TRƯỜNG ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH

KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

HỆ ĐIỀU HÀNH

BÁO CÁO BÀI TẬP LỚN 2

SIMPLE

OPERATING SYSTEM

Giảng viên hướng dẫn: Trần Việt Toản

Lê Nguyên Khang	1913695
Mai Thịnh Phát	1914590
Võ Minh Toang	1915570
Quang Chấn Vĩ	1915961

MỤC LỤC

MỞ ĐẦU

Mục đích: Mục tiêu của bài tập lớn này là mô phỏng các thành phần chính trong một hệ điều hành đơn giản, ví dụ: scheduler, đồng bộ hóa, các hoạt động liên quan đến bộ nhớ vật lý và bộ nhớ ảo.

Nội dung: Sinh viên sẽ được thực hành với 3 phần chính: scheduler, đồng bộ hóa và cơ chế cáp phát bộ nhớ ảo sang bộ nhớ vật lý.

Kết quả: Sau bài tập lớn này, sinh viên có thể hiểu được phần nào nguyên lý hoạt động của một hệ điều hành đơn giản; rút ra vai trò và ý nghĩa của các module chính trong hệ điều hành cũng như cách thức hoạt động của nó.

NỘI DUNG

1. SCHEDULER

1.1. Question

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

Trả lời:

Giải thuật priority feedback queue sử dụng 2 hàng đợi ready_queue và run_queue.

- **ready_queue**: hàng đợi chứa các process theo mức độ ưu tiên, thực thi trước so với hàng đợi run_queue. Khi CPU chuyển sang slot tiếp theo, nó sẽ tìm kiếm trong ready_queue.
- **run_queue:** hàng đợi này chứa các process đang chờ để tiếp tục thực thi sau khi hết slot mà chưa hoàn tất các quá trình của mình. Các process ở hàng đợi này chỉ được tiếp tục ở slot tiếp theo khi *ready_queue* rỗi và được đưa sang hàng đợi *ready_queue* để xét slot tiếp theo.

Cả 2 hàng đợi đều là hàng đợi có độ ưu tiên, dựa trên mức độ ưu tiên của process trong hàng đợi.

Ưu điểm của giải thuật priority feedback queue:

- Sử dụng time slot, tạo sự công bằng về thời gian thực thi giữa các process, tránh tình trạng chiếm CPU sử dụng, trì hoãn vô hạn định.
- Sử dụng 2 hàng đợi và độ ưu tiên priority nên sẽ linh hoạt trong phân chia thực hiện công việc.
- Những process ngắn sẽ nhanh chóng được hoàn hành và nhường thời gian thực thi cho các process khác.

1.2. Implementation

1.2.1. Queue

Đầu tiên, ta hiện thực giải thuật *enqueue* và *dequeue* trong file *queue.c*.

- *enqueue*: Đưa một PCB mới vào hàng đợi.

```
 9 void enqueue(struct queue_t * q, struct pcb_t * proc) {
10     /* TODO: put a new process to queue [q] */
11     //============ this is my code ==============
12     int sizeOFqueue = q->size;
13     if(sizeOFqueue<MAX_QUEUE_SIZE && sizeOFqueue>=0)
14     {
15         q->proc[sizeOFqueue]=proc;
16         q->size = sizeOFqueue+1;
17     }
18     //==========================================
19 }
20
```

- *dequeue*: Lấy PCB có độ ưu tiên cao nhất ra khỏi hàng đợi.

```
21 struct pcb_t * dequeue(struct queue_t * q) {
22     int sizeOfQueue = q->size;
23     if( sizeOfQueue ==1 )
24     {
25         struct pcb_t* return_proc = q->proc[0];
26         q->proc[0]=NULL;
27         q->size =0;
28         return return_proc;
29     }
30     else if( sizeOfQueue >1 && sizeOfQueue <= MAX_QUEUE_SIZE)
31     {
32         struct pcb_t* return_proc= q->proc[0];// process will be returned
33         uint32_t highest_prior = q->proc[0]->priority;
34         int position =0;
35         int i;
36         for(i=0 ; i < sizeOfQueue ;i++)
37         {
38             if( highest_prior < q->proc[i]->priority )
39             {
40                 highest_prior =q->proc[i]->priority;
41                 position = i;
42             }
43         }
44         return_proc = q->proc[position];
45         if(position == (sizeOfQueue -1 ))   // highest at the end of queue
46         {
47             q->proc[position] =NULL;
48             q->size = sizeOfQueue -1;
49             return return_proc;
50         }
51         else
52         {
53             for(i = position+1;i< sizeOfQueue; i++)
54             {
55                 q->proc[i-1]=q->proc[i];
56             }
57             q->proc[sizeOfQueue -1]=NULL;
58             q->size = sizeOfQueue -1;
59             return return_proc;
60 
61         }
62     }
63     return NULL;
64 }
```

