

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



CO2017 - HỆ ĐIỀU HÀNH

Assignment

SIMPLE OPERATING SYSTEM

Nhóm 5 - L08

GVHD: Trần Trương Tuấn Phát
SV: Võ Hoàng - 2113422
Nguyễn Hồng Quân - 2112122
Trần Anh Khoa - 2111541
Phạm Ngọc Khai - 2113650

TP. HỒ CHÍ MINH, THÁNG 4/2023

Mục lục

1	Bảng phân chia công việc	2
2	Lời mở đầu	2
2.1	Tổng quan bài tập lớn	2
2.2	Mục tiêu và kết quả cần đạt được	2
3	Yêu cầu của đề bài	3
3.1	Source code:	3
3.2	Processes	4
3.3	Tạo một process như thế nào?	5
3.4	Chạy mô phỏng như thế nào?	5
4	Hiện thực code và trả lời câu hỏi	6
4.1	Scheduler	6
4.1.1	Hoàn thiện code	6
4.1.1.a	Bổ sung thêm attribute cho struct queue_t	6
4.1.1.b	Hiện thực code trong file queue.c	6
4.1.1.c	Hiện thực code trong file sched.c	7
4.1.1.d	Bổ sung thêm code cho init_scheduler()	7
4.1.1.e	Hàm add_mlq_proc()	7
4.1.1.f	Hàm get_mlq_proc()	8
4.1.1.g	Hàm put_mlq_proc()	8
4.1.1.h	Hàm finish_proc()	9
4.1.2	Output	9
4.1.2.a	Trường hợp input sched	9
4.1.2.b	Trường hợp input sched_0	10
4.1.2.c	Trường hợp input sched_1	12
4.1.2.d	Trường hợp input sched_full	14
4.1.3	Trả lời câu hỏi	17
4.2	Memory Management	17
4.2.1	Hoàn thiện code	17
4.2.1.a	Bổ sung attribute cho vm_rg_struct	17
4.2.1.b	Bổ sung attribute cho vm_rg_struct	18
4.2.1.c	Hoàn thiện ALLOC	18
4.2.1.d	Hoàn thiện FREE	23
4.2.1.e	Hoàn thiện READ/WRITE	24
4.2.1.f	Các hàm kết xuất dữ liệu	29
4.2.2	Output	30
4.2.2.a	os_0_mlq_paging	30
4.2.2.b	os_1_mlq_paging	33
4.2.2.c	os_1_mlq_paging_small_1K	38
4.2.2.d	os_1_mlq_paging_small_4K	44
4.2.2.e	os_1_mlq_paging_small_4K	50
4.2.3	Trả lời câu hỏi	56
4.3	Synchronization	57
4.3.1	Hoàn thiện code	57
4.3.2	Trả lời câu hỏi	58



1 Bảng phân chia công việc

No.	Fullname	Student ID	Problems	Percent of work
1	Võ Hoàng	2113422	- Hiện thực Scheduler - Trả lời câu hỏi	15%
2	Nguyễn Hồng Quân	2112122	- Hiện thực Scheduler - Hiện thực Memory Management - Hiện thực Synchronization - Trả lời câu hỏi	50%
3	Trần Anh Khoa	2111541	- Hiện thực Scheduler - Trả lời câu hỏi	15%
4	Phạm Ngọc Khai	2113650	- Hiện thực Scheduler - Vẽ Gantt chart và viết báo cáo - Trả lời câu hỏi	20%

2 Lời mở đầu

2.1 Tổng quan bài tập lớn

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

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

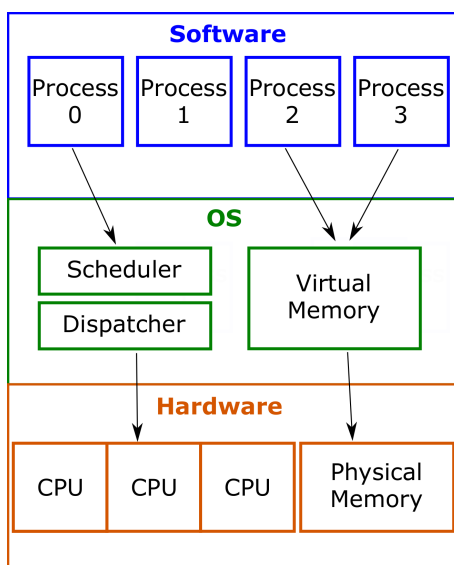
- **Bộ nhớ ảo (Virtual Memory Engine):** cô lập khoảng không gian bộ nhớ của mỗi quá trình với nhau. Mặc dù RAM được chia sẻ bởi nhiều quá trình (multi processeses) nhưng mỗi quá trình đều không biết được sự tồn tại của quá trình khác. Điều đó cho phép mỗi quá trình sở hữu không gian bộ nhớ ảo riêng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

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

2.2 Mục tiêu và kết quả cần đạt được

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

Kết quả cần đạt được: Hiểu được nguyên lý hoạt động của một hệ điều hành đơn giản, đồng thời hiểu được vai trò và ý nghĩa của module của hệ điều hành cũng như cách nó hoạt động



Hình 1: The general view of key modules in this assignment

3 Yêu cầu của đề bài

3.1 Source code:

Header files:

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

Source files:

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

- sched.c: Hiện thực định thời
- mem.c: Hiện thực RAM và bộ nhớ ảo

Make file

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

Output: các mẫu output để đối chiếu kết quả với các input

3.2 Processes

Hệ điều hành quản lý các process bằng PCB của chúng được mô tả như sau:

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

Ý nghĩa:

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

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

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

`alloc [size] [reg]`

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

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

`free [reg]`

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

`read [source] [offset] [destination]`

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

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

`write [data] [destination] [offset]`

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

3.3 Tạo một process như thế nào?

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

```
1 [priority] [N = number of instructions]
2 instruction 0
3 instruction 1
4 ...
5 instruction N-1
```

3.4 Chạy mô phỏng như thế nào?

Để hiện thực quá trình mô phỏng, chúng ta phải mô tả cấu hình của phần cứng và môi trường mà ta mô phỏng. Cấu hình được tạo theo chuẩn:

- Time slice là khoảng thời gian mà quá trình được phép chạy.
- N là số CPU có sẵn
- M là số quá trình sẽ được chạy

```
1 [time slice] [N = Number of CPU] [M = Number of Processes to be run]
2 [time 0] [path 0] [priority 0]
3 [time 1] [path 1] [priority 1]
4 ...
5 [time M-1] [path M-1] [priority M-1]
```

4 Hiện thực code và trả lời câu hỏi

4.1 Scheduler

4.1.1 Hoàn thiện code

4.1.1.a Bổ sung thêm attribute cho struct queue_t

Bổ sung thêm attribute slot_cpu_can_use để giới hạn số lượng cpu có thể sử dụng của mỗi queue trong mlq.

Listing 1: Source code queue_t struct

```
1 struct queue_t
2 {
3     struct pcb_t *proc[MAX_QUEUE_SIZE];
4     int size;
5 #ifdef MLQ_SCHED
6     int slot_cpu_can_use; //use to limit the number of cpu each queue can use
7 #endif
8 };
```

4.1.1.b Hiện thực code trong file queue.c

Hiện thực queue cơ bản theo nguyên tắc FIFO. Chúng ta có **MAX_QUEUE_SIZE = 10** được khai báo trong file queue.h, vậy nên mỗi queue không chứa quá 10 process.

Listing 2: Source code enqueue and dequeue function

```
1 void enqueue(struct queue_t *q, struct pcb_t *proc)
2 {
3     if (q == NULL || proc == NULL)
4         return; // check for null pointers
5
6     if (q->size >= MAX_QUEUE_SIZE)
7     {
8         printf("Queue is full!\n");
9         return;
10    }
11    q->proc[q->size] = proc; // add process to end of array
12    q->size++;              // increment queue size
13 }
14
15 struct pcb_t *dequeue(struct queue_t *q)
16 {
17     if (q == NULL || q->size == 0)
18         return NULL; // check for empty queue
19
20     struct pcb_t *first_proc = q->proc[0];
21
22     // remove first process from queue
23     for (int i = 0; i < q->size - 1; i++)
24     {
```

```
25         q->proc[i] = q->proc[i + 1];  
26     }  
27     q->size--; // decrement queue size  
28     return first_proc;  
29 }
```

4.1.1.c Hiện thực code trong file sched.c

Scheduler có chức năng quản lý việc cập nhật các process sẽ được thực thi cho CPU, cụ thể là quản lý 2 hàng đợi ready và run như ở trên đã mô tả. Cụ thể:

- `init_scheduler()`: bổ sung thêm bước khởi tạo giá trị `slot_cpu_can_use` kiểm soát số lượng cpu một queue có thể sử dụng.
- `get_mlq_proc()`: Trả về process của một quá trình ở hàng đợi sẵn sàng (ready). Nếu hàng đợi khác rỗng tại thời điểm hàm được gọi đồng thời hàng đợi vẫn còn được phép sử dụng thêm cpu, cập nhật lại hàng đợi run bằng process nhận về từ dequeue hàng đợi đó, đồng thời giảm số lượng cpu hàng đợi còn có thể sử dụng. Nếu không đáp ứng yêu cầu, ta duyệt qua hàng đợi khác.
- `put_mlq_proc()`: Chuyển process từ `run_queue` về lại `ready_queue` đồng thời tăng số cpu mà queue đó có thể sử dụng.
- `finish_proc()`: Bổ sung thêm hàm `finish_proc()` thực hiện các thao tác cần thiết sau đó giải phóng vùng nhớ của process đã khởi tạo.

4.1.1.d Bổ sung thêm code cho `init_scheduler()`

Listing 3: Source code `init_scheduler` function

```
1  #ifdef MLQ_SCHED  
2      int i;  
3  
4      for (i = 0; i < MAX_PRIO; i++)  
5      {  
6          mlq_ready_queue[i].size = 0;  
7          // init number of cpu each queue can use maximally  
8          mlq_ready_queue[i].slot_cpu_can_use = MAX_PRIO - i;  
9      }  
10  
11  #endif
```

4.1.1.e Hàm `add_mlq_proc()`

Listing 4: Source code `add_mlq_proc` function

```
1  void add_mlq_proc(struct pcb_t *proc)  
2  {
```



```
3 pthread_mutex_lock(&queue_lock);
4 enqueue(&mlq_ready_queue[proc->prio], proc);
5 pthread_mutex_unlock(&queue_lock);
6 }
```

4.1.1.f Hàm get_mlq_proc()

Listing 5: Source code get_mlq_proc function

```
1 struct pcb_t *get_mlq_proc(void)
2 {
3     struct pcb_t *proc = NULL;
4     unsigned long curr_prio = 0, max_prio = MAX_PRIO;
5     while (curr_prio < max_prio)
6     {
7         if (!empty(&mlq_ready_queue[curr_prio]) &&
8             mlq_ready_queue[curr_prio].slot_cpu_can_use > 0)
9         {
10             pthread_mutex_lock(&queue_lock);
11             proc = dequeue(&mlq_ready_queue[curr_prio]);
12             if (proc != NULL)
13             {
14                 mlq_ready_queue[curr_prio].slot_cpu_can_use--; //decrease slot_cpu_can_use
15             }
16             pthread_mutex_unlock(&queue_lock);
17             break;
18         }
19         else
20         {
21             curr_prio++;
22         }
23     }
24     return proc;
25 }
```

4.1.1.g Hàm put_mlq_proc()

Listing 6: Source code put_mlq_proc function

```
1 void put_mlq_proc(struct pcb_t *proc)
2 {
3     pthread_mutex_lock(&queue_lock);
4     enqueue(&mlq_ready_queue[proc->prio], proc);
5     mlq_ready_queue[proc->prio].slot_cpu_can_use++; //increase slot_cpu_can_use
6     pthread_mutex_unlock(&queue_lock);
7 }
8 }
```

4.1.1.h Hàm finish_proc()

Listing 7: Source code finish_proc function

```
1 void finish_proc(struct pcb_t **proc)
2 {
3     pthread_mutex_lock(&queue_lock);
4     mlq_ready_queue[(*proc)->prio].slot_cpu_can_use++;
5     pthread_mutex_unlock(&queue_lock);
6     free(*proc);
7 }
```

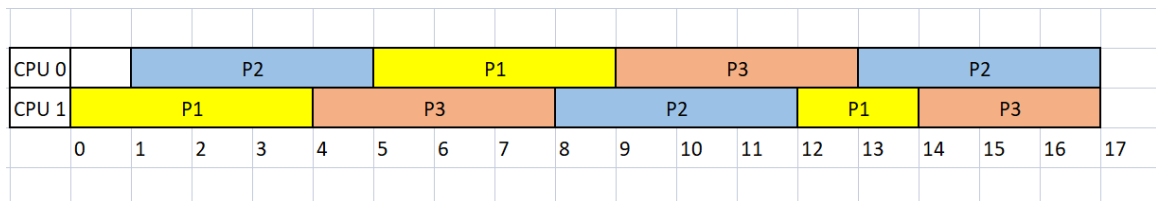
4.1.2 Output

4.1.2.a Trường hợp input sched

```
4 2 3
0 p1s
1 p2s
2 p3s
```

Dữ liệu đầu vào của file sched cho thấy time slice là 4, sử dụng 2 CPU và có 3 process.

