  CPU 0: Dispatched process 8
Time slot 30
Time slot 31      CPU 0: Put process 8 to run queue
                  CPU 0: Dispatched process 8
Time slot 32
Time slot 33      CPU 0: Put process 8 to run queue
                  CPU 0: Dispatched process 8
Time slot 34      CPU 0: Processed 8 has finished
                  CPU 0: Dispatched process 7
Time slot 35
Time slot 36      CPU 0: Put process 7 to run queue
                  CPU 0: Dispatched process 7
Time slot 37
Time slot 38      CPU 0: Put process 7 to run queue
                  CPU 0: Dispatched process 7
Time slot 39
Time slot 40      CPU 0: Put process 7 to run queue
                  CPU 0: Dispatched process 7
Time slot 41
Time slot 42      CPU 0: Put process 7 to run queue
                  CPU 0: Dispatched process 7
Time slot 43
Time slot 44      CPU 0: Put process 7 to run queue
                  CPU 0: Dispatched process 7
Time slot 45
Time slot 46      CPU 0: Put process 7 to run queue
                  CPU 0: Dispatched process 7
Time slot 47
Time slot 48      CPU 0: Put process 7 to run queue
                  CPU 0: Dispatched process 7
```

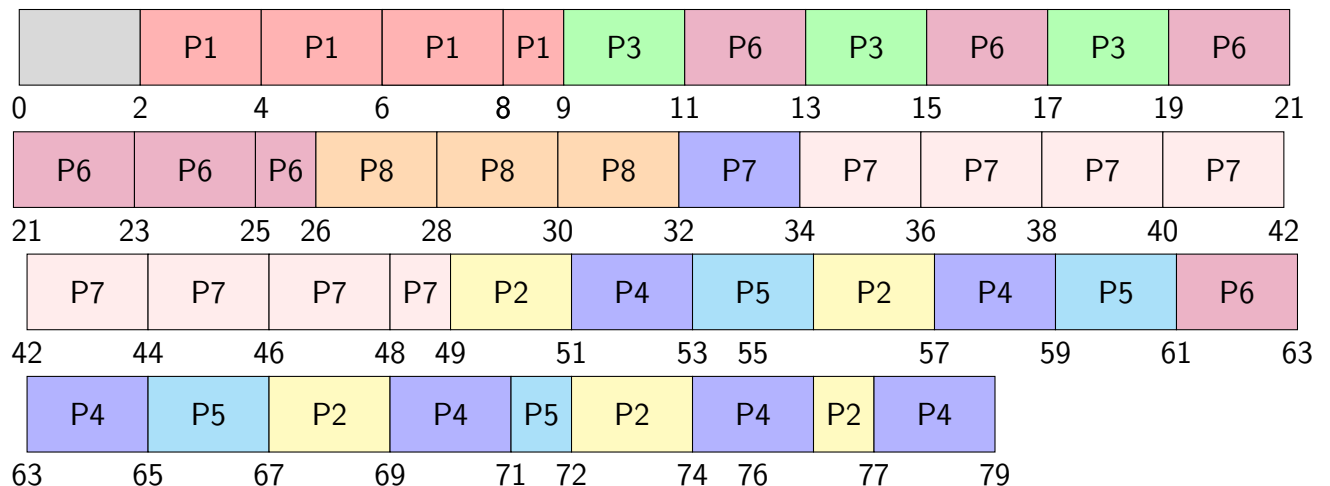
Hình 13: Kết quả định thời từ file `os_1_singleCPU_m1q (1)`

```

Time slot 49      CPU 0: Processed 7 has finished      00000004: 80000005
                  CPU 0: Dispatched process 2      00000008: 80000007
Time slot 50      -----swapping-----
Time slot 51      CPU 0: Put process 5 to run queue
                  CPU 0: Dispatched process 2
Time slot 52      CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 4
Time slot 53      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 5
Time slot 54      CPU 0: Put process 5 to run queue
                  CPU 0: Dispatched process 2
Time slot 55      CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 4
Time slot 56      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 5
Time slot 57      CPU 0: Put process 5 to run queue
                  CPU 0: Dispatched process 2
Time slot 58      CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 4
Time slot 59      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 5
Freeing...
Time slot 60      CPU 0: Put process 5 to run queue
                  CPU 0: Dispatched process 2
Time slot 61      CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 4
Time slot 62      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 5
Time slot 63      CPU 0: Put process 5 to run queue
                  CPU 0: Dispatched process 2
Time slot 64      CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 4
Time slot 65      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 5
Writing...
write region=1 offset=20 value=102
print_pgtbl: 0 - 768
00000000: 80000006
00000004: 80000005
00000008: 80000007
Time slot 67      CPU 0: Put process 5 to run queue
                  CPU 0: Dispatched process 2
Time slot 68      CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 4
Time slot 69      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 5
Time slot 70      CPU 0: Put process 5 to run queue
                  CPU 0: Dispatched process 2
Time slot 71      CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 4
Time slot 72      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 5
Time slot 73      CPU 0: Processed 5 has finished
                  CPU 0: Dispatched process 2
Time slot 74      CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 4
Time slot 75      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 5
Time slot 76      CPU 0: Put process 5 to run queue
                  CPU 0: Dispatched process 2
Time slot 77      CPU 0: Processed 2 has finished
                  CPU 0: Dispatched process 4
Time slot 78      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 5
Time slot 79      CPU 0: Processed 4 has finished
                  CPU 0 stopped

```

Hình 14: Kết quả định thời từ file `os_1_singleCPU_mlq (2)`



Hình 15: Gantt chart cho quá trình định thời `os_1_singleCPU_mlq`



## 2.2 Memory Management

### 2.2.1 Tổng quan

Trong cơ chế quản lý bộ nhớ của hệ điều hành, một lượng vùng nhớ nhất định sẽ được cấp phát cho các tiến trình trên bộ nhớ chính để các tiến trình có thể được thực thi bởi CPU. Tùy vào các cơ chế nhất định mà các tiến trình sẽ được sử dụng bộ nhớ chính theo các cách khác nhau.

**Paging** là cơ chế chia bộ nhớ vật lý thành các khối bằng nhau gọi là **frame**, cũng như chia không gian địa chỉ luận lý (logical address space) thành các **page**. Hệ điều hành sẽ thực hiện ánh xạ các page và frame với nhau, từ đó cung cấp bộ nhớ cho từng tiến trình.

Cơ chế paging được thực hiện dựa trên nhiều thành phần khác nhau. Sau đây là mô tả các thành phần để hiện thực cơ chế paging.

#### 2.2.1.1 The virtual memory mapping in each process

Vùng không gian bộ nhớ ảo được tổ chức như một ánh xạ bộ nhớ cho từng PCB của mỗi một tiến trình. Trong mỗi tiến trình, một bộ nhớ ảo được sử dụng để quản lý địa chỉ cho tiến trình đó. Các tiến trình không dùng chung không gian địa chỉ ảo. Không gian này có thể được chia thành nhiều vùng khác nhau gọi là **virtual memory area**. Từ góc nhìn của tiến trình, vùng địa chỉ ảo này bao gồm các vùng liên tục **vm\_areas**. Trong mỗi area lại được chia ra thành các **memory region** nhỏ hơn. Các region có thể không liền kề nhau.

Muốn áp dụng cơ chế paging, ta cần chia các memory area thành nhiều trang và có một bảng phân trang để quản lý các trang này. CPU tạo ra các địa chỉ, chúng đều có thể truy cập vào một trang chỉ định, được chia ra làm hai phần:

- **Page number:** Cho biết chỉ số trang của địa chỉ này, có thể dùng làm chỉ mục cho bảng phân trang để ánh xạ tới frame tương ứng trong bộ nhớ vật lý.
- **Page offset:** Cho biết vị trí của địa chỉ đó trong cùng 1 trang. Khi kết hợp với base address từ bảng phân trang sẽ tạo ra một địa chỉ vật lý.

#### 2.2.1.2 The system physical memory

Bộ nhớ vật lý được chia ra làm 2 loại: Bộ nhớ RAM và bộ nhớ SWAP.

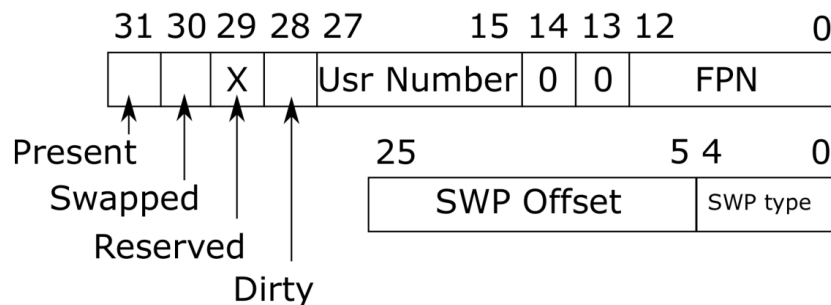
- RAM được CPU truy xuất trực tiếp và các tiến trình chỉ có thể được thực thi sau khi đã nằm trong RAM.
- SWAP là một thiết bị thứ cấp. Để có thể được chạy bởi CPU, các dữ liệu trong SWAP ứng với tiến trình đó phải được nạp lên RAM.

Trong thực tế, kích thước của RAM không đủ lớn để nạp frame cho mọi tiến trình. Do

đó, cơ chế swapping được sử dụng để chuyển các frame qua lại giữa RAM và bộ nhớ vật lý khác (trong Bài tập lớn này sử dụng SWAP). Điều này giúp cho RAM có thể giải phóng nhiều frame cho các tiến trình khác, cũng như cho phép thực thi các chương trình có kích thước lớn hơn nhiều so với kích thước của RAM.

### 2.2.1.3 Paging-based address translation scheme

Cơ chế Dịch (translation) hỗ trợ cho cả segmentation và segmentation with paging. Mỗi tiến trình đều có một bảng phân trang để thực hiện thao tác ánh xạ từ địa chỉ luận lý sang địa chỉ vật lý (từ page tới các frame tương ứng). Cấu trúc của một entry của bảng phân trang trong Bài tập lớn như trong Hình 16.



**Hình 16:** Cấu trúc entry của bảng phân trang

Ý nghĩa của cấu trúc đó như sau:

- Bit 31: present
- Bit 30: swapped
- Bit 29: reserved
- Bit 28: dirty
- Bits 15-27 user-defined numbering if present
- Bits 13-14 zero if present
- Bits 13-14 zero if present
- Bits 0-12 page frame number (FPN) if present
- Bits 25-5: swap offset if swapped
- Bits 4-0: swap type if swapped

### 2.2.1.4 Translation Lookaside Buffer (TLB)

Với các tiến trình có kích thước lớn dẫn đến chi phí cao, TLB được đề xuất để tận dụng khả năng của **cache** nhờ vào đặc điểm tốc độ cao. TLB thường là MEMPHY nhưng hoạt động như một bộ nhớ cache tốc độ cao cho các mục bảng trang (page table entries). Tuy

nhien, bộ nhớ cache có giá thành cao; do đó, nó có dung lượng giới hạn. Sau đây là các công việc cơ bản của TLB:

- TLB accessing method
- TLB setup
- TLB Hit
- TLB Miss

### 2.2.2 Trả lời câu hỏi

**Question:** *In this simple OS, we implement a design of multiple memory segments or memory areas in source code declaration. What is the advantage of the proposed design of multiple segments?*

**Answer:**

Có thể kể đến một vài ưu điểm của việc hiện thực thiết kế nhiều memory segments hay memory areas trong bài tập lớn này như sau:

- Quản lý và sử dụng bộ nhớ một cách hiệu quả hơn, đảm bảo rằng các segment của bộ nhớ được sử dụng theo cách hợp lý và tránh trường hợp sử dụng không hợp lý gây ra xung đột, cũng như tránh gom lại thành 1 segment quá lớn.

- Tối ưu hóa việc phân bổ bộ nhớ cho các tiến trình và giảm thiểu lãng phí bộ nhớ.
- Các segment có thể được ánh xạ sang bộ nhớ vật lý với địa chỉ vật lý khác nhau.

Điều này khiến cho 1 process không nhất thiết phải lưu trữ trong 1 khối liên tục ở bộ nhớ vật lý.

- Không gặp phải vấn đề phân mảnh nội.

**Question:** *What will happen if we divide the address to more than 2-levels in the paging memory management system?*

**Answer:**

Trong hệ thống quản lý bộ nhớ bằng phân trang, khi ta thiết kế địa chỉ thành 2 cấp, ta sẽ có hệ thống phân trang đa cấp. Ở hệ thống này, bộ nhớ ảo sẽ được chia nhỏ thành nhiều vùng.

Khi một tiến trình truy cập bộ nhớ ảo, sẽ truy cập vào vùng địa chỉ cấp trên để tìm ra vùng địa chỉ cấp dưới tương ứng cho đến khi có được địa chỉ trên bộ nhớ vật lý.

**Ưu điểm:**

- Mở rộng khả năng lưu trữ: Bằng cách chia nhỏ địa chỉ thành nhiều cấp, hệ thống có thể hỗ trợ không gian địa chỉ lớn hơn, cho phép quản lý bộ nhớ cho các hệ thống có dung lượng bộ nhớ lớn hơn.

- Phân chia bộ nhớ một cách hiệu quả hơn: Việc có nhiều cấp địa chỉ cho phép

việc tổ chức và quản lý bộ nhớ trở nên hiệu quả hơn, đặc biệt là trong các hệ thống có dung lượng bộ nhớ lớn.

### Nhược điểm:

- Chi phí phát sinh: Việc thêm nhiều cấp địa chỉ có thể tăng độ phức tạp của hệ thống quản lý bộ nhớ, đồng thời cũng tăng chi phí tính toán khi chuyển đổi giữa địa chỉ ảo và địa chỉ vật lý.

- Tăng độ trễ: Việc có nhiều cấp địa chỉ có thể làm tăng độ trễ trong việc truy cập bộ nhớ do việc phải thực hiện nhiều phép tính để xác định địa chỉ vật lý tương ứng, dẫn đến giảm hiệu suất của hệ thống.

**Question:** *What is the advantage and disadvantage of segmentation with paging?*

**Answer:**

### Ưu điểm:

- Tiết kiệm bộ nhớ, sử dụng bộ nhớ hiệu quả.
- Mang các ưu điểm của giải thuật phân trang:
  - Đơn giản việc cấp phát vùng nhớ
  - Khắc phục được phân mảnh ngoại  $\Rightarrow$  Giải quyết vấn đề phân mảnh ngoại của giải thuật phân đoạn

### Nhược điểm của giải thuật:

- Phân mảnh nội của giải thuật phân trang vẫn còn.
- Mức độ phức tạp sẽ cao hơn nhiều so với phân trang.
- Bảng phân trang cần được lưu trữ liên tục trong bộ nhớ.

**Question:** *What will happen if the multi-core system has each CPU core can be run in a different context, and each core has its own MMU and its part of the core (the TLB)? In modern CPU, 2-level TLBs are common now, what is the impact of these new memory hardware configurations to our translation schemes?*

**Answer:**

- Cô lập của TLB: Mỗi lõi CPU có MMU và TLB riêng đồng nghĩa với việc TLB được cô lập giữa các lõi. Điều này đảm bảo rằng TLB của một lõi không can thiệp TLB của một lõi khác. Tuy nhiên, nó cũng dẫn đến việc không thể sử dụng TLB trên lõi khác hỗ trợ khi TLB trên lõi này bị miss.

- Tăng dung lượng TLB: Việc mỗi lõi có TLB riêng giúp tăng tổng dung lượng TLB của hệ thống, từ đó tỷ lệ hit TLB có thể được tăng lên và hiệu suất truy cập bộ nhớ cải thiện.

- Chi phí đồng bộ hóa: Việc có nhiều TLB trong hệ thống sẽ đặt ra yêu cầu rằng tính nhất quán phải được đảm bảo giữa các TLB của các lõi khác nhau. Chi phí đồng bộ

hóa này có thể ảnh hưởng đến hiệu suất hệ thống.

- Tăng độ phức tạp trong việc quản lý bộ nhớ: Quản lý các ánh xạ bộ nhớ và tính nhất quán giữa các TLB trở nên phức tạp hơn với nhiều TLB trong hệ thống. Điều này có thể đòi hỏi các thuật toán và giao thức tinh vi để duy trì tính nhất quán và nhất quán giữa các TLB.

Tóm lại, việc cung cấp cả MMU và TLB riêng cho mỗi lõi trong hệ thống đa lõi có thể cải thiện hiệu suất truy cập bộ nhớ nhưng cũng đặt ra thách thức trong việc quản lý tính nhất quán và đồng bộ hóa bộ nhớ giữa các lõi. Việc tận dụng và quản lý TLB một cách hiệu quả trở nên rất quan trọng để tối đa hóa hiệu suất hệ thống trong các cấu hình như vậy.

### 2.2.3 Hiện thực

Ta sử dụng hàm `MEMPHY_dump` để dump nội dung từ `mp->storage` ra màn hình để theo dõi nội dung bộ nhớ. **Hàm này được hiện thực trong file `mm-memphy.c`.**

