

## Assignment 2: Simple operating system

Nguyễn Xuân Tiến - 1827038

Lương Thành Nhân - 1820047

Nguyễn Khải Hoàng - 1820028

May 22, 2019

# 1 Scheduler

## 1.1 Câu hỏi

Ưu điểm của giải thuật *Priority Feedback Queue* so với các giải thuật khác đã được học? Giải thuật *Priority Feedback Queue* (PFQ) trong bài tập lớn này sử dụng 2 hàng đợi ưu tiên:

- Hàng đợi `ready_queue`: có mức độ ưu tiên cao hơn. Khi CPU chọn một process để thực thi, nó sẽ ưu tiên chọn các process từ hàng đợi `ready_queue` trước.
- Hàng đợi `run_queue`: có mức độ ưu tiên thấp hơn. Các process ở hàng đợi này chỉ được thực thi sau khi thực thi hết tất cả các process ở hàng đợi `ready_queue`

Giải thuật PFQ có một số ưu điểm như sau:

- Tính linh hoạt: Giải thuật PFQ mang những ưu điểm của giải thuật *Multilevel Queue* và giải thuật *Priority Queue*, nhưng linh hoạt hơn khi cho phép các process di chuyển từ 1 hàng đợi sang hàng đợi khác. Ví dụ như khi một process ở hàng đợi `ready_queue` chiếm dụng CPU quá lâu, sử dụng hết *time quantum* của nó mà vẫn chưa thực thi xong, nó sẽ bị di chuyển sang hàng đợi `run_queue` có mức độ ưu tiên thấp hơn.  
Ngoài ra tính linh hoạt còn được thể hiện ở việc các hàng đợi đều có thứ tự, cho phép người dùng gán mức độ ưu tiên cho từng process.
- Tính công bằng: Giải thuật PFQ mang những ưu điểm của giải thuật *Round Robin* khi sử dụng các *time quantum* cho từng process thực thi, tránh được tình trạng trì hoãn vô hạn định.

## 1.2 Hiện thực

### 1.2.1 Enqueue

Đưa một process vào cuối hàng đợi nếu còn chỗ trống. Thông thường một hàng đợi ưu tiên (*priority queue*) sẽ được hiện thực bằng cấu trúc dữ liệu *heap* để có thể nhanh chóng truy xuất ra phần tử có *priority* lớn nhất (hoặc nhỏ nhất). Tuy nhiên trong phạm vi bài tập lớn này, hàng đợi tối đa chỉ có 10 phần tử nên chúng ta chỉ cần hiện thực bằng mảng.

```
1 void enqueue(struct queue_t * q, struct pcb_t * proc) {  
2     /* TODO: put a new process to queue [q] */  
3     if (q->size < MAX_QUEUE_SIZE) {  
4         q->proc[q->size] = proc;  
5         q->size += 1;  
6     }  
7 }
```

### 1.2.2 Dequeue

Duyệt một lượt qua hàng đợi để lấy ra process có mức độ ưu tiên cao nhất, sau đó cập nhật lại hàng đợi bằng cách lần lượt dịch các phần tử phía sau phần tử vừa lấy ra lên một ô trong mảng.

```

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 (q->size == 0) return NULL;
6     int chosen_proc_index = 0;
7     uint32_t max_priority = (q->proc[0])->priority;
8     // Get highest priority process
9     for (int i = 0; i < q->size; ++i){
10         if ((q->proc[i])->priority > max_priority){
11             max_priority = (q->proc[i])->priority;
12             chosen_proc_index = i;
13         }
14     }
15     struct pcb_t * chosen_pcb = (q->proc[chosen_proc_index]);
16     // Remove chosen process from the array and move others up
17     for (int i = chosen_proc_index; i < q->size - 1; ++i)
18         q->proc[i] = q->proc[i+1];
19     q->size -= 1;
20
21     return chosen_pcb;
22 }