Biểu đồ Gantt và kết quả chạy testcase sched



Hình 2: Biểu đồ gantt khi chạy testcase sched

Listing 8: Output of sched

```
1 mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os sched
2 Time slot 0
3 ld_routine
4     Loaded a process at input/proc/p1s, PID: 1 PRI0: 0
5 Time slot 1
6     Loaded a process at input/proc/p2s, PID: 2 PRI0: 0
7     CPU 1: Dispatched process 1
8 Time slot 2
9     Loaded a process at input/proc/p3s, PID: 3 PRI0: 0
10    CPU 0: Dispatched process 2
11 Time slot 3
12 Time slot 4
```

```

13 Time slot 5
14     CPU 1: Put process 1 to run queue
15     CPU 1: Dispatched process 3
16 Time slot 6
17     CPU 0: Put process 2 to run queue
18     CPU 0: Dispatched process 1
19 Time slot 7
20 Time slot 8
21 Time slot 9
22     CPU 1: Put process 3 to run queue
23     CPU 1: Dispatched process 2
24 Time slot 10
25     CPU 0: Put process 1 to run queue
26     CPU 0: Dispatched process 3
27 Time slot 11
28 Time slot 12
29 Time slot 13
30     CPU 1: Put process 2 to run queue
31     CPU 1: Dispatched process 1
32 Time slot 14
33     CPU 0: Put process 3 to run queue
34     CPU 0: Dispatched process 2
35 Time slot 15
36     CPU 1: Processed 1 has finished
37     CPU 1: Dispatched process 3
38 Time slot 16
39 Time slot 17
40 Time slot 18
41     CPU 1: Processed 3 has finished
42     CPU 1 stopped
43     CPU 0: Processed 2 has finished
44     CPU 0 stopped

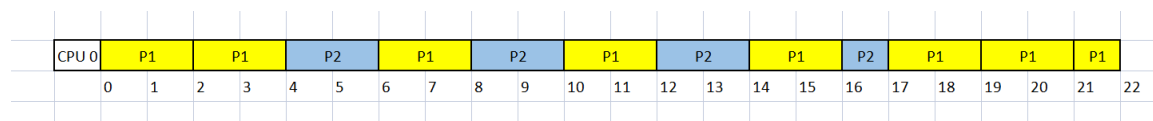
```

4.1.2.b Trường hợp input sched_0

2	1	2
0	s0	
4	s1	

Dữ liệu đầu vào của file sched_0 cho thấy time slice là 2, sử dụng 1 CPU và có 2 process.

Biểu đồ Gantt và kết quả chạy testcase sched_0



Hình 3: Biểu đồ gantt khi chạy testcase sched_0



Listing 9: Output of sched_0

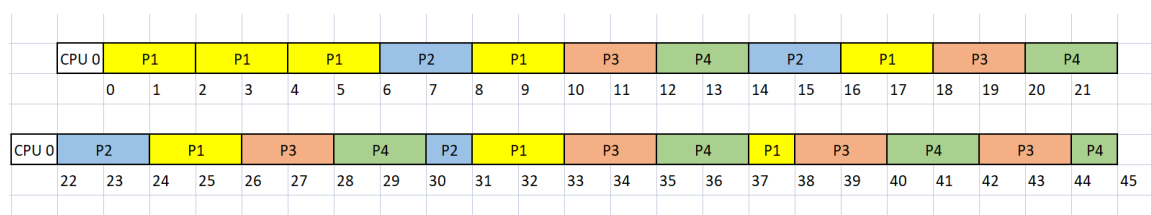
```
1 mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os sched_0
2 Time slot 0
3 ld_routine
4     Loaded a process at input/proc/s0, PID: 1 PRI0: 0
5     CPU 0: Dispatched process 1
6 Time slot 1
7 Time slot 2
8     CPU 0: Put process 1 to run queue
9     CPU 0: Dispatched process 1
10 Time slot 3
11     Loaded a process at input/proc/s1, PID: 2 PRI0: 0
12 Time slot 4
13     CPU 0: Put process 1 to run queue
14     CPU 0: Dispatched process 2
15 Time slot 5
16 Time slot 6
17     CPU 0: Put process 2 to run queue
18     CPU 0: Dispatched process 1
19 Time slot 7
20 Time slot 8
21     CPU 0: Put process 1 to run queue
22     CPU 0: Dispatched process 2
23 Time slot 9
24 Time slot 10
25     CPU 0: Put process 2 to run queue
26     CPU 0: Dispatched process 1
27 Time slot 11
28 Time slot 12
29     CPU 0: Put process 1 to run queue
30     CPU 0: Dispatched process 2
31 Time slot 13
32 Time slot 14
33     CPU 0: Put process 2 to run queue
34     CPU 0: Dispatched process 1
35 Time slot 15
36 Time slot 16
37     CPU 0: Put process 1 to run queue
38     CPU 0: Dispatched process 2
39 Time slot 17
40     CPU 0: Processed 2 has finished
41     CPU 0: Dispatched process 1
42 Time slot 18
43 Time slot 19
44     CPU 0: Put process 1 to run queue
45     CPU 0: Dispatched process 1
46 Time slot 20
47 Time slot 21
48     CPU 0: Put process 1 to run queue
49     CPU 0: Dispatched process 1
50 Time slot 22
51     CPU 0: Processed 1 has finished
52     CPU 0 stopped
```

4.1.2.c Trường hợp input sched_1

2 1 4
0 s0
4 s1
6 s2
7 s3

Dữ liệu đầu vào của file sched_1 cho thấy time slice là 2, sử dụng 1 CPU và có 4 process

Biểu đồ Gantt và kết quả chạy testcase sched_1



Hình 4: Biểu đồ gantt khi chạy testcase sched_1

Listing 10: Output of sched_1

```
1 mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os sched_1
2 Time slot 0
3 ld_routine
4     Loaded a process at input/proc/s0, PID: 1 PRI0: 0
5     CPU 0: Dispatched process 1
6 Time slot 1
7 Time slot 2
8     CPU 0: Put process 1 to run queue
9     CPU 0: Dispatched process 1
10 Time slot 3
11 Time slot 4
12     CPU 0: Put process 1 to run queue
13     CPU 0: Dispatched process 1
14     Loaded a process at input/proc/s1, PID: 2 PRI0: 0
15 Time slot 5
16 Time slot 6
17     CPU 0: Put process 1 to run queue
18     CPU 0: Dispatched process 2
19     Loaded a process at input/proc/s2, PID: 3 PRI0: 0
20 Time slot 7
21     Loaded a process at input/proc/s3, PID: 4 PRI0: 0
22 Time slot 8
23     CPU 0: Put process 2 to run queue
24     CPU 0: Dispatched process 1
25 Time slot 9
26 Time slot 10
27     CPU 0: Put process 1 to run queue
```



```
28         CPU 0: Dispatched process 3
29 Time slot 11
30 Time slot 12
31         CPU 0: Put process 3 to run queue
32         CPU 0: Dispatched process 4
33 Time slot 13
34 Time slot 14
35         CPU 0: Put process 4 to run queue
36         CPU 0: Dispatched process 2
37 Time slot 15
38 Time slot 16
39         CPU 0: Put process 2 to run queue
40         CPU 0: Dispatched process 1
41 Time slot 17
42 Time slot 18
43         CPU 0: Put process 1 to run queue
44         CPU 0: Dispatched process 3
45 Time slot 19
46 Time slot 20
47         CPU 0: Put process 3 to run queue
48         CPU 0: Dispatched process 4
49 Time slot 21
50 Time slot 22
51         CPU 0: Put process 4 to run queue
52         CPU 0: Dispatched process 2
53 Time slot 23
54 Time slot 24
55         CPU 0: Put process 2 to run queue
56         CPU 0: Dispatched process 1
57 Time slot 25
58 Time slot 26
59         CPU 0: Put process 1 to run queue
60         CPU 0: Dispatched process 3
61 Time slot 27
62 Time slot 28
63         CPU 0: Put process 3 to run queue
64         CPU 0: Dispatched process 4
65 Time slot 29
66 Time slot 30
67         CPU 0: Put process 4 to run queue
68         CPU 0: Dispatched process 2
69 Time slot 31
70         CPU 0: Processed 2 has finished
71         CPU 0: Dispatched process 1
72 Time slot 32
73 Time slot 33
74         CPU 0: Put process 1 to run queue
75         CPU 0: Dispatched process 3
76 Time slot 34
77 Time slot 35
78         CPU 0: Put process 3 to run queue
79         CPU 0: Dispatched process 4
80 Time slot 36
```

```

81 Time slot 37
82     CPU 0: Put process 4 to run queue
83     CPU 0: Dispatched process 1
84 Time slot 38
85     CPU 0: Processed 1 has finished
86     CPU 0: Dispatched process 3
87 Time slot 39
88 Time slot 40
89     CPU 0: Put process 3 to run queue
90     CPU 0: Dispatched process 4
91 Time slot 41
92 Time slot 42
93     CPU 0: Put process 4 to run queue
94     CPU 0: Dispatched process 3
95 Time slot 43
96 Time slot 44
97     CPU 0: Processed 3 has finished
98     CPU 0: Dispatched process 4
99 Time slot 45
100    CPU 0: Processed 4 has finished
101    CPU 0 stopped

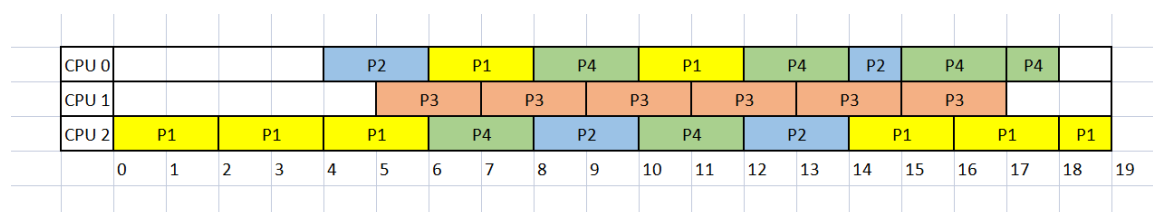
```

4.1.2.d Trường hợp input sched_full

2	3	4
0	s0	1
4	s1	1
5	s2	2
6	s3	1

Nhóm bổ sung thêm một test case phục vụ thử nghiệm tính năng giới hạn số cpu mà mỗi queue trong mlq có thể sử dụng đặt tên là sched_full. Dữ liệu đầu vào của file sched_full cho thấy time slice là 2, sử dụng 3 CPU và có 4 process. Trong file oscfg.c, MAX_PRIO được định nghĩa là 3. Các process trong testcase nằm trong 2 queue với độ ưu tiên 1 và 2 tương ứng có thể sử dụng 2 cpu và 1 cpu.

Biểu đồ Gantt và kết quả chạy testcase sched_full



Hình 5: Biểu đồ gantt khi chạy testcase sched_full

Listing 11: Output of sched_full



```
1 mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os sched_full
2 Time slot 0
3 ld_routine
4     Loaded a process at input/proc/s0, PID: 1 PRI0: 1
5 Time slot 1
6     CPU 1: Dispatched process 1
7 Time slot 2
8 Time slot 3
9     CPU 1: Put process 1 to run queue
10    CPU 1: Dispatched process 1
11 Time slot 4
12    Loaded a process at input/proc/s1, PID: 2 PRI0: 1
13    CPU 0: Dispatched process 2
14 Time slot 5
15    Loaded a process at input/proc/s2, PID: 3 PRI0: 2
16    CPU 1: Put process 1 to run queue
17    CPU 1: Dispatched process 1
18    CPU 2: Dispatched process 3
19 Time slot 6
20    CPU 0: Put process 2 to run queue
21    CPU 0: Dispatched process 2
22    Loaded a process at input/proc/s3, PID: 4 PRI0: 1
23    CPU 1: Put process 1 to run queue
24    CPU 1: Dispatched process 4
25 Time slot 7
26    CPU 2: Put process 3 to run queue
27    CPU 2: Dispatched process 3
28 Time slot 8
29    CPU 0: Put process 2 to run queue
30    CPU 0: Dispatched process 1
31    CPU 1: Put process 4 to run queue
32    CPU 1: Dispatched process 2
33 Time slot 9
34    CPU 2: Put process 3 to run queue
35    CPU 2: Dispatched process 3
36 Time slot 10
37    CPU 0: Put process 1 to run queue
38    CPU 0: Dispatched process 4
39    CPU 1: Put process 2 to run queue
40    CPU 1: Dispatched process 1
41 Time slot 11
42    CPU 2: Put process 3 to run queue
43    CPU 2: Dispatched process 3
44 Time slot 12
45    CPU 0: Put process 4 to run queue
46    CPU 0: Dispatched process 2
47    CPU 1: Put process 1 to run queue
48    CPU 1: Dispatched process 4
49    CPU 0: Processed 2 has finished
50    CPU 0: Dispatched process 1
51 Time slot 13
52    CPU 2: Put process 3 to run queue
53    CPU 2: Dispatched process 3
```




```
54 Time slot 14
55     CPU 1: Put process 4 to run queue
56     CPU 1: Dispatched process 4
57     CPU 0: Put process 1 to run queue
58     CPU 0: Dispatched process 1
59 Time slot 15
60     CPU 2: Put process 3 to run queue
61     CPU 2: Dispatched process 3
62 Time slot 16
63     CPU 1: Put process 4 to run queue
64     CPU 1: Dispatched process 4
65     CPU 0: Put process 1 to run queue
66     CPU 0: Dispatched process 1
67 Time slot 17
68     CPU 2: Processed 3 has finished
69     CPU 2 stopped
70     CPU 0: Processed 1 has finished
71     CPU 0 stopped
72 Time slot 18
73     CPU 1: Put process 4 to run queue
74     CPU 1: Dispatched process 4
75 Time slot 19
76     CPU 1: Processed 4 has finished
77     CPU 1 stopped
78 Time slot 20
```