```
1 int MEMPHY_dump(struct memphy_struct * mp) {
2     /*TODO dump memphy content mp->storage
3      *   for tracing the memory content
4      */
5     for (int i = 0; i < mp->maxsz; i++){
6         if (mp->storage[i] != 0) printf("Index %d: %d\n", i, mp->storage[i]);
7     }
8     return 0;
9 }
```

Các hàm dưới đây được hiện thực trong file `mm-vm.c`:

Ta sử dụng hàm `__alloc()` để cấp phát một vùng nhớ cho tiến trình. Trước tiên, nó thử tìm một vùng nhớ trống có đủ kích thước để cấp phát cho tiến trình bằng cách gọi hàm `get_free_vmrg_area()`. Nếu không tìm thấy vùng nhớ trống đủ lớn, hàm sẽ cố gắng tăng kích thước của vùng nhớ hiện có bằng cách gọi hàm `inc_vma_limit()`. Hàm sẽ cấp phát vùng nhớ tại địa chỉ đã được cấp phát trước đó; nếu xảy ra lỗi, hàm sẽ xử lý việc quản lý vùng nhớ và trả về giá trị khác không để báo lỗi. Giá trị vùng nhớ được cấp phát dư ra sẽ được đưa vào danh sách các vùng nhớ trống của tiến trình.

```
1 int __alloc(struct pcb_t *caller, int vmaid, int rgid, int size, int
   ↪ *alloc_addr)
```

```
2 {
3     if (size <= 0)
4         return -1;
5     /*Allocate at the topproof */
6     caller->mm->allocated[rgid] = 1;
7     struct vm_rg_struct rgnode;
8
9     if (get_free_vmrg_area(caller, vmaid, size, &rgnode) == 0)
10    {
11        caller->mm->symrgtbl[rgid].rg_start = rgnode.rg_start;
12        caller->mm->symrgtbl[rgid].rg_end = rgnode.rg_end;
13
14        *alloc_addr = rgnode.rg_start;
15
16        return 0;
17    }
18
19    /* TODO get_free_vmrg_area FAILED handle the region management (Fig.6)*/
20
21    /*Attempt to increate limit to get space */
22    struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, vmaid);
23    int inc_sz = PAGING_PAGE_ALIGNSZ(size);
24    // int inc_limit_ret
25    int old_sbrk;
26
27    old_sbrk = cur_vma->sbrk;
28
29    /* TODO INCREASE THE LIMIT
30        * inc_vma_limit(caller, vmaid, inc_sz)
31        */
32    inc_vma_limit(caller, vmaid, inc_sz);
33
34    /*Successful increase limit */
35    caller->mm->symrgtbl[rgid].rg_start = old_sbrk;
36    unsigned long new_sbrk = caller->mm->symrgtbl[rgid].rg_end = old_sbrk +
37        ↪ size;
```

```
38 *alloc_addr = old_sbrk;
39
40 if (new_sbrk < cur_vma->sbrk)
41 {
42     struct vm_rg_struct *remain_rg = malloc(sizeof(struct vm_rg_struct));
43     remain_rg->rg_start = new_sbrk;
44     remain_rg->rg_end = cur_vma->sbrk;
45     remain_rg->rg_next = NULL;
46     enlist_vm_freerg_list(caller->mm, remain_rg);
47 }
48
49 return 0;
50 }
```

Ta sử dụng hàm `__free()` giải phóng không gian lưu trữ được liên kết với một khu vực nhất định (được xác định bởi **rgid**) trong vùng nhớ ảo (có id là **vmaid**) của tiến trình.

```
1 int __free(struct pcb_t *caller, int vmaid, int rgid)
2 {
3     struct vm_rg_struct *rgnode = malloc(sizeof(struct vm_rg_struct));
4
5     if(rgid < 0 || rgid > PAGING_MAX_SYMTBL_SZ)
6         return -1;
7
8     /* TODO: Manage the collect freed region to freerg_list */
9     rgnode = get_symrg_byid(caller->mm, rgid);
10    /*enlist the obsoleted memory region */
11    enlist_vm_freerg_list(caller->mm, rgnode);
12
13    return 0;
14 }
```

Ta sử dụng hàm `pg_getpage` để lấy giá trị số trang từ bộ nhớ RAM (**physical memory**) thông qua bảng phân trang. Nếu trang chưa được load vào bộ nhớ RAM, hàm sẽ thực hiện việc swap nội dung của một trang nạn nhân (**victim page** - được chọn theo chiến lược FIFO) hiện tại trong RAM vào bộ nhớ thứ cấp và swap ngược lại giá trị cần truy cập ở bộ nhớ thứ cấp vào trang mới bị swap trong bộ nhớ RAM. Sau khi hoàn thành, hàm cập nhật bảng trang (**page table**) để đánh dấu trang hiện tại đã được load vào bộ nhớ RAM và trả về giá

trị số trang cần tìm.