```

### 1.2.3 Get Process Control Block

Lấy Process Control Block (PCB) của process có mức độ ưu tiên cao nhất từ hàng đợi ready\_queue. Nếu hàng đợi ready\_queue trống vào thời điểm hàm được gọi, chuyển tất cả các PCB từ hàng đợi run\_queue sang hàng đợi ready\_queue.

```

1 struct pcb_t * get_proc(void) {
2     struct pcb_t * proc = NULL;
3     /*TODO: get a process from [ready_queue]. If ready queue
4      * is empty, push all processes in [run_queue] back to
5      * [ready_queue] and return the highest priority one.
6      * Remember to use lock to protect the queue.
7      * */
8     pthread_mutex_lock(&queue_lock);
9     if (!queue_empty()){
10         if(empty(&ready_queue)){//Copy run_queue back to ready_queue
11             for (int i = 0; i < run_queue.size; ++i)
12                 ready_queue.proc[i] = run_queue.proc[i];
13             ready_queue.size = run_queue.size;
14             run_queue.size = 0;
15         }
16         proc = dequeue(&ready_queue);
17     }
18     pthread_mutex_unlock(&queue_lock);
19     return proc;
20 }

```

### 1.3 Kết quả mô phỏng

Kết quả chạy mô phỏng định thời với 2 process (sử dụng file input sched\_0)

```
1 Time slot 0
2   Loaded a process at input/proc/s0, PID: 1
3 Time slot 1
4   CPU 0: Dispatched process 1
5 Time slot 2
6 Time slot 3
7   CPU 0: Put process 1 to run queue
8   CPU 0: Dispatched process 1
9 Time slot 4
10  Loaded a process at input/proc/s1, PID: 2
11 Time slot 5
12  CPU 0: Put process 1 to run queue
13  CPU 0: Dispatched process 2
14 Time slot 6
15 Time slot 7
16  CPU 0: Put process 2 to run queue
17  CPU 0: Dispatched process 2
18 Time slot 8
19 Time slot 9
20  CPU 0: Put process 2 to run queue
21  CPU 0: Dispatched process 1
22 Time slot 10
23 Time slot 11
24  CPU 0: Put process 1 to run queue
25  CPU 0: Dispatched process 2
26 Time slot 12
27 Time slot 13
28  CPU 0: Put process 2 to run queue
29  CPU 0: Dispatched process 1
30 Time slot 14
31 Time slot 15
32  CPU 0: Put process 1 to run queue
33  CPU 0: Dispatched process 2
34 Time slot 16
35  CPU 0: Processed 2 has finished
36  CPU 0: Dispatched process 1
37 Time slot 17
38 Time slot 18
39  CPU 0: Put process 1 to run queue
40  CPU 0: Dispatched process 1
41 Time slot 19
42 Time slot 20
43  CPU 0: Put process 1 to run queue
44  CPU 0: Dispatched process 1
45 Time slot 21
46 Time slot 22
47  CPU 0: Put process 1 to run queue
48  CPU 0: Dispatched process 1
49 Time slot 23
50  CPU 0: Processed 1 has finished
51  CPU 0 stopped
52
53 MEMORY CONTENT:
```

Biểu đồ Gantt:

		p1				p2				p1		p2		p1		p2		p1						
Time slot	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Kết quả chạy mô phỏng định thời với 4 process (sử dụng file input sched\_1)

```

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