4.1.3 Trả lời câu hỏi

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

- Priority Feedback Queue (PFQ) có ý tưởng giống với các giải thuật Round Robin (RR), Priority Scheduling, và Multilevel Queue nên PFQ thừa hưởng các lợi thế sau:
 - Priority Scheduling: sử dụng độ ưu tiên khi lựa chọn process giúp các process quan trọng hơn được xử lý trước
 - Round Robin: sử dụng quantum time giúp các process đều được thực thi luân phiên và tránh tình trạng 'starvation' khi một process có độ ưu tiên thấp có thể sẽ không được thực thi.
 - Multilevel Queue: sử dụng nhiều hàng đợi (cụ thể trong PFQ là ready queue và run queue) giúp lựa chọn hiệu quả hơn.
- Tuy nhiên, ta có thể thấy các giải thuật trên tồn tại một số nhược điểm:
 - Priority Scheduling: một process có priority thấp có thể phải đợi vô thời hạn.
 - Round Robin: thời gian đợi trung bình lớn. Nếu như một process cần được thực thi trước (priority cao) phải đợi lâu thì hệ thống sẽ không được vận hành tốt.
 - Multilevel Queue: Giải thuật có độ phức tạp cao, ngoài ra Process ở level thấp nhất có khả năng không được thực thi hoặc phải đợi thời gian dài do không được cung cấp tài nguyên.
- Khi áp dụng nhiều ý tưởng trên cho giải thuật, PFQ sẽ khắc phục nhược điểm của các giải thuật gốc:
 - Giả sử ta có một process q_0 với priority rất cao và thời gian thực thi quá lớn. Một process mới q_1 được thêm vào. Nếu chỉ có một queue, mặc dù q_1 có priority cao nhưng không bằng q_0 , sau một hay nhiều quantum time thì q_1 vẫn phải đợi rất lâu do thời gian thực thi của q_0 quá lớn. Ở đây, PFQ sử dụng ready_queue để giữ những process đợi được đưa vào thực thi và run_queue để giữ những process đã được thực thi sau một quantum_time. Cơ chế này giúp q_1 sau khi được thêm vào ready_queue, q_0 thực thi sau quantum time sẽ được đưa vào run_queue. Lúc này, giải thuật lựa chọn process có priority cao nhất trong ready_queue để thực thi, nếu q_1 có priority cao nhất thì sẽ được lựa chọn. Điều này tương tự với $q_2, q_3, q_4..$ với các priority khác nhau. Như vậy, giải thuật giúp tất cả process được thực thi luân phiên nhưng vẫn ưu tiên ưu tiên process có priority cao hơn và đảm bảo không process nào chờ đợi vô hạn.

4.2 Memory Management

4.2.1 Hoàn thiện code

4.2.1.a Bổ sung attribute cho vm_rg_struct

Bổ sung thêm attribute is_allocated để đánh dấu region đó đã được cấp phát hay chưa.

Listing 12: Source code vm_rg_struct struct

```
1 struct vm_rg_struct {  
2     unsigned long rg_start;
```

```
3 unsigned long rg_end;
4 int is_allocated;
5
6 struct vm_rg_struct *rg_next;
7 };
```

4.2.1.b Bổ sung attribute cho vm_rg_struct

Bổ sung thêm attribute pte_id, p_owner sử dụng hỗ trợ swap giữa mram và mswp.

Listing 13: Source code framephy_struct struct

```
1 /*
2  * FRAME/MEM PHY struct
3  */
4 struct framephy_struct {
5     int fpn;
6     struct framephy_struct *fp_next;
7
8     /* Resereed for tracking allocated framed */
9     struct mm_struct* owner; //memory management of owner process
10    int pte_id; // page number of vm_area
11    struct pcb_t *p_owner; //process that have vm area link to this physical page
12 };
```

4.2.1.c Hoàn thiện ALLOC

Kiểm tra danh sách free region của area hiện tại, nếu có thì region mới lấy từ danh sách free region, nếu không mở rộng area hiện tại.

Lưu ý: Bộ nhớ cấp phát theo paging nên size sẽ được chỉnh sửa theo kích thước page

Listing 14: Source code __alloc() function

```
1 /*__alloc - allocate a region memory
2  * @caller: caller
3  * @vmaid: ID vm area to alloc memory region
4  * @rgid: memory region ID (used to identify variable in symbole table)
5  * @size: allocated size
6  * @alloc_addr: address of allocated memory region
7  *
8  */
9 int __alloc(struct pcb_t *caller, int vmaid, int rgid, int size, int *alloc_addr)
10 {
11     /*Allocate at the topproof */
12     struct vm_rg_struct rgnode;
13
14     size = PAGING_PAGE_ALIGNSZ(size);
15
16     if (size <= 0)
17     {
18         return -1;
```

```
19 }
20
21 pthread_mutex_lock(&vm_lock);
22
23 if (get_free_vmrg_area(caller, vmaid, size, &rgnode) == 0)
24 {
25
26     caller->mm->symrgtbl[rgid].rg_start = rgnode.rg_start;
27     caller->mm->symrgtbl[rgid].rg_end = rgnode.rg_end;
28     caller->mm->symrgtbl[rgid].is_allocated = 1;
29
30     *alloc_addr = rgnode.rg_start;
31
32     pthread_mutex_unlock(&vm_lock);
33     return 0;
34 }
35
36 /* TODO get_free_vmrg_area FAILED handle the region management (Fig.6)*/
37
38 /*Attempt to increate limit to get space */
39 struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, vmaid);
40 int old_sbrk;
41
42 old_sbrk = cur_vma->sbrk;
43
44 /* TODO INCREASE THE LIMIT
45 */
46 inc_vma_limit(caller, vmaid, size);
47
48 /*Successful increase limit */
49 caller->mm->symrgtbl[rgid].rg_start = old_sbrk;
50 caller->mm->symrgtbl[rgid].rg_end = old_sbrk + size;
51 caller->mm->symrgtbl[rgid].is_allocated = 1;
52
53 *alloc_addr = old_sbrk;
54
55
56 pthread_mutex_unlock(&vm_lock);
57 return 0;
58 }
```

Tăng vm area, liên kết phần bộ nhớ ảo vừa được tăng với bộ nhớ vật lý.

Listing 15: Source code inc_vma_limit() fuction

```
1 /*inc_vma_limit - increase vm area limits to reserve space for new variable
2  *@caller: caller
3  *@vmaid: ID vm area to alloc memory region
4  *@inc_sz: increment size
5  *
6  */
7 int inc_vma_limit(struct pcb_t *caller, int vmaid, int inc_sz)
8 {
```

```
9 struct vm_rg_struct *newrg = malloc(sizeof(struct vm_rg_struct));
10 int inc_amt = PAGING_PAGE_ALIGNSZ(inc_sz);
11 int incnumpage = inc_amt / PAGING_PAGESZ;
12 struct vm_rg_struct *area = get_vm_area_node_at_brk(caller, vmaid, inc_sz, inc_amt);
13 struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, vmaid);
14
15 int old_end = cur_vma->vm_end;
16
17 /*Validate overlap of obtained region */
18 if (validate_overlap_vm_area(caller, vmaid, area->rg_start, area->rg_end) < 0)
19 {
20     fprintf(stderr, "error: overlap when allocated memory region");
21     return -1; /*Overlap and failed allocation */
22 }
23
24 /* The obtained vm area (only)
25  * now will be alloc real ram region */
26 cur_vma->vm_end += inc_sz;
27 cur_vma->sbrk += inc_sz; // Update the sbrk field to reflect the new vm_end value
28 if (vm_map_ram(caller, area->rg_start, area->rg_end,
29               old_end, incnumpage, newrg) < 0)
30     return -1; /* Map the memory to MEMRAM */
31
32 // assign new region to new memory of vm area
33 struct vm_rg_struct *new_rg = malloc(sizeof(struct vm_rg_struct));
34 new_rg->rg_start = old_end;
35 new_rg->rg_end = cur_vma->sbrk;
36
37 return 0;
38 }
```

Listing 16: Source code vm_map_ram() fuction

```
1 /*
2  * vm_map_ram - do the mapping all vm are to ram storage device
3  * @caller    : caller
4  * @astart    : vm area start
5  * @aend      : vm area end
6  * @mapstart  : start mapping point
7  * @incpgnum  : number of mapped page
8  * @ret_rg    : returned region
9  */
10 int vm_map_ram(struct pcb_t *caller, int astart, int aend, int mapstart, int incpgnum,
11               struct vm_rg_struct *ret_rg)
12 {
13     struct framephy_struct *frm_lst = NULL;
14     int ret_alloc;
15
16     /*@bksysnet: author provides a feasible solution of getting frames
17     *FATAL logic in here, wrong behaviour if we have not enough page
18     *i.e. we request 1000 frames meanwhile our RAM has size of 3 frames
19     *Don't try to perform that case in this simple work, it will result
```

```
19  *in endless procedure of swap-off to get frame and we have not provide
20  *duplicate control mechanism, keep it simple
21  */
22  ret_alloc = alloc_pages_range(caller, incpgnum, &frm_lst);
23
24  if (ret_alloc < 0 && ret_alloc != -3000)
25      return -1;
26
27  /* Out of memory */
28  if (ret_alloc == -3000)
29  {
30  #ifdef MMDBG
31      printf("OOM: vm_map_ram out of memory \n");
32  #endif
33      return -1;
34  }
35
36  /* it leaves the case of memory is enough but half in ram, half in swap
37  * do the swaping all to swapper to get the all in ram */
38  vmmap_page_range(caller, mapstart, incpgnum, frm_lst, ret_rg);
39
40  return 0;
41 }
```

Cấp phát ram để liên kết với bộ nhớ ảo vừa mới tạo, nếu ram không đủ chỗ trống, swap victim page sang mswp để có thêm bộ nhớ. Giải thuật tìm victim page trình bày ở phần sau.

Listing 17: Source code alloc_pages_range() function

```
1  /*
2  * alloc_pages_range - allocate req_pgnum of frame in ram
3  * @caller : caller
4  * @req_pgnum : request page num
5  * @frm_lst : frame list
6  */
7
8  int alloc_pages_range(struct pcb_t *caller, int req_pgnum, struct framephy_struct
9  **frm_lst)
10 {
11     int pgit, fpn;
12     struct framephy_struct *newfp_str = NULL;
13
14     for (pgit = 0; pgit < req_pgnum; pgit++)
15     {
16         newfp_str = (struct framephy_struct *)malloc(sizeof(struct framephy_struct));
17         if (MEMPHY_get_freefp(caller->mram, &fpn) == 0)
18         {
19             newfp_str->fpn = fpn;
20             newfp_str->owner = caller->mm;
21         }
22         else
23         { // ERROR CODE of obtaining some but not enough frames
24             int vicpgn, swpfpn;
```

```
24     struct framephy_struct *victim_fp = (struct framephy_struct *)malloc(sizeof(struct
        framephy_struct));
25     if (find_victim_page(caller, &victim_fp) == -1 ||
        MEMPHY_get_freefp(caller->active_mswp, &swpfpn) == -1)
26     {
27         struct framephy_struct *freefp_str;
28         while (*frm_lst != NULL)
29         {
30             freefp_str = *frm_lst;
31             *frm_lst = (*frm_lst)->fp_next;
32             free(freefp_str);
33         }
34         return -3000;
35     }
36     vicpgn = victim_fp->pte_id;
37     uint32_t vicpte = victim_fp->owner->pgd[vicpgn];
38     int vicfpn = PAGING_FPN(vicpte);
39     __swap_cp_page(victim_fp->p_owner->mram, vicfpn, caller->active_mswp, swpfpn);
40     pte_set_swap(&vicpte, 0, swpfpn);
41     newfp_str->fpn = vicfpn;
42     newfp_str->owner = caller->mm;
43 }
44 newfp_str->fp_next = *frm_lst;
45 *frm_lst = newfp_str;
46 }
47
48 return 0;
49 }
```

Listing 18: Source code vmap_page_range() function

```
1  /*
2   * vmap_page_range - map a range of page at aligned address
3   */
4  int vmap_page_range(struct pcb_t *caller,          // process call
5                     int addr,                      // start address which is aligned to
6                     int pgnum,                     // num of mapping page
7                     struct framephy_struct *frames, // list of the mapped frames
8                     struct vm_rg_struct *ret_rg) // return mapped region, the real mapped
9                     fp
10 {
11     mapped
12     struct framephy_struct *fpit = frames;          // Start with the first frame in the list
13     int pgit;
14     int pgn = PAGING_PGN(addr);
15
16     ret_rg->rg_end = ret_rg->rg_start = addr; // at least the very first space is usable
17
18     /* TODO map range of frame to address space
19      * [addr to addr + pgnum*PAGING_PAGESZ
20      * in page table caller->mm->pgd[]
```

```
19  */
20
21  for (pgit = 0; pgit < pgn; pgit++)
22  {
23      // Update the page table entry at the given address
24      pte_set_fpn(&caller->mm->pgd[pgn + pgit], fpit->fpn);
25
26      // Update the region boundaries
27      ret_rg->rg_end += PAGING_PAGESZ;
28
29      /* Tracking for later page replacement activities (if needed)
30       * Enqueue new usage page */
31      enlist_pgn_node(&caller->mm->fifo_pgn, pgn + pgit);
32      enlist_fpn_node(&caller->mm->used_fp_list, fpit->fpn, caller->mm, pgn + pgit,
33                     caller);
34
35      fpit->p_owner = caller;
36      fpit->pte_id = pgn + pgit;
37
38      // Move to the next frame in the list
39      fpit = fpit->fp_next;
40  }
41
42  return 0;
43 }
```

4.2.1.d Hoàn thiện FREE

Thay đổi attribute is_allocated thành giá trị 0 và thêm region và freerg_list.

