

DẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & KĨ THUẬT MÁY TÍNH



OPERATING SYSTEMS

Assignment 2

Simple Operating System

GVHD: La Hoàng Lộc

SV thực hiện: Trịnh Duy Hưng – 1913652

Trần Lê Công Minh – 1910347

Lê Anh Vũ – 1915972

June 2021



Mục lục

1 Scheduler	1
1.1 Câu hỏi	1
1.2 Hiện thực	2
2 Memory Management	8
2.1 Câu hỏi	8
2.2 Hiện thực	9
3 Put it all together	13



1 Scheduler

1.1 Câu hỏi

Question: What is the advantage of using priority feedback queue in comparison with other scheduling algorithms you have learned?

Giải thuật Priority Feedback Queue (hàng đợi phản hồi ưu tiên) sử dụng 2 hàng đợi là ready_queue và run_queue như sau:

- ready_queue: hàng đợi chứa các process ở mức độ ưu tiên thực thi . Khi CPU chuyển sang slot tiếp theo, nó sẽ tìm kiếm process với độ ưu tiên cao nhất trong hàng đợi này.
- run_queue: hàng đợi này chứa các process đang chờ để tiếp tục thực thi sau khi hết slot của nó mà chưa hoàn tất quá trình của mình. Các process ở hàng đợi này chỉ được tiếp tục slot tiếp theo khi ready_queue trống và được đưa sang hàng đợi ready_queue (quá trình feed back) để tiếp tục quá trình thực hiện tiếp theo.
- Cả hai hàng đợi đều là hàng đợi có độ ưu tiên, mức độ ưu tiên dựa trên mức độ ưu tiên của process trong hàng đợi.

Từ đó, Priority Feedback Queue (hàng đợi phản hồi ưu tiên) có những ưu điểm:

- Là một thuật toán có độ linh hoạt cao trong phân chia thực hiện công việc.
- Thuật toán này cho phép processes khác nhau có thể di chuyển giữa 2 hàng đợi là run_queue và ready_queue. Những process ngắn và cần thiết nhất sẽ nhanh chóng được hoàn thành, nhường thời gian thực thi cho các process khác.
- Các process đợi quá lâu trong hàng đợi độ ưu tiên thấp sẽ được di chuyển vào hàng đợi có độ ưu tiên cao. Tạo sự công bằng về thời gian thực thi giữa các process, tránh tình trạng chiếm CPU sử dụng, trì hoãn vô thời định

1.2 Hiện thực

- Hiện thực enqueue() and dequeue() functions trong queue.c

```
● ● ●

1 struct pcb_t * get_proc(void) {
2     /*
3      * 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
9     struct pcb_t * proc = NULL;
10
11    // Queue is empty
12    if (queue_empty())
13        return proc;
14
15    pthread_mutex_lock(&queue_lock);
16
17    // Feed back
18    if (ready_queue.size == 0)
19    {
20        while (run_queue.size > 0)
21        {
22            enqueue(&ready_queue, dequeue(&run_queue));
23        }
24    }
25
26    // Get dispatched process
27    proc = dequeue(&ready_queue);
28
29    pthread_mutex_unlock(&queue_lock);
30
31    return proc;
32 }
```

- Hiện thực get proc() functions trong sched.c

```
● ○ ●
1 void enqueue(struct queue_t * q, struct pcb_t * proc)
2 {
3     /* TODO: put a new process to queue [q] */
4     if (q->size >= MAX_QUEUE_SIZE)
5         return;
6
7     q->proc[q->size++] = proc;
8 }
9
10 struct pcb_t * dequeue(struct queue_t * q)
11 {
12     /*
13      * return a pcb whose priority is the highest
14      * in the queue [q] and remember to remove it from q
15      */
16
17     int i, idx = 0;
18     if (q->size > 0)
19     {
20         // Find the highest-priority process
21         for (i = 1; i < q->size; i++)
22         {
23             if (q->proc[idx].priority < q->proc[i].priority)
24                 idx = i;
25         }
26
27         // Update remaining queue
28         struct pcb_t* rem = q->proc[idx];
29         for (i = idx + 1; i < q->size; i++)
30         {
31             q->proc[i-1] = q->proc[i];
32         }
33
34         q->size--;
35         return rem;
36     }
37
38     return NULL;
39 }
```

