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

OPERATING SYSTEMS (CO2017)

Assignment 02

SIMPLE OPERATING SYSTEM

GVHD: Trần Việt Toản
SV thực hiện: Nguyễn Hồng Dân - 1910916
Trần Văn Thái - 1915121

Hồ Chí Minh city: May, 2021

Mục lục

1 Scheduling

1.1 Hiện thực code

- Hàm enqueue() trong file queue.c

```
 9
10  void enqueue(struct queue_t * q, struct pcb_t * proc)
11  {
12      /* TODO: put a new process to queue [q] */
13      if (q->size == 0)
14      {
15          q->proc[0] = proc;
16          ++(q->size);
17      }
18      else if (q->size < MAX_QUEUE_SIZE)
19      {
20          int ind = q->size;
21
22          while (ind > 0 && (q->proc[ind-1]->priority) < (proc->priority))
23          {
24              --ind;
25          }
26
27          for (int i = q->size; i > ind; --i)
28          {
29              q->proc[i] = q->proc[i-1];
30          }
31          q->proc[ind] = proc;
32          ++(q->size);
33      }
34  }
```

- Nếu hàng đợi rỗng thì thêm process vào và tăng kích thước hàng đợi.
- Nếu kích thước hàng đợi nhỏ hơn kích thước tối đa của hàng đợi thì xếp process trong hàng đợi sao cho process có độ ưu tiên cao nhất được đưa lên đầu. Như vậy process đầu hàng đợi có mức độ ưu tiên cao nhất.

- Hàm dequeue() trong file queue.

```
36  struct pcb_t * dequeue(struct queue_t * q)
37  {
38      /* TODO: return a pcb whose prioprity is the highest
39       * in the queue [q] and remember to remove it from q
40       * */
41      struct pcb_t * highest_proc;
42      if (q->size == 0)
43      {
44          return NULL;
45      }
46      else if (q->size == 1)
47      {
48          q->size = 0;
49          highest_proc = q->proc[0];
50          q->proc[0] = NULL;
51      }
52      else
53      {
54          highest_proc = q->proc[0];
55          for (int i = 0; i < ((q->size) - 1); ++i)
56          {
57              q->proc[i] = q->proc[i+1];
58          }
59          --(q->size);
60      }
61      return highest_proc;
62  }
```

- Kiểm tra hàng đợi, nếu hàng đợi rỗng ra trả về NULL
- Nếu hàng đợi chỉ có 1 process thì trả về process và hàng đợi trở thành hàng đợi rỗng.
- Trong các trường hợp, trả về giá trị process đầu (highest_proc).

- Hàm get_proc() trong file sched.c

```
22  struct pcb_t * get_proc(void)
23  {
24      struct pcb_t * proc = NULL;
25      /*TODO: get a process from [ready_queue]. If ready queue
26       * is empty, push all processes in [run_queue] back to
27       * [ready_queue] and return the highest priority one.
28       * Remember to use lock to protect the queue.
29       * */
30      pthread_mutex_lock(&queue_lock);
31
32      if (ready_queue.size == 0)
33      {
34          while (run_queue.size > 0)
35          {
36              enqueue(&ready_queue,dequeue(&run_queue));
37          }
38      }
39      proc = dequeue(&ready_queue);
40
41      pthread_mutex_unlock(&queue_lock);
42
43      return proc;
44  }
```

- Nếu ready_queue không có process thì di chuyển tất cả process từ run_queue sang ready_queue.
- Trả về proc ở đầu hàng đợi của ready_queue (cũng là process có độ ưu tiên cao nhất).