Listing 19: Source code __free() function

```
1  /*__free - remove a region memory
2  * @caller: caller
3  * @vmaid: ID vm area to alloc memory region
4  * @rgid: memory region ID (used to identify variable in symbole table)
5  * @size: allocated size
6  *
7  */
8  int __free(struct pcb_t *caller, int vmoid, int rgid)
9  {
10     if (rgid < 0 || rgid > PAGING_MAX_SYMTBL_SZ)
11     {
12         return -1;
13     }
14     /* TODO: Manage the collect freed region to freerg_list */
15
16     // Get the free region
17     struct vm_rg_struct *free_rg = get_symrg_byid(caller->mm, rgid);
18
19     if (free_rg->is_allocated != 1)
20     {
```



```
21     printf("Unable to delocated memory region %d\n", rgid);
22     return -1;
23 }
24
25 if (free_rg->rg_start == free_rg->rg_end)
26 {
27     return -1;
28 }
29
30 pthread_mutex_lock(&vm_lock);
31
32 struct vm_rg_struct *rgnode = malloc(sizeof(struct vm_rg_struct));
33
34 // Assign the rgnode with appropriate values
35 rgnode->rg_start = free_rg->rg_start;
36 rgnode->rg_end = free_rg->rg_end;
37 rgnode->rg_next = NULL;
38
39 free_rg->is_allocated = 0;
40
41 /*enlist the obsoleted memory region */
42 enlist_vm_freerg_list(caller->mm, rgnode);
43
44 pthread_mutex_unlock(&vm_lock);
45
46 return 0;
47 }
```

4.2.1.e Hoàn thiện READ/WRITE

Ghi và đọc dữ liệu vào bộ nhớ. Bộ nhớ gồm RAM và SWAP. Để thao tác, dữ liệu cần được đưa lên bộ nhớ chính (RAM) nên sẽ yêu cầu thao tác swap dữ liệu giữa RAM và SWAP.

Listing 20: Source code __read() function

```
1  /*__read - read value in region memory
2  *@caller: caller
3  *@vmaid: ID vm area to alloc memory region
4  *@offset: offset to acess in memory region
5  *@rgid: memory region ID (used to identify variable in symbole table)
6  *@size: allocated size
7  *
8  */
9  int __read(struct pcb_t *caller, int vmaid, int rgid, int offset, BYTE *data)
10 {
11
12     struct vm_rg_struct *curr_g = get_symrg_byid(caller->mm, rgid);
13
14     if (!curr_g->is_allocated)
15     {
16         printf("read region=%d offset=%d\n", rgid, offset);
17         printf("Access violation reading location: memory region %d\n", rgid);
```

```
18     return -1;
19 }
20
21 struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, vmaid);
22
23 if (currq == NULL || cur_vma == NULL) /* Invalid memory identify */
24 {
25     return -1;
26 }
27
28 pthread_mutex_lock(&vm_lock);
29
30 pg_getval(caller->mm, currq->rg_start + offset, data, caller);
31
32 pthread_mutex_unlock(&vm_lock);
33
34 return 0;
35 }
```

Listing 21: Source code __write() function

```
1  /*__write - write a region memory
2  *@caller: caller
3  *@vmaid: ID vm area to alloc memory region
4  *@offset: offset to access in memory region
5  *@rgid: memory region ID (used to identify variable in symbole table)
6  *@size: allocated size
7  *
8  */
9  int __write(struct pcb_t *caller, int vmaid, int rgid, int offset, BYTE value)
10 {
11
12     struct vm_rg_struct *currq = get_symrg_byid(caller->mm, rgid);
13
14     if (!currq->is_allocated)
15     {
16         printf("Access violation writing location: memory region %d\n", rgid);
17         return -1;
18     }
19
20     struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, vmaid);
21
22     if (currq == NULL || cur_vma == NULL) /* Invalid memory identify */
23     {
24         return -1;
25     }
26
27     pthread_mutex_lock(&vm_lock);
28
29     pg_setval(caller->mm, currq->rg_start + offset, value, caller);
30
31     pthread_mutex_unlock(&vm_lock);
```

```
32     return 0;  
33 }
```

Listing 22: Source code pg_getval() function

```
1  /*pg_getval - read value at given offset  
2  *@mm: memory region  
3  *@addr: virtual address to access  
4  *@value: value  
5  *  
6  */  
7  int pg_getval(struct mm_struct *mm, int addr, BYTE *data, struct pcb_t *caller)  
8  {  
9      int pgn = PAGING_PGN(addr);  
10     int off = PAGING_OFFST(addr);  
11     int fpn;  
12  
13     /* Get the page to MEMRAM, swap from MEMSWAP if needed */  
14     if (pg_getpage(mm, pgn, &fpn, caller) != 0)  
15         return -1; /* invalid page access */  
16  
17     int phyaddr = (fpn << PAGING_ADDR_FPN_LOBIT) + off;  
18  
19     MEMPHY_read(caller->mram, phyaddr, data);  
20  
21     return 0;  
22 }
```

Listing 23: Source code pg_setval() function

```
1  /*pg_setval - write value to given offset  
2  *@mm: memory region  
3  *@addr: virtual address to access  
4  *@value: value  
5  *  
6  */  
7  int pg_setval(struct mm_struct *mm, int addr, BYTE value, struct pcb_t *caller)  
8  {  
9      int pgn = PAGING_PGN(addr);  
10     int off = PAGING_OFFST(addr);  
11     int fpn;  
12  
13     /* Get the page to MEMRAM, swap from MEMSWAP if needed */  
14     if (pg_getpage(mm, pgn, &fpn, caller) != 0)  
15         return -1; /* invalid page access */  
16  
17     int phyaddr = (fpn << PAGING_ADDR_FPN_LOBIT) + off;  
18  
19     MEMPHY_write(caller->mram, phyaddr, value);  
20  
21     return 0;  
22 }
```

22 }

READ và WRITE sử dụng `pg_getpage()` để lấy page trên bộ nhớ vật lý và thao tác dữ liệu trên đó. `pf_getpage` kiểm tra và nếu vùng nhớ chỉ định đang nằm ở SWAP thì xác định victim page sau đó swap để đưa vùng nhớ đó vào RAM.

Listing 24: Source code `pg_getpage()` function

```
1  /*pg_getpage - get the page in ram
2  *@mm: memory region
3  *@pagenum: PGN
4  *@framenum: return FPN
5  *@caller: caller
6  *
7  */
8  int pg_getpage(struct mm_struct *mm, int pgn, int *fpn, struct pcb_t *caller)
9  {
10     uint32_t pte = mm->pgd[pgn];
11
12     if (!PAGING_PAGE_PRESENT(pte))
13     { /* Page is not online, make it actively living */
14         int vicpgn, swpfpn;
15         int vicfpn;
16         uint32_t vicpte;
17
18         int tgtfpn = PAGING_SWP(pte); // the target frame storing our variable
19
20         /* TODO: Play with your paging theory here */
21         /* Find victim page */
22         struct framephy_struct *victim_fp = (struct framephy_struct *)malloc(sizeof(struct
                framephy_struct));
23         if (find_victim_page(caller, &victim_fp) == -1)
24             return -1;
25
26         vicpgn = victim_fp->pte_id;
27
28         vicpte = victim_fp->owner->pgd[vicpgn];
29         vicfpn = PAGING_FPN(vicpte);
30
31         /* Get free frame in MEMSWP */
32         if (MEMPHY_get_freefp(caller->active_mswp, &swpfpn) == -1)
33             return -1;
34
35         /* Do swap frame from MEMRAM to MEMSWP and vice versa*/
36         /* Copy victim frame to swap */
37         __swap_cp_page(victim_fp->p_owner->mram, vicfpn, caller->active_mswp, swpfpn);
38         /* Copy target frame from swap to mem */
39         __swap_cp_page(caller->active_mswp, tgtfpn, caller->mram, vicfpn);
40
41         /* Update page table */
42         pte_set_swap(&vicpte, 0, swpfpn);
43
44         /* Update its online status of the target page */
```

```
45     pte_set_fpn(&pte, vicfpn);
46
47     *fpn = PAGING_FPN(pte);
48
49     enlist_pgn_node(&caller->mm->fifo_pgn, pgn);
50
51     enlist_fpn_node(&caller->mram->used_fp_list, *fpn, caller->mm, pgn, caller);
52 }
53
54 *fpn = PAGING_FPN(pte);
55
56 return 0;
57 }
```

find_victim_page() sử dụng giải thuật fifo với queue used_fp_list trong mram, trả về physical page tương ứng.

Listing 25: Source code find_victim_page() function

```
1  /*find_victim_page - find victim page
2  *@caller: caller
3  *@pgn: return physical page
4  *
5  */
6  int find_victim_page(struct pcb_t *caller, struct framephy_struct **re_fp)
7  {
8      struct framephy_struct *fp_q = caller->mram->used_fp_list;
9      struct framephy_struct *prev = NULL;
10
11     /* TODO: Implement the theoretical mechanism to find the victim page */
12
13     if (fp_q == NULL)
14     {
15         // The FIFO queue is empty, which should not happen
16         return -1;
17     }
18
19     // Traverse to the end of the FIFO queue to find the oldest page
20     while (fp_q->fp_next != NULL)
21     {
22         prev = fp_q;
23         fp_q = fp_q->fp_next;
24     }
25
26     // Get the page number of the oldest page
27     *re_fp = fp_q;
28
29     // Remove the oldest page from the FIFO queue
30     if (prev != NULL)
31     {
32         prev->fp_next = NULL;
33     }
34     else
```

```
35 {  
36     caller->mram->used_fp_list = NULL;  
37 }  
38  
39 return 0;  
40 }
```

4.2.1.f Các hàm kết xuất dữ liệu

Hiển thị table entries của process

Listing 26: Source code print_pgtbl() function

```
1 int print_pgtbl(struct pcb_t *caller, uint32_t start, uint32_t end)  
2 {  
3     int pgn_start, pgn_end;  
4     int pgit;  
5  
6     if (end == -1)  
7     {  
8         pgn_start = 0;  
9         struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, 0);  
10        end = cur_vma->vm_end;  
11    }  
12    pgn_start = PAGING_PGN(start);  
13    pgn_end = PAGING_PGN(end);  
14  
15    printf("print_pgtbl: %d - %d", start, end);  
16    if (caller == NULL)  
17    {  
18        printf("NULL caller\n");  
19        return -1;  
20    }  
21    printf("\n");  
22  
23    for (pgit = pgn_start; pgit < pgn_end; pgit++)  
24    {  
25        printf("%08ld: %08x\n", pgit * sizeof(uint32_t), caller->mm->pgd[pgit]);  
26    }  
27  
28    for (pgit = pgn_start; pgit < pgn_end; pgit++)  
29    {  
30        printf("Page Number: %d -> Frame Number: %d\n", pgit,  
31            PAGING_FPN(caller->mm->pgd[pgit]));  
32    }  
33    return 0;  
34 }
```

Hiển thị physical memory.

Listing 27: Source code MEMPHY_dump() function

```
1 int MEMPHY_dump(struct memphy_struct *mp)
2 {
3     /*TODO dump memphy contnt mp->storage
4      *   for tracing the memory content
5      */
6     printf("Memory Dump:\n");
7     printf("-----\n");
8
9     if (mp == NULL || mp->storage == NULL)
10    {
11        printf("Invalid memory structure.\n");
12        return -1;
13    }
14
15    for (int i = 0; i < mp->maxsz; ++i)
16    {
17        if (mp->storage[i] != 0)
18        {
19            printf("BYTE %08x: %d\n", i, mp->storage[i]);
20        }
21    }
22
23    printf("\n");
24    return 0;
25 }
```