```

```

52 Time slot 24
53 Time slot 25
54 CPU 0: Put process 1 to run queue
55 CPU 0: Dispatched process 4
56 Time slot 26
57 Time slot 27
58 CPU 0: Put process 4 to run queue
59 CPU 0: Dispatched process 2
60 Time slot 28
61 CPU 0: Processed 2 has finished
62 CPU 0: Dispatched process 3
63 Time slot 29
64 Time slot 30
65 CPU 0: Put process 3 to run queue
66 CPU 0: Dispatched process 1
67 Time slot 31
68 Time slot 32
69 CPU 0: Put process 1 to run queue
70 CPU 0: Dispatched process 4
71 Time slot 33
72 Time slot 34
73 CPU 0: Put process 4 to run queue
74 CPU 0: Dispatched process 3
75 Time slot 35
76 Time slot 36
77 CPU 0: Put process 3 to run queue
78 CPU 0: Dispatched process 1
79 Time slot 37
80 Time slot 38
81 CPU 0: Put process 1 to run queue
82 CPU 0: Dispatched process 4
83 Time slot 39
84 Time slot 40
85 CPU 0: Put process 4 to run queue
86 CPU 0: Dispatched process 3
87 Time slot 41
88 Time slot 42
89 CPU 0: Processed 3 has finished
90 CPU 0: Dispatched process 1
91 Time slot 43
92 Time slot 44
93 CPU 0: Put process 1 to run queue
94 CPU 0: Dispatched process 4
95 Time slot 45
96 CPU 0: Processed 4 has finished
97 CPU 0: Dispatched process 1
98 Time slot 46
99 CPU 0: Processed 1 has finished
100 CPU 0 stopped
101
102 MEMORY CONTENT:

```

Biểu đồ Gantt:

		p1				p2		p3		p4		p2		p3		p1		p4		p2		p3		
Time slot	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	p1		p4		p2	p3		p1		p4		p3		p1		p4		p3		p1		p4	p1	
Time slot	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46

## 2 Memory Management

### 2.1 Câu hỏi

Nêu những ưu điểm và nhược điểm của phương pháp quản lý bộ nhớ segmentation with paging.

**Ưu điểm:**

- Kết hợp được những ưu điểm của cả 2 phương pháp quản lý bộ nhớ phân trang và phân đoạn.
- Bộ nhớ được sử dụng một cách linh hoạt và hiệu quả: mỗi segment có thể có độ dài khác nhau, kích thước của page table cũng giới hạn bởi độ dài của segment tương ứng.
- Tránh được hiện tượng phân mảnh ngoại (external fragmentation).

**Nhược điểm:**

- Vẫn xảy ra tình trạng phân mảnh nội (internal fragmentation).
- Hiện thực các giải thuật cấp bộ nhớ, thu hồi bộ nhớ phức tạp hơn so với khi chỉ dùng phân đoạn hoặc chỉ dùng phân trang.

### 2.2 Hiện thực

#### 2.2.1 Tìm bảng phân trang

Trong bài tập lớn này, không gian địa chỉ chúng ta đang mô phỏng có độ lớn 20 bit, trong đó 5 bit đầu tiên là segment index, 5 bit tiếp theo là page index, và 10 bit cuối cùng là offset. Với tham số đầu vào là segment table và segment index, hàm `get_page_table` sẽ duyệt lần lượt qua từng phần tử trong segment table và so sánh trường `v_index` với segment index, nếu trùng khớp sẽ trả về page table tương ứng.

```

1 static struct page_table_t * get_page_table(
2     addr_t index, // Segment level index
3     struct seg_table_t * seg_table) { // Segment table
4     int i;
5     for (i = 0; i < seg_table->size; i++) {
6         if (seg_table->table[i].v_index == index)
7             return seg_table->table[i].pages;
8     }
9     return NULL;
10 }

```

### 2.2.2 Dịch địa chỉ luận lý sang địa chỉ vật lý

Sử dụng 5 bit đầu tiên của địa chỉ luận lý (first\_lv) tra vào bảng segment index để lấy được page table. Sau đó dùng 5 bit tiếp theo (second\_lv) tra trong page table để lấy được index của trang vật lý trên RAM. Cuối cùng kết hợp index vừa tìm được với 10 bit offset của địa chỉ luận lý để tạo thành địa chỉ vật lý hoàn chỉnh.