1.2 Test 0

1.2.1 Kết quả

```
------ 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
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
Time slot   7
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   8
Time slot   9
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  1
Time slot  10
Time slot  11
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot  12
Time slot  13
        CPU 0: Put process  2 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
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  1
Time slot  17
Time slot  18
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  19
Time slot  20
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  21
Time slot  22
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  23
        CPU 0: Processed  1 has finished
        CPU 0 stopped

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

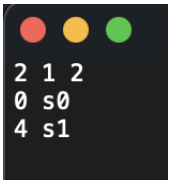
1.2.2 Giản đồ Gantt

Time slot	0	1	2	3	4	5	6	7	8	9	10	11
CPU 0		P1	P1	P1	P1	P1	P2	P2	P2	P1	P1	P2

Time slot	12	13	14	15	16	17	18	19	20	21	22	
CPU 0	P2	P1	P1	P2	P1	P1	P1	P1	P1	P1	P1	

1.2.3 Giải thích kết quả

- File sched_0 có nội dung:



- Tức là process sẽ chạy với time slice là 2, số CPU là 1 và chạy với 2 process. Process 1 sẽ chạy vào time 0 với nội dung trong file s0 và process 2 chạy vào time 4 với nội dung trong file s1.
- P1 là process trong file s0, có priority là 12, có 15 lệnh.
- P2 là process trong file s1, có priority là 20, có 7 lệnh.
- a -> b: từ đầu time slot thứ a tới trước lúc bắt đầu time slot thứ b.

  0 -> 1: P1 được load vào hàng đợi ready_queue.
  1 -> 3: CPU 0 chạy P1 trong ready_queue.
  3 -> 5: P1 chuyển sang hàng đợi run_queue. Lúc time slot 3s, trong hàng đợi ready_queue không có process nào, do đó P1 được qua hàng đợi ready_queue và tiếp tục được CPU 0 chạy.
  4 -> 5: P2 được load vào hàng đợi ready_queue.
  5 -> 7: P1 chuyển sang hàng đợi run_queue, CPU 0 chạy P2 trong ready_queue.
  7 -> 9: P2 chuyển sang hàng đợi run_queue, hàng đợi ready_queue không có process nào nên P1, P2 được chuyển qua hàng đợi ready_queue. Vì P2 có độ ưu tiên cao hơn P1, nên P2 sẽ chạy trước.
  9 -> 11: P2 chuyển sang hàng đợi run_queue, CPU 0 chạy P1 trong ready_queue.
  11 -> 13: P1 chuyển sang hàng đợi run_queue, hàng đợi ready_queue không có process nào nên P1, P2 được chuyển qua hàng đợi ready_queue. Vì P2 có độ ưu tiên cao hơn P1, nên P2 sẽ chạy trước.
  13 -> 15: P2 chuyển sang hàng đợi run_queue, CPU 0 chạy P1 trong ready_queue.
  15 -> 16: P1 chuyển sang hàng đợi run_queue, hàng đợi ready_queue không có process nào nên P1, P2 được chuyển qua hàng đợi ready_queue. Vì P2 có độ ưu tiên cao hơn P1, nên P2 sẽ chạy trước và kết thúc.

16 -> 18: CPU 0 chạy P1 trong ready_queue.
18 -> 20: P1 chuyển sang hàng đợi run_queue. Lúc time slot 18s, trong hàng đợi ready_queue không có process nào, do đó P1 được qua hàng đợi ready_queue và tiếp tục được CPU 0 chạy.
20 -> 22: P1 chuyển sang hàng đợi run_queue. Lúc time slot 20s, trong hàng đợi ready_queue không có process nào, do đó P1 được qua hàng đợi ready_queue và tiếp tục được CPU 0 chạy và kết thúc.

1.3 Test 1

1.3.1 Kết quả

```
------ SCHEDULING TEST 1 --------------------------------------
./os sched_1
Time slot   0
        Loaded a process at input/proc/s0, PID: 1
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
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot   9
Time slot  10
Time slot  11
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  12
Time slot  13
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  14
Time slot  15
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  1
Time slot  16
Time slot  17
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  4
Time slot  18
Time slot  19
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  20
Time slot  21
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  22
Time slot  23
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  1
Time slot  24
Time slot  25
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  4
Time slot  26
Time slot  27
        CPU 0: Put process  4 to run queue
```

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

MEMORY CONTENT:
NOTE: Read file output/sched_1 to verify your result
```

1.3.2 Giản đồ Gantt

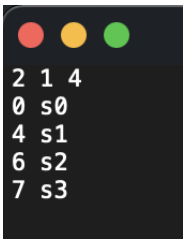
Time slot	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CPU 0		P1	P1	P1	P1	P2	P2	P3	P3	P4	P4	P2	P2	P3	P3	P1	P1

Time slot	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
CPU 0	P4	P4	P2	P2	P3	P3	P1	P1	P4	P4	P2	P3	P3	P1	P1	P4	P4

Time slot	34	35	36	37	38	39	40	41	42	43	44	45
CPU 0	P3	P3	P1	P1	P4	P4	P3	P3	P1	P1	P4	P1

1.3.3 Giải thích kết quả

- File sched_0 có nội dung:



- Tức là process sẽ chạy với time slice là 2, số CPU là 1 và chạy với 4 process. Process 1 sẽ chạy vào time 0 với nội dung trong file s0 và process 2 chạy vào time 4 với nội dung trong file s1, process 3 sẽ chạy vào time 6 với nội dung trong file s2, process 4 chạy vào time 7 với nội dung trong file s3.
- P1 là process trong file s0, có priority là 12, có 15 lệnh.
- P2 là process trong file s1, có priority là 20, có 7 lệnh.
- P3 là process trong file s2, có priority là 20, có 12 lệnh.
- P4 là process trong file s3, có priority là 7, có 11 lệnh.
- a -> b: từ đầu time slot thứ a tới trước lúc bắt đầu time slot thứ b.

  0 -> 1: P1 được load vào hàng đợi ready_queue.
  1 -> 3: P1 chạy.
  3 -> 5: P1 được chuyển sang hàng đợi run_queue, tại thời điểm 3 chưa có process nào trong hàng đợi ready_queue nên P1 được chuyển sang hàng đợi ready_queue và chạy. Thời điểm thứ 4, P2 được load vào hàng đợi ready_queue.
  5 -> 7: P2 chạy, P1 chuyển sang hàng đợi run_queue. Thời điểm 6 P3 được load vào hàng đợi ready_queue.
  7 -> 9: P3 chạy, P2 được chuyển sang hàng đợi run_queue. Thời điểm thứ 7 P4 được load