4.2.2 Output

4.2.2.a os_0_mlq_paging

Listing 28: output of os_0_mlq_paging

```
1 mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os
   os_0_mlq_paging
2 Time slot 0
3 ld_routine
4     Loaded a process at input/proc/p0s, PID: 1 PRI0: 0
5     CPU 1: Dispatched process 1
6 Time slot 1
7 process 1 alloc region 0 size 300
8 print_pgtbl: 0 - 512
9 00000000: 80000001
10 00000004: 80000000
11 Page Number: 0 -> Frame Number: 1
12 Page Number: 1 -> Frame Number: 0
13 Memory Dump:
14 -----
15
16 Time slot 2
17     Loaded a process at input/proc/p1s, PID: 2 PRI0: 15
```



```
18 process 1 alloc region 4 size 300
19 print_pgtbl: 0 - 1024
20 00000000: 80000001
21 00000004: 80000000
22 00000008: 80000003
23 00000012: 80000002
24 Page Number: 0 -> Frame Number: 1
25 Page Number: 1 -> Frame Number: 0
26 Page Number: 2 -> Frame Number: 3
27 Page Number: 3 -> Frame Number: 2
28 Memory Dump:
29 -----
30 CPU 0: Dispatched process 2
31
32 Time slot 3
33 Loaded a process at input/proc/p1s, PID: 3 PRI0: 0
34 process 1 free region 0
35 Time slot 4
36 Loaded a process at input/proc/p1s, PID: 4 PRI0: 0
37 process 1 alloc region 1 size 100
38 print_pgtbl: 0 - 1024
39 00000000: 80000001
40 00000004: 80000000
41 00000008: 80000003
42 00000012: 80000002
43 Page Number: 0 -> Frame Number: 1
44 Page Number: 1 -> Frame Number: 0
45 Page Number: 2 -> Frame Number: 3
46 Page Number: 3 -> Frame Number: 2
47 Memory Dump:
48 -----
49
50 Time slot 5
51 write region=1 offset=20 value=100
52 print_pgtbl: 0 - 1024
53 00000000: 80000001
54 00000004: 80000000
55 00000008: 80000003
56 00000012: 80000002
57 Page Number: 0 -> Frame Number: 1
58 Page Number: 1 -> Frame Number: 0
59 Page Number: 2 -> Frame Number: 3
60 Page Number: 3 -> Frame Number: 2
61 Memory Dump:
62 -----
63 BYTE 00000114: 100
64
65 Time slot 6
66 CPU 1: Put process 1 to run queue
67 CPU 1: Dispatched process 3
68 Time slot 7
69 Time slot 8
70 CPU 0: Put process 2 to run queue
```




```
71      CPU 0: Dispatched process 4
72 Time slot 9
73 Time slot 10
74 Time slot 11
75 Time slot 12
76      CPU 1: Put process 3 to run queue
77      CPU 1: Dispatched process 1
78 read region=1 offset=20 value=100
79 print_pgtbl: 0 - 1024
80 00000000: 80000001
81 00000004: 80000000
82 00000008: 80000003
83 00000012: 80000002
84 Page Number: 0 -> Frame Number: 1
85 Page Number: 1 -> Frame Number: 0
86 Page Number: 2 -> Frame Number: 3
87 Page Number: 3 -> Frame Number: 2
88 Memory Dump:
89 -----
90 BYTE 00000114: 100
91
92 Time slot 13
93 write region=2 offset=20 value=102
94 access violation writing location: memory region 2
95 Memory Dump:
96 -----
97 BYTE 00000114: 100
98
99 Time slot 14
100      CPU 0: Put process 4 to run queue
101      CPU 0: Dispatched process 3
102 read region=2 offset=20
103 access violation reading location: memory region 2
104 Time slot 15
105 write region=3 offset=20 value=103
106 access violation writing location: memory region 3
107 Memory Dump:
108 -----
109 BYTE 00000114: 100
110
111 Time slot 16
112      CPU 1: Processed 1 has finished
113      CPU 1: Dispatched process 4
114 Time slot 17
115 Time slot 18
116      CPU 0: Processed 3 has finished
117      CPU 0: Dispatched process 2
118 Time slot 19
119 Time slot 20
120      CPU 1: Processed 4 has finished
121      CPU 1 stopped
122 Time slot 21
123 Time slot 22
```



```
124 CPU 0: Processed 2 has finished
125 CPU 0 stopped
```

4.2.2.b os_1_mlq_paging

Listing 29: output of os_1_mlq_paging

```
1 mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os
  os_1_mlq_paging
2 Time slot 0
3 ld_routine
4 Time slot 1
5     Loaded a process at input/proc/p0s, PID: 1 PRI0: 130
6     CPU 0: Dispatched process 1
7 Time slot 2
8     Loaded a process at input/proc/s3, PID: 2 PRI0: 39
9 process 1 alloc region 0 size 300
10 print_pgtbl: 0 - 512
11 00000000: 80000001
12 00000004: 80000000
13 Page Number: 0 -> Frame Number: 1
14 Page Number: 1 -> Frame Number: 0
15 Memory Dump:
16 -----
17
18 Time slot 3
19     CPU 2: Dispatched process 2
20     CPU 0: Put process 1 to run queue
21     CPU 0: Dispatched process 1
22 process 1 alloc region 4 size 300
23 print_pgtbl: 0 - 1024
24 00000000: 80000001
25 00000004: 80000000
26 00000008: 80000003
27 00000012: 80000002
28 Page Number: 0 -> Frame Number: 1
29 Page Number: 1 -> Frame Number: 0
30 Page Number: 2 -> Frame Number: 3
31 Page Number: 3 -> Frame Number: 2
32 Memory Dump:
33 -----
34
35 Time slot 4
36     Loaded a process at input/proc/m1s, PID: 3 PRI0: 15
37 process 1 free region 0
38     CPU 3: Dispatched process 3
39 process 3 alloc region 0 size 300
40 print_pgtbl: 0 - 512
41 00000000: 80000005
42 00000004: 80000004
43 Page Number: 0 -> Frame Number: 5
```



```
44 Page Number: 1 -> Frame Number: 4
45 Memory Dump:
46 -----
47 Time slot 5
48     CPU 2: Put process 2 to run queue
49     CPU 2: Dispatched process 2
50     CPU 0: Put process 1 to run queue
51     CPU 0: Dispatched process 1
52 process 1 alloc region 1 size 100
53 print_pgtbl: 0 - 1024
54 00000000: 80000001
55 00000004: 80000000
56 00000008: 80000003
57 00000012: 80000002
58 Page Number: 0 -> Frame Number: 1
59 Page Number: 1 -> Frame Number: 0
60 Page Number: 2 -> Frame Number: 3
61 Page Number: 3 -> Frame Number: 2
62 Memory Dump:
63 -----
64
65
66     Loaded a process at input/proc/s2, PID: 4 PRI0: 120
67 Time slot 6
68     CPU 1: Dispatched process 4
69 write region=1 offset=20 value=100
70 print_pgtbl: 0 - 1024
71 00000000: 80000001
72 00000004: 80000000
73 00000008: 80000003
74 00000012: 80000002
75 Page Number: 0 -> Frame Number: 1
76 Page Number: 1 -> Frame Number: 0
77 Page Number: 2 -> Frame Number: 3
78 Page Number: 3 -> Frame Number: 2
79 Memory Dump:
80 -----
81 BYTE 00000114: 100
82 process 3 alloc region 1 size 100
83 print_pgtbl: 0 - 768
84 00000000: 80000005
85 00000004: 80000004
86 00000008: 80000006
87 Page Number: 0 -> Frame Number: 5
88 Page Number: 1 -> Frame Number: 4
89 Page Number: 2 -> Frame Number: 6
90 Memory Dump:
91 -----
92 BYTE 00000114: 100
93
94
95     Loaded a process at input/proc/m0s, PID: 5 PRI0: 120
96 Time slot 7
```



```
97         CPU 2: Put process 2 to run queue
98         CPU 2: Dispatched process 2
99         CPU 0: Put process 1 to run queue
100        CPU 0: Dispatched process 5
101    process 5 alloc region 0 size 300
102    print_pgtbl: 0 - 512
103    00000000: 80000008
104    00000004: 80000007
105    Page Number: 0 -> Frame Number: 8
106    Page Number: 1 -> Frame Number: 7
107    Memory Dump:
108    -----
109    BYTE 00000114: 100
110        CPU 3: Put process 3 to run queue
111        CPU 3: Dispatched process 3
112    process 3 free region 0
113
114    Time slot 8
115        CPU 1: Put process 4 to run queue
116        CPU 1: Dispatched process 4
117    process 5 alloc region 1 size 100
118    print_pgtbl: 0 - 768
119    00000000: 80000008
120    00000004: 80000007
121    00000008: 80000009
122    Page Number: 0 -> Frame Number: 8
123    Page Number: 1 -> Frame Number: 7
124    Page Number: 2 -> Frame Number: 9
125    Memory Dump:
126    -----
127    BYTE 00000114: 100
128
129    process 3 alloc region 2 size 100
130    print_pgtbl: 0 - 768
131    00000000: 80000005
132    00000004: 80000004
133    00000008: 80000006
134    Page Number: 0 -> Frame Number: 5
135    Page Number: 1 -> Frame Number: 4
136    Page Number: 2 -> Frame Number: 6
137    Memory Dump:
138    -----
139    BYTE 00000114: 100
140
141        Loaded a process at input/proc/p1s, PID: 6 PRI0: 15
142    Time slot 9
143        CPU 3: Put process 3 to run queue
144        CPU 3: Dispatched process 6
145        CPU 2: Put process 2 to run queue
146        CPU 2: Dispatched process 3
147    process 3 free region 2
148        CPU 0: Put process 5 to run queue
149        CPU 0: Dispatched process 2
```



```
150 Time slot 10
151     CPU 1: Put process 4 to run queue
152     CPU 1: Dispatched process 5
153 process 5 free region 0
154 process 3 free region 1
155 Time slot 11
156     CPU 3: Put process 6 to run queue
157     CPU 3: Dispatched process 6
158     CPU 2: Put process 3 to run queue
159     CPU 2: Dispatched process 3
160 Unable to delocated memory region 1
161 process 3 free region 1
162 process 5 alloc region 2 size 100
163 print_pgtbl: 0 - 768
164 00000000: 80000008
165 00000004: 80000007
166 00000008: 80000009
167 Page Number: 0 -> Frame Number: 8
168 Page Number: 1 -> Frame Number: 7
169 Page Number: 2 -> Frame Number: 9
170 Memory Dump:
171 -----
172 BYTE 00000114: 100
173     Loaded a process at input/proc/s0, PID: 7 PRI0: 38
174
175     CPU 0: Put process 2 to run queue
176     CPU 0: Dispatched process 7
177 Time slot 12
178 Unable to delocated memory region 1
179 process 3 free region 1
180     CPU 1: Put process 5 to run queue
181     CPU 1: Dispatched process 2
182 Time slot 13
183     CPU 3: Put process 6 to run queue
184     CPU 3: Dispatched process 6
185     CPU 2: Processed 3 has finished
186     CPU 2: Dispatched process 4
187     CPU 0: Put process 7 to run queue
188     CPU 0: Dispatched process 7
189 Time slot 14
190     CPU 1: Put process 2 to run queue
191     CPU 1: Dispatched process 2
192 Time slot 15
193     CPU 3: Put process 6 to run queue
194     CPU 3: Dispatched process 6
195     CPU 2: Put process 4 to run queue
196     CPU 2: Dispatched process 5
197 write region=1 offset=20 value=102
198 print_pgtbl: 0 - 768
199 00000000: 80000008
200 00000004: 80000007
201 00000008: 80000009
202 Page Number: 0 -> Frame Number: 8
```



```
203 Page Number: 1 -> Frame Number: 7
204 Page Number: 2 -> Frame Number: 9
205 Memory Dump:
206 -----
207 BYTE 00000114: 100
208 BYTE 00000914: 102
209     CPU 1: Processed 2 has finished
210     CPU 1: Dispatched process 4
211
212     CPU 0: Put process 7 to run queue
213     CPU 0: Dispatched process 7
214 Time slot 16
215     Loaded a process at input/proc/s1, PID: 8 PRIO: 0
216 write region=2 offset=1000 value=1
217 access violation writing location: memory region 2
218 Time slot 17
219     CPU 3: Put process 6 to run queue
220     CPU 3: Dispatched process 8
221     CPU 2: Put process 5 to run queue
222     CPU 2: Dispatched process 6
223     CPU 0: Put process 7 to run queue
224     CPU 0: Dispatched process 7
225     CPU 1: Put process 4 to run queue
226     CPU 1: Dispatched process 5
227 write region=2 offset=1000 value=1
228 access violation writing location: memory region 2
229 Time slot 18
230     CPU 1: Processed 5 has finished
231     CPU 1: Dispatched process 4
232     CPU 3: Put process 8 to run queue
233     CPU 3: Dispatched process 8
234 Time slot 19
235     CPU 2: Processed 6 has finished
236     CPU 2: Dispatched process 1
237 read region=1 offset=20 value=100
238 print_pgtbl: 0 - 1024
239 00000000: 80000001
240 00000004: 80000000
241 00000008: 80000003
242 00000012: 80000002
243 Page Number: 0 -> Frame Number: 1
244 Page Number: 1 -> Frame Number: 0
245 Page Number: 2 -> Frame Number: 3
246 Page Number: 3 -> Frame Number: 2
247 Memory Dump:
248 -----
249 BYTE 00000114: 100
250 BYTE 00000914: 102
251     CPU 0: Put process 7 to run queue
252     CPU 0: Dispatched process 7
253
254 Time slot 20
255     CPU 1: Put process 4 to run queue
```



```
256 CPU 1: Dispatched process 4
257 write region=2 offset=20 value=102
258 access violation writing location: memory region 2
259 Memory Dump:
260 -----
261 BYTE 00000114: 100
262 BYTE 00000914: 102
263
264 CPU 3: Put process 8 to run queue
265 CPU 3: Dispatched process 8
266 Time slot 21
267 CPU 0: Put process 7 to run queue
268 CPU 0: Dispatched process 7
269 CPU 2: Put process 1 to run queue
270 CPU 2: Dispatched process 1
271 read region=2 offset=20
272 access violation reading location: memory region 2
273 Time slot 22
274 CPU 1: Processed 4 has finished
275 CPU 1 stopped
276 write region=3 offset=20 value=103
277 access violation writing location: memory region 3
278 Memory Dump:
279 -----
280 BYTE 00000114: 100
281 BYTE 00000914: 102
282
283 CPU 3: Put process 8 to run queue
284 CPU 3: Dispatched process 8
285 Time slot 23
286 CPU 0: Put process 7 to run queue
287 CPU 0: Dispatched process 7
288 CPU 2: Processed 1 has finished
289 CPU 2 stopped
290 CPU 3: Processed 8 has finished
291 CPU 3 stopped
292 Time slot 24
293 Time slot 25
294 CPU 0: Put process 7 to run queue
295 CPU 0: Dispatched process 7
296 Time slot 26
297 CPU 0: Processed 7 has finished
298 CPU 0 stopped
```