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

### 2.2.3 Cấp phát vùng nhớ

#### Kiểm tra số trang nhớ còn trống trên RAM (Dòng 7 - 10)

Duyệt lần lượt qua mảng \_mem\_array lưu trạng thái của RAM để đếm số trang còn trống (có trường proc = 0). Số trang trống cần lớn hơn hoặc bằng số trang mà process yêu cầu thì mới tiến hành quá trình cấp bộ nhớ.

#### Kiểm tra số byte trống trên không gian địa chỉ luận lý (Dòng 11)

Sử dụng con trỏ breakpointer (bp) để kiểm tra số byte còn lại trên không gian địa chỉ luận lý. Nếu con số này lớn hơn hoặc bằng số byte mà process yêu cầu thì mới tiến hành quá trình cấp bộ nhớ.

#### Cập nhật bảng trạng thái các trang trên RAM (Dòng 20 - 27)

Nếu như thỏa mãn 2 yêu cầu trên, quá trình cấp bộ nhớ bắt đầu như sau. Lần lượt duyệt qua mảng \_mem\_stat lưu trạng thái RAM, nếu tìm thấy một trang trống thì cập nhật lại các trường proc, index, next để cấp trang đấy cho process hiện tại. Lặp lại quá trình này cho đến khi đã cấp đủ số trang mà process yêu cầu.

#### Cập nhật segment table và page table tương ứng (Dòng 29 - 43)



Sử dụng con trỏ breakpointer, ta biết được địa chỉ tiếp theo để cấp phát trên không gian địa chỉ luận lý. Từ địa chỉ luận lý, ta sử dụng 5 bit đầu tiên để tra bảng segment table, 5 bit tiếp theo để tra bảng page table và cập nhật lại các trường dữ liệu để lần tiếp theo process truy cập vào địa chỉ luận lý này sẽ dịch ra một địa chỉ vật lý hợp lệ.

```

1  addr_t alloc_mem(uint32_t size, struct pcb_t * proc) {
2      pthread_mutex_lock(&mem_lock);
3      addr_t ret_mem = 0;
4      uint32_t num_pages = ((size % PAGE_SIZE) == 0) ? size / PAGE_SIZE :
5          size / PAGE_SIZE + 1; // Number of pages we will use
6      int mem_avail = 0; // Variable to indicate if we can allocate memory or
7          not
8      uint32_t num_pages_available = 0;
9      for (int i = 0; i < NUM_PAGES; ++i){
10         if (_mem_stat[i].proc == 0) num_pages_available += 1;
11     }
12     uint32_t bytes_available_virtual_space = (RAM_SIZE) - (proc->bp);
13     mem_avail = (num_pages_available >= num_pages &&
14         bytes_available_virtual_space > size)?1:0;
15     if (mem_avail) {
16         /* We could allocate new memory region to the process */
17         ret_mem = proc->bp;
18         proc->bp += num_pages * PAGE_SIZE;
19         uint32_t num_pages_allocated = 0;
20         uint32_t i;
21         uint32_t previous_i;
22         for (i = 0; i < NUM_PAGES; ++i){
23             if (_mem_stat[i].proc != 0) continue;
24             // Find a free page in RAM
25             // Update _mem_stat
26             _mem_stat[i].proc = proc->pid;
27             _mem_stat[i].index = num_pages_allocated;
28             if (num_pages_allocated > 0)
29                 _mem_stat[previous_i].next = i;
30             // Update seg_table of current process
31             addr_t virtual_address = ret_mem + num_pages_allocated * PAGE_SIZE;
32             addr_t first_lv = get_first_lv(virtual_address);
33             addr_t second_lv = get_second_lv(virtual_address);
34             struct page_table_t * page_table = get_page_table(first_lv, proc->
35                 seg_table);
36             if (page_table == NULL){
37                 int last_index = proc->seg_table->size;
38                 proc->seg_table->table[last_index].v_index = first_lv;
39                 proc->seg_table->table[last_index].pages = (struct page_table_t *)
40                 malloc(sizeof(struct page_table_t));
41                 page_table = proc->seg_table->table[last_index].pages;
42                 page_table->size = 0;
43                 proc->seg_table->size += 1;
44             }
45             page_table->table[page_table->size].v_index = second_lv;
46             page_table->table[page_table->size].p_index = i;
47             page_table->size += 1;
48             previous_i = i;
49             num_pages_allocated += 1;
50             if (num_pages_allocated == num_pages){ // Stop when enough pages are
51                 allocated
52                 _mem_stat[i].next = -1;
53                 break;