- Kết quả

- Test 0

- * Input: sched_0

```
2 1 2
0 s0
4 s1
```

* Output: sched_0

```
----- SCHEDULING TEST 0 -----
./os sched_0
Time slot 0
    Loaded a process at input/proc/s0, PID: 1
Time slot 1
    CPU 0: Dispatched process 1
Time slot 2
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 3
Time slot 4
    Loaded a process at input/proc/s1, PID: 2
    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 2
Time slot 7
Time slot 8
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 1
Time slot 9
Time slot 10
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 2
Time slot 11
Time slot 12
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 1
Time slot 13
Time slot 14
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 2
Time slot 15
    CPU 0: Processed 2 has finished
    CPU 0: Dispatched process 1
Time slot 16
Time slot 17
    CPU 0: Put process 1 to run queue
    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
Time slot 23
```

* Gantt diagram



* Giải thích:

Time slice: 2

Numbers of CPU: 1

2 processes to be run

s0 có độ ưu tiên là 12, thực hiện 15 lệnh

s1 có độ ưu tiên là 20, thực hiện 7 lệnh

- Tại time slot 0: Quá trình s0 được tải vào ready_queue, gán cho PID = 1
- Tại time slot 1: Quá trình s0 được cử đi vào CPU
- Tại time slot 2: Quá trình s0 vào run_queue, Thực hiện feedback (vì ready_queue trống), Quá trình s0 được cử đi vào CPU
- Tại time slot 3: Quá trình s0 đang thực thi trong CPU
- Tại time slot 4: Quá trình s1 được tải vào ready_queue, gán cho PID = 2,

Quá trình s0 vào run_queue,

Quá trình s1 được cử đi vào CPU

- Tại time slot 5: Quá trình s1 đang thực thi trong CPU
- Tại time slot 6: Quá trình s1 vào run_queue,
Thực hiện feedback (vì ready_queue trống),
Quá trình s1 được cử đi vào CPU (s1 có độ ưu tiên cao hơn)
- Tại time slot 6: Quá trình s1 vào run_queue,
Thực hiện feedback (vì ready_queue trống),
Quá trình s1 được cử đi vào CPU (s1 có độ ưu tiên cao hơn)
- Tại time slot 7: Quá trình s1 đang thực thi
- Tại time slot 8: Quá trình s1 vào run_queue
Quá trình s0 được cử đi vào CPU
- Tại time slot 9: Quá trình s0 đang thực thi
- Tại time slot 10: Quá trình s0 vào run queue
Thực hiện feedback (vì ready_queue trống)
Quá trình s1 được cử đi vào CPU (s1 có độ ưu tiên cao hơn)
- Tại time slot 15: Quá trình s1 thực thi xong,
Quá trình s0 được cử đi vào CPU
- Tại time slot 22: Quá trình s0 thực thi xong.
Kết thúc.

– Test 1

* Input: sched_1

```
2 1 4
0 s0
4 s1
6 s2
7 s3
```

* Output: sched_1

```
----- SCHEDULING TEST 1 -----
./os sched_1
Time slot 0
    Loaded a process at input/proc/s0, PID: 1
    CPU 0: Dispatched process 1
Time slot 1
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 2
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 3
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot 4
    Loaded a process at input/proc/s1, PID: 2
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 2
Time slot 5
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 3
Time slot 6
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 4
Time slot 7
    Loaded a process at input/proc/s3, PID: 4
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 4
Time slot 8
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 2
Time slot 9
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 3
Time slot 10
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 3
Time slot 11
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 1
Time slot 12
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 4
Time slot 13
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 14
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 4
Time slot 15
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 4
Time slot 16
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 2
Time slot 17
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 2
Time slot 18
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 1
Time slot 19
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 4
Time slot 20
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 3
Time slot 21
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 4
Time slot 22
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 3
Time slot 23
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 2
Time slot 24
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 1
Time slot 25
    CPU 0: Processed 2 has finished
    CPU 0: Dispatched process 3
Time slot 26
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 4
Time slot 27
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 1
Time slot 28
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 2
Time slot 29
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 3
Time slot 30
    CPU 0: Processed 3 has finished
    CPU 0: Dispatched process 1
Time slot 31
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 4
Time slot 32
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 3
Time slot 33
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 1
Time slot 34
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 3
Time slot 35
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 4
Time slot 36
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 1
Time slot 37
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 4
Time slot 38
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 3
Time slot 39
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 40
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 4
Time slot 41
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 1
Time slot 42
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 3
Time slot 43
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 4
Time slot 44
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 1
Time slot 45
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 4
Time slot 46
    CPU 0: Processed 4 has finished
    CPU 0: Dispatched process 1
Time slot 47
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 4
Time slot 48
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 3
Time slot 49
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 1
Time slot 50
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 4
Time slot 51
    CPU 0: Processed 1 has finished
```