4.2.2.c os_1_mlq_paging_small_1K

Listing 30: output of os_1_mlq_paging_small_1K

```
1 mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os
  os_1_mlq_paging_small_1K
2 Time slot 0
```



```
3 ld_routine
4 Time slot 1
5     Loaded a process at input/proc/p0s, PID: 1 PRI0: 130
6     CPU 1: Dispatched process 1
7 Time slot 2
8     Loaded a process at input/proc/s3, PID: 2 PRI0: 39
9 process 1 alloc region 0 size 300
10 print_pgtbl: 0 - 512
11 00000000: 80000001
12 00000004: 80000000
13 Page Number: 0 -> Frame Number: 1
14 Page Number: 1 -> Frame Number: 0
15 Memory Dump:
16 -----
17
18 Time slot 3
19     CPU 2: Dispatched process 2
20     CPU 1: Put process 1 to run queue
21     CPU 1: Dispatched process 1
22 process 1 alloc region 4 size 300
23 print_pgtbl: 0 - 1024
24 00000000: 80000001
25 00000004: 80000000
26 00000008: 80000003
27 00000012: 80000002
28 Page Number: 0 -> Frame Number: 1
29 Page Number: 1 -> Frame Number: 0
30 Page Number: 2 -> Frame Number: 3
31 Page Number: 3 -> Frame Number: 2
32 Memory Dump:
33 -----
34
35 Time slot 4
36 process 1 free region 0
37     Loaded a process at input/proc/m1s, PID: 3 PRI0: 15
38 Time slot 5
39     CPU 2: Put process 2 to run queue
40     CPU 2: Dispatched process 3
41 process 3 alloc region 0 size 300
42 print_pgtbl: 0 - 512
43 00000000: 80000005
44 00000004: 80000004
45 Page Number: 0 -> Frame Number: 5
46 Page Number: 1 -> Frame Number: 4
47 Memory Dump:
48 -----
49
50     CPU 0: Dispatched process 2
51     CPU 1: Put process 1 to run queue
52     CPU 1: Dispatched process 1
53 process 1 alloc region 1 size 100
54 print_pgtbl: 0 - 1024
55 00000000: 80000001
```




```
56 00000004: 80000000
57 00000008: 80000003
58 00000012: 80000002
59 Page Number: 0 -> Frame Number: 1
60 Page Number: 1 -> Frame Number: 0
61 Page Number: 2 -> Frame Number: 3
62 Page Number: 3 -> Frame Number: 2
63 Memory Dump:
64 -----
65
66 Time slot 6
67 process 3 alloc region 1 size 100
68 print_pttbl: 0 - 768
69 00000000: 80000005
70 00000004: 80000004
71 00000008: 80000006
72 Page Number: 0 -> Frame Number: 5
73 Page Number: 1 -> Frame Number: 4
74 Page Number: 2 -> Frame Number: 6
75 Memory Dump:
76 -----
77
78 write region=1 offset=20 value=100
79 print_pttbl: 0 - 1024
80 00000000: 80000001
81 00000004: 80000000
82 00000008: 80000003
83 00000012: 80000002
84 Page Number: 0 -> Frame Number: 1
85 Page Number: 1 -> Frame Number: 0
86 Page Number: 2 -> Frame Number: 3
87 Page Number: 3 -> Frame Number: 2
88 Memory Dump:
89 -----
90 BYTE 00000114: 100
91
92     Loaded a process at input/proc/s2, PID: 4 PRI0: 120
93 Time slot 7
94     CPU 3: Dispatched process 4
95     CPU 2: Put process 3 to run queue
96     CPU 2: Dispatched process 3
97 process 3 free region 0
98     CPU 0: Put process 2 to run queue
99     CPU 0: Dispatched process 2
100    CPU 1: Put process 1 to run queue
101    CPU 1: Dispatched process 1
102 read region=1 offset=20 value=100
103 print_pttbl: 0 - 1024
104 00000000: 80000001
105 00000004: 80000000
106 00000008: 80000003
107 00000012: 80000002
108 Page Number: 0 -> Frame Number: 1
```



```
109 Page Number: 1 -> Frame Number: 0
110 Page Number: 2 -> Frame Number: 3
111 Page Number: 3 -> Frame Number: 2
112 Memory Dump:
113 -----
114 BYTE 00000114: 100
115
116     Loaded a process at input/proc/m0s, PID: 5 PRI0: 120
117 Time slot 8
118 process 3 alloc region 2 size 100
119 print_pgtbl: 0 - 768
120 00000000: 80000005
121 00000004: 80000004
122 00000008: 80000006
123 Page Number: 0 -> Frame Number: 5
124 Page Number: 1 -> Frame Number: 4
125 Page Number: 2 -> Frame Number: 6
126 Memory Dump:
127 -----
128 BYTE 00000114: 100
129
130 write region=2 offset=20 value=102
131 access violation writing location: memory region 2
132 Memory Dump:
133 -----
134 BYTE 00000114: 100
135
136 Time slot 9
137     CPU 3: Put process 4 to run queue
138     CPU 3: Dispatched process 5
139 process 5 alloc region 0 size 300
140 print_pgtbl: 0 - 512
141 00000000: 80000001
142 00000004: 80000007
143 Page Number: 0 -> Frame Number: 1
144 Page Number: 1 -> Frame Number: 7
145 Memory Dump:
146 -----
147 BYTE 00000114: 100
148
149     CPU 2: Put process 3 to run queue
150     CPU 2: Dispatched process 3
151 process 3 free region 2
152     CPU 0: Put process 2 to run queue
153     CPU 0: Dispatched process 2
154     CPU 1: Put process 1 to run queue
155     CPU 1: Dispatched process 4
156     Loaded a process at input/proc/p1s, PID: 6 PRI0: 15
157 Time slot 10
158 process 5 alloc region 1 size 100
159 print_pgtbl: 0 - 768
160 00000000: 80000001
161 00000004: 80000007
```



```
162 00000008: 80000000
163 Page Number: 0 -> Frame Number: 1
164 Page Number: 1 -> Frame Number: 7
165 Page Number: 2 -> Frame Number: 0
166 Memory Dump:
167 -----
168 BYTE 00000114: 100
169
170 process 3 free region 1
171 Time slot 11
172     CPU 3: Put process 5 to run queue
173     CPU 3: Dispatched process 6
174     CPU 2: Put process 3 to run queue
175     CPU 2: Dispatched process 3
176 Unable to delocated memory region 1
177 process 3 free region 1
178     CPU 0: Put process 2 to run queue
179     CPU 0: Dispatched process 2
180     CPU 1: Put process 4 to run queue
181     CPU 1: Dispatched process 5
182 process 5 free region 0
183     Loaded a process at input/proc/s0, PID: 7 PRI0: 38
184 Time slot 12
185 Unable to delocated memory region 1
186 process 3 free region 1
187 process 5 alloc region 2 size 100
188 print_pgtbl: 0 - 768
189 00000000: 80000001
190 00000004: 80000007
191 00000008: 80000000
192 Page Number: 0 -> Frame Number: 1
193 Page Number: 1 -> Frame Number: 7
194 Page Number: 2 -> Frame Number: 0
195 Memory Dump:
196 -----
197 BYTE 00000114: 100
198
199 Time slot 13
200     CPU 3: Put process 6 to run queue
201     CPU 3: Dispatched process 6
202     CPU 2: Processed 3 has finished
203     CPU 2: Dispatched process 7
204     CPU 0: Put process 2 to run queue
205     CPU 0: Dispatched process 2
206     CPU 1: Put process 5 to run queue
207     CPU 1: Dispatched process 4
208 Time slot 14
209     CPU 0: Processed 2 has finished
210     CPU 0: Dispatched process 5
211 write region=1 offset=20 value=102
212 print_pgtbl: 0 - 768
213 00000000: 80000001
214 00000004: 80000007
```



```
215 00000008: 80000000
216 Page Number: 0 -> Frame Number: 1
217 Page Number: 1 -> Frame Number: 7
218 Page Number: 2 -> Frame Number: 0
219 Memory Dump:
220 -----
221 BYTE 00000014: 102
222 BYTE 00000114: 100
223
224 Time slot 15
225     CPU 3: Put process 6 to run queue
226     CPU 1: Put process 4 to run queue
227     CPU 1: Dispatched process 4
228     CPU 3: Dispatched process 6
229     CPU 2: Put process 7 to run queue
230     CPU 2: Dispatched process 7
231 write region=2 offset=1000 value=1
232 access violation writing location: memory region 2
233 Time slot 16
234     CPU 0: Put process 5 to run queue
235     CPU 0: Dispatched process 5
236 write region=2 offset=1000 value=1
237 access violation writing location: memory region 2
238     Loaded a process at input/proc/s1, PID: 8 PRIO: 0
239 Time slot 17
240     CPU 2: Put process 7 to run queue
241     CPU 2: Dispatched process 8
242     CPU 3: Put process 6 to run queue
243     CPU 3: Dispatched process 6
244     CPU 1: Put process 4 to run queue
245     CPU 1: Dispatched process 7
246     CPU 0: Processed 5 has finished
247     CPU 0: Dispatched process 4
248 Time slot 18
249 Time slot 19
250     CPU 2: Put process 8 to run queue
251     CPU 2: Dispatched process 8
252     CPU 0: Put process 4 to run queue
253     CPU 0: Dispatched process 4
254     CPU 1: Put process 7 to run queue
255     CPU 1: Dispatched process 7
256     CPU 3: Put process 6 to run queue
257     CPU 3: Dispatched process 6
258 Time slot 20
259 Time slot 21
260     CPU 0: Processed 4 has finished
261     CPU 0: Dispatched process 1
262 read region=2 offset=20
263 access violation reading location: memory region 2
264     CPU 1: Put process 7 to run queue
265     CPU 1: Dispatched process 7
266     CPU 3: Processed 6 has finished
267     CPU 3 stopped
```



```
268 CPU 2: Put process 8 to run queue
269 CPU 2: Dispatched process 8
270 Time slot 22
271 write region=3 offset=20 value=103
272 access violation writing location: memory region 3
273 Memory Dump:
274 -----
275 BYTE 00000014: 102
276 BYTE 00000114: 100
277
278 Time slot 23
279 CPU 0: Processed 1 has finished
280 CPU 0 stopped
281 CPU 1: Put process 7 to run queue
282 CPU 1: Dispatched process 7
283 CPU 2: Put process 8 to run queue
284 CPU 2: Dispatched process 8
285 Time slot 24
286 CPU 2: Processed 8 has finished
287 CPU 2 stopped
288 Time slot 25
289 CPU 1: Put process 7 to run queue
290 CPU 1: Dispatched process 7
291 Time slot 26
292 Time slot 27
293 CPU 1: Put process 7 to run queue
294 CPU 1: Dispatched process 7
295 Time slot 28
296 CPU 1: Processed 7 has finished
297 CPU 1 stopped
```