```

```

49     }
50 } // end for loop
51 } // end mem_avail
52 pthread_mutex_unlock(&mem_lock);
53 return ret_mem;
54 }

```

### 2.2.4 Thu hồi vùng nhớ

Ngược lại so với quá trình cấp phát bộ nhớ, ở quá trình thu hồi bộ nhớ, ta cần đánh dấu các trang vật lý đã cấp trong mảng `_mem_stat` là trống bằng cách gán trường `proc = 0` (Dòng 4 - 13). Tiếp theo xóa đi các entry tương ứng ở segment table và page table (Dòng 14 - 29).

```

1  int free_mem(addr_t address, struct pcb_t * proc) {
2  pthread_mutex_lock(&mem_lock);
3  // Set flag of physical pages used by the block to zero
4  addr_t physical_address;
5  translate(address, &physical_address, proc);
6  uint32_t index_in_RAM = physical_address / PAGE_SIZE;
7  uint32_t num_pages = 0;
8  while(1){
9      _mem_stat[index_in_RAM].proc = 0;
10     ++num_pages;
11     if (_mem_stat[index_in_RAM].next == -1) break;
12     index_in_RAM = _mem_stat[index_in_RAM].next;
13 }
14 // Remove unused entries in segment table, page tables of the process
15 for (int i = 0; i < num_pages; i++) {
16     addr_t virtual_addr_to_clean = address + i * PAGE_SIZE;
17     addr_t first_lv = get_first_lv(virtual_addr_to_clean);
18     addr_t second_lv = get_second_lv(virtual_addr_to_clean);
19     struct page_table_t * page_table = get_page_table(first_lv, proc->
seg_table);
20     int j;
21     for (j = 0; j < page_table->size; j++) {
22         if (page_table->table[j].v_index == second_lv) {
23             int last_index = page_table->size - 1;
24             page_table->table[j] = page_table->table[last_index];
25             page_table->size -= 1;
26             break;
27         }
28     }
29 }
30 pthread_mutex_unlock(&mem_lock);
31 return 0;
32 }

```

## 2.3 Kết quả mô phỏng

Trạng thái bộ nhớ RAM sau mỗi lệnh alloc\_mem và free\_mem khi mô phỏng như sau:

**Test 0: input file m0**

```
1  ===== Allocate memory =====
2  000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
3  001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
4  002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
5  003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
6  004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
7  005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
8  006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
9  007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
10 008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
11 009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
12 010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
13 011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
14 012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
15 013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
16  ===== Allocate memory =====
17 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
18 001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
19 002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
20 003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
21 004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
22 005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
23 006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
24 007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
25 008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
26 009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
27 010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
28 011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
29 012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
30 013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
31 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
32 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
33  ===== Deallocate memory =====
34 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
35 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
36  ===== Allocate memory =====
37 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
38 001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
39 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
40 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
41  ===== Allocate memory =====
42 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
43 001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
44 002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
45 003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
46 004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
47 005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
48 006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
49 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
50  ===== Final memory state =====
51 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
52 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
53   003e8: 15
54 001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
```

```

55 002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
56 003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
57 004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
58 005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
59 006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
60 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
61 03814: 66
62 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)