```

1 int pg_getpage(struct mm_struct *mm, int pgn, int *fpn, struct pcb_t *caller)
2 {
3     uint32_t pte = mm->pgd[pgn];
4     //printf("-----pte = %X\n", pte);
5     if (!PAGING_PAGE_PRESENT(pte))
6     { /* Page is not online, make it actively living */
7         printf("-----swapping-----\n");
8         int vicpgn, swpfpn;
9         int vicfpn;
10        uint32_t vicpte;
11        int tgtfpn = PAGING_SWP(pte); //the target frame storing our variable
12        /* TODO: Play with your paging theory here */
13        /* Find victim page */
14        if (find_victim_page(caller->mm, &vicpgn) < 0) return -1;
15        vicpte = mm->pgd[vicpgn];
16        vicfpn = PAGING_SWP(vicpte);
17        /* Get free frame in MEMSWP */
18        if (MEMPHY_get_freefp(caller->active_mswp, &swpfpn) < 0) return -1;
19
20        /* Do swap frame from MEMRAM to MEMSWP and vice versa*/
21        /* Copy victim frame to swap */
22        __swap_cp_page(caller->mram, vicfpn, caller->active_mswp, swpfpn);
23        /* Copy target frame from swap to mem */
24        __swap_cp_page(caller->active_mswp, tgtfpn, caller->mram, vicfpn);
25
26        /* Update page table */
27        //pte_set_swap() & mm->pgd;
28        pte_set_swap(&vicpte, 0, swpfpn);
29        /* Update its online status of the target page */
30        //pte_set_fpn() & mm->pgd[pgn];
31        pte_set_fpn(&pte, vicfpn);
32
33        mm->pgd[pgn] = pte;
34        mm->pgd[vicpgn] = vicpte;
35
36        enlist_pgn_node(&caller->mm->fifo_pgn, pgn);

```



```

37     }
38
39     *fpgn = PAGING_FPN(pte);
40
41     return 0;
42 }

```

Ta sử dụng hàm `validate_overlap_vm_area` để kiểm tra xem có sự chồng chéo của các vùng nhớ ảo hay không.

```

1  int validate_overlap_vm_area(struct pcb_t *caller, int vmaid, int vmastart,
   ↪ int vmaend)
2  {
3      struct vm_area_struct *vma = caller->mm->mmap;
4      /* TODO validate the planned memory area is not overlapped */
5      int vmait = 0;
6      while (vma != NULL)
7      {
8          if ((vmait != vmaid) && (vma->vm_start != vma->vm_end))
9          {
10             if (OVERLAP(vma->vm_start, vma->vm_end, vmastart, vmaend))
11                 return -1;
12             }
13             vma = vma->vm_next;
14             vmait++;
15         }
16         return 0;
17     }

```

Ta sử dụng hàm `find_victim` để tìm **page victim** bằng thuật toán FIFO (First in first out) để thực hiện swap.

```

1  int find_victim_page(struct mm_struct *mm, int *retpgn)
2  {
3      struct pgn_t *pg = mm->fifo_pgn;
4      /* TODO: Implement the theoretical mechanism to find the victim page */
5      if (pg == NULL) {
6          return -1;

```

```

7   }
8   if (pg->pg_next == NULL) {
9       *retpgn = pg->pgn;
10      mm->fifo_pgn = NULL;
11      free(pg);
12  }
13  else {
14      while (pg->pg_next->pg_next != NULL) {
15          pg = pg->pg_next;
16      }
17      *retpgn = pg->pg_next->pgn;
18      free(pg->pg_next);
19      pg->pg_next = NULL;
20  }
21  return 0;
22  }

```

Ta sử dụng hàm `tlbread` để đọc dữ liệu từ bộ nhớ trong hệ thống, sử dụng TLB (Translation Lookaside Buffer) để tối ưu hóa thời gian truy cập. **Hàm này được hiện thực trong file `cpu-tlb.c`.**

```

1  int tlbread(struct pcb_t * proc, uint32_t source,
2             uint32_t offset,      uint32_t destination)
3  {
4      //done
5      pthread_mutex_lock(&tlb_lock);
6      BYTE data;
7      int frmnum = -1;
8      BYTE check = 0;
9      int addr = proc->regs[source] + offset;
10     //getting the page number
11     int pgn = PAGING_PGN(addr);
12
13     /* TODO retrieve TLB CACHED frame num of accessing page(s)*/
14     /* by using tlb_cache_read()/tlb_cache_write()*/
15     /* frmnum is return value of tlb_cache_read/write value*/
16     frmnum = tlb_cache_read(proc->tlb, proc->pid, pgn, &check);

```

```
17
18 //check if the address is exists on RAM or not!
19 if(check == -1) {
20     //if not exists -> get PAGE!, if not exists -> ERROR!
21     if(pg_getpage(proc->mm,pgn,&frmnum,proc) != 0){ //Need page is loaded
22         ↪ into RAM
23         pthread_mutex_unlock(&tlb_lock);
24         return -1; /* invalid page access */
25     }
26 }
```

Ta sử dụng hàm `TLBMEMPHY_dump` để dump nội dung từ `mp->storage` ra màn hình để theo dõi nội dung bộ nhớ. Hàm này được hiện thực trong file `cpu-tlbcache.c`.

```
1 int TLBMEMPHY_dump(struct memphy_struct * mp)
2 {
3     /*TODO dump memphy contnt mp->storage
4      *   for tracing the memory content
5      */
6     if (mp == NULL)
7         return -1;
8     MEMPHY_dump(mp);
9
10    return 0;
11 }
```

## 2.2.4 Kết quả chạy thử

Ta chạy thử file `os_1_singleCPU_mlq_paging` nhằm kiểm tra việc hiện thực kỹ thuật Paging cho Memory Management. File này chỉ được cấp phát một CPU nên ta chưa cần quan tâm đến vấn đề đồng bộ.

Nội dung file `os_1_singleCPU_mlq_paging`:

```
input > os_1_singleCPU_mlq_paging
1 2 1 8
2 1048576 16777216 0 0 0
3 1 s4 4
4 2 s3 3
5 4 m1s 2
6 6 s2 3
7 7 m0s 3
8 9 p1s 2
9 11 s0 1
10 16 s1 0
```

Hình 17: Nội dung file input `os_1_singleCPU_mlq_paging`

## Kết quả chạy thử:

```
Time slot 0 ld_routine
Time slot 1 Loaded a process at input/proc/s4, PID: 1 PRIO: 4
Time slot 2 CPU 0: Dispatched process 1
Time slot 3 Loaded a process at input/proc/s3, PID: 2 PRIO: 3
Time slot 4 CPU 0: Put process 1 to run queue
Time slot 5 CPU 0: Dispatched process 1
Time slot 6 Loaded a process at input/proc/m1s, PID: 3 PRIO: 2
Time slot 7 CPU 0: Put process 1 to run queue
Time slot 8 CPU 0: Dispatched process 1
Time slot 9 CPU 0: Processed 1 has finished
Time slot 10 Loaded a process at input/proc/p1s, PID: 6 PRIO: 2
Time slot 11 CPU 0: Dispatched process 3
Time slot 12 CPU 0: Put process 3 to run queue
Time slot 13 CPU 0: Dispatched process 6
Time slot 14 CPU 0: Put process 6 to run queue
Time slot 15 CPU 0: Dispatched process 3
Time slot 16 CPU 0: Put process 3 to run queue
Time slot 17 CPU 0: Dispatched process 6
Time slot 18 CPU 0: Put process 6 to run queue
Time slot 19 CPU 0: Dispatched process 3
Time slot 20 CPU 0: Put process 6 to run queue
Time slot 21 CPU 0: Dispatched process 3
Time slot 22 CPU 0: Processed 3 has finished
Time slot 23 CPU 0: Dispatched process 6
Time slot 24 CPU 0: Put process 6 to run queue
Time slot 25 CPU 0: Dispatched process 6
Time slot 26 CPU 0: Processed 6 has finished
Time slot 27 CPU 0: Dispatched process 8
Time slot 28 CPU 0: Put process 8 to run queue
Time slot 29 CPU 0: Dispatched process 8
Time slot 30 CPU 0: Put process 8 to run queue
Time slot 31 CPU 0: Dispatched process 8
Time slot 32 CPU 0: Put process 8 to run queue
Time slot 33 CPU 0: Dispatched process 8
Time slot 34 CPU 0: Processed 8 has finished
Time slot 35 CPU 0: Dispatched process 7
Time slot 36 CPU 0: Put process 7 to run queue
Time slot 37 CPU 0: Dispatched process 7
Time slot 38 CPU 0: Put process 7 to run queue
Time slot 39 CPU 0: Dispatched process 7
Time slot 40 CPU 0: Put process 7 to run queue
Time slot 41 CPU 0: Dispatched process 7
Time slot 42 CPU 0: Put process 7 to run queue
Time slot 43 CPU 0: Dispatched process 7
Time slot 44 CPU 0: Put process 7 to run queue
Time slot 45 CPU 0: Dispatched process 7
Time slot 46 CPU 0: Put process 7 to run queue
Time slot 47 CPU 0: Dispatched process 7
Time slot 48 CPU 0: Put process 7 to run queue
Time slot 49 CPU 0: Dispatched process 7
Time slot 50 CPU 0: Processed 7 has finished
Time slot 51 CPU 0: Dispatched process 2
Time slot 52 CPU 0: Put process 2 to run queue
Time slot 53 CPU 0: Dispatched process 4
Time slot 54 CPU 0: Put process 4 to run queue
```

Hình 18: Output của file `os_1_singleCPU_mlq_paging` (1)

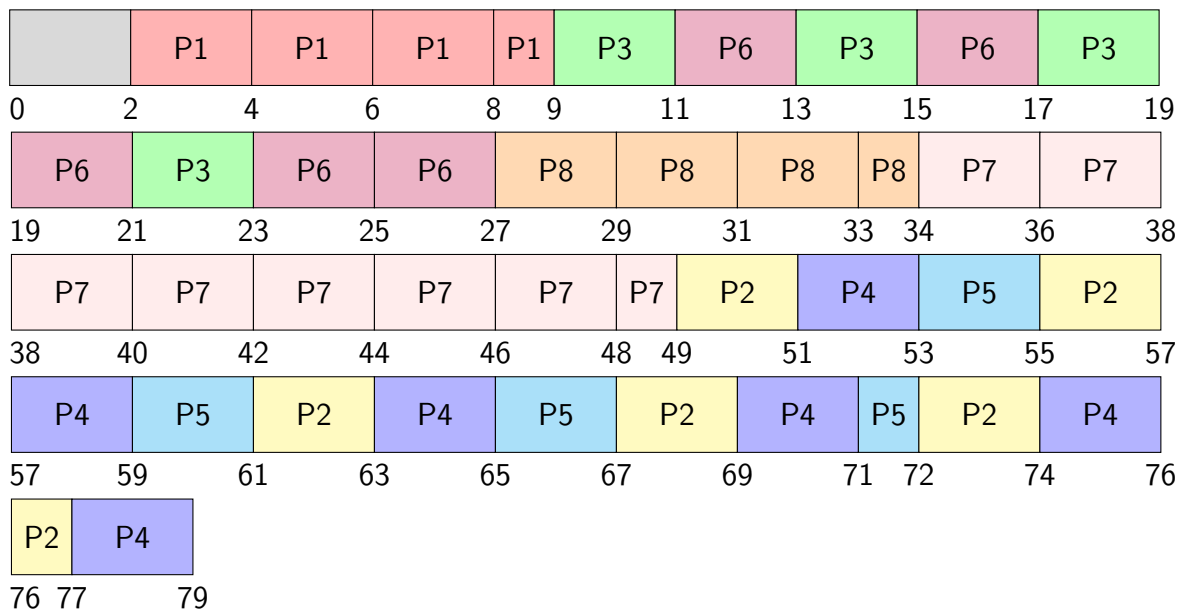
```