4.2.2.d os_1_mlq_paging_small_4K

Listing 31: output of os_1_mlq_paging_small_4K

```
1 mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os
  os_1_mlq_paging_small_4K
2 Time slot 0
3 ld_routine
4 Time slot 1
5     Loaded a process at input/proc/p0s, PID: 1 PRI0: 130
6 Time slot 2
7     CPU 2: Dispatched process 1
8     Loaded a process at input/proc/s3, PID: 2 PRI0: 39
9 Time slot 3
10 process 1 alloc region 0 size 300
11 print_pgtbl: 0 - 512
12 00000000: 80000001
13 00000004: 80000000
14 Page Number: 0 -> Frame Number: 1
15 Page Number: 1 -> Frame Number: 0
```



```
16 Memory Dump:
17 -----
18
19         CPU 3: Dispatched process 2
20 Time slot 4
21         CPU 2: Put process 1 to run queue
22         CPU 2: Dispatched process 1
23 process 1 alloc region 4 size 300
24 print_pgtbl: 0 - 1024
25 00000000: 80000001
26 00000004: 80000000
27 00000008: 80000003
28 00000012: 80000002
29 Page Number: 0 -> Frame Number: 1
30 Page Number: 1 -> Frame Number: 0
31 Page Number: 2 -> Frame Number: 3
32 Page Number: 3 -> Frame Number: 2
33 Memory Dump:
34 -----
35
36         Loaded a process at input/proc/m1s, PID: 3 PRI0: 15
37 Time slot 5
38 process 1 free region 0
39         CPU 0: Dispatched process 3
40 process 3 alloc region 0 size 300
41 print_pgtbl: 0 - 512
42 00000000: 80000005
43 00000004: 80000004
44 Page Number: 0 -> Frame Number: 5
45 Page Number: 1 -> Frame Number: 4
46 Memory Dump:
47 -----
48
49         CPU 3: Put process 2 to run queue
50         CPU 3: Dispatched process 2
51 Time slot 6
52         CPU 2: Put process 1 to run queue
53         CPU 2: Dispatched process 1
54 process 1 alloc region 1 size 100
55 print_pgtbl: 0 - 1024
56 00000000: 80000001
57 00000004: 80000000
58 00000008: 80000003
59 00000012: 80000002
60 Page Number: 0 -> Frame Number: 1
61 Page Number: 1 -> Frame Number: 0
62 Page Number: 2 -> Frame Number: 3
63 Page Number: 3 -> Frame Number: 2
64 Memory Dump:
65 -----
66
67 process 3 alloc region 1 size 100
68 print_pgtbl: 0 - 768
```



```
69 00000000: 80000005
70 00000004: 80000004
71 00000008: 80000006
72 Page Number: 0 -> Frame Number: 5
73 Page Number: 1 -> Frame Number: 4
74 Page Number: 2 -> Frame Number: 6
75 Memory Dump:
76 -----
77
78     Loaded a process at input/proc/s2, PID: 4 PRI0: 120
79 Time slot 7
80 write region=1 offset=20 value=100
81 print_pgtbl: 0 - 1024
82 00000000: 80000001
83 00000004: 80000000
84 00000008: 80000003
85 00000012: 80000002
86 Page Number: 0 -> Frame Number: 1
87 Page Number: 1 -> Frame Number: 0
88 Page Number: 2 -> Frame Number: 3
89 Page Number: 3 -> Frame Number: 2
90 Memory Dump:
91 -----
92 BYTE 00000114: 100
93
94     CPU 0: Put process 3 to run queue
95     CPU 0: Dispatched process 3
96 process 3 free region 0
97     CPU 1: Dispatched process 4
98     Loaded a process at input/proc/m0s, PID: 5 PRI0: 120
99     CPU 3: Put process 2 to run queue
100    CPU 3: Dispatched process 2
101 Time slot 8
102     CPU 2: Put process 1 to run queue
103     CPU 2: Dispatched process 5
104 process 5 alloc region 0 size 300
105 print_pgtbl: 0 - 512
106 00000000: 80000008
107 00000004: 80000007
108 Page Number: 0 -> Frame Number: 8
109 Page Number: 1 -> Frame Number: 7
110 Memory Dump:
111 -----
112 BYTE 00000114: 100
113
114 process 3 alloc region 2 size 100
115 print_pgtbl: 0 - 768
116 00000000: 80000005
117 00000004: 80000004
118 00000008: 80000006
119 Page Number: 0 -> Frame Number: 5
120 Page Number: 1 -> Frame Number: 4
121 Page Number: 2 -> Frame Number: 6
```



```
122 Memory Dump:
123 -----
124 BYTE 00000114: 100
125
126 Time slot 9
127 process 5 alloc region 1 size 100
128 print_pgtbl: 0 - 768
129 00000000: 80000008
130 00000004: 80000007
131 00000008: 80000009
132 Page Number: 0 -> Frame Number: 8
133 Page Number: 1 -> Frame Number: 7
134 Page Number: 2 -> Frame Number: 9
135 Memory Dump:
136 -----
137 BYTE 00000114: 100
138
139 CPU 0: Put process 3 to run queue
140 CPU 0: Dispatched process 3
141 process 3 free region 2
142 CPU 1: Put process 4 to run queue
143 CPU 1: Dispatched process 4
144 CPU 3: Put process 2 to run queue
145 CPU 3: Dispatched process 2
146 Loaded a process at input/proc/p1s, PID: 6 PRI0: 15
147 Time slot 10
148 CPU 2: Put process 5 to run queue
149 CPU 2: Dispatched process 6
150 process 3 free region 1
151 Time slot 11
152 CPU 0: Put process 3 to run queue
153 CPU 0: Dispatched process 3
154 Unable to delocated memory region 1
155 process 3 free region 1
156 CPU 1: Put process 4 to run queue
157 CPU 1: Dispatched process 5
158 process 5 free region 0
159 CPU 3: Put process 2 to run queue
160 CPU 3: Dispatched process 2
161 Loaded a process at input/proc/s0, PID: 7 PRI0: 38
162 Time slot 12
163 CPU 2: Put process 6 to run queue
164 CPU 2: Dispatched process 6
165 Unable to delocated memory region 1
166 process 3 free region 1
167 process 5 alloc region 2 size 100
168 print_pgtbl: 0 - 768
169 00000000: 80000008
170 00000004: 80000007
171 00000008: 80000009
172 Page Number: 0 -> Frame Number: 8
173 Page Number: 1 -> Frame Number: 7
174 Page Number: 2 -> Frame Number: 9
```




```
175 Memory Dump:
176 -----
177 BYTE 00000114: 100
178
179 Time slot 13
180     CPU 0: Processed 3 has finished
181     CPU 0: Dispatched process 7
182     CPU 3: Put process 2 to run queue
183     CPU 3: Dispatched process 2
184     CPU 1: Put process 5 to run queue
185     CPU 1: Dispatched process 4
186 Time slot 14
187     CPU 2: Put process 6 to run queue
188     CPU 2: Dispatched process 6
189     CPU 3: Processed 2 has finished
190     CPU 3: Dispatched process 5
191 write region=1 offset=20 value=102
192 print_ptgbl: 0 - 768
193 00000000: 80000008
194 00000004: 80000007
195 00000008: 80000009
196 Page Number: 0 -> Frame Number: 8
197 Page Number: 1 -> Frame Number: 7
198 Page Number: 2 -> Frame Number: 9
199 Memory Dump:
200 -----
201 BYTE 00000114: 100
202 BYTE 00000914: 102
203
204 Time slot 15
205     CPU 0: Put process 7 to run queue
206     CPU 0: Dispatched process 7
207 write region=2 offset=1000 value=1
208 access violation writing location: memory region 2
209     CPU 1: Put process 4 to run queue
210     CPU 1: Dispatched process 4
211 Time slot 16
212     CPU 2: Put process 6 to run queue
213     CPU 2: Dispatched process 6
214     CPU 3: Put process 5 to run queue
215     CPU 3: Dispatched process 5
216 write region=2 offset=1000 value=1
217 access violation writing location: memory region 2
218     Loaded a process at input/proc/s1, PID: 8 PRI0: 0
219 Time slot 17
220     CPU 0: Put process 7 to run queue
221     CPU 0: Dispatched process 8
222     CPU 3: Processed 5 has finished
223     CPU 3: Dispatched process 7
224     CPU 1: Put process 4 to run queue
225     CPU 1: Dispatched process 4
226 Time slot 18
227     CPU 2: Put process 6 to run queue
```



```
228         CPU 2: Dispatched process 6
229 Time slot 19
230         CPU 0: Put process 8 to run queue
231         CPU 0: Dispatched process 8
232         CPU 3: Put process 7 to run queue
233         CPU 3: Dispatched process 7
234         CPU 1: Put process 4 to run queue
235         CPU 1: Dispatched process 4
236 Time slot 20
237         CPU 2: Processed 6 has finished
238         CPU 2: Dispatched process 1
239 read region=1 offset=20 value=100
240 print_pgtbl: 0 - 1024
241 00000000: 80000001
242 00000004: 80000000
243 00000008: 80000003
244 00000012: 80000002
245 Page Number: 0 -> Frame Number: 1
246 Page Number: 1 -> Frame Number: 0
247 Page Number: 2 -> Frame Number: 3
248 Page Number: 3 -> Frame Number: 2
249 Memory Dump:
250 -----
251 BYTE 00000114: 100
252 BYTE 00000914: 102
253
254 Time slot 21
255 write region=2 offset=20 value=102
256 access violation writing location: memory region 2
257 Memory Dump:
258 -----
259 BYTE 00000114: 100
260 BYTE 00000914: 102
261
262         CPU 0: Put process 8 to run queue
263         CPU 0: Dispatched process 8
264         CPU 3: Put process 7 to run queue
265         CPU 3: Dispatched process 7
266         CPU 1: Processed 4 has finished
267         CPU 1 stopped
268 Time slot 22
269         CPU 2: Put process 1 to run queue
270         CPU 2: Dispatched process 1
271 read region=2 offset=20
272 access violation reading location: memory region 2
273 Time slot 23
274 write region=3 offset=20 value=103
275 access violation writing location: memory region 3
276 Memory Dump:
277 -----
278 BYTE 00000114: 100
279 BYTE 00000914: 102
280
```



```
281 CPU 0: Put process 8 to run queue
282 CPU 0: Dispatched process 8
283 CPU 3: Put process 7 to run queue
284 CPU 3: Dispatched process 7
285 Time slot 24
286 CPU 2: Processed 1 has finished
287 CPU 2 stopped
288 CPU 0: Processed 8 has finished
289 CPU 0 stopped
290 Time slot 25
291 CPU 3: Put process 7 to run queue
292 CPU 3: Dispatched process 7
293 Time slot 26
294 Time slot 27
295 CPU 3: Put process 7 to run queue
296 CPU 3: Dispatched process 7
297 Time slot 28
298 CPU 3: Processed 7 has finished
299 CPU 3 stopped
```