1.2.2. *Scheduler*

Ta hiện thực function *get_proc( )* trong file *sched.c* để lấy PCB của một process trong ready_queue. Nếu hàng đợi đang trống tại thời điểm function được gọi, chúng ta sẽ di chuyển tất cả các PCB của process đang đợi trong run_queue trở về ready_queue trước khi lấy process từ ready_queue.

```
19  struct pcb_t * get_proc(void) {
20      struct pcb_t * proc = NULL;
21      pthread_mutex_lock(&queue_lock);
22
23      if(ready_queue.size==0 && run_queue.size >0)
24      {
25          ready_queue.size=run_queue.size;
26          for(int i=0;i<ready_queue.size ;i++)
27          {
28              ready_queue.proc[i]=run_queue.proc[i];
29              run_queue.proc[i]=NULL;
30          }
31          run_queue.size=0;
32          proc=dequeue(&ready_queue);
33      }
34      else if(ready_queue.size >0)
35      {
36          proc=dequeue(&ready_queue);
37      }
38      pthread_mutex_unlock(&queue_lock);
39      return proc;
40  }
```

1.3. Result

Compile đoạn code sử dụng Makefile:

```
make sched
```

Sau đó chạy thử nghiệm trên cấu hình mẫu:

```
make test_sched
```

Ta thu được kết quả như bên dưới

- sched_0

```
        Loaded a process at input/proc/s0, PID: 1
Time slot   0
Time slot   1
        CPU 0: Dispatched process  1
Time slot   2
Time slot   3
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot   4
        Loaded a process at input/proc/s1, PID: 2
Time slot   5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   6
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
```

process		p1				p2				p1		p2		p2		s1	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

- sched_1

```
        Loaded a process at input/proc/s0, PID: 1
Time slot   0
        CPU 0: Dispatched process  1
Time slot   1
Time slot   2
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot   3
        Loaded a process at input/proc/s1, PID: 2
Time slot   4
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   5
        Loaded a process at input/proc/s2, PID: 3
Time slot   6
        Loaded a process at input/proc/s3, PID: 4
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot   7
Time slot   8
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot   9
Time slot  10
        CPU 0: Put process  4 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  3
Time slot  13
Time slot  14
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  1
Time slot  15
Time slot  16
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  4
Time slot  17
Time slot  18
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  19
Time slot  20
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  21
Time slot  22
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  1
Time slot  23
```

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

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

process	p1				p2		p3		p4		p2		p3		p1		p4		p2		p3		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

process	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	

2. MEMORY MANAGEMENT

2.1. Question

Câu hỏi: What is the advantage and disadvantage of segmentation with paging?

Trả lời:

Ưu điểm của giải thuật:

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

Nhược điểm của giải thuật:

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

2.2. Implementation

2.2.1. Ánh xạ từ địa chỉ ảo sang địa chỉ vật lý

Chúng ta sử dụng 20 bit để biểu diễn địa chỉ, trong đó 5 bit đầu tiên biểu diễn segment index, 5 bit tiếp theo biểu diễn page index và 10 bit cuối biểu diễn offset.

Kế tiếp, chúng ta hiện thực quá trình chuyển đổi từ địa chỉ ảo của một process sang địa chỉ vật lý bằng cách hoàn thành 2 function là *get_page_table( )* và *translate( )* trong file *mem.c*.

get_page_table( ): Tìm bảng phân trang có chỉ mục phân đoạn cho một process.

```
45  static struct page_table_t *get_page_table(addr_t index, struct seg_table_t *seg_table)
46  {
47      if (seg_table == NULL)
48          return NULL;
49      for (int i = 0; i < seg_table->size; i++)
50      {
51          if (seg_table->table[i].v_index == index)
52          {
53              return seg_table->table[i].pages;
54          }
55      }
56      return NULL;
57  }
```

translate( ): sử dụng *get_page_table( )* để chuyển đổi từ địa chỉ ảo sang địa chỉ vật lý.

```c
static int translate(addr_t virtual_addr, addr_t *physical_addr, struct pcb_t *proc)
{
    /* Offset of the virtual address */
    addr_t offset = get_offset(virtual_addr);
    /* The first layer index, find segment virtual */
    addr_t first_lv = get_first_lv(virtual_addr);
    /* The second layer index, find page virtual */
    addr_t second_lv = get_second_lv(virtual_addr);
    /* Search in the first level */
    struct page_table_t *page_table = get_page_table(first_lv, proc->seg_table);
    if (page_table == NULL)
        return 0;
    for (int i = 0; i < page_table->size; i++)
    {
        if (page_table->table[i].v_index == second_lv)
        {
            *physical_addr = (page_table->table[i].p_index << OFFSET_LEN) | (offset);
            return 1;
        }
    }
    return 0;
}
```