Time slot 58
Time slot 59
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 5
Time slot 60
Time slot 61
    CPU 0: Put process 5 to run queue
    CPU 0: Dispatched process 2
Time slot 62
Time slot 63
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 4
Time slot 64
Time slot 65
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 5
TLB hit at write region=1 offset=20 value=102
print_pgtbl: 0 - 768
00000000: 80000006
00000004: 80000005
00000008: 80000007
Time slot 66
TLB miss at write region=2 offset=1000 value=1
print_pgtbl: 0 - 768
00000000: 80000006
00000004: 80000005
00000008: 80000007
Time slot 67
    CPU 0: Put process 5 to run queue
    CPU 0: Dispatched process 2
Time slot 68
Time slot 69
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 4
Time slot 70
Time slot 71
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 5
TLB hit at write region=2 offset=1000 value=1
print_pgtbl: 0 - 768
00000000: c0000000
00000004: 80000005
00000008: 80000007
Time slot 72
    CPU 0: Processed 5 has finished
    CPU 0: Dispatched process 2
Time slot 73
Time slot 74
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 4
Time slot 75
Time slot 76
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 2
Time slot 77
    CPU 0: Processed 2 has finished
    CPU 0: Dispatched process 4
Time slot 78
Time slot 79
    CPU 0: Processed 4 has finished
    CPU 0 stopped
Time slot 68

```

Hình 19: Output của file `os_1_singleCPU_mfq_paging` (2)

### Gantt Chart:



Hình 20: Gantt chart của quá trình định thời `os_1_singleCPU_mfq_paging`

### Giải thích kết quả:

Tại time slot bằng 65, terminal hiển thị thông tin bảng phân trang cho process `m0s` (PID = 5), đồng thời cho biết rằng hiện tại process `m0s` được cấp phát 3 trang nhớ.

Ban đầu, process `m0s` yêu cầu 300 byte cho vùng nhớ 0, tuy nhiên kích thước trang

nhớ được sử dụng trong hệ điều hành này là 256 byte nên hệ thống dành ra 2 trang nhớ để đáp ứng yêu cầu trên. Sau đó, `m0s` lại yêu cầu tiếp 100 byte cho vùng nhớ 1 và được hệ thống đáp ứng bằng 1 trang nhớ.

Bên cạnh đó, còn một yêu cầu khác là 100 byte cho vùng nhớ 2, tuy nhiên trước đó process đã giải phóng vùng nhớ 0 nên vùng nhớ này có thể được sử dụng lại bởi process đó. Do đó, hệ thống không cần cấp phát thêm một trang nhớ nào nữa.

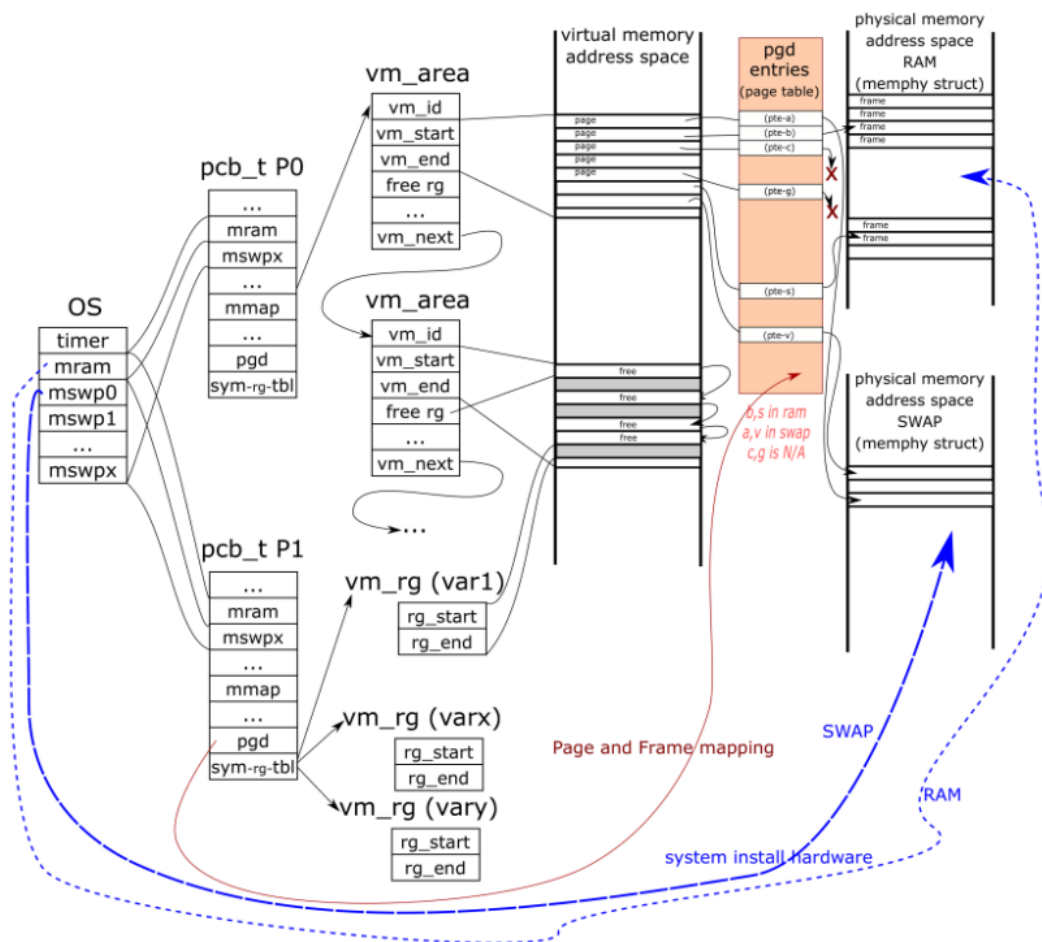
Tại time slot bằng 71, ta thấy nội dung PTE số 0 đã bị thay đổi (từ `80000006` sang `c0000000`). Do lệnh khi thực hiện động tác ghi vào bộ nhớ hệ thống đã thực hiện hoán đổi nên gây ra sự thay đổi về giá trị PTE. Giá trị `c0000000` nghĩa là trang tương ứng đang hiện diện trong RAM (với bit present = 1) và frame number = 0.

## 2.3 Put it all together (Synchronization)

### 2.3.1 Tổng quan

Trong Bài tập lớn, hệ điều hành được xây dựng để được chạy trên nhiều CPU khác nhau. Các CPU có thể được chạy đồng thời với nhau, và nó đều sử dụng chung một bộ nhớ RAM, các bộ nhớ SWAP, hay các queue chứa process control block... Các tài nguyên cần được bảo vệ nhằm tránh xảy ra tranh chấp.

Sau khi tổng hợp hai phần trên, ta đã có được một hệ điều hành đơn giản với một góc nhìn khái quát qua biểu đồ được thể hiện ở Hình 21.



Hình 21: Khái quát về Hệ điều hành đơn giản

### 2.3.2 Trả lời câu hỏi

**Question:** What will happen if the synchronization is not handled in your simple OS? Illustrate the problem of your simple OS by example if you have any. Note: You need to run two versions of your simple OS: the program with/without synchronization, then observe their performance based on demo results and explain their differences.

**Answer:**

Trong một hệ điều hành multi-tasking, đồng bộ hóa rất quan trọng để đảm bảo rằng nhiều tiến trình hoặc luồng không can thiệp vào việc thực thi lẫn nhau và truy cập vào tài nguyên được chia sẻ. Nếu đồng bộ hóa không được xử lý đúng cách, nó có thể dẫn đến tình trạng race conditions, deadlocks và các sự cố khác có thể khiến hệ thống trở nên không ổn định hoặc không phản hồi.

Trong hệ điều hành đơn giản mà nhóm đã hiện thực trong Bài tập lớn này, nếu không giải quyết vấn đề đồng bộ, tình trạng race condition có thể xảy ra. Kết quả là dữ liệu trong các tài nguyên này có thể mất tính toàn vẹn, nhất quán, và làm mất logic của chương trình. Ví dụ, nếu không kiểm soát đồng bộ cho `ready_queue`, các CPU có thể cùng lúc truy cập và thao tác trên nó, dẫn đến xung đột khi lấy ra hoặc thêm vào PCB. Tương tự, nếu không đảm bảo đồng bộ cho bộ nhớ, các CPU có thể gặp phải việc truy cập cùng lúc vào cùng một vùng nhớ, gây ảnh hưởng đến quản lý bộ nhớ.

Để triển khai cơ chế đồng bộ trong hệ điều hành, nhóm đã sử dụng Mutex lock từ thư viện `pthread`. Một mutex được áp dụng để bảo vệ `ready_queue` (`queue_lock`), và một mutex được sử dụng để bảo vệ RAM và SWAP (`mem_lock`). rước khi thực hiện bất kỳ thao tác nào liên quan đến việc thêm hoặc lấy PCB ra khỏi queue, câu lệnh `pthread_mutex_lock(&queue_lock)` được thực hiện để đảm bảo rằng không có CPU nào khác có thể truy cập vào `ready_queue`. Sau khi hoàn tất các thao tác trên `ready_queue`, CPU đó sẽ giải phóng mutex bằng `pthread_mutex_unlock(&queue_lock)`.

Tương tự, ta thực hiện các thao tác liên quan đến RAM và SWAP trong các câu lệnh `alloc/free/read/write`, câu lệnh `pthread_mutex_unlock(&mem_lock)` được thực thi để ngăn các CPU khác truy cập đồng thời vào RAM. Sau khi hoàn tất việc sử dụng bộ nhớ, CPU sẽ giải phóng mutex bằng cách sử dụng lệnh `pthread_mutex_unlock(&mem_lock)`.

### 2.3.3 Kết quả chạy thử

Sau khi hoàn thành xong các thành phần của hệ điều hành (gồm có Scheduler, Memory Management và Synchronization), nhóm sẽ chạy thử một số testcase nhằm kiểm tra toàn bộ Hệ điều hành.

#### 2.3.3.1 File `os_0_mmq_paging`

Nội dung file input `os_0_mmq_paging`:



```
input > ≡ os_0_mlq_paging
1      6 2 4
2      1048576 16777216 0 0 0
3      0 p0s 0
4      2 p1s 15
```

Hình 22: Nội dung file input `os_0_mlq_paging`

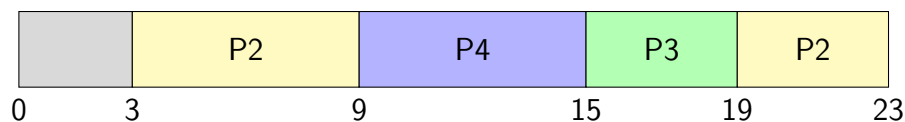
### Kết quả chạy thử:

```
Time slot 0      ld_routine
                  Loaded a process at input/proc/p0s, PID: 1 PRIO: 0
da add
CPU 1: Dispatched process 1
Time slot 1
Time slot 2      Loaded a process at input/proc/p1s, PID: 2 PRIO: 15
da add
Time slot 3      CPU 0: Dispatched process 2
                  Loaded a process at input/proc/p1s, PID: 3 PRIO: 0
da add
Time slot 4      Loaded a process at input/proc/p1s, PID: 4 PRIO: 0
da add
Time slot 5      TLB hit at write region=1 offset=20 value=100
                  print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Time slot 6
Time slot 7      CPU 1: Put process 1 to run queue
                  CPU 1: Dispatched process 3
Time slot 8
Time slot 9      CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 4
Time slot 10
Time slot 11
Time slot 12      CPU 1: Put process 3 to run queue
                  CPU 1: Dispatched process 1
                  TLB hit at read region=1 offset=20
                  print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Time slot 13      TLB hit at write region=3 offset=20 value=103
                  print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Time slot 14
Time slot 15      TLB hit at read region=3 offset=20
                  print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Time slot 16      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 3
Time slot 17
Time slot 18      CPU 1: Processed 1 has finished
                  CPU 1: Dispatched process 4
Time slot 19
Time slot 20      CPU 0: Dispatched process 2
Time slot 21      CPU 1: Processed 4 has finished
                  CPU 1 stopped
Time slot 22
Time slot 23      CPU 0: Processed 2 has finished
                  CPU 0 stopped
```

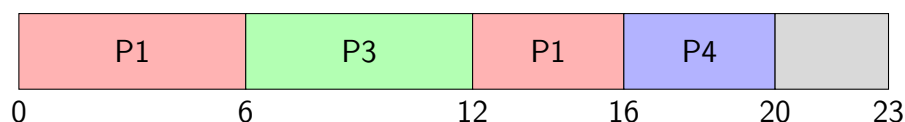
Hình 23: Output của file `os_0_mlq_paging`

### Gantt Chart:

#### CPU 0



#### CPU 1



Hình 24: Gantt chart cho quá trình định thời `os_0_mlq_paging`

### Giải thích kết quả:

Đầu tiên, process `p0s` thực thi lệnh alloc, yêu cầu cấp phát 300 byte vào vùng nhớ

0, do mỗi trang nhớ có kích thước 256 byte nên hệ thống cấp phát 2 trang nhớ để đáp ứng yêu cầu trên.

Tương tự, process **p0s** lại yêu cầu 300 byte vào vùng nhớ 4, do đó hệ thống cấp phát 2 trang nhớ để đáp ứng yêu cầu.

Sau đó, process **p0s** yêu cầu cấp phát 100 byte vào vùng nhớ 1, tuy nhiên trước đó process đã giải phóng vùng nhớ 0 nên vùng nhớ này có thể được sử dụng lại. Do đó, hệ thống không cần cấp phát thêm trang nhớ nào nữa.

### 2.3.3.2 File **os\_1\_mlq\_paging**

Nội dung file input **os\_1\_mlq\_paging**:

```
input > os_1_mlq_paging
1 2 4 8
2 1048576 16777216 0 0 0
3 1 p0s 130
4 2 s3 39
5 4 m1s 15
6 6 s2 120
7 7 m0s 120
8 9 p1s 15
9 11 s0 38
10 16 s1 0
```

Hình 25: Nội dung file input **os\_1\_mlq\_paging**

Kết quả chạy thử:

```
Time slot 0
ld_routine
Loaded a process at input/proc/p0s, PID: 1 PRIO: 130
Time slot 1
CPU 3: Dispatched process 1
Time slot 2
Loaded a process at input/proc/s3, PID: 2 PRIO: 39
CPU 2: Dispatched process 2
Time slot 3
CPU 3: Put process 1 to run queue
CPU 3: Dispatched process 1
Loaded a process at input/proc/m1s, PID: 3 PRIO: 15
Time slot 4
CPU 2: Put process 2 to run queue
CPU 1: Dispatched process 3
Time slot 5
CPU 2: Dispatched process 2
CPU 3: Put process 1 to run queue
CPU 3: Dispatched process 1
Time slot 6
Loaded a process at input/proc/s2, PID: 4 PRIO: 120
TLB miss at write region=1 offset=20 value=102
print_ptbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
CPU 2: Put process 2 to run queue
CPU 0: Dispatched process 4
Time slot 7
CPU 2: Dispatched process 2
CPU 1: Put process 3 to run queue
CPU 1: Dispatched process 3
Loaded a process at input/proc/m0s, PID: 5 PRIO: 120
Time slot 8
CPU 3: Put process 1 to run queue
CPU 3: Dispatched process 5
Time slot 9
CPU 1: Put process 3 to run queue
Loaded a process at input/proc/p1s, PID:
CPU 2: Put process 2 to run queue
CPU 2: Dispatched process 6
CPU 1: Dispatched process 3
CPU 0: Put process 4 to run queue
CPU 0: Dispatched process 2
CPU 3: Put process 5 to run queue
CPU 3: Dispatched process 4
Time slot 10
Loaded a process at input/proc/s0, PID: 7
CPU 1: Put process 3 to run queue
Time slot 11
CPU 0: Put process 2 to run queue
CPU 0: Dispatched process 5
CPU 1: Dispatched process 3
CPU 2: Put process 6 to run queue
CPU 2: Dispatched process 6
Time slot 12
CPU 3: Put process 4 to run queue
CPU 3: Dispatched process 7
CPU 2: Put process 6 to run queue
CPU 0: Put process 5 to run queue
CPU 0: Dispatched process 6
Time slot 13
CPU 2: Dispatched process 2
CPU 1: Processed 3 has finished
CPU 1: Dispatched process 4
CPU 3: Put process 7 to run queue
CPU 3: Dispatched process 5
Time slot 14
TLB hit at write region=1 offset=20 value=102
print_ptbl: 0 - 768
00000000: 80000008
00000004: 80000007
00000008: 80000009
Time slot 15
-----swapping-----
CPU 3: Put process 5 to run queue
CPU 3: Dispatched process 4
Time slot 16
CPU 1: Processed 2 has finished
Loaded a process at input/proc/s1, PID: 8 PRIO: 0
CPU 1: Dispatched process 5
TLB hit at write region=0 offset=0 value=0
print_ptbl: 0 - 768
00000000: c0000000
00000004: 80000007
00000008: 80000009
CPU 2: Put process 7 to run queue
CPU 0: Put process 6 to run queue
CPU 0: Dispatched process 6
CPU 2: Dispatched process 8
Time slot 17
CPU 1: Processed 5 has finished
CPU 1: Dispatched process 7
CPU 3: Put process 4 to run queue
```

Hình 26: Ouput của file **os\_1\_mlq\_paging** (1)