* Giải thích:

Time slice: 2

CPU: 1

Total 4 processes to be run

s0 có độ ưu tiên là 12, thực hiện 15 lệnh

s1 có độ ưu tiên là 20, thực hiện 7 lệnh

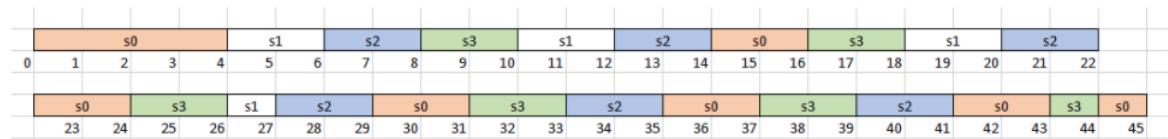
s2 có độ ưu tiên là 20, thực hiện 12 lệnh

s3 có độ ưu tiên là 7, thực hiện 11 lệnh

- Tại time slot 0: Quá trình s0 được tải vào với PID = 1
- Quá trình s0 được cử vào CPU
- Quá trình s0 đang thực thi trong CPU
- Tại time slot 1: Quá trình s0 đang thực thi trong CPU
- Tại time slot 2: Quá trình s0 vào run queue
- Thực hiện feedback
- Quá trình s0 được cử vào CPU

- Tại time slot 3: Quá trình s0 đang thực thi trong CPU
- Tại time slot 4: Quá trình s1 được tải vào với PID = 2 (Ready queue = s1), Quá trình s0 vào run queue, Quá trình s1 được cử đi vào CPU
- Tại time slot 5: Quá trình s2 được tải vào với PID = 3, Quá trình s1 đang thực thi trong CPU
- Tại time slot 6: Quá trình s1 vào run queue Ready queue = s2, Quá trình s2 được cử đi vào CPU
- Tại time slot 7: Quá trình s3 được tải vào với PID = 4, Quá trình s2 đang thực thi trong CPU
- Tại time slot 8: Quá trình s2 vào run queue, Ready queue = s3, Quá trình s3 được cử đi vào CPU
- Tại time slot 9: Quá trình s3 đang thực thi trong CPU
- Tại time slot 10: Quá trình s3 vào run queue Thực hiện feedback (Ready queue =)
Quá trình s1 được cử vào CPU (vì nó có độ ưu tiên cao nhất và nó đến trước quá trình s2)
- Tại time slot 27: Quá trình s1 hoàn thành
- Tại time slot 41: Quá trình s2 hoàn thành
- Tại time slot 44: Quá trình s0 hoàn thành
- Tại time slot 45: Quá trình s3 hoàn thành
Kết thúc.

* **Gantt diagram**





2 Memory Management

2.1 Câu hỏi

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

Ưu điểm:

- Sử dụng ít bộ nhớ
- Kích thước Page table bị giới hạn bởi kích thước Segment table
- Không gây nên External Fragmentation
- Đơn giản hóa việc cấp phát bộ nhớ

Nhược điểm:

- Có thể gây ra Internal Fragmentation
- Mức độ phức tạp sẽ cao hơn nhiều so với Paging
- Page Tables cần được lưu trữ liên tục trong memory

2.2 Hiện thực

- Hiện thực hàm get_page_table và traslate

```
● ● ●
1 /* Search for page table table from the a segment table */
2 static struct page_table_t * get_page_table(
3     addr_t index, // Segment level index
4     struct seg_table_t * seg_table) // first level table
5 {
6     /*
7      * Given the Segment index [index], you must go through each
8      * row of the segment table [seg_table] and check if the v_index
9      * field of the row is equal to the index
10     */
11
12     if (seg_table == NULL)
13         return NULL;
14
15     // Loop through first levels
16     for (int i = 0; i < seg_table->size; i++)
17     {
18         if (seg_table->table[i].v_index == index)
19             return seg_table->table[i].pages;
20     }
21
22     return NULL;
23 }
24 }
```

```
● ● ●
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) // Process uses given virtual address
5 {
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
18    if (page_table == NULL)
19        return 0;
20
21    // int i;
22    for (int i = 0; i < page_table->size; i++)
23    {
24        if (page_table->table[i].v_index == second_lv)
25        {
26            /*
27             * 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
32            // physical page index
33            addr_t p_index = page_table->table[i].p_index;
34            * physical_addr = (p_index << OFFSET_LEN) + offset;
35
36            return 1;
37        }
38    }
39
40    return 0;
41 }
```