```

### Test 1: input file m1

```

1  ===== Allocate memory =====
2 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
3 001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
4 002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
5 003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
6 004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
7 005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
8 006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
9 007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
10 008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
11 009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
12 010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
13 011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
14 012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
15 013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
16  ===== Allocate memory =====
17 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
18 001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
19 002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
20 003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
21 004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
22 005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
23 006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
24 007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
25 008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
26 009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
27 010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
28 011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
29 012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
30 013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
31 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
32 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
33  ===== Deallocate memory =====
34 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
35 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
36  ===== Allocate memory =====
37 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
38 001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
39 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
40 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
41  ===== Allocate memory =====
42 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
43 001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
44 002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
45 003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
46 004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
47 005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
48 006: 01800-01bff - PID: 01 (idx 004, nxt: -01)

```

```

49 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
50 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
51 ===== Deallocate memory=====
52 002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
53 003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
54 004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
55 005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
56 006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
57 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
58 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
59 ===== Deallocate memory=====
60 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
61 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
62 ===== Deallocate memory=====
63
64 ===== Final memory state=====

```

### 3 Putting it all together

Test 0 cho ra kết quả như sau:

```

1 Time slot 0
2   Loaded a process at input/proc/p0, PID: 1
3   CPU 1: Dispatched process 1
4 Time slot 1
5   Loaded a process at input/proc/p1, PID: 2
6 Time slot 2
7   CPU 0: Dispatched process 2
8 Time slot 3
9   Loaded a process at input/proc/p1, PID: 3
10 Time slot 4
11   Loaded a process at input/proc/p1, PID: 4
12 Time slot 5
13 Time slot 6
14   CPU 1: Put process 1 to run queue
15   CPU 1: Dispatched process 3
16 Time slot 7
17 Time slot 8
18   CPU 0: Put process 2 to run queue
19   CPU 0: Dispatched process 4
20 Time slot 9
21 Time slot 10
22 Time slot 11
23 Time slot 12
24   CPU 1: Put process 3 to run queue
25   CPU 1: Dispatched process 1
26 Time slot 13
27 Time slot 14
28   CPU 0: Put process 4 to run queue
29   CPU 0: Dispatched process 2
30 Time slot 15
31 Time slot 16
32   CPU 1: Processed 1 has finished
33   CPU 1: Dispatched process 3
34 Time slot 17
35 Time slot 18

```

```

36 CPU 0: Processed 2 has finished
37 CPU 0: Dispatched process 4
38 Time slot 19
39 Time slot 20
40 CPU 1: Processed 3 has finished
41 CPU 1 stopped
42 Time slot 21
43 Time slot 22
44 CPU 0: Processed 4 has finished
45 CPU 0 stopped
46
47 MEMORY CONTENT:
48 000: 00000-003ff - PID: 03 (idx 000, nxt: 001)
49 001: 00400-007ff - PID: 03 (idx 001, nxt: 006)
50 002: 00800-00bff - PID: 02 (idx 000, nxt: 003)
51 003: 00c00-00fff - PID: 02 (idx 001, nxt: 004)
52 004: 01000-013ff - PID: 02 (idx 002, nxt: 005)
53 005: 01400-017ff - PID: 02 (idx 003, nxt: -01)
54 006: 01800-01bff - PID: 03 (idx 002, nxt: 012)
55 007: 01c00-01fff - PID: 02 (idx 000, nxt: 008)
56 008: 02000-023ff - PID: 02 (idx 001, nxt: 009)
57 009: 02400-027ff - PID: 02 (idx 002, nxt: 010)
58 025e7: 0a
59 010: 02800-02bff - PID: 02 (idx 003, nxt: 011)
60 011: 02c00-02fff - PID: 02 (idx 004, nxt: -01)
61 012: 03000-033ff - PID: 03 (idx 003, nxt: -01)
62 013: 03400-037ff - PID: 04 (idx 000, nxt: 024)
63 014: 03800-03bff - PID: 03 (idx 000, nxt: 015)
64 015: 03c00-03fff - PID: 03 (idx 001, nxt: 016)
65 016: 04000-043ff - PID: 03 (idx 002, nxt: 017)
66 041e7: 0a
67 017: 04400-047ff - PID: 03 (idx 003, nxt: 018)
68 018: 04800-04bff - PID: 03 (idx 004, nxt: -01)
69 019: 04c00-04fff - PID: 04 (idx 000, nxt: 020)
70 020: 05000-053ff - PID: 04 (idx 001, nxt: 021)
71 021: 05400-057ff - PID: 04 (idx 002, nxt: 022)
72 055e7: 0a
73 022: 05800-05bff - PID: 04 (idx 003, nxt: 023)
74 023: 05c00-05fff - PID: 04 (idx 004, nxt: -01)
75 024: 06000-063ff - PID: 04 (idx 001, nxt: 025)
76 025: 06400-067ff - PID: 04 (idx 002, nxt: 026)
77 026: 06800-06bff - PID: 04 (idx 003, nxt: -01)
78 047: 0bc00-0bfff - PID: 01 (idx 000, nxt: -01)
79 0bc14: 64

