

VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



Operating System

Assignment 2

SIMPLE OPERATING SYSTEM

Lecturer: Hoàng Lê Hải Thanh
Group : 07
Students: Trần Minh Trí - 1910637
Trần Quốc Vinh - 1915953
Phạm Duy Quang - 2011899
Võ Văn Hiền - 2020023

Ho Chi Minh City, 11/2021



Mục lục

1	Lời mở đầu	2
1.1	Giới thiệu chung	2
1.2	Tổng quan về Bài tập lớn 2	2
2	Yêu cầu đề bài	3
2.1	Nội dung Bài tập lớn 2	3
2.1.1	Source code:	3
2.2	Process	4
2.2.1	Tạo một process như thế nào?	5
2.2.2	Chạy mô phỏng như thế nào?	5
2.3	Hiện thực	6
2.3.1	Định thời:	6
2.3.2	Quản lý bộ nhớ	7
2.3.3	Đồng bộ	8
3	Code hiện thực	9
3.1	Quản lý bộ nhớ	9
3.1.1	Hàm alloc_mem()	9
3.1.2	Hàm free_mem()	11
3.1.3	Hàm translate()	13
3.1.4	Hàm enqueue()	13
3.1.5	Hàm dequeue()	14
3.2	Định thời	14
	3.2.0.a Hàm get_proc()	14
4	Kết quả:	15
4.1	Định thời	15
4.1.1	Trường hợp input sched_0	15
4.1.2	Biểu đồ Gantt và kết quả của input sched_0	15
4.1.3	Trường hợp input sched_1	16
4.1.4	Biểu đồ Gantt và kết quả của input sched_1	17
4.2	Quản lý bộ nhớ	20
4.2.1	Trường hợp input m0	20
4.2.2	Giải thích kết quả input m0	20
4.2.3	Trường hợp input m1	21
4.2.4	Giải thích kết quả input m1	21
4.3	Đồng bộ hệ điều hành	22
4.3.1	Trường hợp input os_0	22
	4.3.1.a Input os_0	22
	4.3.1.b Biểu đồ Gantt và kết quả của input os_0	22
4.3.2	Trường hợp input os_1	25
	4.3.2.a Input os_1	25
5	Phân công công việc:	30

1 Lời mở đầu

1.1 Giới thiệu chung

- **Mục tiêu:** Bài tập lớn giúp cho sinh viên giả lập các thành phần lớn trong một hệ điều hành đơn giản. Ví dụ: định thời (scheduler), đồng bộ (synchronization), quan hệ giữa bộ nhớ vật lý (physical memory) và bộ nhớ ảo (virtual memory)

- **Nội dung:** Sinh viên phải hiện thực 3 modules: định thời, đồng bộ, cơ chế (mechanism) của cấp phát (allocation) bộ nhớ từ bộ nhớ ảo sang vật lý.

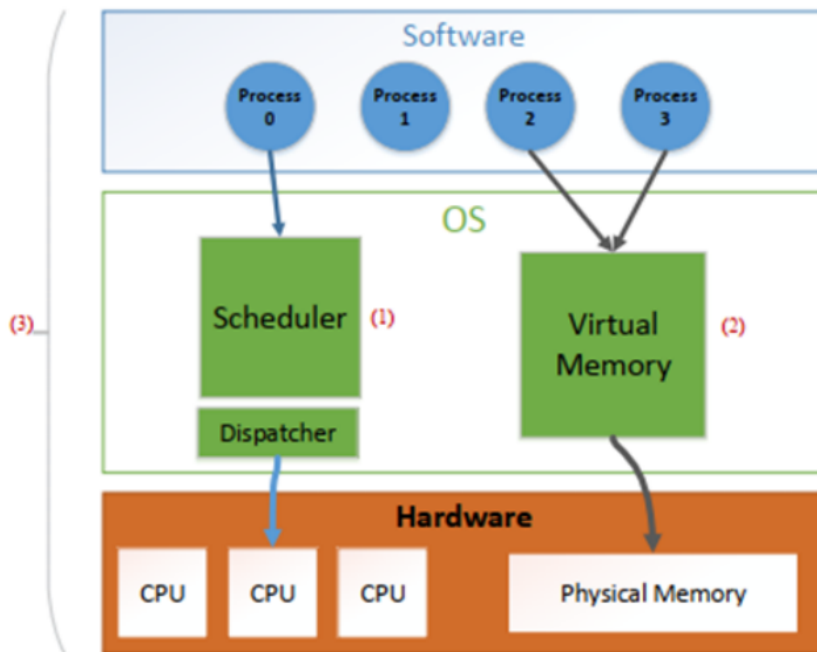
- **Kết quả cần đạt được:** sinh viên hiểu được phần nào đó nguyên lý hoạt động của một hệ điều hành cơ bản. Sinh viên có thể vẽ vai trò và ý nghĩa của mô-đun (key modules) ở hệ điều hành cũng như nó hoạt động như thế nào.

1.2 Tổng quan về Bài tập lớn 2

Bài tập lớn nói về việc mô phỏng một hệ điều hành cơ bản để giúp sinh viên hiểu được khái niệm nền tảng của định thời, đồng bộ và sự quản lý bộ nhớ. Hình 1 thể hiện kiến trúc tổng quát của hệ điều hành mà chúng ta chuẩn bị hiện thực. Nói chung, hệ điều hành phải quản lý 2 nguồn "ảo": CPU(s) và RAM được sử dụng 2 thành phần cốt lõi:

- **Định thời (Scheduler) và điều phối (Dispatcher):** xác định các quá trình được cho phép chạy trên CPU và thứ tự thực hiện của chúng.

- **Bộ nhớ ảo (Virtual Memory Engine):** cô lập khoảng không gian bộ nhớ của mỗi quá trình với nhau. Mặc dù RAM được chia sẻ bởi nhiều quá trình (multi processes) thì mỗi quá trình đều không biết sự tồn tại của nhau. Điều đó cho phép mỗi quá trình sở hữu không gian bộ nhớ ảo riêng cho mình và bộ nhớ ảo sẽ kết nối và dịch các địa chỉ ảo được cung cấp bởi các quá trình thành các địa chỉ vật lý tương ứng.



Hình 1: Kiến trúc tổng quát về các module chính

Thông qua các mô-đun đó, Hệ điều hành cho phép nhiều quá trình do người dùng tạo ra để chia sẻ và sử dụng tài nguyên máy tính ảo. Do đó, trong bài tập lớn này, chúng ta tập trung vào việc hiện thực công cụ định thời/điều phối và bộ nhớ ảo.