- Hiệu thực alloc_mem

```
1 int check_available_free_mem(int num_pages, struct pcb_t * proc){  
2     int count_page = 0;  
3  
4     //check physical space  
5     for (int i = 0; i < NUM_PAGES; i++)  
6     {  
7         if (_mem_stat[i].proc == 0)  
8         {  
9             count_page++;  
10        }  
11  
12        if (count_page == num_pages)  
13            break;  
14    }  
15    if (count_page < num_pages)  
16        return 0;  
17  
18    //check virtual space  
19    if ((proc->bp + (num_pages*PAGE_SIZE) > (1 << ADDRESS_SIZE))  
20        return 0;  
21  
22    return 1;  
23 }  
24  
25 addr_t alloc_mem(uint32_t size, struct pcb_t * proc) {  
26     pthread_mutex_lock(&mem_lock);  
27     addr_t ret_mem = 0;  
28     /*  
29      * Allocate [size] byte in the memory for the  
30      * process [proc] and save the address of the first  
31      * byte in the allocated memory region to [ret_mem].  
32      * */  
33  
34     // Number of pages we will use  
35     uint32_t num_pages = (size % PAGE_SIZE == 0) ? size / PAGE_SIZE : size / PAGE_SIZE + 1;  
36  
37     // We could allocate new memory region or not?  
38     int mem_avail = 0;  
39  
40     /*  
41      * First we must check if the amount of free memory in  
42      * virtual address space and physical address space is  
43      * large enough to represent the amount of required  
44      * memory.  
45      * If so, set 1 to [mem_avail].  
46      *  
47      * Hint: check [proc] bit in each page of _mem_stat  
48      * to know whether this page has been used by a process.  
49      *  
50      * For virtual memory space, check bp (break pointer).  
51      * */  
52  
53     mem_avail = check_available_free_mem(num_pages, proc);  
54  
55     if (mem_avail)  
56     {  
57         /* We could allocate new memory region to the process */  
58         ret_mem = proc->bp;  
59         proc->bp += num_pages * PAGE_SIZE;  
60         /*  
61          * Update status of physical pages which will be allocated  
62          * to [proc] in _mem_stat. Tasks to do:  
63          * - Update [proc], [index], and [next] field  
64          * - Add entries to segment table page tables of [proc]  
65          * to ensure accesses to allocated memory slot is  
66          * valid. */  
67  
68         int idx = 0;  
69         int idx_last_page = -1;  
70         for (int i = 0; i < NUM_PAGES; i++)  
71         {  
72             if (_mem_stat[i].proc == 0)  
73             {  
74                 //update status _mem_stat  
75                 _mem_stat[i].proc = proc->pid;  
76                 _mem_stat[i].index = idx;  
77  
78                 if (idx_last_page > -1)  
79                     _mem_stat[idx_last_page].next = i;  
80  
81                 idx_last_page = i;  
82  
83                 //Add entries to segment table page tables of [proc]  
84                 v_address = ret_mem + idx * PAGE_SIZE;  
85                 addr_t segment_idx = get_first_lv(v_address);  
86                 struct page_table_t * v_page_table = get_page_table(segment_idx, proc->seg_table);  
87  
88                 if (v_page_table == NULL)  
89                 {  
90                     int new_idx = proc->seg_table->size;  
91  
92                     proc->seg_table->table[new_idx].v_index = segment_idx;  
93  
94                     v_page_table = proc->seg_table->table[new_idx].pages  
95                     = (struct page_table_t *) malloc(sizeof(struct page_table_t));  
96  
97                     proc->seg_table->size++;  
98  
99                     v_page_table->size = 0;  
100                }  
101                int new_page_idx = v_page_table->size++;  
102                v_page_table->table[new_page_idx].v_index = get_second_lv(v_address);  
103                v_page_table->table[new_page_idx].p_index = i;  
104  
105                if (++idx == num_pages)  
106                {  
107                    _mem_stat[i].next = -1;  
108                    break;  
109                }  
110            }  
111        }  
112    }  
113  
114    //dump();  
115  
116    pthread_mutex_unlock(&mem_lock);  
117  
118    return ret_mem;  
119 }  
120 }
```