2.2.2. Cấp phát và thu hồi bộ nhớ

Cấp phát bộ nhớ:

- *allocate*: Hàm hỗ trợ cho hàm *alloc_mem*.
  - Input:
  - ret_mem: Địa chỉ của byte đầu tiên trong vùng nhớ sẽ được cấp phát.
  - num_pages: Số lượng trang cấp phát cho process trong không gian địa chỉ ảo.
  - proc: Process cần được cấp phát thêm vùng nhớ.
  - Mục đích: Cập nhật trạng thái của các frame physical sẽ được khởi tạo trong cấu trúc _mem_stat, thêm entries đến segment table, page table.

```
82  void allocate(int ret_mem, int num_pages, struct pcb_t *proc)
83  {
84      int count_pages = 0;
85      int last_index = -1; // use for update [next] in mem_stat
86      for (int i = 0; i < NUM_PAGES; i++)
87      {
88          if (_mem_stat[i].proc != 0)
89              continue;                    // page in used
90          _mem_stat[i].proc = proc->pid;    // page is used by proc
91          _mem_stat[i].index = count_pages; // index in list of allocated pages
92          if (last_index > -1)
93              _mem_stat[last_index].next = i; // update next
94          last_index = i;                     // update last page
95          addr_t virtual_addr = ret_mem + count_pages * PAGE_SIZE; // virtual address of this page
96          addr_t seg = get_first_lv(virtual_addr);
97          struct page_table_t *page_table = get_page_table(seg, proc->seg_table);
98          if (page_table == NULL)
99          {
100             proc->seg_table->table[proc->seg_table->size].v_index = seg;
101             proc->seg_table->table[proc->seg_table->size].pages
102             = (struct page_table_t *)malloc(sizeof(struct page_table_t));
103             page_table = proc->seg_table->table[proc->seg_table->size].pages;
104             proc->seg_table->size++;
105         }
106         page_table->table[page_table->size].v_index = get_second_lv(virtual_addr);
107         page_table->table[page_table->size].p_index = i; // format of i is 10 bit segment and page in address
108         page_table->size++;
109         count_pages++;
110         if (count_pages == num_pages)
111         {
112             _mem_stat[i].next = -1; // last page in list
113             break;
114         }
115     }
```

- *alloc_mem*: Cấp phát bộ nhớ cho process có kích thước size và lưu địa chỉ của byte đầu tiên trong vùng nhớ đã được cấp phát.

```
118 addr_t alloc_mem(uint32_t size, struct pcb_t *proc)
119 {
120     pthread_mutex_lock(&mem_lock);
121     addr_t ret_mem = 0;
122     // Number of pages we will use for this process
123     uint32_t num_pages = size / PAGE_SIZE;
124     if (size % PAGE_SIZE)
125         num_pages++;
126     // memory available? We could allocate new memory region or not?
127     int mem_avail = 0;
128     int free_pages = 0;
129     for (int i = 0; i < NUM_PAGES; i++)
130     {
131         if (_mem_stat[i].proc == 0)
132             free_pages++;
133     }
134     if (free_pages >= num_pages && proc->bp + num_pages * PAGE_SIZE < RAM_SIZE)
135         mem_avail = 1;
136
137     if (mem_avail)
138     {
139         ret_mem = proc->bp;
140         proc->bp += num_pages * PAGE_SIZE;
141         allocate(ret_mem, num_pages, proc);
142     }
143     pthread_mutex_unlock(&mem_lock);
144     return ret_mem;
145 }
```

Thu hồi bộ nhớ:

- *free_mem*: Giải phóng vùng bộ nhớ đã được cấp phát.

```
156  int free_mem(addr_t address, struct pcb_t *proc)
157  {
158      pthread_mutex_lock(&mem_lock);
159      addr_t virtual_addr = address; // virtual address to free
160      addr_t physical_addr = 0;      // physical address to free
161      // Have physical page in memory ?
162      if (translate(virtual_addr, &physical_addr, proc) == 0)
163      {
164          pthread_mutex_unlock(&mem_lock);
165          return 1;
166      }
167      // Clear physical page in memory
168      int num_pages = 0; // number of pages
169      int i = 0;
170      for (i = physical_addr >> OFFSET_LEN; i != -1; i = _mem_stat[i].next)
171      {
172          num_pages++;
173          _mem_stat[i].proc = 0; // clear physical memory
174      }
175
176      // Clear virtual page in process
177      for (i = 0; i < num_pages; i++)
178      {
179          addr_t seg = get_first_lv(virtual_addr + i * PAGE_SIZE);
180          addr_t page = get_second_lv(virtual_addr + i * PAGE_SIZE);
181          // virtual_addr + i * PAGE_SIZE = address of this page
182          struct page_table_t *page_table = get_page_table(seg, proc->seg_table);
183          if (page_table == NULL)
184              continue;
```

```
185         //remove page
186         for (int j = 0; j < page_table->size; j++)
187         {
188             if (page_table->table[j].v_index == page)
189             {
190                 page_table->size--;
191                 page_table->table[j] = page_table->table[page_table->size];
192                 break;
193             }
194         }
195         //remove page table
196         if (page_table->size == 0 && proc->seg_table != NULL)
197         {
198             for (int a = 0; a < proc->seg_table->size; a++)
199             {
200                 if (proc->seg_table->table[a].v_index == seg)
201                 {
202                     proc->seg_table->size--;
203                     proc->seg_table->table[a] = proc->seg_table->table[proc->seg_table->size];
204                     proc->seg_table->table[proc->seg_table->size].v_index = 0;
205                     free(proc->seg_table->table[proc->seg_table->size].pages);
206                 }
207             }
208         }
209     }
210     proc->bp -= num_pages * PAGE_SIZE;
211     pthread_mutex_unlock(&mem_lock);
212     return 0;
213 }
```

2.3. Result

Compile đoạn code sử dụng Makefile:

```
make mem
```

Sau đó chạy thử nghiệm trên cấu hình mẫu:

```
make test_mem
```

Dưới đây là quá trình ghi lại trạng thái của RAM trong chương trình.

```
------ MEMORY MANAGEMENT TEST 0 -----------------
./mem input/proc/m0
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
        003e8: 15
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
        03814: 66
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
NOTE: Read file output/m0 to verify your result
------ MEMORY MANAGEMENT TEST 1 -----------------
./mem input/proc/m1
NOTE: Read file output/m1 to verify your result
```

Giải thích:

- Với input của test 0 như sau:

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

  - Đầu tiên lệnh “alloc 13535 0” sẽ cấp phát 14 từ 000 đến 013 và lưu địa chỉ của byte được cấp phát đầu tiên vào thanh ghi số 0.
  - Lệnh “alloc 1568 1” sẽ cấp phát 2 trang 014, 015 và lưu địa chỉ của byte được cấp phát đầu tiên vào thanh ghi số 0.
  - Lệnh “free 0” sẽ giải phóng vùng nhớ được cấp phát từ lệnh alloc tại thanh ghi số 0. Lúc này chỉ còn lại trang 014 và 015.
  - Lệnh “alloc 1386 2” sẽ cấp phát 2 trang 000 và 001 và lưu địa chỉ của byte được cấp phát vào thanh ghi số 2.
  - Lệnh “alloc 4564 4” sẽ cấp phát 5 trang từ 002 đến 006 và lưu địa chỉ của byte được cấp phát đầu tiên vào thanh ghi số 4.

- Lệnh "write 102 1 20" sẽ viết giá trị dec 102 (hex 66) vào vị trí có địa chỉ là địa chỉ của thanh ghi 1 + offset 20 (hex 14), tức là 03800 + 14 = 03814.
- Lệnh "write 21 2 1000" sẽ viết giá trị dec 21 (hex 15) vào vị trí có địa chỉ là địa chỉ của thanh ghi 2 + offset 1000 (hex 3e8), tức là 00000 + 3e8 = 003e8.

- Với input của test 1:

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

- Với mỗi lệnh alloc có một lệnh free tương ứng.

3. PUT IT ALL TOGETHER

Cuối cùng, chúng ta kết hợp scheduler và Virtual Memory Engine để tạo thành một hệ điều hành hoàn chỉnh.