2 Yêu cầu đề bài

2.1 Nội dung Bài tập lớn 2

2.1.1 Source code:

⇒ Các file Headers

- timer.h: Định nghĩa bộ thời gian của hệ thống
- cpu.h: Định nghĩa hàm được sử dụng để hiện thực CPU ảo
- queue.h: Hàm được sử dụng để hiện thực queue lưu trữ các PCB của process
- sched.h: Định nghĩa hàm được sử dụng để định thời
- mem.h: Hàm được sử dụng bởi Bộ nhớ ảo (Virtual Memory Engine)
- loader.h: Hàm được sử dụng bởi loader nhằm để nạp chương trình từ đĩa vào bộ nhớ
- common.h: Định nghĩa cấu trúc và hàm được sử dụng trong hệ điều hành

⇒ Các file Source

- timer.c: Hiện thực bộ thời gian
- cpu.c: Hiện thực CPU ảo
- queue.c: Hiện thực các phương thức của queue (dựa vào sự ưu tiên – priority)
- paging.c: Sử dụng để kiểm tra chức năng của bộ nhớ ảo
- os.c: hệ điều hành bắt đầu chạy từ file này
- loader.c: Hiện thực loader (nạp)
- sched.c: Hiện thực định thời
- mem.c: Hiện thực RAM và Bộ nhớ ảo

⇒ **Makefile**

⇒ **Input:** các mẫu input cho sẵn được sử dụng cho việc đánh giá.

⇒ **Output:** Các mẫu output của hệ điều hành dùng để đối chiếu

2.2 Process

```
1 // From include/common.h
2 struct pcb_t{
3     uint32_t pid;
4     uint32_t priority;
5     uint32_t code_seg_t * code;
6     addr_t regs;
7     uint32_t pc;
8     struct seg_table_t * seg_table;
9     uint32_t bp;
10 }
```

Listing 1: Cấu trúc struct của process.

⇒ Ý nghĩa các trường trong **struct pcb_t** :

- pid: PID của quá trình
- priority: sự ưu tiên của quá trình. Định thời cho phép các quá trình với sự ưu tiên cao hơn được chạy trước những quá trình với sự ưu tiên thấp hơn
- code_seg_t * code: Text segment của quá trình (Để đơn giản hóa việc mô phỏng, chúng ta không đặt text segment trong RAM)
- regs: Thanh ghi, mỗi quá trình có thể sử dụng đến 10 thanh ghi được đánh thứ tự từ 0 đến 9
- pc: vị trí hiện tại của bộ đếm chương trình
- seg_table_t * seg_table: Page table được sử dụng để dịch địa chỉ ảo thành địa chỉ vật lý
- bp: Điểm dừng (break pointer), sử dụng để quản lý các heap segment

⇒ Tương tự như quá trình thực, mỗi quá trình trong mô phỏng này chỉ là một danh sách các lệnh được CPU thực thi lần lượt từ đầu đến cuối (chúng ta không thực hiện lệnh jump ở đây). Có năm câu lệnh mà một quá trình có thể thực hiện:

- **CALC**: thực hiện tính toán bởi CPU.
- **ALLOC**: Cấp phát số bytes trên bộ nhớ chính (RAM). Cú pháp:

alloc [size] [reg]

với size là lượng bytes mà quá trình muốn cấp phát từ RAM và reg là số thanh ghi sẽ lưu địa chỉ byte đầu tiên được cấp phát ở vùng nhớ

- **FREE**: Giải phóng vùng nhớ đã cấp phát. Cú pháp:

free [reg]

- **READ**: Đọc 1 byte từ bộ nhớ. Cú pháp:

read [source] [offset] [destination]

Đọc 1 byte tại địa chỉ = giá trị thanh ghi nguồn (source) + (offset) và lưu tại điểm đến (destination).

- **WRITE**: Viết giá trị thanh ghi vào bộ nhớ. Cú pháp:

write [data] [destination] [offset]

Viết dữ liệu(data) vào địa chỉ = giá trị thanh ghi tại điểm đến (destination)+ offset

2.2.1 Tạo một process như thế nào?

Nội dung của mỗi quá trình là bản sao chép của một chương trình được chứa ở ổ đĩa. Do đó để tạo một quá trình, chúng ta phải tạo một chương trình mô tả nội dung đó. Một chương trình được định nghĩa bởi một tập tin đơn với:

[priority] [N = number of instructions]
instruction 0
instruction 1
...
instruction N-1

- **priority** là sự ưu tiên của quá trình được tạo ra bởi chương trình. Sự ưu tiên cao hơn, cơ hội thành công cao hơn để một quá trình được lấy bởi CPU từ queue. - N là số câu lệnh

2.2.2 Chạy mô phỏng như thế nào?

Để hiện thực quá trình mô phỏng, chúng ta phải mô tả cấu hình của phần cứng và môi trường mà ta mô phỏng. Cấu hình được tạo theo chuẩn: - **time slide** là khoảng thời gian mà quá trình được phép chạy. - N là số CPU có sẵn - M là số quá trình sẽ được chạy

```
[time slice] [N = Number of CPU] [M = Number of Processes to be run]
[time 0] [path 0]
[time 1] [path 1]
...
[time M-1] [path M-1]
```

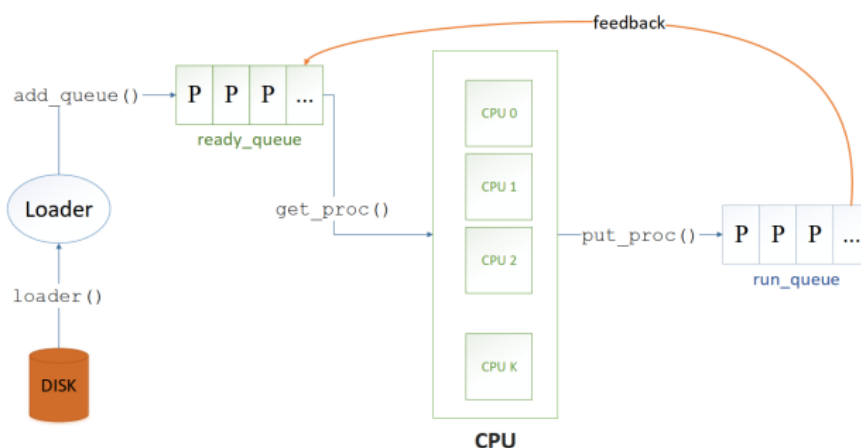
Hình 2: *Time Slice*

2.3 Hiện thực

2.3.1 Định thời:

Đầu tiên chúng tôi triển khai bộ định thời. Hình 2 cho thấy cách hệ điều hành định thời các quá trình như thế nào. Mặc dù hệ điều hành được thiết kế để hoạt động trên nhiều bộ xử lý, trong phần bài tập này, chúng ta giả định hệ thống chỉ có một bộ xử lý. Hệ điều hành sử dụng hàng đợi phản hồi ưu tiên để xác định quá trình nào sẽ được thực thi khi CPU khả dụng. Thiết kế bộ định thời dựa trên thuật toán "hàng đợi phản hồi đa cấp độ" được sử dụng trong nhân Linux.