4.2.2.e os_1_mlq_paging_small_4K

Listing 32: output of os_1_mlq_paging_small_4K

```
1 mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os
  os_1_mlq_paging_small_4K
2 Time slot 0
3 ld_routine
4 Time slot 1
5     Loaded a process at input/proc/p0s, PID: 1 PRI0: 130
6 Time slot 2
7     CPU 2: Dispatched process 1
8     Loaded a process at input/proc/s3, PID: 2 PRI0: 39
9 Time slot 3
10 process 1 alloc region 0 size 300
11 print_pgtbl: 0 - 512
12 00000000: 80000001
13 00000004: 80000000
14 Page Number: 0 -> Frame Number: 1
15 Page Number: 1 -> Frame Number: 0
16 Memory Dump:
17 -----
18
19     CPU 3: Dispatched process 2
20 Time slot 4
21     CPU 2: Put process 1 to run queue
22     CPU 2: Dispatched process 1
23 process 1 alloc region 4 size 300
24 print_pgtbl: 0 - 1024
25 00000000: 80000001
26 00000004: 80000000
```



```
27 00000008: 80000003
28 00000012: 80000002
29 Page Number: 0 -> Frame Number: 1
30 Page Number: 1 -> Frame Number: 0
31 Page Number: 2 -> Frame Number: 3
32 Page Number: 3 -> Frame Number: 2
33 Memory Dump:
34 -----
35
36     Loaded a process at input/proc/m1s, PID: 3 PRI0: 15
37 Time slot 5
38 process 1 free region 0
39     CPU 0: Dispatched process 3
40 process 3 alloc region 0 size 300
41 print_pgtbl: 0 - 512
42 00000000: 80000005
43 00000004: 80000004
44 Page Number: 0 -> Frame Number: 5
45 Page Number: 1 -> Frame Number: 4
46 Memory Dump:
47 -----
48
49     CPU 3: Put process 2 to run queue
50     CPU 3: Dispatched process 2
51 Time slot 6
52     CPU 2: Put process 1 to run queue
53     CPU 2: Dispatched process 1
54 process 1 alloc region 1 size 100
55 print_pgtbl: 0 - 1024
56 00000000: 80000001
57 00000004: 80000000
58 00000008: 80000003
59 00000012: 80000002
60 Page Number: 0 -> Frame Number: 1
61 Page Number: 1 -> Frame Number: 0
62 Page Number: 2 -> Frame Number: 3
63 Page Number: 3 -> Frame Number: 2
64 Memory Dump:
65 -----
66
67 process 3 alloc region 1 size 100
68 print_pgtbl: 0 - 768
69 00000000: 80000005
70 00000004: 80000004
71 00000008: 80000006
72 Page Number: 0 -> Frame Number: 5
73 Page Number: 1 -> Frame Number: 4
74 Page Number: 2 -> Frame Number: 6
75 Memory Dump:
76 -----
77
78     Loaded a process at input/proc/s2, PID: 4 PRI0: 120
79 Time slot 7
```



```
80 write region=1 offset=20 value=100
81 print_pgtbl: 0 - 1024
82 00000000: 80000001
83 00000004: 80000000
84 00000008: 80000003
85 00000012: 80000002
86 Page Number: 0 -> Frame Number: 1
87 Page Number: 1 -> Frame Number: 0
88 Page Number: 2 -> Frame Number: 3
89 Page Number: 3 -> Frame Number: 2
90 Memory Dump:
91 -----
92 BYTE 00000114: 100
93
94     CPU 0: Put process 3 to run queue
95     CPU 0: Dispatched process 3
96 process 3 free region 0
97     CPU 1: Dispatched process 4
98     Loaded a process at input/proc/m0s, PID: 5 PRI0: 120
99     CPU 3: Put process 2 to run queue
100    CPU 3: Dispatched process 2
101 Time slot 8
102     CPU 2: Put process 1 to run queue
103     CPU 2: Dispatched process 5
104 process 5 alloc region 0 size 300
105 print_pgtbl: 0 - 512
106 00000000: 80000008
107 00000004: 80000007
108 Page Number: 0 -> Frame Number: 8
109 Page Number: 1 -> Frame Number: 7
110 Memory Dump:
111 -----
112 BYTE 00000114: 100
113
114 process 3 alloc region 2 size 100
115 print_pgtbl: 0 - 768
116 00000000: 80000005
117 00000004: 80000004
118 00000008: 80000006
119 Page Number: 0 -> Frame Number: 5
120 Page Number: 1 -> Frame Number: 4
121 Page Number: 2 -> Frame Number: 6
122 Memory Dump:
123 -----
124 BYTE 00000114: 100
125
126 Time slot 9
127 process 5 alloc region 1 size 100
128 print_pgtbl: 0 - 768
129 00000000: 80000008
130 00000004: 80000007
131 00000008: 80000009
132 Page Number: 0 -> Frame Number: 8
```



```
133 Page Number: 1 -> Frame Number: 7
134 Page Number: 2 -> Frame Number: 9
135 Memory Dump:
136 -----
137 BYTE 00000114: 100
138
139     CPU 0: Put process 3 to run queue
140     CPU 0: Dispatched process 3
141 process 3 free region 2
142     CPU 1: Put process 4 to run queue
143     CPU 1: Dispatched process 4
144     CPU 3: Put process 2 to run queue
145     CPU 3: Dispatched process 2
146     Loaded a process at input/proc/pls, PID: 6 PRI0: 15
147 Time slot 10
148     CPU 2: Put process 5 to run queue
149     CPU 2: Dispatched process 6
150 process 3 free region 1
151 Time slot 11
152     CPU 0: Put process 3 to run queue
153     CPU 0: Dispatched process 3
154 Unable to delocated memory region 1
155 process 3 free region 1
156     CPU 1: Put process 4 to run queue
157     CPU 1: Dispatched process 5
158 process 5 free region 0
159     CPU 3: Put process 2 to run queue
160     CPU 3: Dispatched process 2
161     Loaded a process at input/proc/s0, PID: 7 PRI0: 38
162 Time slot 12
163     CPU 2: Put process 6 to run queue
164     CPU 2: Dispatched process 6
165 Unable to delocated memory region 1
166 process 3 free region 1
167 process 5 alloc region 2 size 100
168 print_pgtbl: 0 - 768
169 00000000: 80000008
170 00000004: 80000007
171 00000008: 80000009
172 Page Number: 0 -> Frame Number: 8
173 Page Number: 1 -> Frame Number: 7
174 Page Number: 2 -> Frame Number: 9
175 Memory Dump:
176 -----
177 BYTE 00000114: 100
178
179 Time slot 13
180     CPU 0: Processed 3 has finished
181     CPU 0: Dispatched process 7
182     CPU 3: Put process 2 to run queue
183     CPU 3: Dispatched process 2
184     CPU 1: Put process 5 to run queue
185     CPU 1: Dispatched process 4
```



```
186 Time slot 14
187     CPU 2: Put process 6 to run queue
188     CPU 2: Dispatched process 6
189     CPU 3: Processed 2 has finished
190     CPU 3: Dispatched process 5
191 write region=1 offset=20 value=102
192 print_pgtbl: 0 - 768
193 00000000: 80000008
194 00000004: 80000007
195 00000008: 80000009
196 Page Number: 0 -> Frame Number: 8
197 Page Number: 1 -> Frame Number: 7
198 Page Number: 2 -> Frame Number: 9
199 Memory Dump:
200 -----
201 BYTE 00000114: 100
202 BYTE 00000914: 102
203
204 Time slot 15
205     CPU 0: Put process 7 to run queue
206     CPU 0: Dispatched process 7
207 write region=2 offset=1000 value=1
208 access violation writing location: memory region 2
209     CPU 1: Put process 4 to run queue
210     CPU 1: Dispatched process 4
211 Time slot 16
212     CPU 2: Put process 6 to run queue
213     CPU 2: Dispatched process 6
214     CPU 3: Put process 5 to run queue
215     CPU 3: Dispatched process 5
216 write region=2 offset=1000 value=1
217 access violation writing location: memory region 2
218     Loaded a process at input/proc/s1, PID: 8 PRI0: 0
219 Time slot 17
220     CPU 0: Put process 7 to run queue
221     CPU 0: Dispatched process 8
222     CPU 3: Processed 5 has finished
223     CPU 3: Dispatched process 7
224     CPU 1: Put process 4 to run queue
225     CPU 1: Dispatched process 4
226 Time slot 18
227     CPU 2: Put process 6 to run queue
228     CPU 2: Dispatched process 6
229 Time slot 19
230     CPU 0: Put process 8 to run queue
231     CPU 0: Dispatched process 8
232     CPU 3: Put process 7 to run queue
233     CPU 3: Dispatched process 7
234     CPU 1: Put process 4 to run queue
235     CPU 1: Dispatched process 4
236 Time slot 20
237     CPU 2: Processed 6 has finished
238     CPU 2: Dispatched process 1
```



```
239 read region=1 offset=20 value=100
240 print_pgtbl: 0 - 1024
241 00000000: 80000001
242 00000004: 80000000
243 00000008: 80000003
244 00000012: 80000002
245 Page Number: 0 -> Frame Number: 1
246 Page Number: 1 -> Frame Number: 0
247 Page Number: 2 -> Frame Number: 3
248 Page Number: 3 -> Frame Number: 2
249 Memory Dump:
250 -----
251 BYTE 00000114: 100
252 BYTE 00000914: 102
253
254 Time slot 21
255 write region=2 offset=20 value=102
256 access violation writing location: memory region 2
257 Memory Dump:
258 -----
259 BYTE 00000114: 100
260 BYTE 00000914: 102
261
262 CPU 0: Put process 8 to run queue
263 CPU 0: Dispatched process 8
264 CPU 3: Put process 7 to run queue
265 CPU 3: Dispatched process 7
266 CPU 1: Processed 4 has finished
267 CPU 1 stopped
268 Time slot 22
269 CPU 2: Put process 1 to run queue
270 CPU 2: Dispatched process 1
271 read region=2 offset=20
272 access violation reading location: memory region 2
273 Time slot 23
274 write region=3 offset=20 value=103
275 access violation writing location: memory region 3
276 Memory Dump:
277 -----
278 BYTE 00000114: 100
279 BYTE 00000914: 102
280
281 CPU 0: Put process 8 to run queue
282 CPU 0: Dispatched process 8
283 CPU 3: Put process 7 to run queue
284 CPU 3: Dispatched process 7
285 Time slot 24
286 CPU 2: Processed 1 has finished
287 CPU 2 stopped
288 CPU 0: Processed 8 has finished
289 CPU 0 stopped
290 Time slot 25
291 CPU 3: Put process 7 to run queue
```



```
292 CPU 3: Dispatched process 7
293 Time slot 26
294 Time slot 27
295 CPU 3: Put process 7 to run queue
296 CPU 3: Dispatched process 7
297 Time slot 28
298 CPU 3: Processed 7 has finished
299 CPU 3 stopped
```

Lưu ý: Vì tránh bài báo cáo quá dài nên output `os_1_singleCPU_mlq` và `os_1_singleCPU_mlq_paging` nhóm không trình bày trong báo cáo, giảng viên có thể yêu cầu nhóm trình bày khi thuyết trình hoặc chạy source code đã nộp.

4.2.3 Trả lời câu hỏi

Question: In this simple OS, we implement a design of multiple memory segments or memory areas in source code declaration. What is the advantage of the proposed design of multiple segments?

- Trong một hệ thống máy tính, bộ nhớ được sử dụng để lưu trữ dữ liệu và chương trình. Khi sử dụng bộ nhớ, cần đảm bảo rằng các dữ liệu và chương trình không bị xung đột với nhau hoặc không được truy cập một cách trái phép. Do đó, hầu hết các hệ điều hành đều sử dụng phân đoạn bộ nhớ để giải quyết vấn đề này.
- Trong thiết kế về nhiều phân đoạn bộ nhớ, bộ nhớ được chia thành nhiều khu vực khác nhau và mỗi khu vực có thể chứa các dữ liệu hoặc chương trình khác nhau. Điều này giúp giảm thiểu xung đột giữa các dữ liệu và chương trình. Mỗi phân đoạn được quản lý một cách riêng biệt, đảm bảo rằng một chương trình không thể truy cập vào hoặc thay đổi dữ liệu trong các phân đoạn khác.
- Ví dụ, trong một hệ thống máy tính, có thể có một phân đoạn dành cho bộ nhớ đệm, một phân đoạn dành cho bộ nhớ định dạng và một phân đoạn dành cho bộ nhớ thực thi chương trình. Bằng cách phân chia bộ nhớ thành các phân đoạn khác nhau, ta có thể giữ cho các dữ liệu và chương trình của hệ thống hoạt động một cách an toàn và hiệu quả.
- Ngoài ra, việc sử dụng nhiều phân đoạn bộ nhớ cũng giúp tối ưu hóa việc sử dụng bộ nhớ và tăng hiệu suất của hệ thống. Bởi vì mỗi phân đoạn được sử dụng cho một mục đích cụ thể, ta có thể cấp phát và giải phóng bộ nhớ một cách nhanh chóng và hiệu quả. Kết quả là hệ thống hoạt động nhanh hơn và sử dụng bộ nhớ hiệu quả hơn.

Question: What will happen if we divide the address to more than 2-levels in the paging memory management system?

- Trong hệ thống quản lý bộ nhớ phân trang, mỗi địa chỉ bộ nhớ được chia thành hai phần: phần offset trang (page offset) và phần số trang (page number), để xác định vị trí của trang trong bộ nhớ vật lý. Tuy nhiên, nếu ta chia địa chỉ thành nhiều cấp độ hơn hai, tức là sử dụng phân trang nhiều cấp, thì hệ thống sẽ có thể quản lý bộ nhớ với độ chính xác và hiệu quả hơn.
- Ví dụ, nếu chia địa chỉ thành 3 cấp độ, ta có thể sử dụng 3 phần để xác định vị trí của một trang trong bộ nhớ vật lý, ví dụ như số trang (page number), số khối (block number) và offset trang (page offset). Việc này giúp chia bộ nhớ thành các khối nhỏ hơn, cung cấp khả năng quản lý bộ nhớ linh hoạt hơn và giúp hệ thống hoạt động hiệu quả hơn.

- Tuy nhiên, việc sử dụng phân trang nhiều cấp cũng có thể làm tăng độ phức tạp của hệ thống, gây tốn chi phí tính toán và tài nguyên bộ nhớ. Khi chia địa chỉ thành nhiều cấp độ, ta cần sử dụng nhiều bộ nhớ đệm để lưu trữ thông tin về cấu trúc phân trang, gây tốn tài nguyên bộ nhớ và làm cho hệ thống trở nên chậm hơn.
- Ngoài ra, việc sử dụng nhiều cấp độ cũng tạo ra nhiều khối nhỏ hơn, khiến cho bộ nhớ thực tế cần sử dụng nhiều trang hơn để lưu trữ dữ liệu, dẫn đến tăng khối lượng dữ liệu trao đổi giữa bộ nhớ ảo và bộ nhớ vật lý, làm cho hệ thống trở nên chậm hơn.
- Do đó, việc sử dụng nhiều cấp độ trong phân trang bộ nhớ là một lựa chọn tốt để cải thiện hiệu suất của hệ thống, nhưng cần cân nhắc kỹ lưỡng để tránh làm tăng độ phức tạp của hệ thống và tiêu tốn tài nguyên bộ nhớ.

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

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 bằng cách phân trang trong mỗi đoạn.

Hạn chế của giải thuật:

- Phân mảnh nội của giải thuật phân trang vẫn còn.
- Độ phức tạp của giải thuật cao. Bảng phân trang cần phải được lưu trữ liên tục bên trong bộ nhớ.

4.3 Synchronization

4.3.1 Hoàn thiện code

Nhóm sử dụng phương pháp mutex log khi thao tác trên các vùng nhớ chung để tránh xung đột dữ liệu khi chạy song song các luồng. Cụ thể có thể quan sát các đoạn code ở trên.

Listing 33: Synchronization for scheduler

```
1 // synchronized for sched
2 static pthread_mutex_t queue_lock;
```

Listing 34: Synchronization for virtual memory

```
1 // synchronized for vm
2 static pthread_mutex_t vm_lock = PTHREAD_MUTEX_INITIALIZER;
```

Listing 35: Example of synchronization

```
1 void add_mlq_proc(struct pcb_t *proc)
2 {
3     pthread_mutex_lock(&queue_lock);
4     enqueue(&mlq_ready_queue[proc->prio], proc);
5     pthread_mutex_unlock(&queue_lock);
6 }
```

4.3.2 Trả lời câu hỏi

Question: What will happen if the synchronization is not handled in your simple OS? Illustrate by example the problem of your simple OS if you have any.

- Khi không xử lý đồng bộ hóa đúng cách trong hệ thống, nhiều nguồn có thể cùng truy cập vào một biến toàn cục và thực hiện thay đổi đồng thời mà không tuân theo một thứ tự nhất định. Điều này có thể dẫn đến các vấn đề nghiêm trọng như xung đột dữ liệu, khiến cho dữ liệu không được tính toán chính xác và các giao dịch cũng như ủy quyền dữ liệu có thể bị trì hoãn. Nếu điều này xảy ra, sẽ ảnh hưởng đến hiệu quả của các biện pháp kiểm soát quản lý và có thể gây ra các vấn đề khác trong hệ thống. Do đó, đồng bộ hóa là một yếu tố rất quan trọng trong thiết kế và triển khai các hệ thống, và cần được xử lý đúng cách để tránh các vấn đề tiềm tàng.
- Ví dụ: Loại bỏ mutex log khiến mlq_ready_queue không còn cơ chế đồng bộ hoá trong simple os, chạy sched_0, kết quả không hợp lệ vì xảy ra xung đột dữ liệu tại mlq_ready_queue.

```
• mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os sched_0
Time slot 0
ld_routine
    Loaded a process at input/proc/s0, PID: 1 PRI0: 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 PRI0: 1
Time slot 5
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 6
Time slot 7
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 8
Time slot 9
    CPU 0: Put process 1 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 1
Time slot 12
Time slot 13
    CPU 0: Put process 1 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 1
Time slot 16
    CPU 0: Processed 1 has finished
    CPU 0: Dispatched process 2
Time slot 17
Time slot 18
    CPU 0: Put process 2 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 2
Time slot 21
Time slot 22
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot 23
    CPU 0: Processed 2 has finished
    CPU 0 stopped
```

Hình 6: Kết quả chạy testcase sched_0 khi không có muxtex log