vào hàng đợi ready_queue.
9 -> 11: P4 chạy, P3 chuyển sang hàng đợi run_queue.
11 -> 13: P4 chuyển sang hàng đợi run_queue, ready_queue lúc này không có process nào nên 4 process được chuyển sang hàng đợi ready_queue. P2 và P3 có độ ưu tiên bằng nhau và lớn hơn 2 process còn lại, P2 chạy.
13 -> 15: P2 chuyển sang hàng đợi run_queue, P3 có độ ưu tiên lớn nhất trong hàng đợi ready_queue, P3 chạy.
15 -> 17: P3 chuyển sang hàng đợi run_queue, P1 có độ ưu tiên lớn nhất trong hàng đợi ready_queue, P1 chạy.
17 -> 19: P1 chuyển sang hàng đợi run_queue, hàng đợi ready_queue chỉ còn P4, P4 chạy.
19 -> 21: P4 chuyển sang hàng đợi run_queue, ready_queue lúc này không có process nào nên 4 process được chuyển sang hàng đợi ready_queue. P2 và P3 có độ ưu tiên bằng nhau và lớn hơn 2 process còn lại, P2 chạy.
21 -> 23: P2 chuyển sang hàng đợi run_queue, P3 có độ ưu tiên lớn nhất trong hàng đợi ready_queue, P3 chạy.
23 -> 25: P3 chuyển sang hàng đợi run_queue, P1 có độ ưu tiên lớn nhất trong hàng đợi ready_queue, P1 chạy.
25 -> 27: P1 chuyển sang hàng đợi run_queue, hàng đợi ready_queue chỉ còn P4, P4 chạy.
27 -> 28: P4 chuyển sang hàng đợi run_queue, ready_queue lúc này không có process nào nên 4 process được chuyển sang hàng đợi ready_queue. P2 và P3 có độ ưu tiên bằng nhau và lớn hơn 2 process còn lại, P2 chạy và kết thúc.
28 -> 30: P3 có độ ưu tiên lớn nhất trong hàng đợi ready_queue, P3 chạy.
30 -> 32: P3 chuyển sang hàng đợi run_queue, P1 có độ ưu tiên lớn nhất trong hàng đợi ready_queue, P1 chạy.
32 -> 34: P1 chuyển sang hàng đợi run_queue, hàng đợi ready_queue chỉ còn P4, P4 chạy.
34 -> 36: P4 chuyển sang hàng đợi run_queue, ready_queue lúc này không có process nào nên 3 process được chuyển sang hàng đợi ready_queue. P3 có độ ưu tiên bằng nhau và lớn hơn 2 process còn lại, P3 chạy.
36 -> 38: P3 chuyển sang hàng đợi run_queue, P1 có độ ưu tiên lớn nhất trong hàng đợi ready_queue, P1 chạy.
38 -> 40: P1 chuyển sang hàng đợi run_queue, hàng đợi ready_queue chỉ còn P4, P4 chạy.
40 -> 42: P4 chuyển sang hàng đợi run_queue, ready_queue lúc này không có process nào nên 3 process được chuyển sang hàng đợi ready_queue. P3 có độ ưu tiên bằng nhau và lớn hơn 2 process còn lại, P3 chạy và kết thúc.
42 -> 44: P1 có độ ưu tiên lớn nhất trong hàng đợi ready_queue, P1 chạy.
44 -> 45: P1 chuyển sang hàng đợi run_queue, hàng đợi ready_queue chỉ còn P4, P4 chạy và kết thúc.
45 -> 46: ready_queue lúc này không có process nào nên P1 được chuyển sang hàng đợi ready_queue, chạy và kết thúc.

1.4 Trả lời câu hỏi

- **Câu hỏi:** Ưu điểm của việc sử dụng giải thuật **priority feedback queue** khi so sánh với các giải thuật định thời khác đã học là gì?

- **Trả lời:**

+ *Tránh stavation:* Các process chỉ sử dụng CPU một khoảng thời gian nhất định (time slot). Sau đó, process này sẽ chuyển đến run_queue và các process có độ ưu tiên thấp hơn hoặc bằng sẽ được sử dụng CPU. Vì vậy, chúng ta tránh được vấn đề một số process sẽ không được chạy như ở các giải thuật: SJF (các process có thời gian sử dụng CPU dài có thể không được chạy), SRTF, Priority(các process có độ ưu tiên thấp có thể không được chạy).
+ *Các process được thực thi theo độ ưu tiên:* Các process quan trọng có thể được thực thi trước bằng cách gán độ ưu tiên của process đó cao hơn các process đang đợi xử lý. Giúp chúng ta kiểm soát được process nào sẽ được thực hiện trước, process nào được thực hiện sau. Một số giải thuật như: FCFS, SJF, SRTF, RR thì không giúp chúng ta hiện thực được điều này.