```

Biểu đồ Gantt:

CPU 0		p2						p4						p2			p4						
CPU 1	p1						p3						p1			p3							
Time slot	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

Test 1 cho ra kết quả như sau:

```

1 Time slot 0
2   Loaded a process at input/proc/p0, PID: 1
3 Time slot 1
4   CPU 3: Dispatched process 1
5 Time slot 2

```

```

6   Loaded a process at input/proc/s3, PID: 2
7   CPU 2: Dispatched process 2
8   CPU 3: Put process 1 to run queue
9   CPU 3: Dispatched process 1
10  Time slot 3
11  Loaded a process at input/proc/m1, PID: 3
12  CPU 2: Put process 2 to run queue
13  CPU 2: Dispatched process 3
14  Time slot 4
15  CPU 1: Dispatched process 2
16  CPU 3: Put process 1 to run queue
17  CPU 3: Dispatched process 1
18  Time slot 5
19  Loaded a process at input/proc/s2, PID: 4
20  Time slot 6
21  CPU 1: Put process 2 to run queue
22  CPU 1: Dispatched process 4
23  CPU 2: Put process 3 to run queue
24  CPU 2: Dispatched process 2
25  CPU 0: Dispatched process 3
26  CPU 3: Put process 1 to run queue
27  CPU 3: Dispatched process 1
28  Time slot 7
29  Loaded a process at input/proc/m0, PID: 5
30  CPU 2: Put process 2 to run queue
31  Time slot 8
32  CPU 2: Dispatched process 5
33  CPU 1: Put process 4 to run queue
34  CPU 1: Dispatched process 4
35  CPU 0: Put process 3 to run queue
36  CPU 0: Dispatched process 2
37  Loaded a process at input/proc/p1, PID: 6
38  CPU 3: Put process 1 to run queue
39  CPU 3: Dispatched process 6
40  Time slot 9
41  Time slot 10
42  CPU 0: Put process 2 to run queue
43  CPU 0: Dispatched process 2
44  CPU 1: Put process 4 to run queue
45  CPU 1: Dispatched process 3
46  CPU 2: Put process 5 to run queue
47  CPU 2: Dispatched process 1
48  Loaded a process at input/proc/s0, PID: 7
49  CPU 3: Put process 6 to run queue
50  CPU 3: Dispatched process 7
51  Time slot 11
52  CPU 2: Processed 1 has finished
53  Time slot 12
54  CPU 2: Dispatched process 4
55  CPU 0: Put process 2 to run queue
56  CPU 0: Dispatched process 5
57  CPU 1: Put process 3 to run queue
58  CPU 1: Dispatched process 6
59  CPU 3: Put process 7 to run queue
60  CPU 3: Dispatched process 7
61  Time slot 13
62  Time slot 14
63  CPU 1: Put process 6 to run queue