- Hiệu thực free_mem

```
1 void remove_page(addr_t seg_idx, struct seg_table_t * table)
2 {
3     if (table == NULL)
4         return;
5
6     for (int i = 0; i < table->size; i++)
7     {
8         if (table->table[i].v_index == seg_idx)
9         {
10            int last_index = --table->size;
11            table->table[i] = table->table[last_index];
12            table->table[last_index].v_index = 0;
13
14            free(table->table[last_index].pages);
15        }
16    }
17 }
18
19 int free_mem(addr_t address, struct pcb_t * proc)
20 /*
21  * Release memory region allocated by [proc]. The first byte of
22  * this region is indicated by [address]. Task to do:
23  * - Set flag [proc] of physical page use by the memory block
24  * back to zero to indicate that it is free.
25  * - Remove unused entries in segment table and page tables of
26  * the process [proc].
27  * - Remember to use lock to protect the memory from other
28  * processes. */
29
30 pthread_mutex_lock(&mem_lock);
31 addr_t v_address = address;
32 addr_t p_address = 0;
33
34 if (!translate(v_address, &p_address, proc))
35     return 1;
36
37 //clear physical page
38 addr_t p_index = p_address >> OFFSET_LEN;
39 int cnt_pages = 0;
40 for (int i = p_index; i != -1; i = _mem_stat[i].next)
41 {
42     _mem_stat[i].proc = 0;
43     cnt_pages++;
44 }
45
46 //clear virtual page
47 for (int i = 0; i < cnt_pages; i++)
48 {
49     v_address = address + i * PAGE_SIZE;
50     addr_t seg_idx = get_first_lv(v_address);
51     addr_t page_idx = get_second_lv(v_address);
52
53     struct page_table_t * v_page_table = get_page_table(seg_idx, proc->seg_table);
54
55     if (v_page_table != NULL)
56     {
57         for (int j = 0; j < v_page_table->size; j++)
58         {
59             if (v_page_table->table[j].v_index == page_idx)
60             {
61                 v_page_table->table[j] = v_page_table->table[-v_page_table->size];
62                 break;
63             }
64         }
65
66         if (v_page_table->size == 0)
67             remove_page(seg_idx, proc->seg_table);
68     }
69 }
70
71 //update bp if this is the last block memory
72 if (address + cnt_pages * PAGE_SIZE == proc->bp)
73 {
74     addr_t largest_seg_idx = 0;
75     addr_t largest_page_idx = 0;
76     struct page_table_t * v_page_table = NULL;
77
78     for (int i = 0; i < proc->seg_table->size; i++)
79     {
80         if (proc->seg_table->table[i].v_index >= largest_seg_idx)
81         {
82             largest_seg_idx = proc->seg_table->table[i].v_index;
83             v_page_table = proc->seg_table->table[i].pages;
84         }
85     }
86
87     if (v_page_table)
88     {
89         for (int j = 0; j < v_page_table->size; j++)
90         {
91             if (v_page_table->table[j].v_index >= largest_page_idx)
92             {
93                 largest_page_idx = v_page_table->table[j].v_index;
94             }
95         }
96     }
97
98     proc->bp = ((largest_seg_idx << PAGE_LEN) + largest_page_idx + 1) << OFFSET_LEN;
99 }
100
101 //dump();
102
103 pthread_mutex_unlock(&mem_lock);
104
105 return 0;
106 }
```

- Kết quả khi chạy lệnh "make test_mem"

```
os18@ubuntu:~/Assignment_2/source_code$ make test_mem
----- 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: 03c00-03ffff - PID: 01 (idx 001, nxt: -01)
NOTE: Read file output/m0 to verify your result
----- MEMORY MANAGEMENT TEST 1 -----
./mem input/proc/m1
NOTE: Read file output/m1 to verify your result (your implementation should print nothing)
```