2 Memory

2.1 Hiện thực code

- Hàm get_page_table() trong file mem.c

  - Hàm này duyệt trong segment table để trả về page table tương ứng với index.

```
48  /* Search for page table table from the a segment table */
49  static struct page_table_t * get_page_table(
50          addr_t index,    // Segment level index
51          struct seg_table_t * seg_table) { // first level table
52
53      /*
54       * TODO: Given the Segment index [index], you must go through each
55       * row of the segment table [seg_table] and check if the v_index
56       * field of the row is equal to the index
57       *
58       * */
59
60      int i;
61      for (i = 0; i < seg_table->size; i++) {
62          // Enter your code here
63          if(seg_table->table[i].v_index == index)
64              return seg_table->table[i].pages;
65      }
66      return NULL;
67  }
68
```

- Hàm translate() trong file mem.c
  Hàm này dịch địa chỉ ảo(virtual address) thành địa chỉ vật lý(physical address):
  - Tách địa chỉ ảo thành 5 bits segment index, 5 bits page index, 10 bits offset.
  - Từ segment index suy ra page table bằng hàm get page table() ở trên.
  - Từ page table tìm page tương ứng với page index, page đó có chứa trường p_index.
  - Nối trường _index với offset ta được physical address.

  Hàm trả về 1 nếu thành công, ngược lại trả về 0.

```
69  /* Translate virtual address to physical address. If [virtual_addr] is valid,
70   * return 1 and write its physical counterpart to [physical_addr].
71   * Otherwise, return 0 */
72  static int translate(
73          addr_t virtual_addr,    // Given virtual address
74          addr_t * physical_addr, // Physical address to be returned
75          struct pcb_t * proc) {  // Process uses given virtual address
76
77      /* Offset of the virtual address */
78      addr_t offset = get_offset(virtual_addr);
79      /* The first layer index */
80      addr_t first_lv = get_first_lv(virtual_addr);
81      /* The second layer index */
82      addr_t second_lv = get_second_lv(virtual_addr);
83
84      /* Search in the first level */
85      struct page_table_t * page_table = NULL;
86      page_table = get_page_table(first_lv, proc->seg_table);
87      if (page_table == NULL) {
88          return 0;
89      }
90
91      int i;
92      for (i = 0; i < page_table->size; i++) {
93          if (page_table->table[i].v_index == second_lv) {
94              /* TODO: Concatenate the offset of the virtual addess
95               * to [p_index] field of page_table->table[i] to
96               * produce the correct physical address and save it to
97               * [*physical_addr]  */
98              addr_t p_idx = page_table->table[i].p_index;
99              // Concate p_idx with offset
100             *physical_addr = (p_idx << OFFSET_LEN) | offset;
101             return 1;
102         }
103     }
104     return 0;
105 }
```

- Hàm alloc_mem() trong file mem.c
  Hàm này thực hiện cấp phát bộ nhớ cho process. Các bước:
  - *Kiểm tra bộ nhớ có phù hợp không*
    Bộ nhớ phù hợp nếu thỏa cả 2 yếu tố sau:
    + Trong physical address: Kiểm tra số khung có đủ không.
    + Trong virtual address: Kiểm tra vùng nhớ sau khi thêm vào có vượt quá kích thước RAM không.
  - *Nếu bộ nhớ thỏa điều kiện, ta thực hiện:*
    + Từ size tính num_pages (số trang cần cấp phát).
    + Cấp phát các frame trong physical memory (cấu trúc _mem_stat).
    + Tách địa chỉ ảo thành 5 bits segment index, 5 bits page index, 10 bits offset.
    + Tìm page table tương ứng với segment_index.
    + Nếu page table khác NULL: Thêm page vào vị trí size, gán v_index cho page_index, gán p_index tương ứng với vị trí hiện tại trong _mem_stat, cập nhật size.
    + Nếu page table bằng NULL: Cấp phát page table mới, thêm page vào vị trí 0, gán v_index cho page_index, gán p_index tương ứng với vị trí hiện tại trong _mem_stat, cập nhật size lên 1.
    + Tiếp tục cấp phát đến khi đủ num_pages.