Công việc của chúng ta trong phần này là hiện thực thuật toán này bằng cách hoàn thành



Hình 3: *Hoạt động của bộ định thời*

các hàm sau:

- **enqueue()** và **dequeue()** (trong **queue.c**): Chúng ta đã định nghĩa một cấu trúc (**queue_t**) cho một hàng đợi ưu tiên tại **queue.h**. Nhiệm vụ của chúng ta là thực hiện các chức năng đó để giúp đưa một PCB mới vào hàng đợi và nhận được một PCB có mức độ ưu tiên cao nhất trong hàng đợi.
- **get_proc()** (in **Sched.c**): nhận PCB của một tiến trình đang chờ ở **ready_queue**. Nếu hàng đợi trống tại thời điểm hàm được gọi, bạn phải di chuyển tất cả PCB của các quá trình đang chờ ở **run_queue** trở lại **ready_queue** trước khi nhận một quá trình từ **ready_queue**.

Để kiểm tra cách hoạt động của queue, biên dịch mã nguồn bằng Makefile:

```
make sched
```

Và sau đó, chạy bộ định thời với cấu hình mẫu

```
make test_sched
```

QUESTION: Lợi thế của việc sử dụng hàng đợi phản hồi ưu tiên (PFQ) so với các thuật toán định thời khác mà bạn đã học là gì?

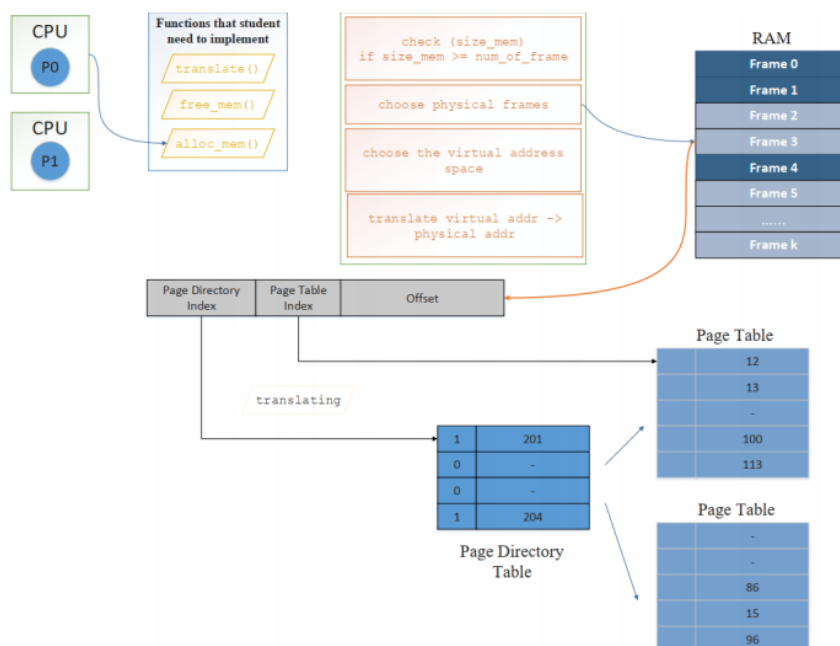
Answer: PFQ giải quyết được những khuyết điểm của FCFS (Thời gian đáp ứng trung bình không tối ưu) hay trong khoảng thời gian cố định như Round Robin đồng thời thiết lập độ ưu tiên của quá trình IO cao hơn quá trình CPU dẫn đến việc quá trình CPU sẽ không được ưu tiên như quá trình IO.

2.3.2 Quản lí bộ nhớ

Bộ nhớ ảo sử dụng cơ chế Phân đoạn với Phân trang để quản lý bộ nhớ. Theo mặc định, kích thước của RAM ảo là 1 MB nên chúng ta phải sử dụng 20 bit để biểu diễn địa chỉ của mỗi byte của nó. Với phân đoạn với cơ chế phân trang, chúng ta sử dụng 5 bit đầu tiên cho chỉ mục phân đoạn, 5 bit tiếp theo cho chỉ mục trang và 10 bit cuối cùng cho phần bù(offset). Sau đó, chúng ta có thể hiện thực quá trình dịch địa chỉ ảo của một quá trình sang địa chỉ vật lý bằng cách hoàn thành các hàm sau:

- `get_page_table ()` (trong `mem.c`): Tìm bảng trang có chỉ mục phân đoạn của một tiến trình.
- `translate ()` (trong `mem.c`): sử dụng hàm `get_page_table ()` để dịch một địa chỉ ảo sang địa chỉ vật lý.

Hình 4 cho thấy cách chúng ta cấp phát các vùng bộ nhớ mới và tạo mục nhập mới trong phân đoạn và bảng trang bên trong một tiến trình. Đặc biệt, đối với mỗi trang mới mà chúng tôi đã cấp phát, chúng tôi phải thêm mục nhập mới vào bảng trang theo số phân đoạn và số trang của trang này.



Hình 4: Hoạt động của bộ nhớ ảo

Nhiệm vụ của chúng ta là hiện thực các hàm `alloc_mem()` và `free_mem`, cả hai đều nằm trong `mem.c` dựa trên thuật toán được mô tả ở trên.

Để kiểm tra cách hoạt động, trước tiên phải biên dịch mã bằng Makefile:

make mem

Và sau đó, chạy bộ nhớ ảo với cấu hình mẫu

make test_mem

QUESTION: Lợi thế và bất lợi của phân đoạn (segmentation) với phân trang (paging) là gì?

Answer: Ưu điểm

- Giảm thiểu hao phí bộ nhớ, khắc phục việc bảng phân trang có kích thước quá lớn.
- Chống phân mảnh ngoại
- Dễ cấp phát phát bộ nhớ vật lý
- Bảo vệ dữ liệu và chia sẻ phân đoạn

Nhược điểm :

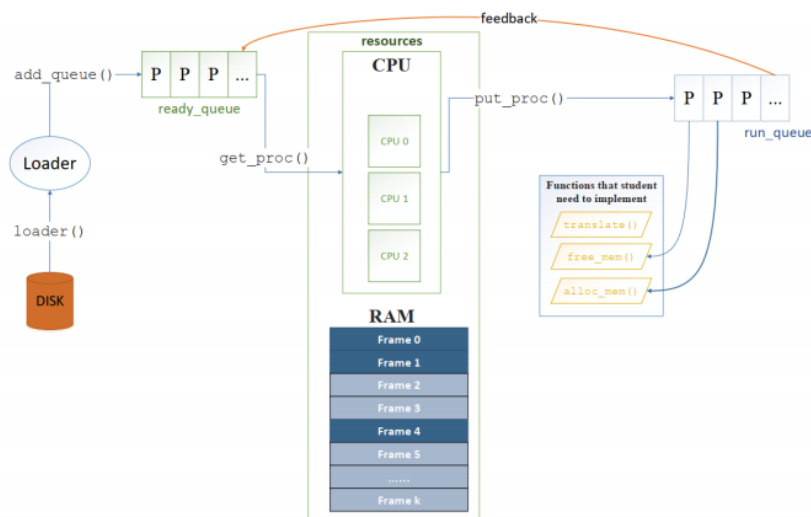
- Khó hiện thực
- Còn tồn tại phân mảnh nội
- Hiệu suất truy cập không cao

2.3.3 Đồng bộ

Cuối cùng, chúng ta kết hợp việc định thời và bộ nhớ ảo để tạo thành một hệ điều hành hoàn chỉnh. Hình 4 thể hiện việc hoàn thành tổ chức của hệ điều hành. Nhiệm vụ cuối cùng là

sự đồng bộ. Vì hệ điều hành chạy trên nhiều nhân, tài nguyên có thể được chia sẻ để truy cập bởi nhiều quá trình tại một thời điểm. Công việc của chúng ta là tìm tài nguyên chia sẻ và dùng cơ chế khóa để bảo vệ chúng.