- 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
```

- Giải thích kết quả:

- Lệnh thứ 1: alloc 13535 sẽ cấp phát 14 trang _mem_stat (từ 000 -> 013) và lưu địa chỉ của byte đầu tiên vào thanh ghi số 0.
- Lệnh thứ 2: alloc 1568 sẽ cấp phát 2 trang (014, 015) và lưu địa chỉ của byte đầu tiên vào thanh ghi số 1.
- Lệnh thứ 3: free 0 sẽ giải phóng vùng nhớ được cấp phát từ lệnh alloc tại thanh ghi số 0, tức là lúc này chỉ có trang 014 và 015.
- Lệnh thứ 4: alloc 1386 sẽ cấp phát 2 trang (000, 001) và lưu địa chỉ của byte đầu tiên vào thanh ghi số 2.
- Lệnh thứ 5: alloc 4564 sẽ cấp phát 5 trang (002 -> 006) và lưu địa chỉ của byte được cấp phát đầu tiên vào thanh ghi số 4 -> Kết quả trạng thái cuối cùng của ram qua các quá trình cấp phát và giải phóng như ảnh trên chúng ta thấy.
- Lệnh thứ 6: write 102 1 20 sẽ viết giá trị 102 vào vị trí có địa chỉ bằng địa chỉ thanh ghi 1 cộng cho offset là 20 -> sẽ ra kết quả là 03814: 66 (102 đổi qua hệ hexa là 66).
- Tương tự cho lệnh 7: kết quả sẽ là 003e8: 15.

3 Put it all together

- Kết quả file input os_0

```
----- OS TEST 0 -----
./os os_0
Time slot 0
    Loaded a process at input/proc/p0, PID: 1
Time slot 1
    CPU 1: Dispatched process 1
Time slot 2
    Loaded a process at input/proc/p1, PID: 2
Time slot 3
    Loaded a process at input/proc/p1, PID: 3
    CPU 0: Dispatched process 2
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 0: 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
    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
```

Grantt Diagram



- Kết quả file input os_1

```
----- OS TEST 1 -----
./os_1
Time slot 0
Time slot 1
    Loaded a process at input/proc/p0, PID: 1
    CPU 2: Dispatched process 1
    Loaded a process at input/proc/s3, PID: 2
    CPU 0: Dispatched process 2
Time slot 2
    CPU 2: Put process 1 to run queue
    CPU 2: Dispatched process 1
Time slot 3
    Loaded a process at input/proc/m1, PID: 3
    CPU 3: Dispatched process 3
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot 4
    CPU 2: Put process 1 to run queue
    CPU 2: Dispatched process 1
Time slot 5
    Loaded a process at input/proc/s2, PID: 4
    CPU 3: Put process 3 to run queue
    CPU 3: Dispatched process 4
    CPU 1: Dispatched process 3
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot 6
    Loaded a process at input/proc/m0, PID: 5
    CPU 2: Put process 1 to run queue
    CPU 2: Dispatched process 5
Time slot 7
    CPU 3: Put process 4 to run queue
    CPU 3: Dispatched process 4
    CPU 1: Put process 3 to run queue
    CPU 1: Dispatched process 1
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot 8
    Loaded a process at input/proc/p1, PID: 6
    CPU 2: Put process 5 to run queue
    CPU 2: Dispatched process 3
Time slot 17
    CPU 3: Put process 8 to run queue
    CPU 3: Dispatched process 5
    CPU 1: Processed 4 has finished
    CPU 1: Dispatched process 8
Time slot 18
    CPU 3: Processed 5 has finished
    CPU 3 stopped
    CPU 2: Put process 6 to run queue
    CPU 2: Dispatched process 6
    CPU 0: Put process 7 to run queue
    CPU 0: Dispatched process 7
Time slot 19
    CPU 1: Put process 8 to run queue
    CPU 1: Dispatched process 8
Time slot 20
    CPU 2: Put process 6 to run queue
    CPU 2: Dispatched process 6
    CPU 0: Put process 7 to run queue
    CPU 0: Dispatched process 7
Time slot 21
    CPU 1: Put process 8 to run queue
    CPU 1: Dispatched process 8
Time slot 22
    CPU 2: Processed 6 has finished
    CPU 2 stopped
    CPU 1: Processed 8 has finished
    CPU 1 stopped
    CPU 0: Put process 7 to run queue
    CPU 0: Dispatched process 7
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
Time slot 28
```