```

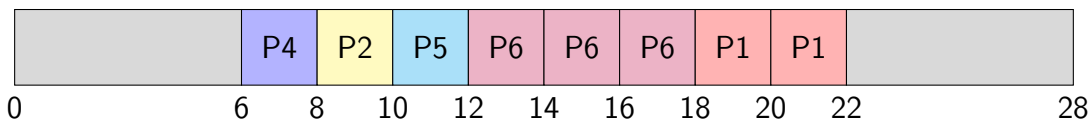
Time slot 18
  CPU 2: Put process 8 to run queue
  CPU 2: Dispatched process 8
  CPU 1: Put process 7 to run queue
  CPU 1: Dispatched process 7
Time slot 19
  CPU 0: Processed 6 has finished
  CPU 0: Dispatched process 1
  TLB miss at read region=1 offset=20
  print_pgtbl: 0 - 1024
  00000000: 80000001
  00000004: 80000000
  00000008: 80000003
  00000012: 80000002
  CPU 3: Put process 4 to run queue
  CPU 3: Dispatched process 4
  TLB hit at write region=3 offset=20 value=103
  print_pgtbl: 0 - 1024
  00000000: 80000001
  00000004: 80000000
  00000008: 80000003
  00000012: 80000002
Time slot 20
  CPU 2: Put process 8 to run queue
  CPU 1: Put process 7 to run queue
  CPU 1: Dispatched process 7
  CPU 2: Dispatched process 8
  CPU 0: Put process 1 to run queue
  CPU 0: Dispatched process 1
  TLB hit at read region=3 offset=20
Time slot 21
  print_pgtbl: 0 - 1024
  00000000: 80000001
  00000004: 80000000
  00000008: 80000003
  00000012: 80000002
  CPU 3: Processed 4 has finished
  CPU 3 stopped
Time slot 22
  CPU 2: Put process 8 to run queue
  CPU 2: Dispatched process 8
  CPU 1: Put process 7 to run queue
  CPU 1: Dispatched process 7
Time slot 23
  CPU 0: Processed 1 has finished
  CPU 0 stopped
  CPU 2: Processed 8 has finished
  CPU 2 stopped
Time slot 24
Time slot 25
  CPU 1: Put process 7 to run queue
  CPU 1: Dispatched process 7
Time slot 26
Time slot 27
  CPU 1: Put process 7 to run queue
  CPU 1: Dispatched process 7
Time slot 28
  CPU 1: Processed 7 has finished
  CPU 1 stopped

```

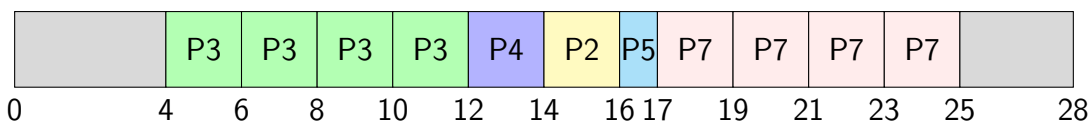
Hình 27: Output của file `os_1_mlq_paging` (2)

Gantt Chart:

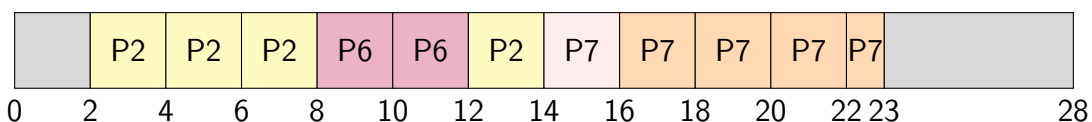
**CPU 0**



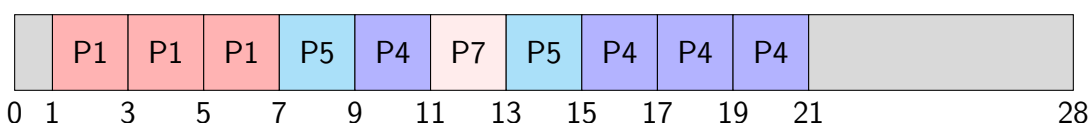
**CPU 1**



**CPU 2**



**CPU 3**



Hình 28: Gantt chart của quá trình định thời `os_1_mlq_paging`

### Giải thích kết quả:

Đầu tiên, process `p0s` yêu cầu cấp phát 300 byte cho vùng nhớ 0 và 300 byte cho vùng nhớ 4. Do đó, hệ thống cung cấp 4 trang nhớ cho process. Mặt khác, tại time slot bằng 6, TLB miss xảy ra, điều này cho thấy rằng khi CPU cố gắng truy cập địa chỉ bộ nhớ ảo nhưng TLB không có thông tin ánh xạ cho địa chỉ này.

Tiếp theo, process `m1s` yêu cầu cấp phát 300 byte cho vùng nhớ 0 và 100 byte cho vùng nhớ 1. Do đó, hệ thống cung cấp cho process 3 trang nhớ. Sau đó process lại yêu cầu cung cấp thêm 100 byte cho vùng nhớ 2, tuy nhiên trước đó process đã giải phóng vùng nhớ 0 nên hệ thống không cung cấp thêm trang nhớ. Đồng thời, ta nhận thấy tại time slot bằng 14, TLB hit xảy ra khi CPU yêu cầu truy cập một địa chỉ bộ nhớ ảo và TLB đã có thông tin ánh xạ cho địa chỉ này.

Tuy nhiên, cũng tại time slot bằng 14, TLB miss xảy ra, do CPU cố gắng truy cập địa chỉ bộ nhớ ảo của câu lệnh `write 1 2 1000` trong process `m0s` nhưng TLB không có thông tin ánh xạ cho địa chỉ này.

#### 2.3.3.3 File `os_1_mlq_paging_small_1K`

Nội dung file input `os_1_mlq_paging_small_1K`:

```
input > ≡ os_1_mlq_paging_small_1K
1      2 4 8
2      2048 16777216 0 0 0
3      1 p0s 130
4      2 s3 39
5      4 m1s 15
6      6 s2 120
7      7 m0s 120
8      9 p1s 15
9      11 s0 38
10     16 s1 0
```

Hình 29: Nội dung file input `os_1_mlq_paging_small_1K`

Kết quả chạy thử:

Time slot 0	ld routine	Time slot 8	Loaded a process at input/proc/pis, PID: 6 PRIO: 15 CPU 2: Put process 3 to run queue CPU 2: Dispatched process 6 CPU 1: Put process 2 to run queue CPU 1: Dispatched process 3	Time slot 14	write region-2 offset=1000 value=1 print pgtbl: 0 - 512 Time slot 15 CPU 1: Put process 4 to run queue CPU 0: Put process 6 to run queue CPU 0: Dispatched process 6
Time slot 1	Loaded a process at input/proc/p0s, PID: 1 PRIO: 130 CPU 3: Dispatched process 1				
Time slot 2	Loaded a process at input/proc/s3, PID: 2 PRIO: 39 CPU 2: Dispatched process 2	Time slot 9	CPU 0: Put process 4 to run queue CPU 0: Dispatched process 2 CPU 3: Put process 5 to run queue CPU 3: Dispatched process 4		
Time slot 3	CPU 3: Put process 1 to run queue CPU 3: Dispatched process 1			00000000: 80000007 00000004: 80000006	CPU 1: Dispatched process 4 CPU 2: Put process 2 to run queue CPU 2: Dispatched process 7 CPU 3: Put process 5 to run queue CPU 3: Dispatched process 2
Time slot 4	Loaded a process at input/proc/m1s, PID: 3 PRIO: 15 CPU 2: Put process 2 to run queue CPU 2: Dispatched process 3	Time slot 10	CPU 2: Put process 6 to run queue CPU 2: Dispatched process 5 CPU 0: Put process 2 to run queue CPU 1: Put process 3 to run queue CPU 1: Dispatched process 3		Loaded a process at input/proc/s1, PID: 8 PRIO: 6
Time slot 5	CPU 1: Dispatched process 2			Time slot 16	CPU 3: Processed 2 has finished CPU 0: Put process 6 to run queue CPU 0: Dispatched process 6 CPU 2: Put process 7 to run queue
Time slot 6	CPU 3: Put process 1 to run queue Loaded a process at input/proc/s2, PID: 4 PRIO: 120 CPU 3: Dispatched process 1	Time slot 11	Loaded a process at input/proc/s0, PID: 7 PRIO: 38 CPU 0: Dispatched process 6 CPU 3: Put process 4 to run queue CPU 3: Dispatched process 7	Time slot 17	CPU 1: Put process 4 to run queue CPU 3: Dispatched process 8 CPU 2: Dispatched process 7 CPU 1: Dispatched process 5
write region-1 offset=20 value=100 print pgtbl: 0 - 1024 00000000: 80000001 00000004: 80000000 00000008: 80000003 00000012: 80000002		Time slot 12	CPU 2: Put process 5 to run queue CPU 2: Dispatched process 2 CPU 0: Put process 6 to run queue		write region=0 offset=0 value=0 print pgtbl: 0 - 512 00000000: c0000000 00000004: 80000006
CPU 2: Put process 3 to run queue CPU 2: Dispatched process 3 CPU 1: Put process 2 to run queue CPU 1: Dispatched process 2		Time slot 13	CPU 1: Processed 3 has finished CPU 1: Dispatched process 4 CPU 0: Dispatched process 6 CPU 3: Put process 7 to run queue CPU 3: Dispatched process 5		CPU 1: Processed 5 has finished CPU 1: Dispatched process 4
Time slot 7	CPU 0: Dispatched process 4 Loaded a process at input/proc/m0s, PID: 5 PRIO: 120 CPU 3: Put process 1 to run queue CPU 3: Dispatched process 5	write region-1 offset=20 value=102 print pgtbl: 0 - 512 00000000: 80000007 00000004: 80000006		Time slot 18	CPU 3: Put process 8 to run queue CPU 3: Dispatched process 8 CPU 2: Put process 7 to run queue CPU 0: Processed 6 has finished

**Hình 30:** Output của file `os_1_mfq_paging_small_1K` (1)

```

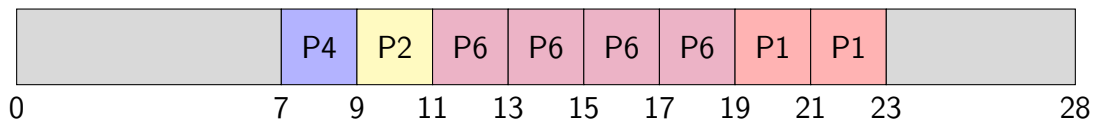
CPU 0: Processed 6 has finished
CPU 0: Dispatched process 1
read region=1 offset=20 value=100
print_pttbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Time slot 19
CPU 2: Dispatched process 7
CPU 1: Put process 4 to run queue
CPU 1: Dispatched process 4
Time slot 20
write region=3 offset=20 value=103
print_pttbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
CPU 3: Put process 8 to run queue
CPU 3: Dispatched process 8
Time slot 21
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 1
read region=3 offset=20 value=103
CPU 2: Put process 7 to run queue
print_pttbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
CPU 2: Dispatched process 7
Time slot 22
CPU 1: Processed 4 has finished
CPU 1 stopped
CPU 3: Put process 8 to run queue
CPU 2: Put process 7 to run queue
CPU 2: Dispatched process 7
Time slot 23
CPU 0: Processed 1 has finished
CPU 0 stopped
CPU 3: Dispatched process 8
CPU 3: Processed 8 has finished
CPU 3 stopped
Time slot 24
Time slot 25
Time slot 26
Time slot 27
Time slot 28

```

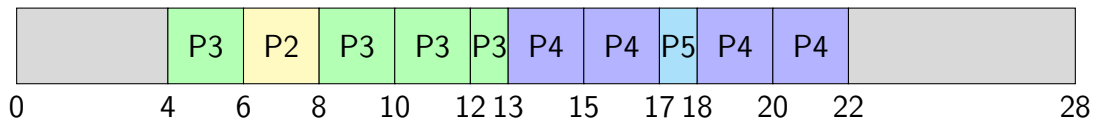
**Hình 31:** Output của file `os_1_mfq_paging_small_1K` (2)

## Gantt Chart:

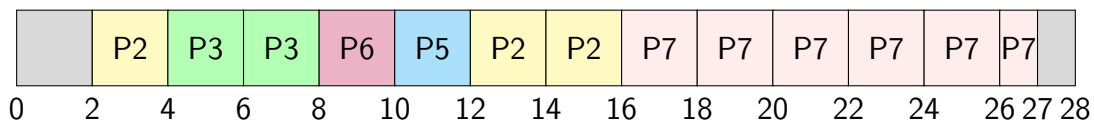
### CPU 0



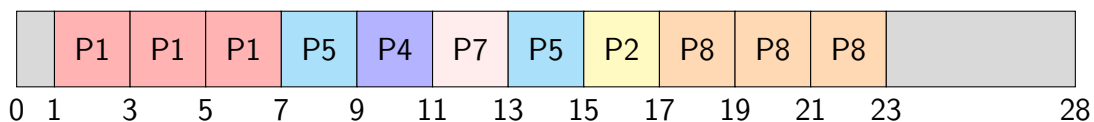
### CPU 1



### CPU 2



### CPU 3



Hình 32: Gantt chart của quá trình định thời `os_1_mlq_paging_small_1K`

## Giải thích kết quả:

Đầu tiên, proces `p0s` (PID = 1) yêu cầu cấp phát 300 byte cho vùng nhớ 0, 300 byte cho vùng nhớ 4. Do đó, hệ thống cấp phát cho process 4 trang nhớ. Sau đó, process lại yêu cầu cấp phát 100 byte cho vùng nhớ 1, do trước đó process thực thi lệnh `free` để giải phóng vùng nhớ, nên hệ thống sử dụng lại vùng nhớ này và không cấp phát thêm.

Tiếp theo, process `m1s` (PID = 3) yêu cầu cấp phát 300 byte cho vùng nhớ 0, 100 byte cho vùng nhớ 1. Do đó, hệ thống cấp cho process 2 trang nhớ. Một yêu cầu khác nữa là 100 byte cho vùng nhớ 2, do trước đó process đã giải phóng vùng nhớ nên hệ thống không cung cấp thêm trang nhớ nào nữa. Tại time slot bằng 17, ta nhận thấy nội dung PTE số 0 đã bị thay đổi (từ `80000007` tại time slot bằng đến `c0000000`), do lệnh khi thực hiện động tác ghi vào bộ nhớ hệ thống đã thực hiện hoán đổi nên gây ra sự thay đổi về giá trị PTE. Điều đó cho thấy, trang nhớ hiện tại đang hiện diện trong RAM (có bit present bằng 1) và frame number = 0.

### 2.3.3.4 File `os_1_mlq_paging_small_4K`

Nội dung file input `os_1_mlq_paging_small_4K`:

```
input > os_1_mlq_paging_small_4K
1 2 4 8
2 4096 16777216 0 0 0
3 1 p0s 130
4 2 s3 39
5 4 m1s 15
6 6 s2 120
7 7 m0s 120
8 9 p1s 15
9 11 s0 38
10 16 s1 0
```

Hình 33: Nội dung file input `os_1_mlq_paging_small_4K`

Kết quả chạy thử:

```
Time slot 0 CPU 2: Dispatched process 4 CPU 1: Dispatched process 4
ld_routine CPU 0: Dispatched process 7
Time slot 1 CPU 3: Put process 1 to run queue CPU 3: Put process 6 to run queue
Loaded a process at input/proc/p0s, PID: 1 PRIO: 130 CPU 3: Dispatched process 6
CPU 3: Dispatched process 1
Time slot 2 CPU 3: Put process 3 to run queue
Loaded a process at input/proc/s3, PID: 2 PRIO: 39 CPU 3: Dispatched process 6
CPU 2: Dispatched process 2
Time slot 3 CPU 0: Put process 5 to run queue
Loaded a process at input/proc/m1s, PID: 3 PRIO: 15 CPU 0: Dispatched process 3
CPU 0: Dispatched process 3
Time slot 4 CPU 1: Put process 2 to run queue
Loaded a process at input/proc/s0, PID: 7 PRIO: 38 CPU 1: Dispatched process 2
CPU 2: Put process 4 to run queue CPU 2: Dispatched process 7
CPU 2: Dispatched process 2 CPU 3: Put process 6 to run queue
CPU 3: Dispatched process 6
Time slot 5 CPU 3: Put process 3 to run queue
Loaded a process at input/proc/s2, PID: 4 PRIO: 120 CPU 3: Dispatched process 6
CPU 3: Dispatched process 3
Time slot 6 CPU 0: Processed 3 has finished
Loaded a process at input/proc/s0, PID: 5 PRIO: 120 CPU 0: Dispatched process 5
CPU 0: Dispatched process 5
Time slot 7 CPU 2: Put process 2 to run queue
Loaded a process at input/proc/s1, PID: 8 PRIO: 0 CPU 2: Put process 7 to run queue
CPU 2: Dispatched process 8
CPU 1: Put process 4 to run queue CPU 1: Dispatched process 5
CPU 1: Dispatched process 5
Time slot 8 write region=1 offset=20 value=100
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
00000008: 80000005
00000012: 80000004
CPU 0: Put process 4 to run queue
CPU 0: Dispatched process 5
CPU 2: Put process 2 to run queue
Time slot 9 CPU 1: Put process 2 to run queue
Loaded a process at input/proc/p1s, PID: 6 PRIO: 15 CPU 1: Dispatched process 2
CPU 3: Put process 3 to run queue CPU 2: Put process 4 to run queue
CPU 3: Dispatched process 6 CPU 3: Dispatched process 6
Time slot 10 CPU 0: Put process 5 to run queue
Loaded a process at input/proc/s1, PID: 8 PRIO: 0 CPU 0: Dispatched process 3
Time slot 11 CPU 1: Put process 2 to run queue
Loaded a process at input/proc/s0, PID: 7 PRIO: 38 CPU 1: Dispatched process 2
CPU 2: Put process 4 to run queue CPU 2: Dispatched process 7
CPU 2: Dispatched process 2 CPU 3: Put process 6 to run queue
CPU 3: Dispatched process 6
Time slot 12 CPU 0: Processed 3 has finished
Loaded a process at input/proc/s1, PID: 8 PRIO: 0 CPU 0: Dispatched process 5
CPU 0: Dispatched process 5
Time slot 13 CPU 1: Put process 2 to run queue
Loaded a process at input/proc/s0, PID: 7 PRIO: 38 CPU 1: Dispatched process 2
CPU 2: Put process 4 to run queue CPU 2: Dispatched process 7
CPU 2: Dispatched process 2 CPU 3: Put process 6 to run queue
CPU 3: Dispatched process 6
Time slot 14 CPU 0: Put process 5 to run queue
Loaded a process at input/proc/s1, PID: 8 PRIO: 0 CPU 0: Dispatched process 3
Time slot 15 CPU 1: Put process 2 to run queue
Loaded a process at input/proc/s0, PID: 7 PRIO: 38 CPU 1: Dispatched process 2
CPU 2: Put process 4 to run queue CPU 2: Dispatched process 7
CPU 2: Dispatched process 2 CPU 3: Put process 6 to run queue
CPU 3: Dispatched process 6
Time slot 16 CPU 0: Processed 3 has finished
Loaded a process at input/proc/s1, PID: 8 PRIO: 0 CPU 0: Dispatched process 5
CPU 0: Dispatched process 5
Time slot 17 CPU 1: Put process 2 to run queue
Loaded a process at input/proc/s0, PID: 7 PRIO: 38 CPU 1: Dispatched process 2
CPU 2: Put process 4 to run queue CPU 2: Dispatched process 7
CPU 2: Dispatched process 2 CPU 3: Put process 6 to run queue
CPU 3: Dispatched process 6
Time slot 18 CPU 0: Processed 3 has finished
Loaded a process at input/proc/s1, PID: 8 PRIO: 0 CPU 0: Dispatched process 5
CPU 0: Dispatched process 5
Time slot 19 CPU 1: Put process 2 to run queue
Loaded a process at input/proc/s0, PID: 7 PRIO: 38 CPU 1: Dispatched process 2
CPU 2: Put process 4 to run queue CPU 2: Dispatched process 7
CPU 2: Dispatched process 2 CPU 3: Put process 6 to run queue
CPU 3: Dispatched process 6
Time slot 20 CPU 0: Processed 3 has finished
Loaded a process at input/proc/s1, PID: 8 PRIO: 0 CPU 0: Dispatched process 5
CPU 0: Dispatched process 5
```

Hình 34: Output của file `os_1_mlq_paging_small_4K` (1)

```

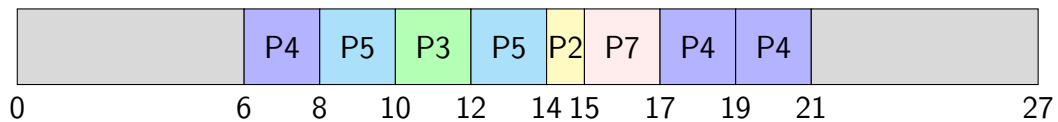
00000008: 80000005                                Time slot 26
00000012: 80000004                                CPU 1: Put process 7 to run queue
write region=3 offset=20 value=103        CPU 1: Dispatched process 7
print_pgtbl: 0 - 1024                      Time slot 27
Time slot 21                                CPU 1: Processed 7 has finished
CPU 0: Processed 4 has finished            CPU 1 stopped
CPU 0 stopped
00000000: 80000001
00000004: 80000000
00000008: 80000005
00000012: 80000004
CPU 2: Put process 8 to run queue
CPU 2: Dispatched process 8
CPU 3: Put process 1 to run queue
CPU 3: Dispatched process 1
read region=3 offset=20 value=103
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000005
00000012: 80000004
Time slot 22
CPU 1: Put process 7 to run queue
CPU 1: Dispatched process 7
CPU 2: Put process 8 to run queue
CPU 2: Dispatched process 8
Time slot 23
CPU 3: Processed 1 has finished
CPU 3 stopped
CPU 2: Processed 8 has finished
CPU 2 stopped
Time slot 24
CPU 1: Put process 7 to run queue
CPU 1: Dispatched process 7
Time slot 25
Time slot 26

```

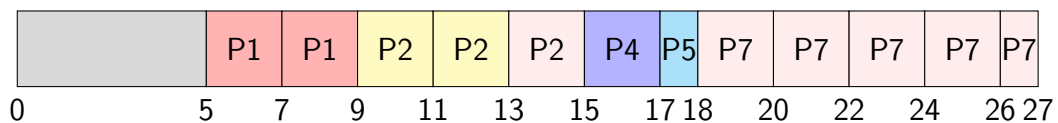
Hình 35: Output của file `os_1_mlq_paging_small_4K` (2)

Gantt Chart:

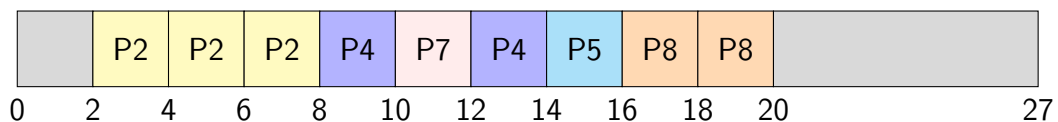
**CPU 0**



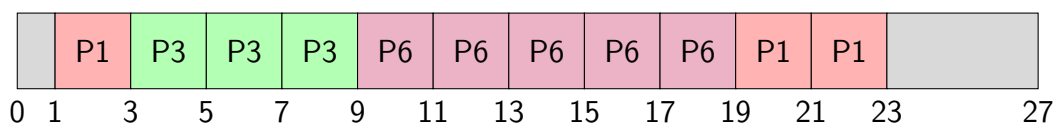
**CPU 1**



**CPU 2**



**CPU 3**



Hình 36: Gantt chart của quá trình định thời `os_1_mlq_paging_small_4K`



### 2.3.3.5 File `os_1_tlbsz_singleCPU_mlq`

Nội dung file input `os_1_tlbsz_singleCPU_mlq`:

```
input > os_1_tlbsz_singleCPU_mlq
1 2 1 8
2 40000
3 1 s4 4
4 2 s3 3
5 4 m1s 2
6 6 s2 3
7 7 m0s 3
8 9 p1s 2
9 11 s0 1
10 16 s1 0
```

Hình 37: Nội dung file input `os_1_tlbsz_singleCPU_mlq`

Kết quả chạy thử:

```
Time slot 0
ld_routine
Time slot 1
    Loaded a process at input/proc/s4, PID: 1 PRIO: 4
da add
Time slot 2
    CPU 0: Dispatched process 1
    Loaded a process at input/proc/s3, PID: 2 PRIO: 3
da add
Time slot 3
Time slot 4
    Loaded a process at input/proc/m1s, PID: 3 PRIO: 2
da add
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 5
Time slot 6
    Loaded a process at input/proc/s2, PID: 4 PRIO: 3
da add
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 7
    Loaded a process at input/proc/m0s, PID: 5 PRIO: 3
da add
Time slot 8
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 9
    CPU 0: Processed 1 has finished
    CPU 0: Dispatched process 3
    Loaded a process at input/proc/p1s, PID: 6 PRIO: 2
da add
Time slot 10
Time slot 11
    CPU 0: Put process 3 to run queue
    Loaded a process at input/proc/s0, PID: 7 PRIO: 1
da add
    CPU 0: Dispatched process 6
Time slot 12
Time slot 13
    CPU 0: Put process 6 to run queue
    CPU 0: Dispatched process 3
Time slot 14
Time slot 15
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 6
Time slot 16
    Loaded a process at input/proc/s1, PID: 8 PRIO: 0
da add
Time slot 17
    CPU 0: Put process 6 to run queue
    CPU 0: Dispatched process 3
Time slot 18
Time slot 19
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 6
Time slot 20
Time slot 21
    CPU 0: Put process 6 to run queue
    CPU 0: Dispatched process 3
Time slot 22
Time slot 23
    CPU 0: Processed 3 has finished
    CPU 0: Dispatched process 6
Time slot 24
Time slot 25
    CPU 0: Put process 6 to run queue
    CPU 0: Dispatched process 6
Time slot 26
Time slot 27
    CPU 0: Processed 6 has finished
    CPU 0: Dispatched process 8
Time slot 28
Time slot 29
    CPU 0: Put process 8 to run queue
    CPU 0: Dispatched process 8
Time slot 30
Time slot 31
    CPU 0: Put process 8 to run queue
```

Hình 38: Output của file `os_1_tlbsz_singleCPU_mlq` (1)

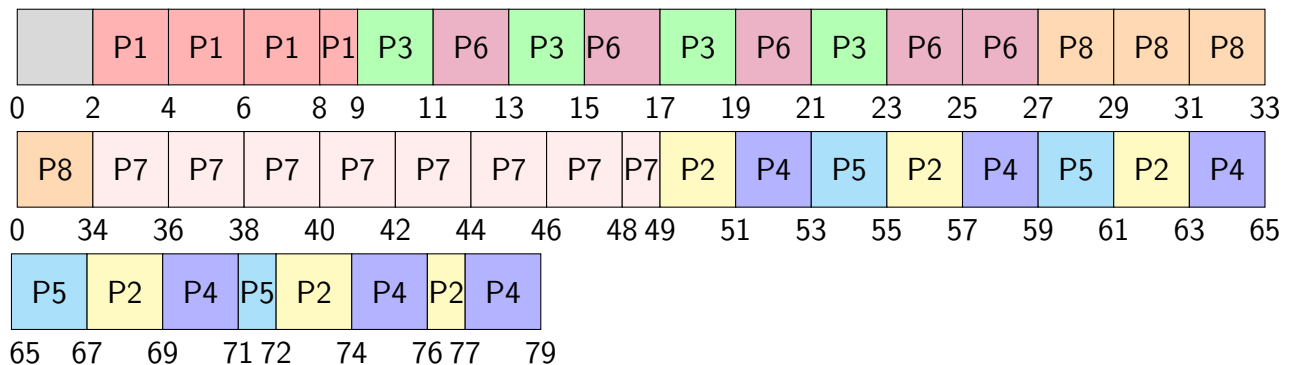
```

Time slot 31      CPU 0: Put process 8 to run queue
                  CPU 0: Dispatched process 8
Time slot 32
Time slot 33      CPU 0: Put process 8 to run queue
                  CPU 0: Dispatched process 8
Time slot 34      CPU 0: Processed 8 has finished
                  CPU 0: Dispatched process 7
Time slot 35
Time slot 36      CPU 0: Put process 7 to run queue
                  CPU 0: Dispatched process 7
Time slot 37
Time slot 38      CPU 0: Put process 7 to run queue
                  CPU 0: Dispatched process 7
Time slot 39
Time slot 40      CPU 0: Put process 7 to run queue
                  CPU 0: Dispatched process 7
Time slot 41      CPU 0: Put process 7 to run queue
                  CPU 0: Dispatched process 7
Time slot 42
Time slot 43
Time slot 44      CPU 0: Put process 7 to run queue
                  CPU 0: Dispatched process 7
Time slot 45
Time slot 46      CPU 0: Put process 7 to run queue
                  CPU 0: Dispatched process 7
Time slot 47
Time slot 48      CPU 0: Put process 7 to run queue
                  CPU 0: Dispatched process 7
Time slot 49      CPU 0: Processed 7 has finished
                  CPU 0: Dispatched process 2
Time slot 50
Time slot 51      CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 4
Time slot 52
Time slot 53      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 5
Time slot 54
Time slot 55      CPU 0: Put process 5 to run queue
                  CPU 0: Dispatched process 2
Time slot 56
Time slot 57      CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 4
Time slot 58
Time slot 59      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 5
Time slot 60
Time slot 61      CPU 0: Put process 5 to run queue
                  CPU 0: Dispatched process 2
Time slot 62
Time slot 63      CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 4
Time slot 64
Time slot 65      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 5
Time slot 66      TLB hit at write region=1 offset=20 value=102
                  print_pgtbl: 0 - 768
                  00000000: 80000006
Time slot 67      CPU 0: Put process 5 to run queue
                  CPU 0: Dispatched process 2
Time slot 68
Time slot 69      CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 4
Time slot 70
Time slot 71      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 5
Time slot 72      TLB hit at write region=0 offset=0 value=0
                  print_pgtbl: 0 - 768
                  00000000: c0000000
                  00000004: 80000005
                  00000008: 80000007
Time slot 73      CPU 0: Processed 5 has finished
                  CPU 0: Dispatched process 2
Time slot 74
Time slot 75      CPU 0: Put process 2 to run queue
                  CPU 0: Dispatched process 4
Time slot 76
Time slot 77      CPU 0: Put process 4 to run queue
                  CPU 0: Dispatched process 2
Time slot 78      CPU 0: Processed 2 has finished
                  CPU 0: Dispatched process 4
Time slot 79

```

Hình 39: Output của file `os_1_tlbsz_singleCPU_mlq` (2)

### Gantt Chart:



Hình 40: Gantt chart của quá trình định thời `os_1_tlbsz_singleCPU_mlq`

### Giải thích kết quả:

Tại thời điểm time slot bằng 66, TLB hit xảy ra khi CPU yêu cầu truy cập một địa chỉ bộ nhớ ảo, và TLB đã có thông tin ánh xạ cho địa chỉ này.

Tại thời điểm time slot bằng 67, TLB miss xảy ra, điều này cho thấy rằng khi CPU

cố gắng truy cập địa chỉ bộ nhớ ảo nhưng TLB không có thông tin ánh xạ cho địa chỉ này. Khi TLB miss xảy ra hệ điều hành sẽ tìm kiếm trong page table để tìm xem địa chỉ bộ nhớ ảo đó được ánh xạ đến đâu trong bộ nhớ vật lý.

Sau khi xác định được ánh xạ mới, hệ điều hành sẽ cập nhật TLB với thông tin mới này. Điều này giúp tối ưu hóa các truy cập bộ nhớ sau này bằng cách lưu trữ thông tin ánh xạ trong TLB để tránh TLB miss trong tương lai.

### 3 Tổng kết

Ta có thể thấy được hệ điều hành đóng vai trò vô cùng quan trọng trong các máy tính đa dụng (general-purpose computer) cũng như máy tính cá nhân (personal computer).

Trong bài tập lớn này, nhóm đã có cơ hội hiện thực một hệ điều hành đơn giản có các tính năng cơ bản như quản lý bộ nhớ, quản lý tiến trình và lập lịch. Bằng cách sử dụng ngôn ngữ lập trình C và các khái niệm cơ bản của hệ điều hành, nhóm đã xây dựng được một môi trường thử nghiệm để hiểu rõ hơn về cách làm việc và hoạt động của một hệ điều hành thực tế.

Nhóm đã gặp nhiều thử thách trong việc hiện thực hệ điều hành đơn giản, nhưng những khó khăn đó đã giúp nhóm nắm vững hơn về các nguyên lý cơ bản của hệ điều hành và làm quen với quá trình phát triển hệ điều hành.

Bên cạnh đó, nhóm cũng có cơ hội áp dụng kiến thức lý thuyết vào thực tiễn và hiểu rõ hơn về cách một hệ điều hành hoạt động từ bên trong.

Tuy nhiên, nhóm cũng còn những hạn chế và sai sót trong các bước hiện thực, cũng như phân bố thời gian để hoàn thành bài tập lớn.

Tóm lại, sau khi thực hiện bài tập lớn, nhóm đã có thêm những kiến thức về hệ điều hành cũng như cách hiện thực một hệ điều hành đơn giản. Và nhóm cũng đã có những kinh nghiệm để phát triển trong tương lai.

## Tài liệu

- [1] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne (04/05/2018), *Operating System Concept, Tenth Edition*.
- [2] GeeksforGeeks (17/02/2023), *Paged Segmentation and Segmented Paging*. Truy cập tại <https://www.geeksforgeeks.org/paged-segmentation-and-segmented-paging/>.
- [3] GeeksforGeeks (05/05/2023), *Multilevel Queue (MLQ) CPU Scheduling*. Truy cập tại <https://www.geeksforgeeks.org/multilevel-queue-mlq-cpu-scheduling/>.
- [4] GeeksforGeeks (19/01/2024), *Virtual Memory in Operating System*. Truy cập tại <https://www.geeksforgeeks.org/virtual-memory-in-operating-system/>.
- [5] GeeksforGeeks (30/04/2024), *Segmentation in Operating System*. Truy cập tại <https://www.geeksforgeeks.org/segmentation-in-operating-system/>.