Cuối cùng là kiểm tra hoạt động của toàn bộ hệ điều hành bằng Makefile:



Hình 5: Mô hình của Hệ điều hành

make all

Và sau đó, chạy bộ nhớ ảo với cấu hình mẫu

make test_all

3 Code hiện thực

3.1 Quản lí bộ nhớ

3.1.1 Hàm `alloc_mem()`

```
1 addr_t alloc_mem(uint32_t size, struct pcb_t *proc)
2 {
3     pthread_mutex_lock(&mem_lock);
4     addr_t ret_mem = 0;
5     /* TODO: Allocate [size] byte in the memory for the
6      * process [proc] and save the address of the first
7      * byte in the allocated memory region to [ret_mem].
8      */
9
10    uint32_t num_pages = ((size % PAGE_SIZE) == 0) ? size / PAGE_SIZE : size /
11    PAGE_SIZE + 1; // Number of pages we will use
12    int mem_avail = 0; // We could allocate new
13    memory region or not?
14
15    /* First we must check if the amount of free memory in
```

```
14  * virtual address space and physical address space is
15  * large enough to represent the amount of required
16  * memory. If so, set 1 to [mem_avail].
17  * Hint: check [proc] bit in each page of _mem_stat
18  * to know whether this page has been used by a process.
19  * For virtual memory space, check bp (break pointer).
20  * */
21  int count = 0;
22  for (int i = 0; i < NUM_PAGES; i++)
23  {
24      if (_mem_stat[i].proc == 0)
25      {
26          count++;
27          if (count == num_pages && proc->bp + num_pages * PAGE_SIZE <= RAM_SIZE)
28          {
29              mem_avail = 1;
30              break;
31          }
32      }
33  }
34  if (mem_avail)
35  {
36      /* We could allocate new memory region to the process */
37      ret_mem = proc->bp;
38      proc->bp += num_pages * PAGE_SIZE;
39      /* Update status of physical pages which will be allocated
40       * to [proc] in _mem_stat. Tasks to do:
41       * - Update [proc], [index], and [next] field
42       * - Add entries to segment table page tables of [proc]
43       *   to ensure accesses to allocated memory slot is
44       *   valid. */
45      int index_of_proc_page = 0;
46      int pre_index_of_proc_page = 0;
47      for (int i = 0; i < NUM_PAGES; i++)
48      {
49          if (_mem_stat[i].proc == 0)
50          {
51              _mem_stat[i].proc = proc->pid;
52              _mem_stat[i].index = index_of_proc_page;
53              if (_mem_stat[i].index != 0)
54              {
55                  _mem_stat[pre_index_of_proc_page].next = i;
56              }
57              pre_index_of_proc_page = i;
58
59              struct seg_table_t *seg_table = proc->seg_table;
60              if (seg_table->table[0].pages == NULL)
61              {
62                  seg_table->size = 0;
63              }
64
65              addr_t location_virtual_addr = (index_of_proc_page << OFFSET_LEN) +
ret_mem;
66              int found = 0;
67              for (int j = 0; j < seg_table->size; j++)
68              {
69                  if (seg_table->table[j].v_index == get_first_lv(location_virtual_addr))
70                  // Tim thay virtual_index o day.
71                  {
72                      struct page_table_t *page_table = seg_table->table[j].pages;
73                      page_table->table[page_table->size].v_index = get_second_lv(
location_virtual_addr);
```

```
73         page_table->table[page_table->size].p_index = i;
74         page_table->size++;
75         found = 1;
76         break;
77     }
78 }
79
80     if (!found) // Không tìm thấy virtual_index thì khởi tạo table mới
81     {
82         seg_table->table[seg_table->size].v_index = get_first_lv(
location_virtual_addr);
83         seg_table->table[seg_table->size].pages = (struct page_table_t *)malloc(
sizeof(struct page_table_t));
84         seg_table->table[seg_table->size].pages->table[0].v_index =
get_second_lv(location_virtual_addr);
85         seg_table->table[seg_table->size].pages->table[0].p_index = i;
86
87         seg_table->table[seg_table->size].pages->size = 1;
88
89         seg_table->size++;
90     }
91
92     index_of_proc_page++;
93     if (index_of_proc_page == num_pages) // phần tử cuối thì ta set next = -1
94     {
95         _mem_stat[i].next = -1;
96         break;
97     }
98 }
99 }
100 }
101 pthread_mutex_unlock(&mem_lock);
102 return ret_mem;
103 }
```

Listing 2: Code hàm alloc_mem() ở file mem.c

3.1.2 Hàm free_mem()

```
1 int free_mem(addr_t address, struct pcb_t *proc)
2 {
3     /*TODO: Release memory region allocated by [proc]. The first byte of
4      * this region is indicated by [address]. Task to do:
5      * - Set flag [proc] of physical page use by the memory block
6      *   back to zero to indicate that it is free.
7      * - Remove unused entries in segment table and page tables of
8      *   the process [proc].
9      * - Remember to use lock to protect the memory from other
10     *   processes. */
11
12     pthread_mutex_lock(&mem_lock);
13     struct page_table_t *page_table = get_page_table(get_first_lv(address), proc->
seg_table);
14
15     int valid = 0;
16     if (page_table != NULL)
17     {
18         for (int i = 0; i < page_table->size; i++)
19         {
20             if (page_table->table[i].v_index == get_second_lv(address))
```

```
21     {
22         addr_t physical_address;
23         if (translate(address, &physical_address, proc))
24         {
25             int physical_index = physical_address >> OFFSET_LEN;
26             int num_pages = 0;
27
28             do
29             {
30                 addr_t current_virtual_address = (num_pages << OFFSET_LEN) + address;
31                 _mem_stat[physical_index].proc = 0;
32
33                 struct seg_table_t *cur_seg_table = proc->seg_table;
34                 int found = 0;
35
36                 for (int k = 0; k < cur_seg_table->size && !found; k++)
37                 {
38                     if (cur_seg_table->table[k].v_index == get_first_lv(
current_virtual_address))
39                     {
40                         struct page_table_t *cur_page_table = cur_seg_table->table[k].
pages;
41                         for (int l = 0; l < cur_page_table->size; l++)
42                         {
43                             if (cur_page_table->table[l].v_index == get_second_lv(
current_virtual_address))
44                             {
45                                 for (int m = l; m < cur_page_table->size - 1; m++)
46                                 {
47                                     cur_page_table->table[m] = cur_page_table->table[m + 1];
48                                 }
49                                 cur_page_table->size--;
50                                 if (cur_page_table->size == 0)
51                                 {
52                                     free(proc->seg_table->table[k].pages);
53                                     for (int n = k; n < proc->seg_table->size - 1; n++)
54                                     {
55                                         proc->seg_table->table[n] = proc->seg_table->table[n + 1];
56                                     }
57                                     proc->seg_table->size--;
58                                 }
59                                 found = 1;
60                                 break;
61                             }
62                         }
63                     }
64                 }
65                 physical_index = _mem_stat[physical_index].next;
66                 num_pages++;
67             } while (physical_index != -1);
68             valid = 1;
69         }
70         break;
71     }
72 }
73 }
74
75 pthread_mutex_unlock(&mem_lock);
76 if (!valid)
77     return 1;
78 else
79     return 0;
```