```
Time slot 9
    CPU 3: Put process 4 to run queue
    CPU 3: Dispatched process 6
    CPU 1: Put process 1 to run queue
    CPU 1: Dispatched process 4
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 5
Time slot 10
    Loaded a process at input/proc/s0, PID: 7
    CPU 2: Put process 3 to run queue
    CPU 2: Dispatched process 7
Time slot 11
    CPU 3: Put process 6 to run queue
    CPU 3: Dispatched process 1
    CPU 1: Put process 4 to run queue
    CPU 1: Dispatched process 4
    CPU 0: Put process 5 to run queue
    CPU 0: Dispatched process 2
Time slot 12
    CPU 2: Put process 7 to run queue
    CPU 2: Dispatched process 3
Time slot 13
    CPU 3: Processed 1 has finished
    CPU 3: Dispatched process 6
    CPU 1: Put process 4 to run queue
    CPU 1: Dispatched process 4
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 7
Time slot 14
    CPU 2: Processed 3 has finished
    CPU 2: Dispatched process 5
Time slot 15
    Loaded a process at input/proc/s1, PID: 8
    CPU 3: Put process 6 to run queue
    CPU 3: Dispatched process 8
    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 2
Time slot 16
    CPU 2: Put process 5 to run queue
    CPU 2: Dispatched process 6
    CPU 0: Processed 2 has finished
    CPU 0: Dispatched process 7
Time slot 17
    CPU 3: Put process 8 to run queue
    CPU 3: Dispatched process 5
    CPU 1: Processed 4 has finished
    CPU 1: Dispatched process 8
Time slot 18
    CPU 3: Processed 5 has finished
    CPU 3 stopped
    CPU 2: Put process 6 to run queue
    CPU 2: Dispatched process 6
    CPU 0: Put process 7 to run queue
    CPU 0: Dispatched process 7
Time slot 19
    CPU 1: Put process 8 to run queue
    CPU 1: Dispatched process 8
Time slot 20
    CPU 2: Put process 6 to run queue
    CPU 2: Dispatched process 6
    CPU 0: Put process 7 to run queue
    CPU 0: Dispatched process 7
Time slot 21
    CPU 1: Put process 8 to run queue
    CPU 1: Dispatched process 8
Time slot 22
    CPU 2: Processed 6 has finished
    CPU 2 stopped
    CPU 1: Processed 8 has finished
    CPU 1 stopped
    CPU 0: Put process 7 to run queue
    CPU 0: Dispatched process 7
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
Time slot 28
```

MEMORY CONTENT:

000: 00000-003ff - PID: 05 (idx 000, nxt: 001)
 001: 00400-007ff - PID: 05 (idx 001, nxt: 002)
 002: 00800-00bff - PID: 05 (idx 002, nxt: 003)
 003: 00c00-00fff - PID: 05 (idx 003, nxt: 004)
 004: 01000-013ff - PID: 05 (idx 004, nxt: -01)
 007: 01c00-01fff - PID: 06 (idx 000, nxt: 008)
 008: 02000-023ff - PID: 06 (idx 001, nxt: 009)
 009: 02400-027ff - PID: 06 (idx 002, nxt: 010)
 025e0: 0a
 010: 02800-02bff - PID: 06 (idx 003, nxt: 011)
 011: 02c00-02fff - PID: 06 (idx 004, nxt: -01)
 012: 03000-033ff - PID: 05 (idx 000, nxt: 013)
 033e0: 15
 013: 03400-037ff - PID: 05 (idx 001, nxt: -01)
 016: 04000-043ff - PID: 06 (idx 000, nxt: 017)
 017: 04400-047ff - PID: 06 (idx 001, nxt: 018)
 018: 04800-04bff - PID: 06 (idx 002, nxt: 019)
 019: 04c00-04fff - PID: 06 (idx 003, nxt: -01)
 021: 05400-057ff - PID: 01 (idx 000, nxt: -01)
 05414: 64
 024: 06000-063ff - PID: 05 (idx 000, nxt: 025)
 06014: 66
 025: 06400-067ff - PID: 05 (idx 001, nxt: -01)

NOTE: Read file output/os_1 to verify your result
 root@192.168.1.11:~/Assignment_3/resource_code#

Grantt Diagram