  Hàm trả về break pointer trước khi cấp phát (địa chỉ ảo). Trong toàn quá trình cấp phát, dùng lock để ngăn chặn truy cập bộ nhớ từ các process khác.

```
107  addr_t alloc_mem(uint32_t size, struct pcb_t * proc) {
108      pthread_mutex_lock(&mem_lock);
109      addr_t ret_mem = 0;
110      /* TODO: Allocate [size] byte in the memory for the
111       * process [proc] and save the address of the first
112       * byte in the allocated memory region to [ret_mem].
113       * */
114
115      uint32_t num_pages = (size % PAGE_SIZE) ? size / PAGE_SIZE :
116          size / PAGE_SIZE + 1; // Number of pages we will use
117
118      num_pages = (size % PAGE_SIZE) ? size / PAGE_SIZE + 1:
119          size / PAGE_SIZE; // Number of pages we will use
120      int mem_avail = 0; // We could allocate new memory region or not?
121
122      /* First we must check if the amount of free memory in
123       * virtual address space and physical address space is
124       * large enough to represent the amount of required
125       * memory. If so, set 1 to [mem_avail].
126       * Hint: check [proc] bit in each page of _mem_stat
127       * to know whether this page has been used by a process.
128       * For virtual memory space, check bp (break pointer).
129       * */
130
131      // Check free amount for required memory
132      int free_physical = 0;
133      int i;
134      for(i = 0; i < NUM_PAGES; i++){
135          if(_mem_stat[i].proc == 0){
136              free_physical++;
137              if(free_physical == num_pages){
138                  mem_avail = 1;
139                  break;
140              }
141          }
142      }
143      if (proc->bp + num_pages * PAGE_SIZE > RAM_SIZE)
144          mem_avail = 0;
145
```

```
146     if (mem_avail) {
147         /* We could allocate new memory region to the process */
148         // Virtual address is returned
149         ret_mem = proc->bp;
150         proc->bp += num_pages * PAGE_SIZE;
151         /* Update status of physical pages which will be allocated
152          * to [proc] in _mem_stat. Tasks to do:
153          *  - Update [proc], [index], and [next] field
154          *  - Add entries to segment table page tables of [proc]
155          *    to ensure accesses to allocated memory slot is
156          *    valid. */
157 
158         int pre_frame = 0;
159         int count_page = 0;
160         for(i = 0; i < NUM_PAGES; i++){
161             if(_mem_stat[i].proc == 0){ // Free
162                 _mem_stat[i].proc = proc->pid;
163                 _mem_stat[i].index = count_page;
164                 _mem_stat[i].next = -1; // Insert to the end?
165                 // First block
166                 if(count_page == 0);
167                 else
168                     _mem_stat[pre_frame].next = i;
169                 pre_frame = i;
170 
171                 // Get segment index and page table index of proc
172                 addr_t v_addr = ret_mem + count_page*PAGE_SIZE;
173                 addr_t seg_idx = get_first_lv(v_addr);
174                 addr_t page_idx = get_second_lv(v_addr);
175 
176                 struct page_table_t * page_table = get_page_table(seg_idx, proc->seg_table);
177                 if(page_table != NULL){
178                     int page_size = proc->seg_table->table[seg_idx].pages->size++;
179                     page_table->table[page_size].v_index = page_idx;
180                     page_table->table[page_size].p_index = i;
181                 }
182                 else {
183                     int segment_size = proc->seg_table->size++;
184                     proc->seg_table->table[segment_size].pages =
185                         (struct page_table_t *)malloc(sizeof(struct page_table_t));
186                     proc->seg_table->table[segment_size].pages->size = 1;
187                     proc->seg_table->table[segment_size].v_index = seg_idx;
188                     proc->seg_table->table[segment_size].pages->table[0].v_index = page_idx;
189                     proc->seg_table->table[segment_size].pages->table[0].p_index = i;
190 
191                 }
192                 if(++count_page == num_pages){
193                     break;
194                 }
195             }
196         }
197     }
198     pthread_mutex_unlock(&mem_lock);
199     printf("======== ALLOC size = %5d, pid = %d =========\n",size,proc->pid);
200     dump();
201     return ret_mem;
202 }
```

- Hàm free_mem() trong file mem.c
  Hàm thực hiện giải phóng bộ nhớ của một process. Các bước:
  - *Chuyển virtual address sang physical address (tiếp tục nếu thành công).*
  - *Trong physical memory: Lấy index của frame, gán trường proc của frame đó về 0 để ám chỉ frame đó đã trống. Lấy index kế bằng next.*
  - *Trong virtual address:*
    + Tách địa chỉ ảo thành 5 bits segment index, 5 bits page index, 10 bits offset.
    + Duyệt để tìm page table tương ứng với segment index, sau đó xóa page tương ứng với page index, cập nhật lại size.
    + Nếu sau khi xóa page table đó không còn page, free page table đó.
  - *Thực hiện xóa cho đến khi next = -1 (không còn frame nào của process đó mà chưa được đưa về trạng thái chưa sử dụng.*

  Trong toàn quá trình cấp phát, dùng lock để ngăn chặn truy cập bộ nhớ từ các process khác.

```
204  int free_mem(addr_t address, struct pcb_t * proc) {
205      /*TODO: Release memory region allocated by [proc]. The first byte of
206       * this region is indicated by [address]. Task to do:
207       *  - Set flag [proc] of physical page use by the memory block
208       *    back to zero to indicate that it is free.
209       *  - Remove unused entries in segment table and page tables of
210       *    the process [proc].
211       *  - Remember to use lock to protect the memory from other
212       *    processes.  */
213      pthread_mutex_lock(&mem_lock);
214      addr_t physical_addr;
215      if (translate(address, &physical_addr, proc) != 0){
216          int count_page = 0;
217          int next = physical_addr >> OFFSET_LEN;
218          while (next != -1)
219          {
220              _mem_stat[next].proc = 0;
221              next = _mem_stat[next].next;
222
223              // Free seg table va page table
224              addr_t v_addr = address + count_page*PAGE_SIZE;
225              addr_t seg_idx = get_first_lv(v_addr);
226              addr_t page_idx = get_second_lv(v_addr);
227
228              for (int i = 0; i < proc->seg_table->size; i++)
229              {
230                  if (proc->seg_table->table[i].v_index != seg_idx)
231                      continue;
232
233                  struct page_table_t * page_table = proc->seg_table->table[i].pages;
234                  for (int m = 0; m < page_table->size; m++)
235                  {
236                      if (page_table->table[m].v_index == page_idx)
237                      {
238                          // Merge and delete last element
239                          int k = 0;
```

```
228             for (int i = 0; i < proc->seg_table->size; i++)
229             {
230                 if (proc->seg_table->table[i].v_index != seg_idx)
231                     continue;
232 
233                 struct page_table_t * page_table int page_table_t::size e[i].pages;
234                 for (int m = 0; m < page_table->size; m++)
235                 {
236                     if (page_table->table[m].v_index == page_idx)
237                     {
238                         // Merge and delete last element
239                         int k = 0;
240                         for (k = m; k < page_table->size - 1; k++)
241                         {
242                             page_table->table[k].v_index = page_table->table[k + 1].v_index;
243                             page_table->table[k].p_index = page_table->table[k + 1].p_index;
244                         }
245                         page_table->table[k].v_index = 0;
246                         page_table->table[k].p_index = 0;
247                         page_table->size--;
248                         break;
249                     }
250                 }
251                 if (proc->seg_table->table[i].pages->size == 0)
252                 {
253                     // Free empty page table
254                     free(proc->seg_table->table[i].pages);
255                     int m = 0;
256                     for (m = i; m < proc->seg_table->size - 1; m++)
257                     {
258                         proc->seg_table->table[m].v_index = proc->seg_table->table[m + 1].v_index;
259                         proc->seg_table->table[m].pages = proc->seg_table->table[m + 1].pages;
260                     }
261                     proc->seg_table->table[m].v_index = 0;
262                     proc->seg_table->table[m].pages = NULL;
263                     proc->seg_table->size--;
264                 }
265                 break;
266             }
267             count_page++;
268         }
269     }
270     printf("=============== FREE pid = %d ================\n",proc->pid);
271     dump();
272     pthread_mutex_unlock(&mem_lock);
273 
274     return 0;
275 }
276 
```

2.2 Test 0

2.2.1 Kết quả

```
==================   RESULT   ==================
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
```

2.2.2 Trạng thái của RAM sau mỗi lời gọi alloc và free

Page	Địa chỉ	Lệnh (1)	Lệnh (2)	Lệnh (3)	Lệnh (4)	Lệnh (5)
0	00000-003ff	P0	P0	0	P2	P2
1	00400-007ff	P0	P0	0	P2	P2
2	00800-00bff	P0	P0	0	0	P4
3	00c00-00fff	P0	P0	0	0	P4
4	01000-013ff	P0	P0	0	0	P4
5	01400-017ff	P0	P0	0	0	P4
6	01800-01bff	P0	P0	0	0	P4
7	01c00-01fff	P0	P0	0	0	0
8	02000-023ff	P0	P0	0	0	0
9	02400-027ff	P0	P0	0	0	0
10	02800-02bff	P0	P0	0	0	0
11	02c00-02fff	P0	P0	0	0	0
12	03000-033ff	P0	P0	0	0	0
13	03400-037ff	P0	P0	0	0	0
14	03800-03bff	0	P1	P1	P1	P1
15	03c00-03fff	0	P1	P1	P1	P1

Bảng 1: *RAM's status - Test m0*

2.2.3 Giải thích kết quả

File m0 có nội dung:

```
1 7
(1) alloc 13535 0
(2) alloc 1568 1
(3) free 0
(4) alloc 1386 2
(5) alloc 4564 4
(6) write 102 1 20
(7) write 21 2 1000
```

+ 1 7 : Độ ưu tiên là 1, số lệnh là 7
+ *Lệnh 1* - alloc 13535 0: Cấp phát 14 trang _mem_stat (từ 000 -> 013) và lưu địa chỉ của byte đầu tiên vào thanh ghi 0.
+ *Lệnh 2* - alloc 1568 1: Cấp phát 02 trang _mem_stat (từ 014 -> 015) và lưu địa chỉ của byte đầu tiên vào thanh ghi 1.
+ *Lệnh 3* - free 0: Giải phóng vùng nhớ được cấp phát từ lệnh alloc ở thanh ghi 0 (xóa trang 001 -> 013).
+ *Lệnh 4* - alloc 1386 2: Cấp phát 02 trang _mem_stat (từ 000 -> 001) và lưu địa chỉ của byte đầu tiên vào thanh ghi 2.
+ *Lệnh 5* - alloc 4564 4: Cấp phát 05 trang _mem_stat (từ 002 -> 006) và lưu địa chỉ của byte đầu tiên vào thanh ghi 4.

+ *Lệnh 6* - write 102 1 20: Viết giá trị 102 vào vị trí có địa chỉ bằng địa chỉ thanh ghi 1 cộng cho offset (20). Kết quả sau lệnh này là 03814: 66 (102(DEC) tương ứng với 66(HEX)).

+ *Lệnh 7* - write 21 2 1000: Viết giá trị 21 vào vị trí có địa chỉ bằng địa chỉ thanh ghi 2 cộng cho offset (1000). Kết quả sau lệnh này là 003e8: 15 (15(DEC) tương ứng với 21(HEX)).

2.3 Test 1

2.3.1 Kết quả

```
================   RESULT   ================
NOTE: Read file output/m1 to verify your result (your implementation should print nothing)
```

2.3.2 Trạng thái của RAM sau mỗi lời gọi alloc và free

Page	Địa chỉ	L (1)	L (2)	L (3)	L (4)	L (5)	L (6)	L (7)	L (8)
0	00000-003ff	P0	P0	0	P2	P2	0	0	0
1	00400-007ff	P0	P0	0	P2	P2	0	0	0
2	00800-00bff	P0	P0	0	0	P4	P4	0	0
3	00c00-00fff	P0	P0	0	0	P4	P4	0	0
4	01000-013ff	P0	P0	0	0	P4	P4	0	0
5	01400-017ff	P0	P0	0	0	P4	P4	0	0
6	01800-01bff	P0	P0	0	0	P4	P4	0	0
7	01c00-01fff	P0	P0	0	0	0	0	0	0
8	02000-023ff	P0	P0	0	0	0	0	0	0
9	02400-027ff	P0	P0	0	0	0	0	0	0
10	02800-02bff	P0	P0	0	0	0	0	0	0
11	02c00-02fff	P0	P0	0	0	0	0	0	0
12	03000-033ff	P0	P0	0	0	0	0	0	0
13	03400-037ff	P0	P0	0	0	0	0	0	0
14	03800-03bff	0	P1	P1	P1	P1	P1	P1	0
15	03c00-03fff	0	P1	P1	P1	P1	P1	P1	0

Bảng 2: *RAM's tatus - Test m1*

2.3.3 Giải thích kết quả

File m1 có nội dung:

```
1 8
(1) alloc 13535 0
(2) alloc 1568 1
(3) free 0
(4) alloc 1386 2
(5) alloc 4564 4
(6) free 2
(7) free 4
(8) free 1
```

+ 1 8 : Độ ưu tiên là 1, số lệnh là 8
+ *Lệnh 1* - alloc 13535 0: Cấp phát 14 trang _mem_stat (từ 000 -> 013) và lưu địa chỉ của byte đầu tiên vào thanh ghi 0.
+ *Lệnh 2* - alloc 1568 1: Cấp phát 02 trang _mem_stat (từ 014 -> 015) và lưu địa chỉ của byte đầu tiên vào thanh ghi 1.
+ *Lệnh 3* - free 0: Giải phóng vùng nhớ được cấp phát từ lệnh alloc ở thanh ghi 0 (xóa trang 001 -> 013).
+ *Lệnh 4* - alloc 1386 2: Cấp phát 02 trang _mem_stat (từ 000 -> 001) và lưu địa chỉ của byte đầu tiên vào thanh ghi 2.
+ *Lệnh 5* - alloc 4564 4: Cấp phát 05 trang _mem_stat (từ 002 -> 006) và lưu địa chỉ của byte đầu tiên vào thanh ghi 4.
+ *Lệnh 6* - free 2: Giải phóng vùng nhớ được cấp phát từ lệnh alloc ở thanh ghi 2 (xóa trang 000 -> 001).
+ *Lệnh 7* - free 4: Giải phóng vùng nhớ được cấp phát từ lệnh alloc ở thanh ghi 4 (xóa trang 002 -> 006).
+ *Lệnh 8* - free 1: Giải phóng vùng nhớ được cấp phát từ lệnh alloc ở thanh ghi 1 (xóa trang 014 -> 015). Sau lệnh này tất cả bộ nhớ đã được giải phóng.

2.4 Trả lời câu hỏi

- **Câu hỏi:** Ưu điểm và nhược điểm của **segmentaton with paging** là gì?
- **Trả lời:**
  + *Ưu điểm:*
    * Đơn giản việc cấp phát, giúp tiết kiệm bộ nhớ.
    * Kích thước bảng trang được giới hạn bởi kích thước phân đoạn.
    * Hạn chế tối đa hiện tượng phân mảnh ngoại.
    * Chia sẻ được từng trang riêng biệt như Paging.
    * Chia sẻ được toàn bộ segment bằng việc chia sẻ entry trong bảng phân đoạn của segment đó, hoặc chia sẻ cả bản phân trang của nó.
    * Hoán đổi giữa các trang có kích thước bằng nhau và các khung trang (page frame) dễ dàng.
  + *Nhược điểm:*
    * Có thể bị phân mảnh nội.
    * Mức độ phức tạp sẽ cao hơn so với phân trang.
    * Bảng trang cần được lưu trữ liên tục trong bộ nhớ.
    * Chương trình tự chia không gian ảo của mình ra nhiều partition khác nhau để chứa những thông tin độc lập cần được xử lý, trong quá trình chạy, nếu một trong các partition không đủ chỗ chứa thông tin thì chương trình sẽ bị dừng đột ngột.

3 Overall

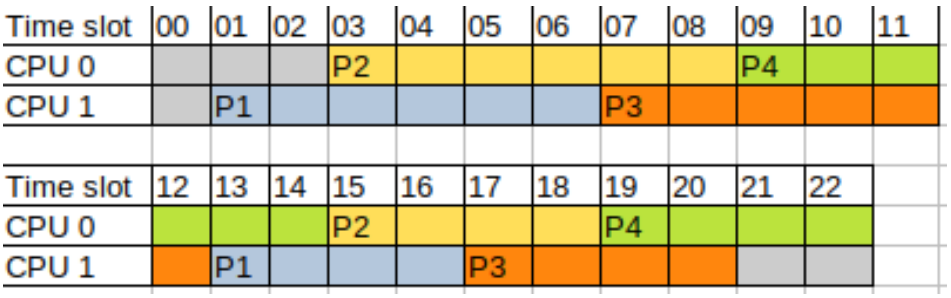
3.1 Test 0

3.1.1 Kết quả

```
----- 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
        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
```

```
MEMORY CONTENT:
000: 00000-003ff - PID: 05 (idx 000, nxt: 001)
003e8: 15
001: 00400-007ff - PID: 05 (idx 001, nxt: -01)
002: 00800-00bff - PID: 05 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 05 (idx 001, nxt: 004)
004: 01000-013ff - PID: 05 (idx 002, nxt: 005)
005: 01400-017ff - PID: 05 (idx 003, nxt: 006)
006: 01800-01bff - PID: 05 (idx 004, nxt: -01)
011: 02c00-02fff - PID: 06 (idx 000, nxt: 012)
012: 03000-033ff - PID: 06 (idx 001, nxt: 013)
013: 03400-037ff - PID: 06 (idx 002, nxt: 014)
014: 03800-03bff - 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)
031: 07c00-07fff - PID: 06 (idx 000, nxt: 032)
032: 08000-083ff - PID: 06 (idx 001, nxt: 033)
033: 08400-087ff - PID: 06 (idx 002, nxt: 034)
085e7: 0a
034: 08800-08bff - PID: 06 (idx 003, nxt: 035)
035: 08c00-08fff - PID: 06 (idx 004, nxt: -01)
NOTE: Read file output/os_1 to verify your result
```

3.1.2 Giản đồ Gantt



3.2 Test 1

3.2.1 Kết quả

```
----- OS TEST 1 -----------------------------------------------------
./os os_1
Time slot 0
Time slot 1
        Loaded a process at input/proc/p0, PID: 1
Time slot 2
        Loaded a process at input/proc/s3, PID: 2
        CPU 3: Dispatched process 1
Time slot 3
        CPU 2: Dispatched process 2
Time slot 4
        CPU 3: Put process 1 to run queue
        CPU 3: Dispatched process 1
        Loaded a process at input/proc/m1, PID: 3
Time slot 5
        CPU 2: Put process 2 to run queue
        CPU 2: Dispatched process 3
        CPU 1: Dispatched process 2
Time slot 6
        Loaded a process at input/proc/s2, PID: 4
        CPU 3: Put process 1 to run queue
        CPU 3: Dispatched process 4
Time slot 7
        CPU 2: Put process 3 to run queue
        CPU 2: Dispatched process 1
        CPU 1: Put process 2 to run queue
        CPU 1: Dispatched process 3
        CPU 0: Dispatched process 2
        Loaded a process at input/proc/m0, PID: 5
Time slot 8
        CPU 3: Put process 4 to run queue
        CPU 3: Dispatched process 5
Time slot 9
        Loaded a process at input/proc/p1, PID: 6
        CPU 1: Put process 3 to run queue
        CPU 1: Dispatched process 6
        CPU 0: Put process 2 to run queue
        CPU 2: Put process 1 to run queue
        CPU 2: Dispatched process 4
        CPU 0: Dispatched process 3
Time slot 10
        CPU 3: Put process 5 to run queue
        CPU 3: Dispatched process 1
Time slot 11
        Loaded a process at input/proc/s0, PID: 7
        CPU 1: Put process 6 to run queue
        CPU 1: Dispatched process 7
        CPU 0: Put process 3 to run queue
        CPU 2: Put process 4 to run queue
        CPU 2: Dispatched process 4
        CPU 0: Dispatched process 2
```

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

```
Time slot 22
        CPU 3: Put process 6 to run queue
        CPU 3: Dispatched process 6
        CPU 2: Put process 7 to run queue
        CPU 2: Dispatched process 7
Time slot 23
        CPU 1: Put process 8 to run queue
        CPU 1: Dispatched process 8
Time slot 24
        CPU 3: Processed 6 has finished
        CPU 3 stopped
        CPU 2: Put process 7 to run queue
        CPU 2: Dispatched process 7
        CPU 1: Processed 8 has finished
        CPU 1 stopped
Time slot 25
Time slot 26
        CPU 2: Put process 7 to run queue
        CPU 2: Dispatched process 7
Time slot 27
Time slot 28
        CPU 2: Put process 7 to run queue
        CPU 2: Dispatched process 7
Time slot 29
        CPU 2: Processed 7 has finished
        CPU 2 stopped
```

3.2.2 Giản đồ Gantt

Time slot	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14
CPU 0								P2		P3		P2		P7	
CPU 1						P2		P3		P6		P7		P4	
CPU 2				P2		P3		P1		P4		P4		P6	
CPU 3			P1				P4		P5		P1		P5		

Time slot	15	16	17	18	19	20	21	22	23	24	25	26	27	28
CPU 0	P2		P4		P5									
CPU 1			P6		P8		P8		P8					
CPU 2	P1		P5		P2	P7		P7		P7		P7		P7
CPU 3		P8		P7		P6		P6						

4 Reference

Tài liệu

[1] Abraham Silberschatz, Peter Baer Galvin and Greg Gagne. *Operating System Concepts*.

[2] Two Level Paging and Multi Level Paging in OS, https://www.geeksforgeeks.org/two-level-paging-and-multi-level-paging-in-os/