Compile toàn bộ code sử dụng Makefile:

```
make all
```

Sau đó kiểm tra kết quả:

```
make test_all
```

Đây là kết quả thu được:

- File os_0:

```
./os os_0
        Loaded a process at input/proc/p0, PID: 1
Time slot   0
Time slot   1
        CPU 1: Dispatched process  1
Time slot   2
        Loaded a process at input/proc/p1, PID: 2
Time slot   3
        CPU 0: Dispatched process  2
        Loaded a process at input/proc/p1, PID: 3
Time slot   4
        Loaded a process at input/proc/p1, PID: 4
Time slot   5
Time slot   6
Time slot   7
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  3
Time slot   8
Time slot   9
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  4
Time slot  10
Time slot  11
Time slot  12
Time slot  13
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  1
Time slot  14
Time slot  15
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  16
Time slot  17
        CPU 1: Processed  1 has finished
        CPU 1: Dispatched process  3
Time slot  18
Time slot  19
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  4
Time slot  20
Time slot  21
        CPU 1: Processed  3 has finished
        CPU 1 stopped
Time slot  22
Time slot  23
        CPU 0: Processed  4 has finished
        CPU 0 stopped
```

CPU 0

process				p2						p4						p2				p4				
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

CPU 1

process		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	23

```
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)
        01414: 64
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)
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)
```

- File os_1:

```
./os os_1
Time slot   0
        Loaded a process at input/proc/p0, PID: 1
        CPU 3: Dispatched process  1
Time slot   1
        Loaded a process at input/proc/s3, PID: 2
Time slot   2
        CPU 2: Dispatched process  2
        CPU 3: Put process  1 to run queue
        CPU 3: Dispatched process  1
Time slot   3
        Loaded a process at input/proc/m1, PID: 3
Time slot   4
        CPU 2: Put process  2 to run queue
        CPU 2: Dispatched process  2
        CPU 1: Dispatched process  3
        CPU 3: Put process  1 to run queue
        CPU 3: Dispatched process  1
Time slot   5
        Loaded a process at input/proc/s2, PID: 4
```

```
Time slot   6
        CPU 2: Put process  2 to run queue
        CPU 2: Dispatched process  2
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
        CPU 0: Dispatched process  4
        Loaded a process at input/proc/m0, PID: 5
        CPU 3: Put process  1 to run queue
        CPU 3: Dispatched process  5
Time slot   7
Time slot   8
        CPU 2: Put process  2 to run queue
        CPU 2: Dispatched process  2
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  1
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
        Loaded a process at input/proc/p1, PID: 6
        CPU 3: Put process  5 to run queue
        CPU 3: Dispatched process  3
Time slot   9
```

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

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

```
Time slot  18
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  19
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
        CPU 3: Put process  6 to run queue
        CPU 3: Dispatched process  6
Time slot  20
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  21
        CPU 0: Processed  4 has finished
        CPU 0 stopped
        CPU 3: Processed  6 has finished
        CPU 3 stopped
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
```

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

CPU 0																											
process							p4				p5		p2	p6				p4									
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	24	25	26	27

CPU 1																											
process					p3			p1		p2		p4		p5													
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	24	25	26	27

CPU 2																											
process											p6		p1		p7												
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	24	25	26	27

CPU 3																											
process							p5		p3		p7		p3		p4		p5										
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	24	25	26	27

```
MEMORY CONTENT:
000: 00000-003ff - PID: 06 (idx 000, nxt: 001)
001: 00400-007ff - PID: 06 (idx 001, nxt: 007)
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)
        01414: 64
006: 01800-01bff - PID: 05 (idx 004, nxt: -01)
007: 01c00-01fff - PID: 06 (idx 002, nxt: 008)
        01de7: 0a
008: 02000-023ff - PID: 06 (idx 003, nxt: 009)
009: 02400-027ff - PID: 06 (idx 004, nxt: -01)
010: 02800-02bff - PID: 05 (idx 000, nxt: 011)
        02be8: 15
011: 02c00-02fff - 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: 020)
019: 04c00-04fff - PID: 01 (idx 000, nxt: -01)
020: 05000-053ff - PID: 06 (idx 003, nxt: -01)
024: 06000-063ff - PID: 05 (idx 000, nxt: 025)
        06014: 66
025: 06400-067ff - PID: 05 (idx 001, nxt: -01)
```

KẾT LUẬN

Qua bài tập lớn này, nhóm chúng em đã hiểu được phần nào nguyên lý hoạt động của một hệ điều hành đơn giản cũng như nâng cao khả năng làm việc nhóm. Bài báo cáo có thể còn nhiều sai sót, mong nhận được góp ý từ các giảng viên.

TÀI LIỆU THAM KHẢO

1. Operating System Concepts, Silberschatz, Galvin, and Gagne, 10th Ed., John Wiley & Sons, Inc., 2018.

2. Operating Systems: Three Easy Pieces, Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau, http://pages.cs.wisc.edu/~remzi/OSTEP/, 2019.

3. Segment Paging, JavatPoint, https://www.javatpoint.com/os-segmented-paging, 2021.