80 }

Listing 3: Code của hàm free_mem() ở file mem.c

3.1.3 Hàm translate()

```
1 static int translate(  
2     addr_t virtual_addr,    // Given virtual address  
3     addr_t *physical_addr, // Physical address to be returned  
4     struct pcb_t *proc)  
5 { // Process uses given virtual address  
6  
7     /* Offset of the virtual address */  
8     addr_t offset = get_offset(virtual_addr);  
9     /* The first layer index */  
10    addr_t first_lv = get_first_lv(virtual_addr);  
11    /* The second layer index */  
12    addr_t second_lv = get_second_lv(virtual_addr);  
13  
14    /* Search in the first level */  
15    struct page_table_t *page_table = NULL;  
16    page_table = get_page_table(first_lv, proc->seg_table);  
17    if (page_table == NULL)  
18    {  
19        return 0;  
20    }  
21  
22    int i;  
23    for (i = 0; i < page_table->size; i++)  
24    {  
25        if (page_table->table[i].v_index == second_lv)  
26        {  
27            /* TODO: Concatenate the offset of the virtual address  
28             * to [p_index] field of page_table->table[i] to  
29             * produce the correct physical address and save it to  
30             * [*physical_addr] */  
31            *physical_addr = (page_table->table[i].p_index << OFFSET_LEN) + offset;  
32            return 1;  
33        }  
34    }  
35    return 0;  
36 }
```

Listing 4: Hàm translate() ở file mem.c

3.1.4 Hàm enqueue()

```
1 void enqueue(struct queue_t *q, struct pcb_t *proc)  
2 {  
3     /* TODO: put a new process to queue [q] */  
4     // If queue is full -> ERROR  
5     if (q->size == MAX_QUEUE_SIZE)  
6         return;  
7     // If queue is not full -> SUCCESS  
8     q->proc[q->size++] = proc;  
9 }
```

Listing 5: Hàm enqueue() ở file queue.c

3.1.5 Hàm dequeue()

```
1 struct pcb_t * dequeue(struct queue_t * q) {
2     /* TODO: return a pcb whose priority is the highest
3      * in the queue [q] and remember to remove it from q
4      * */
5     if (empty(q)) return NULL;
6
7     int pos = 0, maxPriority = q->proc[0]->priority;
8     for (int i = 1; i < q->size; i++){
9         if (maxPriority > q->proc[i]->priority){
10             maxPriority = q->proc[i]->priority;
11             pos = i;
12         }
13     }
14     struct pcb_t * result = q->proc[pos];
15     for (int i = pos; i < q->size - 1; i++){
16         q->proc[i] = q->proc[i+1];
17     }
18     q->proc[--q->size] = NULL;
19     return result;
20 }
```

Listing 6: Hàm dequeue() ở file queue.c

3.2 Định thời

3.2.0.a Hàm get_proc()

```
1 struct pcb_t *get_proc(void)
2 {
3     struct pcb_t *proc = NULL;
4     /*TODO: get a process from [ready_queue]. If ready queue
5      * is empty, push all processes in [run_queue] back to
6      * [ready_queue] and return the highest priority one.
7      * Remember to use lock to protect the queue.
8      * */
9
10    if(empty(&ready_queue)) {
11        while (!empty(&run_queue)){
12            pthread_mutex_lock(&queue_lock);
13            struct pcb_t * ptemp = dequeue(&run_queue);
14            enqueue(&ready_queue, ptemp);
15            pthread_mutex_unlock(&queue_lock);
16        }
17    }
18    pthread_mutex_lock(&queue_lock);
19    proc = dequeue(&ready_queue);
20    pthread_mutex_unlock(&queue_lock);
21    return proc;
22 }
```

Listing 7: Hàm get_proc() ở file sched.c



4 Kết quả:

4.1 Định thời

4.1.1 Trường hợp input sched_0

2	1	2
0	s0	
4	s1	

Trong đó:

2 : Quantum time

1 : Chạy trên 1 CPU

2 : Có 2 tiến trình s0 và s1

0,4: Thời điểm đến của tiến trình s0 và s1

s0,s1: tên tiến trình

4.1.2 Biểu đồ Gantt và kết quả của input sched_0

P1				P2			P1		P2		P1		P2		P1		P2		P1			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

Hình 6: Biểu đồ Gantt input sched_0


```

----- SCHEDULING TEST 0 -----
./os sched_0
    Loaded a process at input/proc/s0, PID: 1
Time slot 0
    CPU 0: Dispatched process 1
Time slot 1
Time slot 2
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 3
    Loaded a process at input/proc/s1, PID: 2
Time slot 4
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 2
Time slot 5
Time slot 6
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 1
Time slot 7
Time slot 8
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 2
Time slot 9
Time slot 10
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 1
Time slot 11
Time slot 12
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 2
Time slot 13
Time slot 14
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 1
Time slot 15
Time slot 16
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 2
Time slot 17
    CPU 0: Processed 2 has finished
    CPU 0: Dispatched process 1
Time slot 18
Time slot 19
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 20
Time slot 21
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 22
    CPU 0: Processed 1 has finished
    CPU 0 stopped

MEMORY CONTENT:
NOTE: Read file output/sched_0 to verify your result

```

Hình 7: Kết quả input sched_0

4.1.3 Trường hợp input sched_1

2	1	4
0	s0	
4	s1	
6	s2	
7	s3	

Trong đó:

2 : Quantum time



1 : Chạy trên 1 CPU
4 : Có 4 tiến trình
0,4,6,7: Thời điểm đến của tiến trình s0 và s1
s0,s1,s2,s3: tên tiến trình

4.1.4 Biểu đồ Gantt và kết quả của input sched_1

```
----- SCHEDULING TEST 1 -----
./os sched_1
    Loaded a process at input/proc/s0, PID: 1
Time slot 0
Time slot 1
    CPU 0: Dispatched process 1
Time slot 2
Time slot 3
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 4
    Loaded a process at input/proc/s1, PID: 2
Time slot 5
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 2
Time slot 6
    Loaded a process at input/proc/s2, PID: 3
Time slot 7
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 3
    Loaded a process at input/proc/s3, PID: 4
Time slot 8
Time slot 9
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 4
Time slot 10
Time slot 11
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 4
Time slot 12
Time slot 13
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 1
Time slot 14
Time slot 15
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 2
Time slot 16
Time slot 17
    CPU 0: Put process 2 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 4
Time slot 20
Time slot 21
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 1
Time slot 22
Time slot 23
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 2
Time slot 24
```

Hình 8: Kết quả phần 1 input sched_1

```

CPU 0: Dispatched process 2
Time slot 24
Time slot 25
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 3
Time slot 26
Time slot 27
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 4
Time slot 28
Time slot 29
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 1
Time slot 30
Time slot 31
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 2
Time slot 32
    CPU 0: Processed 2 has finished
    CPU 0: Dispatched process 3
Time slot 33
Time slot 34
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 4
Time slot 35
Time slot 36
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 1
Time slot 37
Time slot 38
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 3
Time slot 39
Time slot 40
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 4
Time slot 41
    CPU 0: Processed 4 has finished
    CPU 0: Dispatched process 1
Time slot 42
Time slot 43
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 3
Time slot 44
Time slot 45
    CPU 0: Processed 3 has finished
    CPU 0: Dispatched process 1
Time slot 46
    CPU 0: Processed 1 has finished
    CPU 0 stopped

```

MEMORY CONTENT:

Hình 9: *Kết quả phần 2 input sched 1*

	P1				P2		P3		P4				P1		P2		P3		P4		P1		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
P2		P3		P4		P1		P2		P3		P4		P1		P3		P4		P1			
23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46

Hình 10: Biểu đồ Gantt và kết quả của sched 1

4.2 Quản lí bộ nhớ

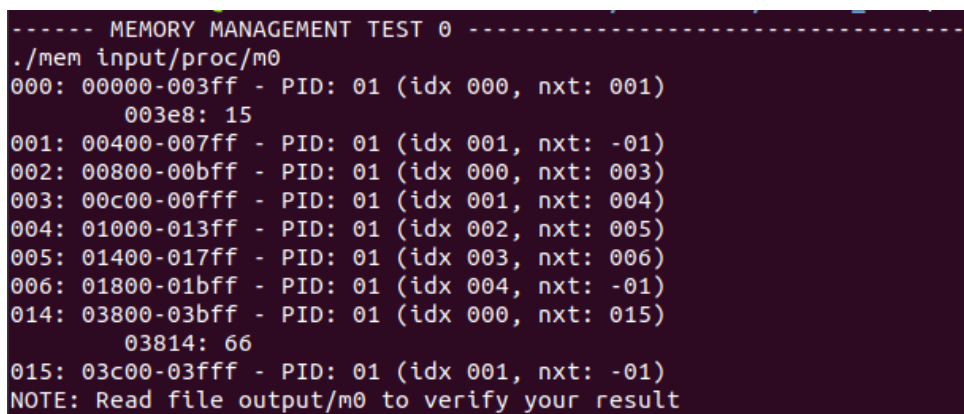
4.2.1 Trường hợp input m0

```
1 7  
alloc 13535 0  
alloc 1568 1  
free 0  
alloc 1386 2  
alloc 4564 4  
write 102 1 20  
write 21 2 1000
```

Trong đó:

1 : Mức độ ưu tiên

7 : Số câu lệnh



```
----- MEMORY MANAGEMENT TEST 0 -----  
./mem input/proc/m0  
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)  
003e8: 15  
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)  
002: 00800-00bff - PID: 01 (idx 000, nxt: 003)  
003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)  
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)  
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)  
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)  
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)  
03814: 66  
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)  
NOTE: Read file output/m0 to verify your result
```

Hình 11: Kết quả input m0

4.2.2 Giải thích kết quả input m0

- Giải thích:

+ **alloc 13535 0**

Cấp phát vùng nhớ (A) kích thước 13535/1024 \approx 14 trang (0-13) cho tiến trình 1. Thanh ghi 0 lưu địa chỉ đầu tiên cho vùng nhớ A là 0x00000

+ **alloc 1568 1**

Cấp phát vùng nhớ (B) kích thước 1568/1024 \approx 2 trang (14-15) cho tiến trình 1. Thanh ghi 1 lưu địa chỉ đầu tiên của vùng nhớ B là 0x03800

+ **free 0**

Giải phóng vùng nhớ có địa chỉ bắt đầu được lưu trong thanh ghi 0(Giải phóng vùng nhớ (A)).

+**alloc 1386 2**

Cấp phát vùng nhớ (C) kích thước 1386/1024 \approx 2 trang cho tiến trình 1. Thanh ghi 2 lưu địa chỉ đầu tiên của vùng nhớ C

+**alloc 4564 4** Cấp phát vùng nhớ (D) kích thước 4564/1024 \approx 5 trang. Thanh ghi 4 lưu địa chỉ đầu tiên của vùng nhớ D.

+ **write 102 1 20** Ghi giá trị 0x66 vào địa chỉ 0x03800 + 0x00014 = 0x03814.

+ **write 21 2 10000** Ghi giá trị 0x15 vào địa chỉ 0x00000 + 0x003e8 = 0x003e8

4.2.3 Trường hợp input m1

⇒ Đầu vào:

```
1 8
alloc 13535 0
alloc 1568 1
free 0
alloc 1386 2
alloc 4564 4
free 2
free 4
free 1
```

Trong đó:

1 : Mức độ ưu tiên

8 : Số câu lệnh

```
----- MEMORY MANAGEMENT TEST 1 -----
./mem input/proc/m1
NOTE: Read file output/m1 to verify your result (your implementation should print nothing)
```

Hình 12: Kết quả input m1

4.2.4 Giải thích kết quả input m1

+ **alloc 13535 0**

Cấp phát vùng nhớ (A) kích thước 13535/1024 \approx 14 trang (0-13) cho tiến trình 2. Thanh ghi 0 lưu địa chỉ đầu tiên cho vùng nhớ A là 0x00000

+ **alloc 1568 1**

Cấp phát vùng nhớ (B) kích thước 1568/1024 \approx 2 trang (14-15) cho tiến trình 2. Thanh ghi 1 lưu địa chỉ đầu tiên của vùng nhớ B là 0x03800

+ **free 0**

Giải phóng vùng nhớ có địa chỉ bắt đầu được lưu trong thanh ghi 0 (Giải phóng vùng nhớ (A)).

+ **alloc 1386 2**

Cấp phát vùng nhớ (C) kích thước 1386/1024 \approx 2 trang cho tiến trình 2. Thanh ghi 2 lưu địa chỉ đầu tiên của vùng nhớ C

+ **alloc 4564 4**

Cấp phát vùng nhớ (D) kích thước 4564/1024 \approx 5 trang cho tiến trình 2. Thanh ghi 4 lưu địa chỉ đầu tiên của vùng nhớ D.

+ **free 2**

Giải phóng vùng nhớ có địa chỉ bắt đầu được lưu trong thanh ghi 1. Giải phóng vùng nhớ (B)

+ **free 4**

Giải phóng vùng nhớ có địa chỉ bắt đầu được lưu trong thanh ghi 4. Giải phóng vùng nhớ (D)

+ **free 1**

Giải phóng vùng nhớ có địa chỉ bắt đầu được lưu trong thanh ghi 1. Giải phóng vùng nhớ (B)



4.3 Đồng bộ hệ điều hành

4.3.1 Trường hợp input os_0

4.3.1.a Input os_0

6	2	4
0	p0	
2	p1	

Trong đó:

6 : Quantum time

2 : Chạy trên 2 CPU

4 : Có 4 tiến trình

0,2: Thời điểm đến của tiến trình p0 và p1

p0,p1: tên tiến trình

4.3.1.b Biểu đồ Gantt và kết quả của input os_0

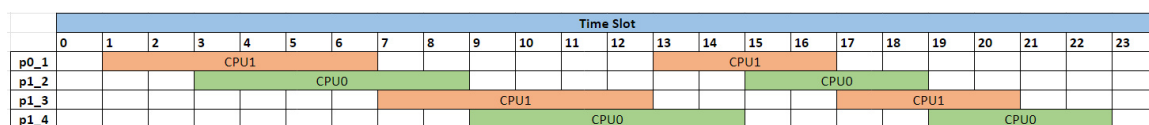
```
----- OS TEST 0 -----
./os os_0
    Loaded a process at input/proc/p0, PID: 1
Time slot 0
Time slot 1
    CPU 1: Dispatched process 1
Time slot 2
    Loaded a process at input/proc/p1, PID: 2
Time slot 3
    CPU 0: Dispatched process 2
    Loaded a process at input/proc/p1, PID: 3
Time slot 4
    Loaded a process at input/proc/p1, PID: 4
Time slot 5
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
Time slot 13
    CPU 1: Put process 3 to run queue
    CPU 1: Dispatched process 1
Time slot 14
Time slot 15
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 2
Time slot 16
Time slot 17
    CPU 1: Processed 1 has finished
    CPU 1: Dispatched process 3
Time slot 18
Time slot 19
    CPU 0: Processed 2 has finished
    CPU 0: Dispatched process 4
Time slot 20
Time slot 21
    CPU 1: Processed 3 has finished
    CPU 1 stopped
Time slot 22
Time slot 23
    CPU 0: Processed 4 has finished
    CPU 0 stopped
```

Hình 13: Kết quả phần 1 của input os_0


```
File Edit View Search Terminal Help
Time slot 20
Time slot 21
    CPU 1: Processed 3 has finished
    CPU 1 stopped
Time slot 22
Time slot 23
    CPU 0: Processed 4 has finished
    CPU 0 stopped

MEMORY CONTENT:
000: 00000-003ff - PID: 02 (idx 000, nxt: 001)
001: 00400-007ff - PID: 02 (idx 001, nxt: 007)
002: 00800-00bff - PID: 02 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 02 (idx 001, nxt: 004)
004: 01000-013ff - PID: 02 (idx 002, nxt: 005)
005: 01400-017ff - PID: 02 (idx 003, nxt: -01)
006: 01800-01bff - PID: 03 (idx 000, nxt: 011)
007: 01c00-01fff - PID: 02 (idx 002, nxt: 008)
    01de7: 0a
008: 02000-023ff - PID: 02 (idx 003, nxt: 009)
009: 02400-027ff - PID: 02 (idx 004, nxt: -01)
010: 02800-02bff - PID: 01 (idx 000, nxt: -01)
    02814: 64
011: 02c00-02fff - PID: 03 (idx 001, nxt: 012)
012: 03000-033ff - PID: 03 (idx 002, nxt: 013)
013: 03400-037ff - PID: 03 (idx 003, nxt: -01)
014: 03800-03bff - PID: 04 (idx 000, nxt: 025)
015: 03c00-03fff - PID: 03 (idx 000, nxt: 016)
016: 04000-043ff - PID: 03 (idx 001, nxt: 017)
017: 04400-047ff - PID: 03 (idx 002, nxt: 018)
    045e7: 0a
018: 04800-04bff - PID: 03 (idx 003, nxt: 019)
019: 04c00-04fff - PID: 03 (idx 004, nxt: -01)
020: 05000-053ff - PID: 04 (idx 000, nxt: 021)
021: 05400-057ff - PID: 04 (idx 001, nxt: 022)
022: 05800-05bff - PID: 04 (idx 002, nxt: 023)
    059e7: 0a
023: 05c00-05fff - PID: 04 (idx 003, nxt: 024)
024: 06000-063ff - PID: 04 (idx 004, nxt: -01)
025: 06400-067ff - PID: 04 (idx 001, nxt: 026)
026: 06800-06bff - PID: 04 (idx 002, nxt: 027)
027: 06c00-06fff - PID: 04 (idx 003, nxt: -01)
NOTE: Read file output/os_0 to verify your result
```

Hình 14: Kết quả phần 2 của input os_0



Hình 15: Biểu đồ Gantt của os_0

4.3.2 Trường hợp input os_1

4.3.2.a Input os_1

2 4 8
1 p0
2 s3
4 m1
6 s2
7 m0
9 p1
11 s0
16 s1

```
----- OS TEST 1 -----
./os os_1
Time slot 0
    Loaded a process at input/proc/p0, PID: 1
    CPU 3: Dispatched process 1
Time slot 1
    Loaded a process at input/proc/s3, PID: 2
Time slot 2
    CPU 2: Dispatched process 2
    CPU 3: Put process 1 to run queue
    CPU 3: Dispatched process 1
Time slot 3
    Loaded a process at input/proc/m1, PID: 3
Time slot 4
    CPU 2: Put process 2 to run queue
    CPU 2: Dispatched process 2
    CPU 1: Dispatched process 3
    CPU 3: Put process 1 to run queue
    CPU 3: Dispatched process 1
Time slot 5
    Loaded a process at input/proc/s2, PID: 4
Time slot 6
    CPU 2: Put process 2 to run queue
    CPU 2: Dispatched process 2
    CPU 1: Put process 3 to run queue
    CPU 1: Dispatched process 3
    CPU 0: Dispatched process 4
    Loaded a process at input/proc/m0, PID: 5
    CPU 3: Put process 1 to run queue
    CPU 3: Dispatched process 5
Time slot 7
Time slot 8
    CPU 2: Put process 2 to run queue
    CPU 2: Dispatched process 1
    CPU 1: Put process 3 to run queue
    CPU 1: Dispatched process 2
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 3
    Loaded a process at input/proc/p1, PID: 6
    CPU 3: Put process 5 to run queue
    CPU 3: Dispatched process 6
Time slot 9
Time slot 10
    CPU 2: Put process 1 to run queue
    CPU 2: Dispatched process 4
    CPU 1: Put process 2 to run queue
    CPU 1: Dispatched process 5
    CPU 0: Put process 3 to run queue
```

Hình 16: Kết quả phần 1 của input os_1

```
CPU 3: Dispatched process 6
Time slot 9
Time slot 10
CPU 2: Put process 1 to run queue
CPU 2: Dispatched process 4
CPU 1: Put process 2 to run queue
CPU 1: Dispatched process 5
CPU 0: Put process 3 to run queue
CPU 0: Dispatched process 1
Loaded a process at input/proc/s0, PID: 7
CPU 3: Put process 6 to run queue
CPU 3: Dispatched process 2
Time slot 11
CPU 2: Put process 4 to run queue
CPU 2: Dispatched process 7
Time slot 12
CPU 1: Put process 5 to run queue
CPU 1: Dispatched process 3
CPU 0: Processed 1 has finished
CPU 0: Dispatched process 6
CPU 3: Put process 2 to run queue
CPU 3: Dispatched process 5
Time slot 13
CPU 2: Put process 7 to run queue
CPU 2: Dispatched process 4
Time slot 14
CPU 1: Processed 3 has finished
CPU 1: Dispatched process 2
CPU 0: Put process 6 to run queue
CPU 0: Dispatched process 7
CPU 3: Put process 5 to run queue
CPU 3: Dispatched process 6
Time slot 15
CPU 1: Processed 2 has finished
CPU 1: Dispatched process 5
Loaded a process at input/proc/s1, PID: 8
CPU 2: Put process 4 to run queue
CPU 2: Dispatched process 8
Time slot 16
CPU 1: Processed 5 has finished
CPU 1: Dispatched process 4
CPU 0: Put process 7 to run queue
CPU 0: Dispatched process 7
CPU 3: Put process 6 to run queue
CPU 3: Dispatched process 6
Time slot 17
CPU 2: Put process 8 to run queue
CPU 2: Dispatched process 8
```

Hình 17: Kết quả phần 2 của input os_1


```
Time slot 17
  CPU 2: Put process 8 to run queue
  CPU 2: Dispatched process 8
Time slot 18
  CPU 1: Put process 4 to run queue
  CPU 1: Dispatched process 4
  CPU 0: Put process 7 to run queue
  CPU 0: Dispatched process 7
  CPU 3: Put process 6 to run queue
  CPU 3: Dispatched process 6
Time slot 19
  CPU 2: Put process 8 to run queue
  CPU 2: Dispatched process 8
Time slot 20
  CPU 1: Put process 4 to run queue
  CPU 1: Dispatched process 4
  CPU 0: Put process 7 to run queue
  CPU 0: Dispatched process 7
  CPU 3: Processed 6 has finished
  CPU 3 stopped
Time slot 21
  CPU 2: Put process 8 to run queue
  CPU 2: Dispatched process 8
Time slot 22
  CPU 1: Processed 4 has finished
  CPU 1 stopped
  CPU 0: Put process 7 to run queue
  CPU 0: Dispatched process 7
  CPU 2: Processed 8 has finished
  CPU 2 stopped
Time slot 23
Time slot 24
  CPU 0: Put process 7 to run queue
  CPU 0: Dispatched process 7
Time slot 25
Time slot 26
  CPU 0: Put process 7 to run queue
  CPU 0: Dispatched process 7
Time slot 27
  CPU 0: Processed 7 has finished
  CPU 0 stopped

MEMORY CONTENT:
000: 00000-003ff - PID: 05 (idx 000, nxt: 001)
      003e8: 15
001: 00400-007ff - PID: 05 (idx 001, nxt: -01)
006: 01800-01bff - PID: 06 (idx 000, nxt: 007)
007: 01c00-01fff - PID: 06 (idx 001, nxt: 000)
```

Hình 18: Kết quả phần 3 của input os_1

```

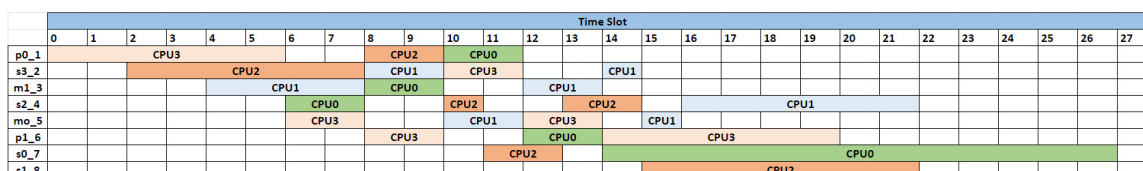
CPU 2: Put process 8 to run queue
CPU 2: Dispatched process 8
Time slot 22
CPU 1: Processed 4 has finished
CPU 1 stopped
CPU 0: Put process 7 to run queue
CPU 0: Dispatched process 7
CPU 2: Processed 8 has finished
CPU 2 stopped
Time slot 23
Time slot 24
CPU 0: Put process 7 to run queue
CPU 0: Dispatched process 7
Time slot 25
Time slot 26
CPU 0: Put process 7 to run queue
CPU 0: Dispatched process 7
Time slot 27
CPU 0: Processed 7 has finished
CPU 0 stopped

MEMORY CONTENT:
000: 00000-003ff - PID: 05 (idx 000, nxt: 001)
      003e8: 15
001: 00400-007ff - PID: 05 (idx 001, nxt: -01)
006: 01800-01bfff - PID: 06 (idx 000, nxt: 007)
007: 01c00-01ffff - PID: 06 (idx 001, nxt: 008)
008: 02000-023fff - PID: 06 (idx 002, nxt: 009)
009: 02400-027fff - PID: 06 (idx 003, nxt: -01)
019: 04c00-04ffff - PID: 01 (idx 000, nxt: -01)
      04c14: 64
022: 05800-05bfff - PID: 06 (idx 000, nxt: 023)
023: 05c00-05ffff - PID: 06 (idx 001, nxt: 031)
024: 06000-063fff - PID: 05 (idx 000, nxt: 025)
      06014: 66
025: 06400-067fff - PID: 05 (idx 001, nxt: -01)
031: 07c00-07ffff - PID: 06 (idx 002, nxt: 032)
      07de7: 0a
032: 08000-083fff - PID: 06 (idx 003, nxt: 033)
033: 08400-087fff - PID: 06 (idx 004, nxt: -01)
049: 0c400-0c7fff - PID: 05 (idx 000, nxt: 050)
050: 0c800-0cbfff - PID: 05 (idx 001, nxt: 051)
051: 0cc00-0cffff - PID: 05 (idx 002, nxt: 052)
052: 0d000-0d3fff - PID: 05 (idx 003, nxt: 053)
053: 0d400-0d7fff - PID: 05 (idx 004, nxt: -01)

NOTE: Read file output/os_1 to verify your result

```

Hình 19: Kết quả phần 4 của input os_1



Hình 20: Biểu đồ Gantt của os_1



5 Phân công công việc:

- Trần Minh Trí: Soạn báo cáo, tổng hợp code.
- Trần Quốc Vinh: Lên ý tưởng, tìm hiểu đề.
- Phạm Duy Quang: Lên ý tưởng, tìm hiểu đề, vẽ biểu đồ Gantt.
- Võ Văn Hiền: Lên ý tưởng, code, tìm hiểu đề.