```

```

64 CPU 1: Dispatched process 2
65 CPU 0: Put process 5 to run queue
66 CPU 0: Dispatched process 3
67 CPU 2: Put process 4 to run queue
68 CPU 2: Dispatched process 4
69 CPU 3: Put process 7 to run queue
70 CPU 3: Dispatched process 6
71 Time slot 15
72 CPU 1: Processed 2 has finished
73 CPU 1: Dispatched process 5
74 Loaded a process at input/proc/s1, PID: 8
75 CPU 2: Put process 4 to run queue
76 CPU 2: Dispatched process 8
77 Time slot 16
78 CPU 0: Processed 3 has finished
79 CPU 0: Dispatched process 4
80 CPU 3: Put process 6 to run queue
81 CPU 3: Dispatched process 7
82 Time slot 17
83 CPU 1: Put process 5 to run queue
84 CPU 1: Dispatched process 6
85 CPU 2: Put process 8 to run queue
86 CPU 2: Dispatched process 5
87 Time slot 18
88 CPU 0: Put process 4 to run queue
89 CPU 0: Dispatched process 8
90 CPU 3: Put process 7 to run queue
91 CPU 2: Processed 5 has finished
92 Time slot 19
93 CPU 2: Dispatched process 7
94 CPU 3: Dispatched process 4
95 CPU 1: Put process 6 to run queue
96 CPU 1: Dispatched process 6
97 Time slot 20
98 CPU 0: Put process 8 to run queue
99 CPU 0: Dispatched process 8
100 CPU 3: Processed 4 has finished
101 CPU 3 stopped
102 CPU 2: Put process 7 to run queue
103 CPU 2: Dispatched process 7
104 Time slot 21
105 CPU 1: Processed 6 has finished
106 CPU 1 stopped
107 CPU 0: Put process 8 to run queue
108 CPU 0: Dispatched process 8
109 Time slot 22
110 CPU 2: Put process 7 to run queue
111 CPU 2: Dispatched process 7
112 Time slot 23
113 CPU 0: Processed 8 has finished
114 CPU 0 stopped
115 Time slot 24
116 Time slot 25
117 CPU 2: Put process 7 to run queue
118 CPU 2: Dispatched process 7
119 Time slot 26
120 Time slot 27
121 CPU 2: Put process 7 to run queue

```



```

122 CPU 2: Dispatched process 7
123 Time slot 28
124 CPU 2: Processed 7 has finished
125 CPU 2 stopped
126
127 MEMORY CONTENT:
128 000: 00000-003ff - PID: 01 (idx 000, nxt: -01)
129 00014: 64
130 001: 00400-007ff - PID: 05 (idx 000, nxt: 006)
131 006: 01800-01bff - PID: 05 (idx 001, nxt: 007)
132 007: 01c00-01fff - PID: 05 (idx 002, nxt: 008)
133 008: 02000-023ff - PID: 05 (idx 003, nxt: 009)
134 009: 02400-027ff - PID: 05 (idx 004, nxt: -01)
135 010: 02800-02bff - PID: 06 (idx 000, nxt: 011)
136 011: 02c00-02fff - PID: 06 (idx 001, nxt: 012)
137 012: 03000-033ff - PID: 06 (idx 002, nxt: 013)
138 013: 03400-037ff - PID: 06 (idx 003, nxt: -01)
139 024: 06000-063ff - PID: 05 (idx 000, nxt: 025)
140 06014: 66
141 025: 06400-067ff - PID: 05 (idx 001, nxt: -01)
142 026: 06800-06bff - PID: 06 (idx 000, nxt: 027)
143 027: 06c00-06fff - PID: 06 (idx 001, nxt: 028)
144 028: 07000-073ff - PID: 06 (idx 002, nxt: 029)
145 071e7: 0a
146 029: 07400-077ff - PID: 06 (idx 003, nxt: 030)
147 030: 07800-07bff - PID: 06 (idx 004, nxt: -01)
148 052: 0d000-0d3ff - PID: 05 (idx 000, nxt: 053)
149 0d3e8: 15
150 053: 0d400-0d7ff - PID: 05 (idx 001, nxt: -01)

```

Biểu đồ Gantt:

