

TRƯỜNG ĐẠI HỌC BÁCH KHOA - ĐHQG TPHCM  
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



BÁO CÁO BÀI TẬP LỚN  
MÔN: HỆ ĐIỀU HÀNH (CO2017)

ĐỀ TÀI: SIMPLE OPERATING SYSTEM

NHÓM 7 - LỚP L10 - HỌC KỲ 232

GVHD: Th.s Trương Tuấn Phát

STT	Họ và tên	MSSV	Đóng góp
1	Lý Nguyên Khang	2211437	30%
2	Nguyễn Trường Giang	2210829	20%
3	Nguyễn Đình Bằng	2210298	20%
4	Đỗ Thị Thùy Vân	2213921	20%
5	Đoàn Công Hải	2210878	10%

Thành phố Hồ Chí Minh, tháng 5 năm 2024



## Mục lục

<b>1 Scheduler</b>	<b>4</b>
1.1 Tổng quan . . . . .	4
1.1.1 Khái niệm cơ bản . . . . .	4
1.1.2 Bộ quản lý quy trình của nhóm . . . . .	4
1.1.3 Thực thi theo giải thuật Round Robin . . . . .	4
1.1.4 Giải thuật Multi Level Queue (MLQ) . . . . .	4
1.2 Implementation . . . . .	5
1.2.1 enqueue() . . . . .	5
1.2.2 dequeue(): . . . . .	5
1.2.3 get_mlq_proc() . . . . .	6
1.3 Result . . . . .	7
1.3.1 sched . . . . .	7
1.3.2 sched_0 . . . . .	9
1.3.3 sched_1 . . . . .	10
1.4 Question . . . . .	12
<b>2 Memory Management</b>	<b>13</b>
2.1 Tổng quan . . . . .	13
2.1.1 Ý tưởng chung . . . . .	13
2.1.2 Paging . . . . .	13
2.2 Implementation . . . . .	14
2.2.1 MEMPHY_dump(struct memphy_struct * mp) . . . . .	14
2.2.2 int __alloc(struct pcb_t *caller, int vmaid, int rgid, int size, int *alloc_addr) . . . . .	14
2.2.3 int __free(struct pcb_t *caller, int vmaid, int rgid)) . . . . .	15
2.2.4 int pg_getpage(struct mm_struct *mm, int pgn, int *fpn, struct pcb_t *caller) . . . . .	16
2.2.5 int find_victim_page(struct mm_struct *mm, int *retpgn) . . . . .	18
2.2.6 int vmap_page_range(struct pcb_t *caller, int addr, int pgnum, struct framephy_struct *frames, struct vm_rg_struct *ret_rg)) . . . . .	18
2.2.7 int alloc_pages_range(struct pcb_t *caller, int req_pgnum, struct framephy_struct** frm_lst) . . . . .	19
2.2.8 int __read(struct pcb_t *caller, int vmaid, int rgid, int offset, BYTE *data) . . . . .	21
2.2.9 int __write(struct pcb_t *caller, int vmaid, int rgid, int offset, BYTE value) . . . . .	22
2.3 Result . . . . .	24
2.3.1 os_0_mlq_paging . . . . .	24
2.3.2 os_1_mlq_paging_small_1K . . . . .	26
2.3.3 os_1_mlq_paging_small_4K . . . . .	28
2.3.4 os_1_singleCPU_mlq_paging . . . . .	30
2.4 Question . . . . .	32
<b>3 TLB (Translation Lookaside Buffer)</b>	<b>34</b>
3.1 Tổng quan . . . . .	34
3.1.1 Khái niệm . . . . .	34
3.1.2 Chức năng . . . . .	34
3.1.3 Ưu và nhược điểm . . . . .	34
3.2 Implementation . . . . .	35
3.2.1 cpu-tlb.c . . . . .	35
3.2.2 cpu-tlbcache.c . . . . .	38
3.3 Result . . . . .	39



3.3.1	os_1_singleCPU_mlq . . . . .	39
3.3.2	os_1_tlbsz_singleCPU_mlq . . . . .	41
3.3.3	os_1_mlq_paging . . . . .	43
3.4	Question . . . . .	45
<b>4</b>	<b>Put All Together</b>	<b>46</b>
<b>5</b>	<b>Kết luận</b>	<b>49</b>
<b>6</b>	<b>Tài liệu tham khảo</b>	<b>50</b>



# 1 Scheduler

## 1.1 Tổng quan

### 1.1.1 Khái niệm cơ bản

Bộ quản lý quy trình (Scheduler) là một thành phần nền tảng của hệ điều hành, chịu trách nhiệm quản lý thực thi quy trình, phân bổ tài nguyên hệ thống và đảm bảo sử dụng CPU hiệu quả và công bằng. Nó quyết định quy trình nào được chạy dựa trên các yếu tố như mức ưu tiên, sẵn có tài nguyên và thuật toán lập lịch, tối ưu hóa hiệu suất hệ thống và duy trì trải nghiệm người dùng mượt mà.

### 1.1.2 Bộ quản lí quy trình của nhóm

Trong hệ điều hành đơn giản này, bộ quản lý sử dụng nhiều hàng đợi sẵn sàng (ready queue), mỗi hàng đợi gắn với một giá trị ưu tiên cố định, để xác định quy trình nào được thực thi khi CPU có sẵn. Khi một chương trình mới được nạp:

- Một quy trình mới được tạo và một Khối Điều Khiển Quy Trình (PCB) được gán cho nó.
- Nội dung chương trình được sao chép vào phân đoạn văn bản của quy trình, được chỉ định bởi con trỏ mã trong PCB.
- Sau đó, PCB được đặt vào hàng đợi sẵn sàng tương ứng với mức ưu tiên của quy trình.

### 1.1.3 Thực thi theo giải thuật Round Robin

CPU thực thi các quy trình theo kiểu vòng tròn, cho phép mỗi quy trình một khoảng thời gian nhất định để chạy (gọi là lát cắt thời gian - time slice). Khi lát cắt thời gian hết hạn, CPU xếp hàng lại quy trình đó vào hàng đợi sẵn sàng được liên kết và chọn quy trình tiếp theo để tiếp tục thực thi.

### 1.1.4 Giải thuật Multi Level Queue (MLQ)

Hệ thống tuân theo giải thuật MLQ (Multi Level Queue), trong đó số bước di chuyển trong danh sách hàng đợi sẵn sàng được xác định bởi một công thức cố định dựa trên mức ưu tiên, được gọi là "slot" (slot = MAX PRIO - prio). Mỗi hàng đợi có một số lượng slot cố định để sử dụng CPU và khi tất cả các slot đều được chiếm dụng, hệ thống sẽ chuyển sang hàng đợi tiếp theo, để lại công việc còn lại cho các slot trong tương lai, ngay cả khi nó yêu cầu hoàn thành toàn bộ vòng lặp của hàng đợi sẵn sàng. Thiết kế của bộ quản lý đơn giản hóa việc thêm một quy trình vào hàng đợi sẵn sàng phù hợp bằng cách sử dụng khớp mức ưu tiên. Giải thuật MLQ được thực hiện thông qua hoạt động "get proc", hoạt động này sẽ lấy một quy trình và phân phái nó cho CPU để thực thi.



## 1.2 Implementation

### 1.2.1 enqueue()

Trong file này, chúng ta cần triển khai các hàm sau: **enqueue()**: Thêm một tiến trình mới vào một hàng đợi **q**, sắp xếp hàng đợi theo thứ tự ưu tiên (tiến trình có mức ưu tiên cao hơn sẽ đứng trước).

```
1 void enqueue(struct queue_t * q, struct pcb_t * proc) {
2     /* TODO: put a new process to queue [q] */
3     if (q->size == MAX_QUEUE_SIZE)
4         return;
5     if (q->size == 0){
6         q->proc[0] = proc;
7     }
8     else{
9         int i;
10        for (i = q->size; i > 0 && proc->priority >= q->proc[i - 1]-
11            ↪ priority; i--){
12            q->proc[i] = q->proc[i - 1];
13        }
14        q->proc[i] = proc;
15    }
16    q->size++;
}
```

### 1.2.2 dequeue():

Được thiết kế để lấy một tiến trình (process) có ưu tiên cao nhất trong hàng đợi **q** và loại bỏ nó khỏi hàng đợi.

```
1 struct pcb_t * dequeue(struct queue_t * q) {
2     if (empty(q))
3         return NULL;
4     if(q->size>0){
5         struct pcb_t * temp = q->proc[q->size-1];
6         q->proc[q->size-1]=NULL;
7         q->size--;
8         return temp;
9     }
10    return NULL;
11 }
```



### 1.2.3 get\_mlq\_proc()

Được thiết kế để lấy một tiến trình (process) từ hàng đợi ưu tiên (priority queue) *mlq\_ready\_queue* trong MLQ (Multi-Level Queue).

```
1   struct pcb_t * get_mlq_proc(void) {
2       struct pcb_t * proc = NULL;
3       /*TODO: get a process from PRIORITY [ready_queue].
4        * Remember to use lock to protect the queue.
5        */
6       pthread_mutex_lock(&queue_lock);
7       for (INDEX = INDEX ; INDEX < MAX_PRIO; INDEX++){
8           if (!empty(&mlq_ready_queue[INDEX]))
9           {
10               proc = dequeue(&mlq_ready_queue[INDEX]);
11               break;
12           }
13       };
14       INDEX =0;
15       pthread_mutex_unlock(&queue_lock);
16       return proc;
17 }
```



### 1.3 Result

- ★ Preprocessor directives cần để chạy chương trình:

```
include > /.../os.cfg.h > mmu_fixing.h
    You, 1 second ago | 1 author (You)
1 #ifndef OS_CFG_H
2 #define OS_CFG_H
3
4 #define MLQ_SCHED 1
5 #define MAX_PRIO 140
6
7 // #define CPU_TLB
8 // #define CPUTLB_FIXED_TLBSZ
9 #define MM_PAGING      You, 2 days ago • Merge
10 #define MM_FIXED_MEMSZ
11 #define VMDBG 1
12 #define MMDBG 1
13 #define IODUMP 1
14 #define PAGETBL_DUMP 1
15
16 #endif
17
```

#### 1.3.1 sched

##### sched input

```
4 2 4
0 s1 4
1 s2 4
3 s3 4
4 s4 0
```

Output:

```

1 Time slot 0
2 ld_routine
3     Loaded a process at input/proc/s1, PID: 1 PRIO: 4
4     CPU 0: Dispatched process 1
5 Time slot 1
6     Loaded a process at input/proc/s2, PID: 2 PRIO: 4
7     CPU 1: Dispatched process 2
8 Time slot 2
9 Time slot 3
10    Loaded a process at input/proc/s3, PID: 3 PRIO: 4
11 Time slot 4
12    Loaded a process at input/proc/s4, PID: 4 PRIO: 0
13    CPU 0: Put process 1 to run queue
14    CPU 0: Dispatched process 4
15 Time slot 5
16    CPU 1: Put process 2 to run queue
17    CPU 1: Dispatched process 3
18 Time slot 6
19 Time slot 7
20 Time slot 8
21    CPU 0: Put process 4 to run queue
22    CPU 0: Dispatched process 4
23 Time slot 9
24    CPU 1: Put process 3 to run queue
25    CPU 1: Dispatched process 1
26 Time slot 10
27 Time slot 11
28 Time slot 12
29    CPU 1: Processed 1 has finished
30    CPU 1: Dispatched process 2
31    CPU 0: Put process 4 to run queue
32    CPU 0: Dispatched process 4
33 Time slot 13
34 Time slot 14
35 Time slot 15
36 Time slot 16
37    CPU 1: Put process 2 to run queue
38    CPU 1: Dispatched process 3
39    CPU 0: Put process 4 to run queue
40    CPU 0: Dispatched process 4
41 Time slot 17
42 Time slot 18
43 Time slot 19
44 Time slot 20
45    CPU 1: Put process 3 to run queue
46    CPU 1: Dispatched process 2
47    CPU 0: Put process 4 to run queue
48    CPU 0: Dispatched process 4
49 Time slot 21
50 Time slot 22
51 Time slot 23
52 Time slot 24
53    CPU 1: Processed 2 has finished
54    CPU 1: Dispatched process 3
55    CPU 0: Put process 4 to run queue
56    CPU 0: Dispatched process 4
57 Time slot 25
58 Time slot 26
59 Time slot 27
60    CPU 1: Processed 3 has finished
61    CPU 1 stopped
62 Time slot 28
63    CPU 0: Put process 4 to run queue
64    CPU 0: Dispatched process 4
65 Time slot 29
66 Time slot 30
67 Time slot 31
68 Time slot 32
69    CPU 0: Put process 4 to run queue
70    CPU 0: Dispatched process 4
71 Time slot 33
72    CPU 0: Processed 4 has finished
73    CPU 0 stopped

```

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	
CPU 0	s1				s4																			
CPU 1	s2				s3			s1		s2			s3			s2								
Time slot	23	24	25	26	27	28	29	30	31	32	33													
CPU 0	s4																							
CPU 1	s2	s3																						

### Giải thích

Do CPU 0 và CPU 1 chạy song song nên cách lấy vào là ngẫu nhiên, tùy thuộc vào thread nào get process () trước nên đôi khi là CPU 0 hoặc CPU 1 sẽ chạy trước.

Timeslot 0: Tiến trình đầu tiên được tải và thực thi trên CPU 0. Timeslot 1: Tiến trình thứ hai được tải và thực thi trên CPU 1. Timeslot 4: Tiến trình thứ tư được tải và thực thi trên CPU 0, trong khi tiến trình đầu tiên vẫn đang chạy. Timeslot 5: Tiến trình thứ ba được tải và thực thi trên CPU 1, trong khi tiến trình thứ hai vẫn đang chạy. Timeslot 9: Tiến trình thứ tư được thực thi xong trên CPU 0 và tiến trình thứ nhất được tiếp tục chạy. Timeslot 12: Tiến trình đầu tiên thực thi xong và CPU 1 tiếp tục chạy tiến trình thứ hai. Timeslot 16: Tiến trình thứ hai thực thi xong và CPU 0 tiếp tục chạy tiến trình thứ tư. Timeslot 20: Tiến trình thứ tư thực thi xong và CPU 1 tiếp tục chạy tiến trình thứ ba. Timeslot 24: Tiến trình thứ ba thực thi xong và CPU 0 tiếp tục chạy tiến trình thứ tư. Timeslot 27: Tiến trình thứ ba thực thi xong và CPU 1 dừng hoạt động sau khi không còn tiến trình nào trong hàng đợi. Timeslot 33: Tiến trình thứ tư thực thi xong và CPU 0 dừng hoạt động sau khi không còn tiến trình nào trong hàng đợi.



### 1.3.2 sched\_0

#### sched\_0 input

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

Output:

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

Biểu đồ Grantt:

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
CPU 0		s0		s1		s0		s1		s0		s1		s0		s1		s0		s1		s0	

#### Giải thích

- Timeslot 0-2: Tiến trình đầu tiên được tải và thực thi trên CPU 0.
- Timeslot 3-16: CPU tiếp tục thực thi tiến trình và đẩy nó vào hàng đợi thực thi sau mỗi time slot.
- Timeslot 17: Tiến trình hoàn thành, CPU tiếp tục thực thi tiến trình tiếp theo trong hàng đợi.
- Timeslot 18-21: Sau khi tiến trình thứ nhất hoàn thành, CPU tiếp tục thực thi tiến trình này từ hàng đợi.
- Timeslot 22: Tiến trình thứ nhất hoàn thành và CPU dừng hoạt động.



### 1.3.3 sched\_1

#### sched\_1 input

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

Output:

```
1 Time slot 0
2 ld_routine
3 Loaded a process at input/proc/s0, PID: 1 PI
4 CPU 0: Dispatched process 1
5 Time slot 1
6 Loaded a process at input/proc/s0, PID: 2 PI
7 Time slot 2
8 Loaded a process at input/proc/s0, PID: 3 PI
9 CPU 0: Put process 1 to run queue
10 CPU 0: Dispatched process 2
11 Time slot 3
12 Loaded a process at input/proc/s0, PID: 4 PI
13 Time slot 4
14 CPU 0: Put process 2 to run queue
15 CPU 0: Dispatched process 3
16 Time slot 5
17 Time slot 6
18 CPU 0: Put process 3 to run queue
19 CPU 0: Dispatched process 1
20 Time slot 7
21 Time slot 8
22 CPU 0: Put process 1 to run queue
23 CPU 0: Dispatched process 2
24 Time slot 9
25 Time slot 10
26 CPU 0: Put process 2 to run queue
27 CPU 0: Dispatched process 3
28 Time slot 11
29 Time slot 12
30 CPU 0: Put process 3 to run queue
31 CPU 0: Dispatched process 1
32 Time slot 13
33 Time slot 14
34 CPU 0: Put process 1 to run queue
35 CPU 0: Dispatched process 2
36 Time slot 15
37 Time slot 16
38 CPU 0: Put process 2 to run queue
39 CPU 0: Dispatched process 3
40 Time slot 17
41 Time slot 18
42 CPU 0: Put process 3 to run queue
43 CPU 0: Dispatched process 1
44 Time slot 19
45 CPU 0: Dispatched process 1
46 Time slot 19
47 CPU 0: Put process 1 to run queue
48 CPU 0: Dispatched process 2
49 Time slot 21
50 CPU 0: Put process 2 to run queue
51 CPU 0: Dispatched process 3
52 Time slot 23
53 CPU 0: Put process 3 to run queue
54 CPU 0: Dispatched process 1
55 Time slot 25
56 CPU 0: Put process 1 to run queue
57 CPU 0: Dispatched process 2
58 CPU 0: Put process 2 to run queue
59 CPU 0: Dispatched process 3
60 Time slot 27
61 Time slot 28
62 CPU 0: Put process 3 to run queue
63 CPU 0: Dispatched process 1
64 Time slot 29
65 Time slot 30
66 CPU 0: Put process 1 to run queue
67 CPU 0: Dispatched process 2
68 Time slot 31
69 Time slot 32
70 CPU 0: Put process 2 to run queue
71 CPU 0: Dispatched process 3
72 Time slot 33
73 Time slot 34
74 CPU 0: Put process 3 to run queue
75 CPU 0: Dispatched process 1
76 Time slot 35
77 Time slot 36
78 CPU 0: Put process 1 to run queue
79 CPU 0: Dispatched process 2
80 Time slot 37
81 Time slot 38
82 CPU 0: Put process 2 to run queue
83 CPU 0: Dispatched process 3
84 Time slot 39
85 Time slot 40
86 CPU 0: Put process 3 to run queue
87 CPU 0: Dispatched process 1
88 Time slot 41
89 Time slot 42
90 CPU 0: Put process 1 to run queue
91 CPU 0: Dispatched process 2
92 Time slot 43
93 CPU 0: Processed 1 has finished
94 CPU 0: Dispatched process 3
95 Time slot 44
96 CPU 0: Processed 2 has finished
97 CPU 0: Dispatched process 4
98 Time slot 45
99 CPU 0: Processed 3 has finished
100 CPU 0: Dispatched process 4
101 Time slot 46
102 Time slot 47
103 CPU 0: Put process 4 to run queue
104 CPU 0: Dispatched process 4
105 Time slot 48
106 Time slot 49
107 CPU 0: Put process 4 to run queue
108 CPU 0: Dispatched process 4
109 Time slot 50
110 Time slot 51
111 CPU 0: Put process 4 to run queue
112 CPU 0: Dispatched process 4
113 Time slot 52
114 Time slot 53
115 CPU 0: Put process 4 to run queue
116 CPU 0: Dispatched process 4
117 Time slot 54
118 Time slot 55
119 CPU 0: Put process 4 to run queue
120 CPU 0: Dispatched process 4
121 Time slot 56
122 Time slot 57
123 CPU 0: Put process 4 to run queue
124 CPU 0: Dispatched process 4
125 Time slot 58
126 Time slot 59
127 CPU 0: Put process 4 to run queue
128 CPU 0: Dispatched process 4
129 Time slot 60
130 CPU 0: Processed 4 has finished
131 CPU 0 stopped
132
```

Biểu đồ Grantt:

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
CPU 0	s0	s1	s2																				
Time slot	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43		
CPU 0	s2	s0	s1																				
Time slot	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60						
CPU 0	s2																						



### Giải thích

- Timeslot 0-2: Tiến trình đầu tiên được tải và thực thi trên CPU 0, sau đó được đẩy vào hàng đợi thực thi và tiến trình thứ hai được thực thi.
- Timeslot 3-43: Tiến trình thứ hai, thứ ba và thứ tư lần lượt được tải và thực thi. Mỗi khi một tiến trình hoàn thành, nó được đẩy vào hàng đợi thực thi và tiến trình tiếp theo được chọn để thực thi.
- Timeslot 43-59: Sau khi tiến trình thứ tư hoàn thành, tiến trình thứ năm được thực thi. Tuy nhiên, trong mỗi time slot, chỉ có tiến trình thứ năm được thực thi. Điều này có thể chỉ ra rằng sau khi tiến trình thứ tư hoàn thành, không có tiến trình nào được tải vào hệ thống hoặc tiến trình tiếp theo đã gặp lỗi và dừng lại.
- Timeslot 60: Cuối cùng, CPU dừng hoạt động sau khi tiến trình thứ năm hoàn thành và không có tiến trình nào khác được thực thi.



## 1.4 Question

### Question

What is the advantage of the scheduling strategy used in this assignment in comparison with other scheduling algorithms you have learned?

### Answer

Giải thuật Priority Queue được sử dụng trong bài tập lớn này có những ưu điểm so với những giải thuật định thời khác:

1. Phân cấp theo mức độ ưu tiên: Các tác vụ được lên lịch dựa trên mức độ ưu tiên của chúng, cho phép các tác vụ quan trọng hoặc ưu tiên cao được thực thi nhanh chóng.
2. Linh hoạt: Mức độ ưu tiên có thể được điều chỉnh linh hoạt trong quá trình chạy, phù hợp với những thay đổi về tầm quan trọng hoặc tính cấp bách của tác vụ.
3. Đa dạng: Các tiêu chí ưu tiên có thể được tùy chỉnh để phù hợp với các yêu cầu cụ thể của hệ thống, chẳng hạn như thời hạn, nhu cầu tài nguyên hoặc các yếu tố khác.
4. Hỗ trợ thời gian thực: Lập lịch theo ưu tiên phù hợp cho các hệ thống thời gian thực với các ràng buộc thời gian nghiêm ngặt.

So sánh với các thuật toán lập lịch khác: Các thuật toán lập lịch khác có một số hạn chế so với Hàng đợi ưu tiên (Priority Queue):

1. First Come First Served - FCFS: Quy trình yêu cầu CPU đầu tiên sẽ được cấp phát CPU trước, các quy trình khác phải đợi nó hoàn thành trước khi được thực thi. Điều này dẫn đến thời gian chờ trung bình lâu hơn và hiện tượng đoàn tàu (convoy effect).
2. Round Robin - RR: RR phân bổ một khoảng thời gian cố định (gọi là quantum) cho mỗi tác vụ theo chu kỳ, cho phép chia sẻ CPU một cách công bằng giữa các tác vụ. Tuy nhiên, việc chọn thời gian quantum thích hợp có thể là một thách thức: quá ngắn làm tăng chi phí chuyển đổi ngữ cảnh, quá dài làm giảm khả năng phản hồi và trở thành FCFS.
3. Shortest Job First - SJF: Quy trình có thời gian bùng nổ (burst time) ít nhất được xử lý trước, nghĩa là một quy trình dài có thể không bao giờ được thực thi và hệ thống cứ tiếp tục thực thi các quy trình ngắn. Điều này dẫn đến tình trạng chết đói (starvation).
4. Multilevel Queue: Các tiến trình được thực thi tùy thuộc vào mức độ ưu tiên của cấp độ hàng đợi cụ thể mà tiến trình đó thuộc. Điều này dẫn đến tình trạng chết đói của các tiến trình ở các cấp độ thấp hơn của hàng đợi đa cấp.
5. Ngoài ra, lập lịch Priority Queue có thể ngăn chặn tình trạng “chết đói” bằng cách sử dụng kỹ thuật lão hóa (aging).

Tuy nhiên, giải thuật này vẫn còn 1 vài hạn chế như phức tạp và tốn kém hơn, nguy cơ bị tắc nghẽn hoặc bị hạn chế về kích thước hay khó khăn trong việc gỡ lỗi nhưng nhìn chung Priority Queue là một công cụ mạnh mẽ và linh hoạt cho việc xử lý các tác vụ theo mức độ ưu tiên, nhưng cần cân nhắc những hạn chế tiềm ẩn trước khi sử dụng.



## 2 Memory Management

### 2.1 Tổng quan

#### 2.1.1 Ý tưởng chung

Mỗi tiến trình (process) có không gian địa chỉ ảo riêng biệt, được tổ chức thành nhiều vùng nhớ ảo (virtual memory area) liên tục. Mỗi vùng nhớ ảo có phạm vi từ `vm_start` đến `vm_end`, và phần có thể sử dụng thực sự được giới hạn bởi con trỏ `sbrk`. Trong mỗi vùng nhớ ảo, có các vùng nhớ (memory region) được quản lý bằng danh sách `vm_freeerg_list`, đại diện cho các biến trong chương trình.

Việc ánh xạ từ địa chỉ ảo sang địa chỉ vật lý được quản lý bằng cấu trúc `mm_struct`, bao gồm các thành phần như page table directory `pgd`, danh sách các vùng nhớ ảo `mmap`, và bảng ký hiệu `symsrgtbl` để quản lý các biến.

Hệ thống vật lý gồm có RAM và SWAP, được quản lý bằng các cấu trúc `framephy_struct` và `memphy_struct`. RAM có thể truy cập trực tiếp từ bus địa chỉ CPU, trong khi SWAP chỉ có thể truy cập gián tiếp thông qua việc di chuyển dữ liệu đến RAM.

Việc ánh xạ từ địa chỉ ảo sang địa chỉ vật lý được thực hiện thông qua cơ chế phân trang (paging), sử dụng bảng trang (page table) và bộ đệm tìm kiếm bảng trang (translation lookaside buffer - TLB). Cơ chế này sẽ được mô tả chi tiết trong phần tiếp theo.

#### 2.1.2 Paging

Cơ chế paging (phân trang) là kỹ thuật quản lý bộ nhớ ảo, trong đó bộ nhớ vật lý được chia thành các khung (frame) có kích thước cố định, trong bài tập lớn này là 256 byte hoặc 512 byte. Không gian địa chỉ ảo của mỗi tiến trình cũng được chia thành các trang (page) có cùng kích thước với khung vật lý.

Khi một tiến trình truy cập một địa chỉ ảo, hệ thống sẽ dịch địa chỉ ảo này thành địa chỉ vật lý thông qua bảng trang (page table) của tiến trình. Bảng trang lưu trữ sự ánh xạ từ mỗi trang ảo đến khung vật lý tương ứng trong RAM.

Nếu trang ảo được yêu cầu không có trong RAM, hệ thống sẽ thực hiện cơ chế hoán đổi (swapping), tức là nạp trang ảo đó từ vùng bộ nhớ hoán đổi (swap) vào một khung vật lý trống trong RAM. Để tối ưu hóa hiệu suất, hệ thống sử dụng bộ đệm dịch địa chỉ (TLB) để lưu trữ một số ánh xạ địa chỉ ảo - địa chỉ vật lý thường được sử dụng.



## 2.2 Implementation

### 2.2.1 MEMPHY\_dump(struct memphy\_struct \* mp)

Hàm MEMPHY\_dump được sử dụng để in ra nội dung của bộ nhớ được quản lý bởi cấu trúc memphy\_struct:

```
1 int MEMPHY_dump(struct memphy_struct * mp)
2 {
3     /* TODO: Dump memphy content mp->storage for tracing the memory content
4      * ↪ */
5     if(!mp || !mp->storage) return -1;
6     printf("Dumping memphy content\n");
7     fprintf(output_file,"Dumping memphy content\n");
8     for(int i = 0; i < mp->maxsz; i++){
9         if(mp->storage[i] != 0){
10             printf("Address: %d, Value: %d\n", i, mp->storage[i]);
11             fprintf(output_file,"Address: %d, Value: %d\n", i, mp->storage[i]);
12         }
13     }
14 }
```

### 2.2.2 int \_\_alloc(struct pcb\_t \*caller, int vmaid, int rgid, int size, int \*alloc\_addr)

Hàm \_\_alloc được sử dụng để cấp phát bộ nhớ cho một vùng nhớ ảo (virtual memory area - VMA) cụ thể trong quá trình điều khiển bảng (process control block - PCB)

1. Hàm cố gắng cấp phát bộ nhớ bằng cách sử dụng hàm get\_free\_vmrq\_area. Nếu thành công, nó sẽ thiết lập các giá trị rg\_start và rg\_end cho vùng nhớ được cấp phát và trả về 0.
2. Nếu bước 1 thất bại, hàm sẽ cố gắng tăng giới hạn bộ nhớ của VMA để có thêm không gian. Điều này được thực hiện bằng cách tính toán kích thước tăng (inc\_sz) và gọi hàm inc\_vma\_limit.
3. Nếu kích thước mới (new\_sbrk) nhỏ hơn vm\_end của VMA hiện tại, nó sẽ cập nhật rg\_start và rg\_end của vùng nhớ và sbrk của VMA, sau đó trả về 0.
4. Cuối cùng, nếu tất cả các bước trên đều thành công, hàm sẽ cập nhật rg\_start và rg\_end của vùng nhớ, sbrk của mmap và địa chỉ cấp phát (alloc\_addr), sau đó trả về 0.

```
1 int __alloc(struct pcb_t *caller, int vmaid, int rgid, int size, int *
2     ↪ alloc_addr){
3     /*Allocate at the top of */
4     struct vm_rg_struct rgnode;
5     if (get_free_vmrq_area(caller, vmaid, size, &rgnode) == 0)
6     {
7         caller->mm->symrgtbl[rgid].rg_start = rgnode.rg_start;
8         caller->mm->symrgtbl[rgid].rg_end = rgnode.rg_end;
9         *alloc_addr = rgnode.rg_start;
10    }
11    /* TODO get_free_vmrq_area FAILED handle the region management (Fig.6) */
12    /* Attempt to increase limit to get space */
13    struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, vmaid);
```



```
14
15 // int inc_sz = PAGING_PAGE_ALIGNNSZ(size); //original inc_sz
16 int inc_sz = PAGING_PAGE_ALIGNNSZ((size + cur_vma->sbrk) - cur_vma->vm_end)
17     ↪ ;
18 //int inc_limit_ret
19 int old_sbrk = cur_vma->sbrk ;
20 int new_sbrk = old_sbrk + size;
21
22 if(new_sbrk < cur_vma->vm_end){
23     caller->mm->symrgtbl[rgid].rg_start = old_sbrk;
24     caller->mm->symrgtbl[rgid].rg_end = new_sbrk;
25     cur_vma->sbrk = new_sbrk;
26     return 0;
27 }
28 /* TODO INCREASE THE LIMIT
29 * inc_vma_limit(caller, vmaid, inc_sz)
30 */
31 inc_vma_limit(caller, vmaid, inc_sz);
32
33 /*Successful increase limit */
34 caller->mm->symrgtbl[rgid].rg_start = old_sbrk;
35 caller->mm->symrgtbl[rgid].rg_end = old_sbrk + size;
36 caller->mm->mmap->sbrk = new_sbrk;
37
38 *alloc_addr = old_sbrk;
39 //printf("alloc rg %d start: %d, end: %d\n", rgid, caller->mm->symrgtbl[
40     ↪ rgid].rg_start, caller->mm->symrgtbl[rgid].rg_end);
41 return 0;
42 }
```

### 2.2.3 int \_\_free(struct pcb\_t \*caller, int vmaid, int rgid))

Hàm `__free` được sử dụng để giải phóng một vùng nhớ ảo (virtual memory area - VMA) cụ thể từ quá trình điều khiển bảng (process control block - PCB).

1. Hàm kiểm tra xem `rgid` có nằm trong phạm vi hợp lệ không, sau đó tìm VMA tương ứng với `vmaid` trong PCB. Nếu không thấy, trả về -1..
2. Sau đó, hàm lấy vùng nhớ cần giải phóng (`freed_rg`) từ bảng `symrgtbl` của PCB. Nếu `freed_rg` không tồn tại hoặc đã được giải phóng (điều này được kiểm tra bằng cách so sánh `rg_start` và `rg_end`), in ra lỗi và trả về -1.
3. Cập nhật `rg_start` và `rg_end` của `rgnode` để phản ánh vùng nhớ được giải phóng.
4. Đặt `rg_start` và `rg_end` của `freed_rg` về 0 để đánh dấu rằng vùng nhớ đã được giải phóng.
5. Thêm `rgnode` vào danh sách `vm_freerg_list` của `mmap` để quản lý các vùng nhớ đã được giải phóng.
6. Cuối cùng, in ra thông tin về các vùng nhớ đã được giải phóng và trả về 0.
7. Trong hàm này, thông báo “Double free” được in ra khi vùng nhớ cần giải phóng (`freed_rg`) đã được giải phóng trước đó. Điều này được kiểm tra bằng cách so sánh `rg_start` và `rg_end` của `freed_rg`. Nếu hai



giá trị này bằng nhau, điều đó có nghĩa là vùng nhớ đã được giải phóng (do rg\_start và rg\_end được đặt về 0 sau khi giải phóng vùng nhớ), và do đó thông báo “Double free” được in ra.

```
1 int __free(struct pcb_t *caller, int vmaid, int rgid){
2     struct vm_rg_struct *rgnode = malloc(sizeof(struct vm_rg_struct)); //khai
3         ↪ bao khu vực trong' moi bang heap
4     if(rgid < 0 || rgid > PAGING_MAX_SYMTBL_SZ)
5         return -1;
6
7     /* TODO: Manage the collect freed region to freerg_list */
8     struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, vmaid);
9     if(cur_vma == NULL) return -1;
10
11    struct vm_rg_struct *freed_rg = &(caller->mm->symrgtbl[rgid]);
12    if(freed_rg->rg_start==freed_rg->rg_end){
13        printf("Double\u00eufree\n");
14        fprintf(output_file,"Double\u00eufree\n");
15        return -1;
16    }
17    if(freed_rg == NULL){
18        printf("Segmentation\u00eufault\n");
19        fprintf(output_file,"Segmentation\u00eufault\n");
20        return -1;
21
22    rgnode->rg_start = freed_rg->rg_start;
23    rgnode->rg_end = freed_rg->rg_end;
24
25    freed_rg->rg_start = freed_rg->rg_end = 0;
26    /*enlist the obsoleted memory region */
27    enlist_vm_freerg_list(caller->mm, rgnode);
28    printf("Check\u00eufree\u00eurg\n");
29    fprintf(output_file,"Check\u00eufree\u00eurg\n");
30    print_list_rg(caller->mm->mmap->vm_freerg_list);
31    _print_rg_alloc(caller->mm->symrgtbl);
32    return 0;
}
```

#### 2.2.4 int pg\_getpage(struct mm\_struct \*mm, int pgn, int \*fpn, struct pcb\_t \*caller)

Hàm pg\_getpage được sử dụng để quản lý việc truy cập trang trong hệ thống phân trang. Nó nhận vào một cấu trúc mm\_struct (đại diện cho không gian địa chỉ ảo của một tiến trình), một số trang (pgn), một con trỏ đến số trang vật lý (fpn), và một cấu trúc pcb\_t (đại diện cho một tiến trình).

Nếu trang được yêu cầu không hiện diện (PAGING\_PAGE\_PRESENT(pte) trả về false), hàm sẽ thực hiện các bước để đưa trang đó vào bộ nhớ. Điều này bao gồm việc tìm một trang nạn nhân để thay thế (nếu cần), lấy một khung trang tự do từ bộ nhớ hoán đổi (MEMSWP), và sau đó hoán đổi trang nạn nhân ra khỏi bộ nhớ và hoán đổi trang được yêu cầu vào bộ nhớ.

Nếu trang đã có, hàm chỉ đơn giản trả về số trang vật lý (fpn) của trang đó.

Hàm trả về 0 nếu thành công, và -1 nếu có lỗi xảy ra (ví dụ: không tìm thấy trang nạn nhân hoặc không có

khung trang tự do nào trong MEMSWP).

```
1 int pg_getpage(struct mm_struct *mm, int pgn, int *fpn, struct pcb_t *caller
2   ↪ ) {
3     uint32_t pte = mm->pgd[pgn];
4
5     if (!PAGING_PAGE_PRESENT(pte))
6     { /* Page is not online, make it actively living */
7       int vicpgn, swpfpn;
8       int tgtfpn = PAGING_SWPOFF(pte);
9       int tgtswp_type = PAGING_SWPTYP(pte);
10      /* TODO: Play with your paging theory here */
11      /* Find victim page */
12      if(find_victim_page(caller->mm, &vicpgn) < 0)
13        return -1; /* No victim page found */
14      while(vicpgn == pgn){
15        if(find_victim_page(caller->mm, &vicpgn) < 0)
16          return -1; /* No victim page found */
17      }
18      //uint32_t vicpte = caller->mm->pgd[vicpgn]; không dùng
19
20      /* Get free frame in MEMSWP */
21      if(MEMPHY_get_freefp(caller->active_mswp, &swpfpn) < 0)
22      {
23        struct memphy_struct *mswpit = caller->mswp[tgtswp_type];
24        if(MEMPHY_get_freefp(mswpit, &swpfpn) < 0)
25          return -1; /* No free frame found */
26        __swap_cp_page(mswpit, swpfpn, caller->mram, vicpgn);
27        caller->active_mswp = mswpit;
28      }
29      else{
30        __swap_cp_page(caller->active_mswp, swpfpn, caller->mram, vicpgn);
31
32        __swap_cp_page(caller->mswp[tgtswp_type], tgtfpn, caller->mram, vicpgn
33          ↪ );
34        MEMPHY_put_freefp(caller->mswp[tgtswp_type], tgtfpn);
35        pte_set_swap(&caller->mm->pgd[vicpgn], tgtswp_type, swpfpn);
36        pte_set_fpn(&caller->mm->pgd[pgn], vicpgn);
37
38        enlist_pgn_node(&caller->mm->fifo_pgn, pgn);
39        pte = caller->mm->pgd[pgn];
40      }
41      *fpn = PAGING_FPN(pte);
42      return 0;
43    }
44
45    *fpn = PAGING_FPN(pte);
46    return 0;
47 }
```



### 2.2.5 int find\_victim\_page(struct mm\_struct \*mm, int \*retpgn)

Hàm `find_victim_page` được sử dụng để tìm một trang nạn nhân (victim page) trong không gian địa chỉ ảo của một tiến trình (được đại diện bởi cấu trúc `mm_struct`). Trang nạn nhân là trang sẽ bị thay thế khi không còn khung trang tự do (free frame) nào trong bộ nhớ.

Hàm nhận vào một cấu trúc `mm_struct` và một con trỏ đến một số nguyên (`retpgn`), nơi nó sẽ trả về số trang của trang nạn nhân.

Hàm sử dụng một cấu trúc dữ liệu hàng đợi (FIFO) để theo dõi các trang. Nó duyệt qua hàng đợi cho đến khi tìm thấy trang cuối cùng (trang nạn nhân), sau đó gán số trang của trang nạn nhân cho `*retpgn` và loại bỏ trang đó khỏi hàng đợi.

Hàm trả về 0 nếu thành công, và -1 nếu không tìm thấy trang nạn nhân (nghĩa là hàng đợi trống).

```
1 int find_victim_page(struct mm_struct *mm, int *retpgn) {
2     struct pgn_t *pg = mm->fifo_pgn;
3
4     /* TODO: Implement the theoretical mechanism to find the victim page */
5     if(pg == NULL)
6         return -1;
7
8     while(pg->pg_next && pg->pg_next->pg_next != NULL)
9         pg = pg->pg_next;
10    if(pg->pg_next){
11        *retpgn = pg->pg_next->pgn;
12        free(pg->pg_next);
13        pg->pg_next = NULL;
14    }
15    else{
16        *retpgn = pg->pgn;
17        free(pg);
18        mm->fifo_pgn = NULL;
19    }
20    return 0;
21 }
```

### 2.2.6 int vmap\_page\_range(struct pcb\_t \*caller, int addr, int pgnum, struct framephy\_struct \*frames, struct vm\_rg\_struct \*ret\_rg))

Hàm `vmap_page_range` được sử dụng để ánh xạ một dãy trang vào không gian địa chỉ ảo của một process.

Hàm nhận vào một cấu trúc `pcb_t` (đại diện cho một process), một địa chỉ bắt đầu (đã được căn chỉnh theo `PAGING_PAGESZ`), số lượng trang cần ánh xạ, một danh sách các khung trang đã ánh xạ, và một cấu trúc `vm_rg_struct` để trả về vùng đã ánh xạ.

Hàm sử dụng một vòng lặp để duyệt qua tất cả các trang cần ánh xạ. Đối với mỗi trang, nó sẽ cập nhật bảng trang của tiến trình (`caller->mm->pgd[]`) để ánh xạ trang đó tới một khung trang tương ứng (`fpit->fpn`), và thêm số trang vào hàng đợi FIFO của tiến trình (`caller->mm->fifo_pgn`).

Sau đó trả về 0 nếu thành công.



```
1 int vmap_page_range(struct pcb_t *caller, // process call
2                     int addr, // start address which is aligned
3                     ↪ to pagesz
4                     int pgn, // num of mapping page
5                     struct framephy_struct *frames, // list of the mapped frames
6                     struct vm_rg_struct *ret_rg) // return mapped region, the real
7                     ↪ mapped fp
8 {
9     // no guarantee all given pages
10    ↪ are mapped
11    //uint32_t * pte = malloc(sizeof(uint32_t));
12    struct framephy_struct *fpit = malloc(sizeof(struct framephy_struct));
13    //int fpn;
14    int pgit = 0;
15    int pgn = PAGING_PGN(addr);
16
17    ret_rg->rg_end = ret_rg->rg_start = addr; // at least the very first space
18    ↪ is usable
19
20    /* TODO map range of frame to address space
21     *      [addr to addr + pgn*PAGING_PAGESZ
22     *      in page table caller->mm->pgd[]]
23     */
24
25    /* Tracking for later page replacement activities (if needed)
26     * Enqueue new usage page */
27    for(pgit = 0; pgit < pgn; pgit++)
28    {
29        fpit = fpit->fp_next;
30        pgn = PAGING_PGN((addr + pgit * PAGING_PAGESZ));
31        if(fpit){
32            pte_set_fp(&caller->mm->pgd[pgn], fpit->fpn);
33            enlist_pgn_node(&caller->mm->fifo_pgn, pgn);
34        }
35    }
36    ret_rg->rg_end += (pgit - 1) * PAGING_PAGESZ;
37    return 0;
38 }
```

### 2.2.7 int alloc\_pages\_range(struct pcb\_t \*caller, int req\_pgn, struct framephy\_struct\*\* frm\_lst)

Hàm `alloc_pages_range` được sử dụng để cấp phát một dãy trang cho một tiến trình.

Hàm nhận vào một cấu trúc `pcb_t` (đại diện cho một tiến trình), số lượng trang cần cấp phát (`req_pgn`), và một danh sách các khung trang (`frm_lst`).

Hàm sử dụng một vòng lặp để duyệt qua tất cả các trang cần cấp phát. Đối với mỗi trang, nó sẽ kiểm tra xem có



khung trang nào trống không (`MEMPHY_get_freefp`). Nếu có, nó sẽ cấp phát một khung trang mới (`newfp_str`), gán số khung trang (`fpn`) và chủ sở hữu của khung trang (`caller->mm`), sau đó thêm khung trang mới này vào danh sách khung trang (`frm_lst`).

Nếu không có khung trang nào trống, hàm sẽ tìm một trang nạn (`find_victim_page`), sao chép trang nạn này vào một khung trang trống trong bộ nhớ swap (`_swap_cp_page`), sau đó đặt trang nạn này vào trạng thái swap (`pte_set_swap`).

Hàm trả về 0 nếu thành công, hoặc -3000 nếu hết bộ nhớ.

```
1 int alloc_pages_range(struct pcb_t *caller, int req_pgn, struct
2   ↪ framephy_struct** frm_lst)
3 {
4   int pgit, fpn;
5   struct framephy_struct *newfp_str;
6
7   for(pgit = 0; pgit < req_pgn; pgit++)
8   {
9     if(MEMPHY_get_freefp(caller->mram, &fpn) == 0)
10    {
11      newfp_str = (struct framephy_struct *)malloc (sizeof(struct
12        ↪ framephy_struct));
13      newfp_str->fpn = fpn;
14      newfp_str->owner = caller->mm;
15      if(!*frm_lst){
16        *frm_lst = newfp_str;
17      }
18      else{
19        newfp_str->fp_next = *frm_lst;
20        *frm_lst = newfp_str;
21      }
22    }
23    else { // ERROR CODE of obtaining somes but not enough frames
24      int victim_fpn;
25      int victim_pgn;
26      int victim_pte;
27      int swpfpn = -1;
28      if(find_victim_page(caller->mm, &victim_pgn) < 0){
29        return -3000; // Out of memory
30      }
31      victim_pte = caller->mm->pgd[victim_pgn];
32      victim_fpn = PAGING_FPN(victim_pte);
33
34      newfp_str = (struct framephy_struct *)malloc (sizeof(struct
35        ↪ framephy_struct));
36      newfp_str->fpn = victim_fpn;
37      newfp_str->owner = caller->mm;
38
39      if(!*frm_lst){
40        *frm_lst = newfp_str;
41      }
42    }
43  }
44 }
```



```
38
39     }
40     else{
41         newfp_str->fp_next = *frm_lst;
42         *frm_lst = newfp_str;
43     }
44
45     int i = 0;
46     if(MEMPHY_get_freefp(caller->active_mswp, &swpfpn) == 0){
47         __swap_cp_page(caller->mram, victim_fpn, caller->active_mswp, swpfpn
48         ↳ );
49         struct memphy_struct *mswp = (struct memphy_struct *)caller->mswp;
50         for (i = 0; i < PAGING_MAX_MMSWP; i++){
51             if(mswp + i == caller->active_mswp){
52                 break;
53             }
54         }
55         else{
56             struct memphy_struct *mswp = (struct memphy_struct *)caller->mswp;
57             swpfpn = -1;
58             for (i = 0; i < PAGING_MAX_MMSWP; i++){
59                 if(MEMPHY_get_freefp(mswp + i, &swpfpn) == 0){
60                     break;
61                 }
62             }
63             if(swpfpn < 0){
64                 return -3000; // Out of memory
65             }
66             pte_set_swap(&caller->mm->pgd[victim_pgn], i, swpfpn);
67         }
68     }
69     return 0;
70 }
```

### 2.2.8 int \_\_read(struct pcb\_t \*caller, int vmaid, int rgid, int offset, BYTE \*data)

Hàm \_\_read được sử dụng để đọc dữ liệu từ một vùng nhớ ảo.

1. Hàm lấy vùng nhớ (currsg) bằng cách sử dụng ID rgid từ tiến trình caller.
2. Nếu vùng nhớ này không có dữ liệu (nghĩa là rg\_end bằng rg\_start), hàm sẽ in ra lỗi "Segmentation fault" và trả về -1.
3. Hàm lấy vùng nhớ ảo (cur\_vma) bằng cách sử dụng ID vmaid từ tiến trình caller.
4. Nếu vùng nhớ đặc biệt hoặc vùng nhớ ảo không tồn tại (nghĩa là currsg hoặc cur\_vma là NULL), hàm trả về -1.
5. Hàm đọc giá trị từ vùng nhớ đặc biệt tại vị trí rg\_start + offset và lưu giá trị đọc được vào data.
6. Hàm trả về 0, nghĩa là việc đọc dữ liệu thành công.



```
1 int __read(struct pcb_t *caller, int vmaid, int rgid, int offset, BYTE *data
2   ↲ )
3 {
4     struct vm_rg_struct *currng = get_symrg_byid(caller->mm, rgid);
5     if(currng->rg_end==currng->rg_start){
6       printf("Segmentation fault\n");
7       fprintf(output_file,"Segmentation fault\n");
8       return -1;
9     };
10    struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, vmaid);
11    if(currng == NULL || cur_vma == NULL) /* Invalid memory identify */
12      return -1;
13    pg_getval(caller->mm, currng->rg_start + offset, data, caller);
14    return 0;
}
```

### 2.2.9 int \_\_write(struct pcb\_t \*caller, int vmaid, int rgid, int offset, BYTE value)

Hàm \_\_write được sử dụng để ghi một giá trị vào một vùng nhớ cụ thể.

- Đầu tiên, hàm lấy phạm vi bằng cách sử dụng hàm `get_symrg_byid` với `caller->mm` và `rgid` làm tham số.
- Nếu phạm vi này không có độ dài (nghĩa là `rg_end` bằng `rg_start`), hàm sẽ in ra lỗi "Segmentation fault" và trả về 0.
- Tiếp theo, nó lấy vùng nhớ ảo bằng cách sử dụng hàm `get_vma_by_num` với `caller->mm` và `vmaid` làm tham số.
- Nếu phạm vi hoặc vùng nhớ ảo không tồn tại (nghĩa là `currng` hoặc `cur_vma` là NULL), hàm sẽ in ra lỗi "Invalid memory identify" và trả về -1.
- Cuối cùng, nếu không có lỗi nào xảy ra, hàm sẽ gọi `pg_setval` để ghi giá trị vào vị trí tương ứng trong vùng nhớ và trả về 0.

```
1 int __write(struct pcb_t *caller, int vmaid, int rgid, int offset, BYTE
2   ↲ value)
3 {
4     struct vm_rg_struct *currng = get_symrg_byid(caller->mm, rgid);
5     if(currng->rg_end==currng->rg_start){
6       printf("Segmentation fault\n");
7       fprintf(output_file,"Segmentation fault\n");
8       return 0;
9     };
10    struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, vmaid);
11    if(currng == NULL || cur_vma == NULL) /* Invalid memory identify */
12    {
13      printf("Invalid memory identify\n");
14      fprintf(output_file,"Invalid memory identify\n");
15      return -1;
}
```



```
16 pg_setval(caller->mm, currrg->rg_start + offset, value, caller);  
17     return 0;  
18 }
```

## 2.3 Result

★ Preprocessor directives cần để chạy chương trình:

```
100, 2 days ago | 1 author (100)
1 #ifndef OSCFG_H
2 #define OSCFG_H
3
4 #define MLQ_SCHED 1
5 #define MAX_PRIO 140
6
7 // #define CPU_TLB
8 // #define CPUTLB_FIXED_TLBSZ
9 #define MM_PAGING
10 // #define MM_FIXED_MEMSZ
11 #define VMDBG 1      You, 2 days ago • Merge
12 #define MMDBG 1
13 #define IODUMP 1
14 #define PAGETBL_DUMP 1
15
16 #endif
17
```

### 2.3.1 os\_0\_mlq\_paging

#### os\_0\_mlq\_paging input

```
6 2 2
1048576 16777216 0 0 0
0 p0s 0
2 p1s 15
```

Output:

```
1 Time slot 0
2 ↴ td_routine
3   Loaded a process at input/proc/p0s, PID: 1 PRIO: 0
4   CPU 0: Dispatched process 1
5 Time slot 1
6 ↴ Time slot 2
7   Loaded a process at input/proc/p1s, PID: 2 PRIO: 15
8   CPU 1: Dispatched process 2
9 Time slot 3
10 Check free rg 0
11 print_list_rg:
12 rg[0->300]
13 rg[0->0]
14
15 Check remaining registers
16 rg 4
17 rg[300->600]
18
19 Time slot 4
20 Time slot 5
21 TLB hit at write region=1 offset=20 value=100
22 print_pgtbl: 0 - 512
23 00000000: 80000001
24 00000004: 80000000
25 Dumping memphy content
26 Time slot 6
27   CPU 0: Put process 1 to run queue
28   CPU 0: Dispatched process 1
29 TLB hit at read region=1 offset=20
30 print_pgtbl: 0 - 512
31 00000000: 80000001
32 00000004: 80000000
33 Dumping memphy content
34 Address: 476, Value: 100
35 data readed : 100
36 Time slot 7
37 Segmentation fault write
38 Time slot 8
39 Segmentation fault read
40   CPU 1: Put process 2 to run queue
41   CPU 1: Dispatched process 2
42 Time slot 9
43 Check free rg 4
44 print_list_rg:
45 rg[300->600]
46 rg[0->0]
47
48 Check remaining registers
49 rg 1
50 rg[200->300]
51
52 Time slot 10
53   CPU 0: Processed 1 has finished
54   CPU 0 stopped
55 Time slot 11
56 Time slot 12
57 Time slot 13
58   CPU 1: Processed 2 has finished
59   CPU 1 stopped
60
```



### Giải thích

- timeslot 0: Tải một tiến trình từ `input/proc/p0s` với PID (Process ID) là 1 và độ ưu tiên là 0. CPU 1 sau đó được giao tiến trình này để xử lý.
- timeslot 2: Một tiến trình khác được tải từ `input/proc/p1s` với PID là 2 và độ ưu tiên là 15. CPU 0 sau đó được giao tiến trình này để xử lý.
- timeslot 3: Hệ thống kiểm tra các vùng nhớ tự do và các thanh ghi còn lại. Có một vùng nhớ từ 0 đến 300 và một thanh ghi từ 300 đến 600 còn trống.
- timeslot 5: Một TLB (Translation Lookaside Buffer) hit xảy ra khi ghi vào vùng nhớ 1 tại offset 20 với giá trị 100. Điều này có nghĩa là địa chỉ vật lý tương ứng với địa chỉ ảo đã được tìm thấy trong TLB, giúp tăng tốc quá trình dịch địa chỉ.
- timeslot 6: CPU 1 đưa tiến trình 1 vào hàng đợi chờ và sau đó tiếp tục xử lý tiến trình này. Một TLB hit khác xảy ra khi đọc từ vùng nhớ 1 tại offset 20.
- timeslot 7: Một lỗi phân đoạn (Segmentation fault) xảy ra khi cố gắng ghi vào một vùng nhớ không hợp lệ.
- timeslot 8: Một lỗi phân đoạn khác xảy ra khi cố gắng đọc từ một vùng nhớ không hợp lệ. CPU 0 đưa tiến trình 2 vào hàng đợi chờ và sau đó tiếp tục xử lý tiến trình này.
- timeslot 9: Hệ thống kiểm tra các vùng nhớ tự do và các thanh ghi còn lại một lần nữa.
- timeslot 10: CPU 1 hoàn thành xử lý tiến trình 1 và dừng lại.
- timeslot 11: Không có sự kiện nào xảy ra trong khung thời gian này.
- timeslot 13: CPU 0 hoàn thành xử lý tiến trình 2 và dừng lại.

### 2.3.2 os\_1\_mlq.paging\_small\_1K

Với input:

#### os\_1\_mlq.paging\_small\_1K input

```

2 4 8
2048 16777216 0 0 0
1 p0s 130
2 s3 39
4 m1s 15
6 s2 120
7 m0s 120
9 p1s 15
11 s0 38
16 s1 0

```

Output:

```

1 Time slot 0
2 Id routine
3 Time slot 1
4 Loaded a process at input/proc/p0s, PID: 0 PRIO: 124
5 CPU 2: Dispatched process 1
6 Time slot 2
7 Loaded a process at input/proc/s3, PID: 3 PRIO: 39
8 CPU 0: Dispatched process 2
9 Time slot 3
10 CPU 1: Put process 3 to run queue
11 CPU 0: Dispatched process 3
12 Time slot 4
13 CPU 2: Dispatched process 4
14 CPU 0: Put process 2 to run queue
15 CPU 0: Dispatched process 2
16 Check free rg 0
17 print_list_rg:
18 rg[0->300]
19 rg[0->0]
20
21 Check remaining registers
22 Time slot 5
23 rg[300->600]
24
25 CPU 3: Dispatched process 3
26 Time slot 5
27 CPU 2: Put process 3 to run queue
28 CPU 2: Dispatched process 1
29 Time slot 6
30 CPU 3: Dispatched process at input/proc/p1s, PID: 0 PRIO: 124
31 CPU 1: Put process 2 to run queue
32 CPU 3: Dispatched process 3
33 CPU 0: Put process 2 to run queue
34 CPU 3: Dispatched process 2
35 CPU 1: Dispatched process 4
36 Check free rg 0
37 print_list_rg:
38 rg[0->300]
39 rg[0->0]
40
41 Check remaining registers
42 Time slot 7
43 Time slot 7
44 CPU 1: Write region@ offset=20 value=100
45 print_pttbl: 0 - 512
46 00000000: 80000000
47 00000004: 80000000
48 00000004: 80000000
49 00000004: 80000000
50 Time slot 8
51 CPU 2: Time slot 7
52 CPU 2: Put process 3 to run queue
53 CPU 2: Dispatched process 5
54 Time slot 8
55 CPU 3: Put process 4 to run queue
56 CPU 3: Dispatched process 4
57 CPU 3: Put process 3 to run queue
58 CPU 3: Dispatched process 3
59 Check remaining rg 2
60 print_list_rg:
61 rg[200->300]
62 rg[0->0]
63 Dumping memphy content
64 Address: #70, Value: 100
65 Time slot 9
66 CPU 0: Put process 2 to run queue
67 CPU 2: Put process 3 to run queue
68 CPU 2: Dispatched process 2
69 CPU 0: Dispatched process 2
70 Time slot 9
71 CPU 3: Dispatched process at input/proc/p1s, PID: 0 PRIO: 124
72 CPU 2: Put process 3 to run queue
73 CPU 2: Dispatched process
74 Check free rg 0
75 print_list_rg:
76 rg[200->600]
77 rg[0->0]
78
79 Check remaining registers
80 Time slot 10
81 Time slot 10
82 CPU 3: Processed 3 has finished
83 CPU 3: Dispatched process 5
84 Check free rg 0
85 print_list_rg:
86 rg[0->300]
87 CPU 0: Put process 2 to run queue
88 CPU 0: Put process 4 to run queue
89 CPU 0: Dispatched process 2
90 CPU 3: Dispatched process 4
91 CPU 3: Dispatched process
92
93 Check remaining registers
94 print_pttbl: 0 - 512
95 00000000: 80000000
96 00000004: 80000000
97 00000004: 80000000
98 00000004: 80000000
99 Time slot 11
100 CPU 2: Dispatched process 6
101 CPU 3: Put process 5 to run queue
102 CPU 3: Dispatched process 7
103 CPU 0: Dispatched process 2 to run queue
104 CPU 0: Dispatched process 3 to run queue
105 CPU 0: Dispatched process 4 to run queue
106 CPU 1: Dispatched process 5 to run queue
107 CPU 1: Dispatched process 5
108 TLB miss at write region@ offset=20 value=102
109 print_pttbl: 0 - 512
110 00000000: 80000000
111 00000004: 80000000
112 Dumping memphy content
113 Address: #70, Value: 100
114 Time slot 12
115 CPU 0: Processed 2 has finished
116 CPU 2: Put process 6 to run queue
117 CPU 2: Dispatched process 6
118 CPU 2: Dispatched process 4
119 Segmentation fault write
120 Time slot 12
121 CPU 3: Put process 7 to run queue
122 CPU 3: Dispatched process 7
123 CPU 0: Processed 1 has finished
124 CPU 1: Dispatched process 6
125 CPU 1: Dispatched process 7
126 print_pttbl: 0 - 512
127 00000000: 80000000
128 00000004: 80000000
129 Dumping memphy content
130 Address: #70, Value: 100
131 Data readed : 100
132 Time slot 13
133 CPU 0: Processed 1 has finished
134 CPU 0: Put process 4 to run queue
135 CPU 1: Dispatched process 6 to run queue
136 CPU 1: Dispatched process 6
137 CPU 2: Dispatched process 6
138 Segmentation fault write
139 Time slot 13
140 CPU 3: Dispatched process at input/proc/s1, PID: 0 PRIO: 0
141 CPU 3: Put process 3 to run queue
142 CPU 3: Dispatched process 8
143 CPU 3: Put process 7 to run queue
144 CPU 3: Dispatched process 7
145 Time slot 14
146 CPU 0: Put process 4 to run queue
147 CPU 0: Dispatched process 4
148 CPU 2: Put process 6 to run queue
149 CPU 2: Dispatched process 6
150 CPU 3: Put process 7 to run queue
151 CPU 3: Dispatched process 7
152 CPU 3: Put process 7 to run queue
153 CPU 3: Dispatched process 8
154 CPU 3: Dispatched process 8
155 Time slot 15
156 CPU 2: Put process 6 to run queue
157 CPU 2: Dispatched process 6
158 CPU 0: Dispatched process 5 has finished
159 CPU 0: Dispatched process 6
160 CPU 0: Dispatched process 8
161 Segmentation fault read
162 Time slot 20
163 CPU 0: Put process 8 to run queue
164 CPU 0: Dispatched process 8
165 Check free rg 4
166 CPU 2: Processed 6 has finished
167 CPU 2: stopped
168 CPU 3: Put process 7 to run queue
169 CPU 3: Dispatched process 7
170 print_list_rg:
171 rg[300->600]
172 rg[0->0]
173 Check remaining registers
174 rg 0
175 rg[200->300]
176
177 Time slot 21
178 CPU 0: Processed 1 has finished
179 CPU 0: stopped
180 Time slot 21
181 CPU 3: Put process 7 to run queue
182 CPU 3: Dispatched process 7
183 CPU 3: Put process 8 to run queue
184 CPU 3: Dispatched process 8
185 Time slot 22
186 CPU 1: Processed 8 has finished
187 CPU 1: stopped
188 Time slot 22
189 CPU 3: Put process 7 to run queue
190 CPU 3: Dispatched process 7
191 Time slot 25
192 Time slot 25
193 CPU 3: Put process 7 to run queue
194 CPU 3: Dispatched process 7
195 Time slot 27
196 CPU 3: Dispatched process 7
197 Time slot 27
198 CPU 3: Processed 7 has finished
199 CPU 3: stopped

```



## Giải thích

- **Time slot 0:** Quá trình tải bắt đầu.
- **Time slot 1:** Một quá trình từ tệp `input/proc/p0s` được tải với PID là 1 và ưu tiên là 130. CPU 0 phân phối quá trình này.
- **Time slot 2:** Một quá trình từ tệp `input/proc/s3` được tải với PID là 2 và ưu tiên là 39. CPU 2 phân phối quá trình này.
- **Time slot 3:** CPU 0 đặt quá trình 1 trở lại vào hàng đợi chạy và phân phối nó một lần nữa.
- **Time slot 4:** Một quá trình từ tệp `input/proc/m1s` được tải với PID là 3 và ưu tiên là 15. CPU 3 phân phối quá trình này. CPU 2 đặt quá trình 2 trở lại vào hàng đợi chạy và phân phối nó một lần nữa.
- **Time slot 5:** CPU 0 đặt quá trình 1 trở lại vào hàng đợi chạy và phân phối nó một lần nữa.
- **Time slot 6:** Một quá trình từ tệp `input/proc/s2` được tải với PID là 4 và ưu tiên là 120. CPU 1 phân phối quá trình này. CPU 2 và CPU 3 đặt các quá trình tương ứng của họ trở lại vào hàng đợi chạy và phân phối chúng một lần nữa.
- **Time slot 7:** Một quá trình từ tệp `input/proc/m0s` được tải với PID là 5 và ưu tiên là 120. CPU 0 phân phối quá trình này.
- **Time slot 8:** CPU 1 và CPU 3 đặt các quá trình tương ứng của họ trở lại vào hàng đợi chạy và phân phối chúng một lần nữa. CPU 2 làm tương tự với quá trình 2.
- **Time slot 9:** Một quá trình từ tệp `input/proc/p1s` được tải với PID là 6 và ưu tiên là 15. CPU 0 phân phối quá trình này.
- **Time slot 10:** CPU 1 và CPU 3 đặt các quá trình tương ứng của họ trở lại vào hàng đợi chạy và phân phối chúng một lần nữa. CPU 2 làm tương tự với quá trình 2.
- **Time slot 11:** Một quá trình từ tệp `input/proc/s0` được tải với PID là 7 và ưu tiên là 38. CPU 0 phân phối quá trình 6.
- **Time slot 12:** CPU 1 phân phối quá trình 7. CPU 2 phân phối quá trình 2. CPU 3 hoàn thành xử lý quá trình 3 và phân phối quá trình 5.
- **Time slot 13:** CPU 0 phân phối quá trình 6. CPU 2 hoàn thành xử lý quá trình 2.
- **Time slot 14:** CPU 3 phân phối quá trình 5. CPU 1 phân phối quá trình 7.
- **Time slot 15:** CPU 2 phân phối quá trình 4. CPU 0 phân phối quá trình 6. CPU 3 hoàn thành xử lý quá trình 5 và phân phối quá trình 1.
- **Time slot 16:** Một quá trình từ tệp `input/proc/s1` được tải với PID là 8 và ưu tiên là 0. CPU 1 phân phối quá trình này.
- **Time slot 17:** CPU 2 phân phối quá trình 7. CPU 3 phân phối quá trình 4. CPU 0 phân phối quá trình 6.
- **Time slot 18:** CPU 1 phân phối quá trình 8.
- **Time slot 19:** CPU 2 phân phối quá trình 7. CPU 3 phân phối quá trình 4. CPU 0 hoàn thành xử lý quá trình 6 và phân phối quá trình 1.
- **Time slot 20:** CPU 1 phân phối quá trình 8.
- **Time slot 21:** CPU 0 hoàn thành xử lý quá trình 1 và dừng lại. CPU 3 hoàn thành xử lý quá trình 4 và dừng lại. CPU 2 phân phối quá trình 7.
- **Time slot 22:** CPU 1 phân phối quá trình 8.
- **Time slot 23:** CPU 2 phân phối quá trình 7. CPU 1 hoàn thành xử lý quá trình 8 và dừng lại.
- **Time slot 24:** Không có hành động nào được thực hiện.
- **Time slot 25:** CPU 2 phân phối quá trình 7.
- **Time slot 26:** Không có hành động nào được thực hiện.
- **Time slot 27:** CPU 2 phân phối quá trình 7.
- **Time slot 28:** CPU 2 hoàn thành xử lý quá trình 7 và dừng lại.

### 2.3.3 os\_1\_mlq\_paging\_small\_4K

## os\_1\_mlq\_paging\_small\_4K input

2 4 8  
4096 16777216 0 0 0  
1 p0s 130  
2 s3 39  
4 m1s 15  
6 s2 120  
7 m0s 120  
9 p1s 15  
11 s0 38  
16 s1 0

Output:

```

Time slot 0
1d routine
3 Time slot 1
4 Loaded a process at input/proc/#0, PID: # PRI: 10
CPU 0: Dispatched process 1
5 Time slot 2
6 Loaded a process at input/proc/#4, PID: 2 PRI: 30
CPU 1: Dispatched process 2
7 Time slot 3
8 Loaded a process at input/proc/#3, PID: 2 PRI: 30
CPU 1: Dispatched process 2
9 Time slot 4
10 CPU 0: Put process 1 to run queue
11 CPU 0: Dispatched process 1
12 Time slot 5
13 Loaded a process at input/proc/mis, PID: 3 PRI: 15
CPU 1: Dispatched process 3
14 Check free rg @
15 print_list_rg:
16 rg[>300]
17 rg[<=300]
18 CPU 1: Put process 1 to run queue
19 CPU 1: Dispatched process 2
20 Time slot 6
21 CPU 0: Put process 2 to run queue
22 CPU 1: Dispatched process 2
23 Check remaining registers
24 rg @
25 rg[>300->600]
26 Time slot 7
27 CPU 0: Put process 1 to run queue
28 CPU 1: Dispatched process 1
29 Time slot 8
30 Loaded a process at input/proc/bis, PID: 6 PRI: 15
CPU 1: Dispatched process 2
31 Check free rg @
32 print_list_rg:
33 rg[>300]
34 rg[<=300]
35 Check remaining registers
36 rg @
37 print_list_rg:
38 rg[>300]
39 rg[<=300]
40 Check remaining registers
41 rg @
42 Check remaining registers
43 rg @
44 Dumping memory content
45 CPU 2: Dispatched process 4
46 Time slot 9
47 CPU 0: Put process 3 to run queue
48 CPU 1: Dispatched process 3
49 CPU 2: Put process 4 to run queue
50 CPU 0: Dispatched process 5
51 CPU 1: Dispatched process 3
52 CPU 2: Dispatched process 4
53 CPU 0: Put process 1 to run queue
54 CPU 1: Dispatched process 5
55 CPU 2: Dispatched process 4
56 CPU 0: Put process 2 to run queue
57 CPU 1: Dispatched process 3
58 CPU 2: Dispatched process 4
59 CPU 0: Put process 3 to run queue
60 CPU 1: Dispatched process 3
61 CPU 2: Dispatched process 2
62 CPU 0: Put process 4 to run queue
63 CPU 1: Dispatched process 2
64 CPU 2: Dispatched process 3
65 Check remaining registers
66 rg @
67 rg[>300->400]
68 CPU 1: Put process 2 to run queue
69 CPU 1: Dispatched process 2
70 Time slot 10
71 CPU 0: Put process 5 to run queue
72 Check free rg @
73 print_list_rg:
74 rg[>200->400]
75 CPU 0: Put process 3 to run queue
76 CPU 1: Dispatched process 6
77 rg[<=300]
78 Check remaining registers
79 Time slot 11
80 Loaded a process at input/proc/s0, PID: 4 PRI: 120
CPU 1: Dispatched process 1
81 Time slot 12
82 CPU 0: Put process 2 to run queue
83 CPU 1: Dispatched process 2
84 CPU 2: Put process 3 to run queue
85 CPU 3: Dispatched process 3
86 Check free rg @
87 print_list_rg:
88 rg[>300]
89 rg[<=300]
90 Check remaining registers
91 rg @
92 Check remaining registers
93 rg @
94 Dumping memory content
95 CPU 0: Processed 3 has finished
96 CPU 1: Dispatched process 4
97 Time slot 13
98 Loaded a process at input/proc/s0, PID: 7 PRI: 38
CPU 0: Put process 6 to run queue
99 CPU 1: Put process 7 to run queue
100 CPU 2: Put process 8 to run queue
101 CPU 3: Put process 9 to run queue
102 Time slot 14
103 CPU 0: Put process 4 to run queue
104 CPU 1: Dispatched process 7
105 CPU 2: Put process 5 to run queue
106 CPU 3: Dispatched process 6
107 CPU 0: Put process 2 to run queue
108 CPU 1: Dispatched process 2
109 CPU 2: Processed 2 has finished
110 CPU 3: Put process 6 to run queue
111 CPU 0: Dispatched process 6
112 CPU 1: Dispatched process 5
113 CPU 2: Dispatched process 5
114 CPU 3: Dispatched process 5
115 TLS region@ offset=20 value=102
116 print_pttbl: 0 - 312
117 00000000: 00000005
118 00000004: 00000000
119 Dumping memory content
120 Address: 476, Value: 100
121 Time slot 15
122 Segmentation fault write
123 CPU 0: Put process 7 to run queue
124 CPU 1: Dispatched process 7
125 CPU 2: Put process 4 to run queue
126 CPU 3: Dispatched process 4
127 Time slot 16
128 CPU 0: Put process 6 to run queue
129 CPU 1: Processed 5 has finished
130 CPU 2: Dispatched process 4
131 TLS region@ offset=20
132 print_pttbl: 0 - 312
133 00000000: 00000000
134 Dumping memory content
135 Address: 476, Value: 100
136 CPU 0: Dispatched process 6
137 CPU 1: Dispatched process 6
138 CPU 2: Dispatched process 7
139 CPU 3: Dispatched process 7
140 Loaded a process at input/proc/s1, PID: 8 PRI: 0
141 CPU 2: Put process 4 to run queue
142 CPU 3: Dispatched process 5
143 CPU 0: Put process 2 to run queue
144 CPU 1: Dispatched process 2
145 CPU 3: Dispatched process 7
146 Segmentation fault write
147 Time slot 17
148 CPU 0: Dispatched process 6
149 CPU 1: Dispatched process 6
150 Time slot 18
151 CPU 2: Put process 6 to run queue
152 CPU 3: Dispatched process 8
153 CPU 0: Put process 7 to run queue
154 CPU 1: Dispatched process 7
155 Time slot 19
156 CPU 2: Processed 6 has finished
157 CPU 3: Dispatched process 1
158 Segmentation fault read
159 CPU 0: Put process 6 to run queue
160 CPU 1: Dispatched process 6
161 Time slot 20
162 CPU 2: Put process 8 to run queue
163 CPU 3: Dispatched process 8
164 CPU 0: Put process 7 to run queue
165 CPU 1: Dispatched process 7
166 Check free rg @
167 print_list_rg:
168 rg[>300->500]
169 rg[<=300]
170 Check remaining registers
171 CPU 0: Processed 6 has finished
172 Time slot 21
173 rg @
174 rg[>200->300]
175 CPU 0 stopped
176 Time slot 22
177 CPU 1: Processed 1 has finished
178 CPU 2: Stopped
179 Time slot 23
180 CPU 1: Put process 2 to run queue
181 CPU 2: Dispatched process 8
182 CPU 3: Put process 7 to run queue
183 CPU 4: Dispatched process 7
184 Time slot 24
185 CPU 2: Processed 8 has finished
186 CPU 2: Stopped
187 Time slot 25
188 CPU 3: Put process 7 to run queue
189 CPU 4: Dispatched process 7
190 Time slot 26
191 Time slot 25
192 Time slot 26
193 CPU 3: Put process 7 to run queue
194 CPU 4: Dispatched process 7
195 Time slot 27
196 CPU 3: Processed 7 has finished
197 CPU 4: Stopped

```



## Giải thích

- Time slot 0: Bắt đầu một chu kỳ mới. ld\_routine: Một routine tải được gọi.
- Time slot 1: Một tiến trình được tải từ input/proc/p0s với PID là 1 và độ ưu tiên là 130. CPU 3 được chọn để thực thi tiến trình này.
- Time slot 2: Một tiến trình khác được tải từ input/proc/s3 với PID là 2 và độ ưu tiên là 39. CPU 0 được chọn để thực thi tiến trình này.
- Time slot 3: CPU 3 đưa tiến trình 1 vào hàng đợi chờ và tiếp tục thực thi tiến trình 1.
- Time slot 4: Một tiến trình mới được tải từ input/proc/m1s với PID là 3 và độ ưu tiên là 15. CPU 0 đưa tiến trình 2 vào hàng đợi chờ và tiếp tục thực thi tiến trình 2. CPU 2 được chọn để thực thi tiến trình 3.
- Time slot 5: CPU 3 đưa tiến trình 1 vào hàng đợi chờ và tiếp tục thực thi tiến trình 1.
- Time slot 6: Một tiến trình mới được tải từ input/proc/s2 với PID là 4 và độ ưu tiên là 120. CPU 0 và CPU 2 đưa các tiến trình hiện tại của họ vào hàng đợi chờ và tiếp tục thực thi tiến trình mới. CPU 1 được chọn để thực thi tiến trình 4.
- Time slot 7: Một tiến trình mới được tải từ input/proc/m0s với PID là 5 và độ ưu tiên là 120. CPU 3 đưa tiến trình 1 vào hàng đợi chờ và tiếp tục thực thi tiến trình 5.
- Time slot 8: CPU 2 và CPU 0 đưa các tiến trình hiện tại của họ vào hàng đợi chờ và tiếp tục thực thi tiến trình mới. CPU 1 đưa tiến trình 4 vào hàng đợi chờ và tiếp tục thực thi tiến trình 4.
- Time slot 9: Một tiến trình mới được tải từ input/proc/p1s với PID là 6 và độ ưu tiên là 15. CPU 3 đưa tiến trình 5 vào hàng đợi chờ và tiếp tục thực thi tiến trình 6.
- ...
- Time slot 16: Một tiến trình mới được tải từ input/proc/s1 với PID là 8 và độ ưu tiên là 0. CPU 0 đưa tiến trình 7 vào hàng đợi chờ và tiếp tục thực thi tiến trình 8.
- Time slot 17: CPU 3, CPU 2 và CPU 1 đưa các tiến trình hiện tại của họ vào hàng đợi chờ và tiếp tục thực thi tiến trình mới.
- Time slot 18: CPU 0 đưa tiến trình 8 vào hàng đợi chờ và tiếp tục thực thi tiến trình 8.
- Time slot 19: CPU 3 thông báo rằng tiến trình 6 đã hoàn thành và tiếp tục thực thi tiến trình 1. CPU 2 và CPU 1 đưa các tiến trình hiện tại của họ vào hàng đợi chờ và tiếp tục thực thi tiến trình mới.
- Time slot 20: CPU 0 đưa tiến trình 8 vào hàng đợi chờ và tiếp tục thực thi tiến trình 8.
- Time slot 21: CPU 3 thông báo rằng tiến trình 1 đã hoàn thành và dừng lại. CPU 2 và CPU 1 đưa các tiến trình hiện tại của họ vào hàng đợi chờ và tiếp tục thực thi tiến trình mới. CPU 1 thông báo rằng tiến trình 4 đã hoàn thành và dừng lại.
- Time slot 22: CPU 0 đưa tiến trình 8 vào hàng đợi chờ và tiếp tục thực thi tiến trình 8.
- Time slot 23: CPU 0 thông báo rằng tiến trình 8 đã hoàn thành và dừng lại. CPU 2 đưa tiến trình 7 vào hàng đợi chờ và tiếp tục thực thi tiến trình 7.
- Time slot 24 và Time slot 25: Không có sự kiện nào xảy ra.
- Time slot 26 và Time slot 27: CPU 2 đưa tiến trình 7 vào hàng đợi chờ và tiếp tục thực thi tiến trình 7.
- Time slot 28: CPU 2 thông báo rằng tiến trình 7 đã hoàn thành và dừng lại.



#### 2.3.4 os\_1\_singleCPU\_mlq\_paging

os\_1\_singleCPU\_mlq\_paging input

```
2 1 8
1048576 16777216 0 0 0
1 s4 4
2 s3 3
4 mls 2
6 s2 3
7 m0s 3
9 p1s 2
11 s0 1
16 s1 0
```

Output:

```
1 TIME slot 0      51 TIME slot 37
2 id_routine       52 CPU 0: Put process 7 to run queue
3 TIME slot 1      53 CPU 0: Dispatched process 8
4 Loaded a process at input/proc/s4, PID: 54 TIME slot 18
5 CPU 0: Dispatched process 1 55 CPU 0: Put process 8 to run queue
6 TIME slot 2      56 CPU 0: Dispatched process 8
7 Loaded a process at input/proc/s3, PID: 57 CPU 0: Dispatched process 8
8 CPU 0: Put process 1 to run queue
9 CPU 0: Dispatched process 2 58 TIME slot 20
10 CPU 0: Dispatched process 3 59 CPU 0: Put process 8 to run queue
11 TIME slot 4      60 CPU 0: Dispatched process 8
12 Loaded a process at input/proc/m1s, PID: 61 CPU 0: Put process 6 to run queue
13 TIME slot 5      62 CPU 0: Dispatched process 2
14 CPU 0: Put process 2 to run queue
15 CPU 0: Dispatched process 3 63 CPU 0: Put process 8 to run queue
16 TIME slot 6      64 CPU 0: Dispatched process 8
17 CPU 0: Put process at input/proc/s2, PID: 65 CPU 0: Dispatched process 8
18 TIME slot 7      66 CPU 0: Dispatched process 24
19 Loaded a process at input/proc/m0s, PID: 67 CPU 0: Dispatched process 8 has finished
20 CPU 0: Put process 3 to run queue 68 CPU 0: Dispatched process 7
21 CPU 0: Dispatched process 3 69 TIME slot 25
22 CPU 0: Put process 7 to run queue
23 Check free rg 6 70 TIME slot 26
24 print_list_rg: 71 TIME slot 27
25 rg[0->300] 72 TIME slot 28
26 rg[0->0] 73 TIME slot 29
27 Check remaining registers 74 CPU 0: Put process 7 to run queue
28 rg 1 75 CPU 0: Dispatched process 7
29 rg[300->400]
30 TIME slot 8      76 CPU 0: Put process 6 to run queue
31 TIME slot 9      77 CPU 0: Dispatched process 3
32 Loaded a process at input/proc/p1s, PID: 78 CPU 0: Put process 7 to run queue
33 CPU 0: Put process 3 to run queue
34 CPU 0: Dispatched process 6 79 CPU 0: Dispatched process 7
35 TIME slot 10     80 CPU 0: Put process 7 has finished
36 TIME slot 11     81 CPU 0: Dispatched process 3
37 Loaded a process at input/proc/s0, PID: 82 CPU 0: Dispatched process 2
38 CPU 0: Put process 6 to run queue
39 CPU 0: Dispatched process 7 83 print_list_rg:
40 CPU 0: Dispatched process 7 84 rg[200->300]
41 TIME slot 12     85 rg[0->0]
42 TIME slot 13     86 CPU 0: Put process 7 to run queue
43 CPU 0: Dispatched process 7 87 Check remaining registers
44 CPU 0: Dispatched process 7 88 rg 1
45 TIME slot 14     89 rg[300->400]
46 TIME slot 15     90 CPU 0: Put process 7 to run queue
47 CPU 0: Dispatched process 7 91 TIME slot 34
48 CPU 0: Dispatched process 7 92 Check free rg 1
49 TIME slot 16     93 print_list_rg:
50 Loaded a process at input/proc/s1, PID: 94 rg[0->400]
```



### Giải thích

- Time slot 0: Quá trình tải được bắt đầu.
- Time slot 1: Một tiến trình từ `input/proc/s4` được tải với PID 1 và ưu tiên 4. Nó được gửi đến CPU 0.
- Time slot 2: Một tiến trình khác từ `input/proc/s3` được tải với PID 2 và ưu tiên 3.
- Time slot 3: Tiến trình 1 được đưa vào hàng đợi chạy và tiến trình 2 được gửi đi.
- Time slots 4-7: Nhiều tiến trình khác được tải và gửi đi. Hệ thống kiểm tra các thanh ghi trống và in ra danh sách.
- Time slots 8-16: Nhiều tiến trình khác được tải và gửi đi. Tiến trình mới được gửi vào hàng đợi chạy khi có tiến trình được gửi đi.
- Time slot 24: Tiến trình 8 đã hoàn thành. Tiến trình 7 được gửi đi.
- Time slots 25-33: Tiến trình 7 được đưa vào hàng đợi chạy và gửi đi lặp đi lặp lại. Nó hoàn thành tại time slot 33.
- Time slots 34-45: Tiến trình 3 và 6 được đưa vào hàng đợi chạy, gửi đi và hoàn thành.
- Time slots 46-52: Tiến trình 5 được đưa vào hàng đợi chạy, gửi đi và gặp lỗi phân đoạn. Nó hoàn thành tại time slot 52.
- Time slots 53-61: Tiến trình 2 được đưa vào hàng đợi chạy, gửi đi và hoàn thành.
- Time slots 62-73: Tiến trình 4 được đưa vào hàng đợi chạy, gửi đi và hoàn thành.
- Time slots 74-78: Tiến trình 1 được đưa vào hàng đợi chạy, gửi đi và hoàn thành. CPU 0 dừng lại.



## 2.4 Question

### 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?

### Answer

Ưu điểm của việc thiết kế với nhiều phân đoạn bộ nhớ (memory segment) hoặc nhiều vùng bộ nhớ (memory area) trong bài tập lớn này bao gồm:

1. Cải thiện tính ổn định và bảo mật của hệ thống. Bằng cách chia bộ nhớ thành nhiều phân đoạn, hệ điều hành có thể cách ly tốt hơn các tiến trình khác nhau. Điều này giúp ngăn ngừa truy cập bộ nhớ hoặc sự can thiệp không mong muốn giữa các phần khác nhau của hệ thống.
2. Quản lý bộ nhớ tốt hơn. Với nhiều phân đoạn bộ nhớ, hệ điều hành có thể dễ dàng quản lý và thao tác với việc sử dụng bộ nhớ của từng tiến trình, giúp việc quản lý bộ nhớ trở nên linh hoạt hơn.
3. Hệ điều hành có thể cấp phát và giải phóng các vùng bộ nhớ theo yêu cầu mà không cần phải liên tục chiếm toàn bộ không gian bộ nhớ ảo của một tiến trình.
4. Việc phân tách các vùng bộ nhớ đơn giản hóa việc triển khai cơ chế phân trang và chuyển đổi địa chỉ từ ảo sang vật lý. Hệ điều hành có thể dễ dàng ánh xạ các vùng bộ nhớ khác nhau của một tiến trình vào các khung bộ nhớ vật lý.
5. Thiết kế với nhiều vùng bộ nhớ, cùng với danh sách theo dõi các vùng bộ nhớ trống `vm_free_list` cung cấp nền tảng để triển khai các cơ chế cấp phát bộ nhớ động, như `malloc()` và `free()` trong mỗi tiến trình.

### Question

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

### Answer

Việc chia địa chỉ thành nhiều hơn 2 cấp trong hệ thống quản lý bộ nhớ sẽ có ảnh hưởng sau:

1. Tăng tính linh hoạt và mức độ chi tiết của quản lý bộ nhớ. Với nhiều mức trang, không gian địa chỉ ảo có thể được chia thành các khối nhỏ hơn (các trang), cho phép sử dụng bộ nhớ vật lý hiệu quả hơn, điều này có thể hữu ích khi xử lý không gian địa chỉ lớn và phức tạp.
2. Tăng độ phức tạp của cấu trúc bảng trang (page-table), vì nó yêu cầu thêm các biến để dịch từ địa chỉ ảo sang địa chỉ vật lý, làm tăng chi phí về mặt bộ nhớ và thời gian dịch.
3. Nhiều cấp phân trang có thể dẫn đến nhiều lần truy cập bộ nhớ hơn, điều này có thể ảnh hưởng đến hiệu suất chung của hệ thống, đặc biệt là đối với các ứng dụng có nhiều truy cập bộ nhớ. Độ phức tạp tăng của cấu trúc bảng trang cũng có thể yêu cầu các cơ chế quản lý và cache phức tạp hơn, chẳng hạn như Bộ đệm Chuyển đổi Địa chỉ (TLB), để giảm thiểu tác động về hiệu suất.
4. Xử lý nhiều mức trang yêu cầu các thuật toán và cấu trúc dữ liệu phức tạp hơn để quản lý bảng trang, cấp phát và giải phóng các khung bộ nhớ vật lý, và xử lý các lỗi trang. Điều này có thể làm cho toàn bộ phần công việc quản lý bộ nhớ trở nên phức tạp hơn và có thể dẫn đến nhiều lỗi hơn.

### Question

What is the advantage and disadvantage of segmentation with paging?



### Answer

Ưu điểm và nhược điểm của phân đoạn (Segmentation) kết hợp với phân trang (Paging) trong quản lý bộ nhớ là:

1. Ưu điểm: Linh hoạt. Phân đoạn bộ nhớ kết hợp với trang cung cấp sự linh hoạt hơn trong quản lý bộ nhớ so với sử dụng chỉ phương pháp phân trang hoặc chỉ phân đoạn. Nó cho phép các chương trình có các phân đoạn bộ nhớ có kích thước thay đổi, đồng thời vẫn sử dụng được các lợi ích của các trang bộ nhớ cố định kích thước để phân bổ và truy cập bộ nhớ một cách hiệu quả.
2. Nhược điểm: Độ phức tạp tăng. Việc triển khai phân đoạn bộ nhớ kết hợp với trang phức tạp hơn so với sử dụng chỉ một trong hai kỹ thuật trên. Nó yêu cầu thêm các thành phần phần cứng và phần mềm để xử lý cả bảng phân vùng và bảng trang, cũng như các cơ chế dịch địa chỉ cần phải bao gồm cả phân vùng và phân trang.

### Question

What will happen if the multi-core system has each CPU core can be run in a different context, and each core has its own MMU and its part of the core (the TLB)? In modern CPU, 2-level TLBs are common now, what is the impact of these new memory hardware configurations to our translation schemes?

### Answer

Nếu một hệ thống đa nhân (multi-core system) có mỗi nhân CPU chạy trong một ngữ cảnh khác nhau, và mỗi nhân có một MMU và phần lõi TLB riêng thì:

1. Mỗi CPU sẽ có không gian nhớ ảo riêng, cho phép các tiến trình chạy trên các nhân khác nhau có không gian địa chỉ ảo riêng của chúng.
2. Với mỗi lõi MMU riêng biệt, việc dịch từ địa chỉ ảo sang địa chỉ vật lý sẽ được thực hiện độc lập, nghĩa là cùng một địa chỉ ảo có thể ánh xạ sang các địa chỉ vật lý trên các MMU khác nhau.
3. TLB lưu các ánh xạ từ địa chỉ ảo sang địa chỉ vật lý, sẽ được cục bộ hóa trên mỗi lõi CPU, cho phép tra cứu các ánh xạ địa chỉ nhanh hơn, vì TLB chỉ cần tìm kiếm trong bộ nhớ đệm của riêng nó, thay vì trong một TLB chung.
4. Với không gian nhớ ảo và dịch địa chỉ độc lập, các tiến trình chạy trên các nhân khác nhau có thể truy cập bộ nhớ đồng thời mà không ảnh hưởng lẫn nhau, có thể giúp cải thiện hiệu suất của hệ thống.
5. Việc duy trì các không gian nhớ ảo và TLB riêng biệt trên mỗi lõi có thể dẫn đến việc sử dụng bộ nhớ nhiều hơn, vì mỗi lõi cần phải lưu trữ các bảng trang và mục nhập TLB của riêng nó.

Sự xuất hiện của TLB 2 cấp trong các CPU hiện đại có những tác động sau đối với các phương pháp dịch địa chỉ trong triển khai hệ điều hành:

1. Với TLB 2 cấp, quá trình dịch địa chỉ cần phải kiểm tra trước TLB cấp 1, nếu không tìm thấy ánh xạ, thì phải kiểm tra TLB cấp 2 lớn hơn. Điều này có thể gây thêm một chút độ trễ so với TLB một cấp.
2. TLB cấp 2 lớn hơn TLB cấp 1, có thể lưu trữ nhiều mục nhập ánh xạ hơn, bao phủ phạm vi rộng hơn các ánh xạ từ địa chỉ ảo sang vật lý. Điều này có thể giảm số lần phải đi tới bảng phân trang (page-table).



### 3 TLB (Translation Lookaside Buffer)

#### 3.1 Tổng quan

##### 3.1.1 Khái niệm

TLB là một thành phần quan trọng trong kiến trúc máy tính để tăng tốc độ dịch địa chỉ ảo thành địa chỉ vật lý khi thực hiện việc truy cập bộ nhớ. Đây là một bộ nhớ cache nhỏ được tích hợp trực tiếp vào bộ điều khiển bộ nhớ (Memory Management Unit - MMU) của một bộ xử lý, cung cấp một phép ánh xạ nhanh chóng từ địa chỉ ảo được sử dụng bởi các lệnh và dữ liệu trong chương trình sang địa chỉ vật lý tương ứng.

##### 3.1.2 Chức năng

TLB hoạt động như một bộ nhớ cache tốc độ cao cho các bản dịch địa chỉ ảo sang địa chỉ vật lý.

- **Phương thức truy cập TLB:** TLB sử dụng các kỹ thuật ánh xạ cache (direct mapped, set associative, fully associative) để lưu trữ các bản dịch.
- **Cài đặt TLB:** TLB chứa các bản dịch địa chỉ ảo sang địa chỉ vật lý được sử dụng gần đây nhất.
- **Truy cập địa chỉ ảo:** Khi CPU tạo ra một địa chỉ ảo → kiểm tra TLB:
  - Nếu tìm thấy bản dịch (TLB hit): địa chỉ vật lý tương ứng được lấy ra.
  - Nếu không tìm thấy bản dịch (TLB miss):
    - \* Số trang được dùng làm chỉ mục để truy cập bảng trang trong bộ nhớ chính.
    - \* Nếu trang không có trong bộ nhớ chính, lỗi trang (page fault) xảy ra và TLB được cập nhật với bản dịch mới.

##### 3.1.3 Ưu và nhược điểm

###### • Ưu điểm:

- TLB giúp giảm thời gian truy cập dữ liệu trong bộ nhớ bằng cách lưu trữ tạm thời các bản dịch địa chỉ ảo sang địa chỉ vật lý.
- TLB cải thiện hiệu suất tổng thể của hệ thống bằng cách giảm thiểu số lần truy cập vào bảng trang chậm hơn.

###### • Nhược điểm:

- Trong quản lý bộ nhớ, mỗi tiến trình có một bảng trang riêng (page table) lưu trữ các bản dịch địa chỉ ảo sang địa chỉ vật lý.
- Việc truy cập bảng trang tốn nhiều thời gian vì nó nằm trong bộ nhớ chính, chậm hơn nhiều so với bộ nhớ cache
- Với các tiến trình có kích thước lớn, việc truy cập bảng trang liên tục gây ra độ trễ đáng kể.



### 3.2 Implementation

```
You, 1 second ago | 1 author (You)
1 #ifndef OSFCFG_H
2 #define OSFCFG_H
3
4 #define MLQ_SCHED 1
5 #define MAX_PRIO 140
6
7 #define CPU_TLB
8 #define CPUTLB_FIXED_TLBSZ
9 #define MM_PAGING
10 #define MM_FIXED_MEMSZ | You, 1 second ago *
11 #define VMDBG 1
12 #define MMDBG 1
13 #define IODUMP 1
14 #define PAGEtbl_DUMP 1
15
16 #endif
17
```

Hình 3.1. Các preprocessor directives (chỉ thị tiền xử lý) cần để chạy TLB

```
You, 1 second ago | 1 author (You)
1 #ifndef OSFCFG_H
2 #define OSFCFG_H
3
4 #define MLQ_SCHED 1
5 #define MAX_PRIO 140
6
7 #define CPU_TLB
8 // #define CPUTLB_FIXED_TLBSZ
9 #define MM_PAGING
// #define MM_FIXED_MEMSZ
10 #define VMDBG 1
#define MMDBG 1
11 #define IODUMP 1
#define PAGEtbl_DUMP 1
12
13 #endif
```

Hình 3.2. Các preprocessor directives cần để chạy TLB và MMU (không có TLB size)

```
You, 1 second ago | 1 author (You)
1 #ifndef OSFCFG_H
2 #define OSFCFG_H
3
4 #define MLQ_SCHED 1
5 #define MAX_PRIO 140
6
7 #define CPU_TLB
8 #define CPUTLB_FIXED_TLBSZ | You, 1 second ago + Uncommitted char
9 #define MM_PAGING
10 // #define MM_FIXED_MEMSZ
11 #define VMDBG 1
12 #define MMDBG 1
13 #define IODUMP 1
14 #define PAGEtbl_DUMP 1
15
16 #endif
17
```

Hình 3.3. Các preprocessor directives cần để chạy TLB và MMU

#### 3.2.1 cpu-tlb.c

tlballoc(): cấp phát một trang (page) cho một quy trình (process) trong bộ nhớ và cập nhật thông tin về trang đó trong bộ nhớ đệm TLB.

```
1 int tlballoc(struct pcb_t *proc, uint32_t size, uint32_t reg_index)
2 {
3     int addr, val;
4
5     /* By default using vmaid = 0 */
6     val = __alloc(proc, 0, reg_index, size, &addr);
7     /* TODO update TLB CACHED frame num of the new allocated page(s) */
8     /* by using tlb_cache_read()/tlb_cache_write() */
9     int pgnum = PAGING_PGN(addr);
10    int frmnum = PAGING_FPN(proc->mm->pgd[pgnum]);
11    tlb_cache_write(proc->tlb, proc->pid, pgnum, frmnum);
12    return val;
13 }
```

tlbread(): được sử dụng để đọc dữ liệu từ bộ nhớ ảo của một quy trình thông qua TLB (Translation Lookaside Buffer) và cập nhật thông tin trong TLB sau mỗi lần đọc.

```
1 int tlbread(struct pcb_t * proc, uint32_t source,
2             uint32_t offset, uint32_t destination) {
3     int val;
4     BYTE data, frmnum = -1;
5     int pgnum = PAGING_PGN(proc->mm->symrgtbl[source].rg_start + offset);
6     int off = PAGING_OFFSET(proc->mm->symrgtbl[source].rg_start + offset);
7     tlb_cache_read(proc->tlb, proc->pid, pgnum, &frmnum);
8     /* TODO retrieve TLB CACHED frame num of accessing page(s) */
9     /* by using tlb_cache_read()/tlb_cache_write() */
10    /* frmnum is return value of tlb_cache_read/write value*/
11    #ifdef IODUMP
12        if (frmnum >= 0)
```



```
13     printf("TLB hit at read region=%d offset=%d\n",
14             source, offset);
15     else
16         printf("TLB miss at read region=%d offset=%d\n",
17                 source, offset);
18 #ifdef PAGETBL_DUMP
19     print_pgtbl(proc, 0, -1); //print max TBL
20 #endif
21     MEMPHY_dump(proc->mram);
22 #endif
23     /* TODO update TLB CACHED with frame num of recent accessing page(s) */
24     /* by using tlb_cache_read()/tlb_cache_write()*/
25     if(frmnum >= 0){
26         int phyaddr = (frmnum << PAGING_ADDR_FPN_LOBIT) + off;
27         MEMPHY_read(proc->mram, phyaddr, &data);
28         // tlbwrite(proc,data,destination,offset);
29     }
30     else{
31         val = __read(proc, 0, source, offset, &data);
32         frmnum = PAGING_FPN(proc->mm->pgd[pgnum]);
33         tlb_cache_write(proc->tlb, proc->pid, pgnum, frmnum);
34         // tlbwrite(proc,data,destination,offset);
35     }
36     TLBMEMPHY_dump(proc->tlb);
37     // destination = (uint32_t) data;
38     printf("data readed : %d\n", data);
39     return val;
40 }
```

tlbwrite() : được sử dụng để ghi dữ liệu vào bộ nhớ ảo của một quy trình thông qua TLB (Translation Lookaside Buffer) và cập nhật thông tin trong TLB sau mỗi lần ghi.

```
1 int tlbwrite(struct pcb_t * proc, BYTE data,
2             uint32_t destination, uint32_t offset){
3     int val;
4     BYTE frmnum = -1;
5     int pgnum = PAGING_PGN(proc->mm->symrgtbl[destination].rg_start + offset);
6     int off = PAGING_OFFSET(proc->mm->symrgtbl[destination].rg_start + offset);
7     tlb_cache_read(proc->tlb, proc->pid, pgnum, &frmnum);
8     /* TODO retrieve TLB CACHED frame num of accessing page(s) */
9     /* by using tlb_cache_read()/tlb_cache_write()
10    frmnum is return value of tlb_cache_read/write value*/
11
12 #ifdef IODUMP
13     if (frmnum >= 0)
14         printf("TLB hit at write region=%d offset=%d value=%d\n",
15                destination, offset, data);
16     else
17         printf("TLB miss at write region=%d offset=%d value=%d\n",
18                destination, offset, data);
19 }
```



```
18         destination, offset, data);
19 #ifdef PAGETBL_DUMP
20     print_pgtbl(proc, 0, -1); //print max TBL
21 #endif
22     MEMPHY_dump(proc->mram);
23 #endif
24
25 /* TODO update TLB CACHED with frame num of recent accessing page(s)*/
26 /* by using tlb_cache_read()/tlb_cache_write()*/
27 if(frmnum >= 0){//ghi truc tiep vao bo nho
28     int phyaddr = (frmnum << PAGING_ADDR_FPN_LOBIT) + off;
29     MEMPHY_write(proc->mram, phyaddr, data);
30 }
31 else{
32     val = __write(proc, 0, destination, offset, data);
33     uint32_t pte = proc->mm->pgd[pgnum];
34     frmnum = PAGING_FPN(pte);
35     tlb_cache_write(proc->tlb, proc->pid, pgnum, frmnum);
36 }
37 return val;
38 }
```



### 3.2.2 cpu-tlbcache.c

tlb\_cache\_read() : được sử dụng để đọc dữ liệu từ bộ nhớ cache TLB (Translation Lookaside Buffer) và trả về giá trị của dữ liệu được đọc.

```
1 int tlb_cache_read(struct memphy_struct * mp, int pid, int pgnum, BYTE*
2   ↪ value)
3 {
4     if(tlb_entries[pgnum].valid==0) return 0;
5
6     if(tlb_entries[pgnum].pid==pid){
7       TLBMEMPHY_read(mp,pgnum,value);
8       return 1;
9     };
10    return 0;
11 }
```

tlb\_cache\_write() : được sử dụng để ghi dữ liệu vào bộ nhớ cache TLB (Translation Lookaside Buffer) và cập nhật thông tin liên quan đến trang được ghi vào TLB.

```
1 int tlb_cache_write(struct memphy_struct *mp, int pid, int pgnum, BYTE value
2   ↪ )
3 {
4   TLBMEMPHY_write(mp,pgnum,value);
5   tlb_entries[pgnum].pid=pid;
6   tlb_entries[pgnum].valid=1;
7   return 0;
8 }
```

TLBMEMPHY\_read() : được sử dụng để đọc dữ liệu từ bộ nhớ vật lý (physical memory) thông qua TLB (Translation Lookaside Buffer).

```
1 int TLBMEMPHY_read(struct memphy_struct * mp, int addr, BYTE *value){
2   if (mp == NULL)
3     return -1;
4   /* TLB cached is random access by native */
5   *value = mp->storage[addr];
6   return 0;
7 }
```

TLBMEMPHY\_write() : được sử dụng để ghi dữ liệu vào bộ nhớ vật lý thông qua TLB (Translation Lookaside Buffer).

```
1 int TLBMEMPHY_write(struct memphy_struct * mp, int addr, BYTE data){
2   if (mp == NULL)
3     return -1;
4   /* TLB cached is random access by native */
5   mp->storage[addr] = data;
6   return 0;
7 }
```

### 3.3 Result

### 3.3.1 os\_1\_singleCPU\_mlq

## os\_1\_singleCPU\_mlq input

2 1 8  
1 s4 4  
2 s3 3  
4 m1s 2  
6 s2 3  
7 m0s 3  
9 p1s 2  
11 s0 1  
16 s1 0

m0s	m1s	p1s
1 6	1 6	1 11
alloc 300 0	alloc 300 0	calc
alloc 100 1	alloc 100 1	calc
free 0	free 0	calc
alloc 100 2	alloc 100 2	calc
write 102 1 20	free 2	calc
write 1 2 1000	free 1	calc

## Output:

```

1 Time slot 0 51 Time slot 17 101 rg[0->0] 152 Check remaining registers 166 CPU 0: Dispatched process 1
2 Id_routine 52 CPU 0: Put process 7 to run queue 102 rg[0->0] 153 rg[0->0=>0] 167 CPU 0: Dispatched process 1
3 Time slot 1 53 CPU 0: Dispatched process 8 103 Check remaining registers 168 Time slot 72
4 Loaded a process at input/proc/s4, PID: 54 Time slot 18 104 rg[0->0] 154 rg[0->0=>0] 169 Time slot 73
5 CPU 0: Dispatched process 1 55 Time slot 19 105 CPU 0: Put process 8 to run queue 155 Time slot 74
6 CPU 0: Dispatched process 1 56 Time slot 20 106 CPU 0: Processed 3 has finished 156 Time slot 75
7 Loaded a process at input/proc/s3, PID: 57 Time slot 21 107 CPU 0: Dispatched process 6 157 Time slot 76
8 Time slot 2 58 Time slot 22 108 CPU 0: Put process 5 to run queue 158 CPU 0: Dispatched process 2 203 CPU 0: Put process 1 to run queue
9 CPU 0: Put process 1 to run queue 59 Time slot 23 109 CPU 0: Put process 7 to run queue 159 CPU 0: Dispatched process 1 204 CPU 0: Dispatched process 1
10 CPU 0: Dispatched process 2 60 Time slot 24 110 CPU 0: Put process 6 to run queue 160 Time slot 77
11 Time slot 4 61 CPU 0: Dispatched process 8 111 CPU 0: Dispatched process 6 161 CPU 0: Put process 2 to run queue 207 CPU 0: Put process 1 to run queue
12 Loaded a process at input/proc/m1s, PID 62 Time slot 25 112 Time slot 38 162 CPU 0: Dispatched process 4 208 CPU 0: Dispatched process 1
13 Time slot 5 63 CPU 0: Dispatched process 7 113 Time slot 39 163 CPU 0: Dispatched process 4 209 Time slot 78
14 CPU 0: Put process 2 to run queue 64 CPU 0: Put process 8 to run queue 114 CPU 0: Put process 6 to run queue 164 Time slot 59
15 CPU 0: Dispatched process 3 65 Time slot 40 115 CPU 0: Dispatched process 6 165 CPU 0: Put process 4 to run queue 210 Time slot 79
16 Time slot 6 66 Time slot 41 116 CPU 0: Processed 8 has finished 166 CPU 0: Dispatched process 5 211 Time slot 80
17 Loaded a process at input/proc/s2, PID: 67 Time slot 25 117 CPU 0: Dispatched process 7 167 CPU 0: Dispatched process 5 212 Time slot 80
18 Time slot 7 68 Time slot 42 118 CPU 0: Put process 6 to run queue 168 TBL miss at write region1 offset>20 value 213 Time slot 81
19 Loaded a process at input/proc/mbs, PID 69 Time slot 43 119 CPU 0: Dispatched process 6 169 CPU 0: Dispatched process 6 214 Time slot 81
20 CPU 0: Put process 3 to run queue 70 Time slot 44 120 CPU 0: Put process 7 to run queue 170 CPU 0: 0000000000000003 215 CPU 0: Put process 1 to run queue
21 CPU 0: Dispatched process 3 71 Time slot 45 121 CPU 0: Put process 4 to run queue 171 CPU 0: 0000000000000002 216 CPU 0: Dispatched process 1
22 Check free rg 0 72 CPU 0: Dispatched process 7 122 CPU 0: Put process 6 to run queue 172 Dumping memphy content 217 Time slot 82
23 print_list_rg 73 Time slot 27 123 CPU 0: Dispatched process 6 173 CPU 0: Put process 1 to run queue 218 Time slot 83
24 rg[0=>300] 74 Time slot 28 124 Time slot 46 174 Separation fault write 219 CPU 0: Put process 1 to run queue
25 rg[0=>0] 75 CPU 0: Put process 7 to run queue 125 CPU 0: Dispatched process 2 175 Time slot 62 220 CPU 0: Dispatched process 1
26 rg[0=>0=>0] 76 CPU 0: Dispatched process 7 126 CPU 0: Dispatched process 2 176 CPU 0: Processed 5 has finished 221 Time slot 84
27 Check remaining registers 77 Time slot 29 127 CPU 0: Dispatched process 2 177 CPU 0: Processed 6 has finished 222 Time slot 85
28 rg 1 78 Time slot 30 128 CPU 0: Put process 2 to run queue 178 Time slot 63 223 CPU 0: Dispatched process 1 to run queue
29 rg[0=>0=>0] 79 CPU 0: Put process 7 to run queue 129 CPU 0: Put process 2 to run queue 179 Time slot 64 224 CPU 0: Dispatched process 1
30 CPU 0: Dispatched process 7 80 Time slot 31 130 CPU 0: Dispatched process 4 180 CPU 0: Put process 2 to run queue 225 Time slot 86
31 Time slot 8 81 Time slot 32 131 CPU 0: Dispatched process 4 181 CPU 0: Dispatched process 4 226 Time slot 86
32 Time slot 9 82 Time slot 33 132 CPU 0: Put process 5 to run queue 182 Time slot 65 227 Time slot 87
33 Loaded a process at input/proc/p1s, PID 83 CPU 0: Put process 7 to run queue 133 CPU 0: Put process 4 to run queue 183 Time slot 66 228 Time slot 87
34 CPU 0: Put process 3 to run queue 84 CPU 0: Dispatched process 7 134 CPU 0: Dispatched process 5 184 CPU 0: Put process 4 to run queue 229 CPU 0: Dispatched process 1
35 CPU 0: Dispatched process 6 85 Time slot 33 135 Time slot 49 185 CPU 0: Dispatched process 2 230 Time slot 88
36 CPU 0: Dispatched process 6 86 CPU 0: Processed 7 has finished 136 CPU 0: Dispatched process 3 186 Time slot 67 231 CPU 0: Put process 1 to run queue
37 Time slot 11 87 Time slot 34 137 CPU 0: Put process 5 to run queue 187 CPU 0: Processed 2 has finished 232 CPU 0: Dispatched process 1
38 CPU 0: Dispatched process 3 88 Check free rg 2 138 CPU 0: Dispatched process 2 188 CPU 0: Dispatched process 4 233 Time slot 91
39 CPU 0: Put process 6 to run queue 89 print_list_rg 139 CPU 0: Time slot 51 189 Time slot 68 234 CPU 0: Put process 1 to run queue
40 CPU 0: Dispatched process 7 90 rg[0=>300] 140 CPU 0: Time slot 52 190 Time slot 69 235 CPU 0: Put process 1 to run queue
41 Time slot 12 91 CPU 0: Put process 2 to run queue 141 CPU 0: Put process 4 to run queue 191 CPU 0: Put process 4 to run queue 236 CPU 0: Dispatched process 1
42 Time slot 13 92 CPU 0: Dispatched process 4 142 CPU 0: Dispatched process 4 192 CPU 0: Dispatched process 4 237 Time slot 92
43 CPU 0: Put process 7 to run queue 93 Check remaining registers 143 Time slot 53 193 Time slot 70 238 CPU 0: Put process 1 to run queue
44 CPU 0: Dispatched process 7 94 rg 1 144 Time slot 54 194 Time slot 71 239 CPU 0: Dispatched process 1
45 Time slot 14 95 rg[0=>0=>0] 145 CPU 0: Put process 4 to run queue 195 CPU 0: Processed 4 has finished 240 Time slot 93
46 Time slot 15 96 Time slot 55 146 CPU 0: Dispatched process 5 196 CPU 0: Dispatched process 1 241 CPU 0: Put process 1 to run queue
47 CPU 0: Put process 7 to run queue 97 Time slot 34 147 Check Free rg 0 197 Time slot 72 242 CPU 0: Dispatched process 1
48 CPU 0: Dispatched process 7 98 Check free rg 1 148 CPU 0: Dispatched process 5 198 Time slot 73 243 CPU 0: Put process 1 to run queue
49 CPU 0: Dispatched process 7 99 rg[0=>300] 149 rg[0=>0] 199 CPU 0: Put process 1 to run queue 244 CPU 0: Dispatched process 1
50 CPU 0: Dispatched process 7 100 Time slot 56 200 CPU 0: Dispatched process 1 245 CPU 0: stopped
51 Loaded a process at input/proc/s1, PID: 100 rg[0=>0=>0] 201 CPU 0: Dispatched process 1

```



## Giải thích

- Time slot 0-33:
  - Tiến trình PID 1 được tải lên CPU 0 ở time slot 1 với ưu tiên là 4, sau đó tiến trình này được chạy và hoàn thành ở time slot 33.
  - Tiến trình PID 2 được tải lên ở time slot 2 với ưu tiên là 3 và được chạy trên CPU 0 từ time slot 3 đến time slot 33.
  - ...
  - Dòng "print\_list\_rg" hiển thị thông tin về phân vùng của thanh ghi. Trong trường hợp này, có một phân vùng trong thanh ghi 0 từ 0 đến 300, và không có giá trị nào được gán cho nó.
  - Dòng "Check remaining registers" chỉ ra rằng còn một thanh ghi còn lại, thanh ghi 1, với một phân vùng từ 300 đến 400 được sử dụng.
  - ...
- Time slot 34-44:
  - Sau khi tiến trình PID 7 kết thúc ở time slot 33, CPU 0 tiếp tục chạy tiến trình PID 3 (từ time slot 34 đến time slot 35)
  - ...
- Time slot 45-49:
  - Tiến trình PID 6 kết thúc ở time slot 44. CPU 0 tiếp tục chạy tiến trình PID 2 (từ time slot 45 đến time slot 49).
- Tại time slot 50:
  - Tiến trình PID 5 được đưa vào hàng đợi chạy và CPU 0 chuyển sang chạy tiến trình PID 2.
- Tại time slot 51-52:
  - Tiến trình PID 2 được chạy và sau đó được đưa vào hàng đợi. Tiến trình PID 4 được chạy tiếp theo.
- Tại time slot 53-54:
  - Tiến trình PID 4 được chạy và sau đó được đưa vào hàng đợi. Tiến trình PID 5 được chạy tiếp theo.
  - Sau đó, dòng "Check remaining registers" hiển thị thông tin về các thanh ghi còn lại. Trong trường hợp này, thanh ghi 1 có phân vùng từ 300 đến 400.
- Tại time slot 55-56:
  - Tiến trình PID 5 được chạy và sau đó được đưa vào hàng đợi. Tiến trình PID 2 được chạy tiếp theo.
- Tại time slot 57-58:
  - Tiến trình PID 2 được chạy và sau đó được đưa vào hàng đợi. Tiến trình PID 4 được chạy tiếp theo.
- Tại time slot 59-60:
  - Tiến trình PID 4 được chạy và sau đó được đưa vào hàng đợi.
- Ở time slot 61:
  - Thông báo "TLB miss at write region=1 offset=20 value=102" cho biết có một TLB miss xảy ra khi cố ghi vào vùng 1 của bộ nhớ, ở vị trí 20, với giá trị 102.
  - Một lỗi "Segmentation fault write" xảy ra khi có một TLB miss ở vùng ghi và giá trị 102 không thể được ghi.
- Time slot 64 đến 67:
  - Tiến trình PID 5 kết thúc và tiến trình PID 2 được chạy lại ở time slot 62-63. Sau đó, tiến trình PID 4 được chạy lại.
- Time slot 71 đến 98:
  - Tiến trình PID 4 kết thúc và CPU 0 chuyển sang chạy tiến trình PID 1 .
  - Sau đó, tiến trình PID 1 kết thúc và CPU 0 dừng lại ở time slot 98.

### 3.3.2 os\_1\_tlbsz\_singleCPU\_mlq

**os\_1\_tlbsz\_singleCPU\_mlq input**

2 1 8  
40000  
1048576 16777216 0 0 0  
1 s4 4  
2 s3 3  
4 m1s 2  
6 s2 3  
7 m0s 3  
9 p1s 2  
11 s0 1  
16 s1 0

m0s	m1s	p1s
1 6	1 6	1 11
alloc 300 0	alloc 300 0	calc
alloc 100 1	alloc 100 1	calc
free 0	free 0	calc
alloc 100 2	alloc 100 2	calc
write 102 1 20	free 2	calc
write 1 2 1000	free 1	calc

## Output:



## Giải thích

- Time slot 0-33:
  - Tiến trình PID 1 được tải lên CPU 0 ở time slot 1 với ưu tiên là 4, sau đó tiến trình này được chạy và hoàn thành ở time slot 33.
  - ...
  - Tiến trình PID 4 được tải lên ở time slot 6 với ưu tiên là 3 và được chạy trên CPU 0 từ time slot 7 đến time slot 33.
  - Tiến trình PID 5 được tải lên ở time slot 7 với ưu tiên là 3 và được chạy trên CPU 0 từ time slot 7 đến time slot 33.
  - ...
- Time slot 34-44:
  - Sau khi tiến trình PID 7 kết thúc ở time slot 33, CPU 0 tiếp tục chạy tiến trình PID 3 (từ time slot 34 đến time slot 35)
  - Dòng "Check remaining registers" chỉ ra rằng có một thanh ghi còn lại, thanh ghi 1, có phân vùng từ 300 đến 400.
  - Sau đó tiếp tục chạy tiến trình PID 6 (từ time slot 35 đến time slot 44).
- Time slot 45-49:
  - Tiến trình PID 6 kết thúc ở time slot 44. CPU 0 tiếp tục chạy tiến trình PID 2 (từ time slot 45 đến time slot 49).
- Tại time slot 50:
  - Tiến trình PID 5 được đưa vào hàng đợi chạy và CPU 0 chuyển sang chạy tiến trình PID 2.
- Tại time slot 51-52:
  - Tiến trình PID 2 được chạy và sau đó được đưa vào hàng đợi. Tiến trình PID 4 được chạy tiếp theo.
- Tại time slot 53-54:
  - Tiến trình PID 4 được chạy và sau đó được đưa vào hàng đợi. Tiến trình PID 5 được chạy tiếp theo.
  - Sau đó, dòng "Check remaining registers" hiển thị thông tin về các thanh ghi còn lại. Trong trường hợp này, thanh ghi 1 có phân vùng từ 300 đến 400.
- Tại time slot 55-56:
  - Tiến trình PID 5 được chạy và sau đó được đưa vào hàng đợi. Tiến trình PID 2 được chạy tiếp theo.
- Tại time slot 57-58:
  - Tiến trình PID 2 được chạy và sau đó được đưa vào hàng đợi. Tiến trình PID 4 được chạy tiếp theo.
- Tại time slot 59-60:
  - Tiến trình PID 4 được chạy và sau đó được đưa vào hàng đợi.
- Ở time slot 61:
  - Thông báo "TLB miss at write region=1 offset=20 value=102" cho biết có một TLB miss xảy ra khi cố ghi vào vùng 1 của bộ nhớ, ở vị trí 20, với giá trị 102.
  - Một lỗi "Segmentation fault write" xảy ra khi có một TLB miss ở vùng ghi và giá trị 102 không thể được ghi.
- Time slot 64 đến 67:
  - Tiến trình PID 5 kết thúc và tiến trình PID 2 được chạy lại ở time slot 62-63. Sau đó, tiến trình PID 4 được chạy lại.
- Time slot 71 đến 98:
  - Tiến trình PID 4 kết thúc và CPU 0 chuyển sang chạy tiến trình PID 1 .
  - Sau đó, tiến trình PID 1 kết thúc và CPU 0 dừng lại ở time slot 98.

### 3.3.3 os 1 mlq paging

## os\_1\_mlq\_paging input

2 1 8  
1048576 16777216 0 0 0  
1 s4 4  
2 s3 3  
4 m1s 2  
6 s2 3  
7 m0s 3  
9 p1s 2  
11 s0 1  
16 s1 0

m0s	m1s	p1s	p0s
1 6	1 6	1 11	1 10
alloc 300 0	alloc 300 0	calc	calc
alloc 100 1	alloc 100 1	calc	alloc 300 0
free 0	free 0	calc	alloc 100 4
alloc 100 2	alloc 100 2	calc	free 0
write 102 1 20	free 2	calc	alloc 100 1
write 1 2 1000	free 1	calc	write 100 1 20
		calc	read 1 20 20
		calc	write 103 3 20
		calc	read 3 20 20
		calc	free 4

Output:

```

1 Time slot 0
2   Time slot 1
3     Time slot 1
4       Loaded a process at input/proc/pbs, PID: 1 PRIO: 100
5         CPU 2: Dispatched process 1
6       Time slot 1
7         Loaded a process at input/proc/g3, PID: 2 PRIO: 39
8           CPU 3: Dispatched process 2
9         Time slot 3
10           CPU 2: Put process 1 to run queue
11             CPU 2: Dispatched process 1
12           Time slot 4
13             CPU 2: Put process 2 to run queue
14               CPU 3: Dispatched process 2
15             Check free rg
16             print_list_rg:
17               rg[0->300]
18               rg[300->400]
19             Check remaining registers
20             Time slot 5
21               CPU 2: Put process 1 to run queue
22                 CPU 2: Dispatched process 1
23               Time slot 6
24                 CPU 2: Put process 2 to run queue
25                   CPU 3: Dispatched process 2
26             Check remaining registers
27             Time slot 7
28               CPU 2: Put process 1 to run queue
29                 CPU 2: Dispatched process 1
30               Time slot 8
31                 CPU 2: Put process 2 to run queue
32                   CPU 3: Dispatched process 2
33             Check free rg
34             print_list_rg:
35               rg[0->300]
36               rg[300->400]
37             Dumping memory content
38             CPU 1: Put process 3 to run queue
39             CPU 1: Dispatched process 3
40             Check free rg
41             print_list_rg:
42               rg[0->300]
43               rg[300->400]
44             Check remaining registers
45             Time slot 9
46               CPU 1: Put process 4 to run queue
47                 CPU 1: Dispatched process 4
48               Time slot 10
49                 CPU 2: Put process 3 to run queue
50                   CPU 3: Dispatched process 5
51             Check free rg
52             print_list_rg:
53               rg[0->300]
54               rg[300->400]
55             Check remaining registers
56             Time slot 11
57               CPU 1: Put process 3 to run queue
58                 CPU 1: Dispatched process 3
59               Time slot 12
60                 CPU 2: Put process 2 to run queue
61                   CPU 3: Dispatched process 4
62             Check free rg
63             print_list_rg:
64               rg[0->300]
65               rg[300->400]
66             Check remaining registers
67             Time slot 13
68               CPU 1: Put process 4 to run queue
69                 CPU 1: Dispatched process 4
70               Time slot 14
71                 CPU 2: Put process 3 to run queue
72                   CPU 3: Dispatched process 6
73             Check free rg 1
74             print_list_rg:
75               rg[0->300]
76               rg[300->400]
77             Check remaining registers
78             Time slot 15
79               CPU 2: Put process 1 to run queue
80                 CPU 2: Dispatched process 1
81               Time slot 16
82                 CPU 2: Put process 2 to run queue
83                   CPU 3: Dispatched process 7
84             Check free rg 1
85             print_list_rg:
86               rg[0->300]
87               rg[300->400]
88             Dumping memory content
89             CPU 1: Put process 3 to run queue
90             CPU 1: Dispatched process 3
91             Check free rg 0
92             print_list_rg:
93               rg[0->300]
94               rg[300->400]
95             Check remaining registers
96             Time slot 17
97               CPU 1: Put process 4 to run queue
98                 CPU 1: Dispatched process 4
99               Time slot 18
100                 CPU 2: Put process 5 to run queue
101                   CPU 3: Dispatched process 8
102             Check free rg 0
103             print_list_rg:
104               rg[0->300]
105               rg[300->400]
106             Dumping memory content
107             CPU 1: Put process 4 to run queue
108               CPU 1: Dispatched process 5
109             Check free rg 0
110             print_list_rg:
111               rg[0->300]
112             CPU 2: Put process 2 to run queue
113               CPU 3: Dispatched process 2
114             Time slot 19
115               CPU 2: Put process 3 to have finished
116                 CPU 2: Dispatched process 8
117               CPU 2: Put process 6 to run queue
118                 CPU 3: Dispatched process 4
119             Segmentation fault write
120             Time slot 20
121               CPU 0: Processed 3 has finished
122                 CPU 1: Put process 7 to run queue
123                   CPU 2: Dispatched process 7
124                     CPU 3: Dispatched process 6
125             Tlb miss at read region offset=20
126             print_ptbl: 0 - 512
127             00000000: 00000000
128             Dumping memory content
129             CPU 0: Processed 4 has finished
130             Address: 476, Value: 100
131             Address: 1088, Value: 102
132             dump_ptbl: 100
133             Time slot 15
134             Segmentation fault write
135             CPU 0: Put process 8 to run queue
136               CPU 1: Dispatched process 8
137             CPU 0: Put process 9 to run queue
138               CPU 3: Dispatched process 4
139             Time slot 16
140             Loaded a process at input/proc/s1, PID: 8 PRIO: 0
141               CPU 0: Put process 7 to run queue
142                 CPU 1: Dispatched process 8
143               CPU 0: Put process 1 to run queue
144                 CPU 3: Dispatched process 7
145             Time slot 17
146               CPU 3: Put process 4 to run queue
147                 CPU 3: Dispatched process 4
148               CPU 2: Put process 6 to run queue
149                 CPU 0: Put process 7 to run queue
150                   CPU 1: Dispatched process 8
151                     CPU 2: Dispatched process 7
152                       CPU 3: Dispatched process 8
153                         CPU 1: Put process 8 to run queue
154                           CPU 3: Dispatched process 8
155                         Time slot 18
156                           CPU 3: Processed 4 has finished
157                             CPU 3: Dispatched process 1
158                           Segmentation fault read
159                           CPU 0: Put process 6 to run queue
160                             CPU 2: Dispatched process 6
161                           Time slot 20
162                             CPU 2: Processed 0 has finished
163                               CPU 3: Dispatched process 8
164                                 CPU 0: Put process 8 to run queue
165                                   CPU 1: Dispatched process 8
166                                     CPU 0: Put process 7 to run queue
167                                       CPU 3: Dispatched process 7
168                                         Check free rg
169                                         print_list_rg:
170                                           rg[300->x000]
171                                           rg[0->0]
172             Check remaining registers
173             Time slot 21
174               CPU 3: Processed 1 has finished
175                 CPU 0: Stopped
176               Time slot 22
177                 CPU 1: Put process 8 to run queue
178                   CPU 2: Dispatched process 8
179                     CPU 3: Dispatched process 7
180               Time slot 23
181                 CPU 1: Put process 8 to run queue
182                   CPU 2: Dispatched process 8
183                     CPU 3: Dispatched process 7
184               Time slot 24
185                 CPU 1: Processed 8 has finished
186                   CPU 1: Stopped
187               Time slot 25
188                 CPU 0: Put process 7 to run queue
189                   CPU 1: Dispatched process 7
190                     CPU 0: Dispatched process 7
191               Time slot 26
192                 CPU 0: Put process 7 to run queue
193                   CPU 1: Dispatched process 7
194                     CPU 0: Dispatched process 7
195               Time slot 27
196                 CPU 0: Processed 7 has finished
197                   CPU 0: Stopped
198

```



## Giải thích

Có một số lỗi xảy ra trong quá trình, bao gồm lỗi TLB miss và segmentation fault.

- Time slot 0: Hàm ld\_routine được gọi.
- Time slot 1: Quá trình được tải lên từ tệp p0s với PID là 1 và ưu tiên là 130. CPU 2 chạy quá trình này.
- Time slot 2: Quá trình được tải lên từ tệp s3 với PID là 2 và ưu tiên là 39. CPU 3 chạy quá trình này.
- Time slot 3: CPU 2 đặt quá trình 1 vào hàng đợi chạy và chạy quá trình 1.
- Time slot 4: Quá trình được tải lên từ tệp m1s với PID là 3 và ưu tiên là 15. CPU 3 đặt quá trình 2 vào hàng đợi chạy và chạy quá trình 2. Kiểm tra sự trống trơn của thanh ghi 0 và in ra thông tin của thanh ghi.
- Time slot 5: CPU 2 đặt quá trình 1 vào hàng đợi chạy và chạy quá trình 1.
- Time slot 6: Quá trình được tải lên từ tệp s2 với PID là 4 và ưu tiên là 120. CPU 0 chạy quá trình này. CPU 3 đặt quá trình 2 vào hàng đợi chạy và chạy quá trình 2.
- Time slot 7: Quá trình được tải lên từ tệp m0s với PID là 5 và ưu tiên là 120. CPU 2 đặt quá trình 1 vào hàng đợi chạy và chạy quá trình 5.
- Time slot 8: CPU 1 đặt quá trình 3 vào hàng đợi chạy và chạy quá trình 3. CPU 3 đặt quá trình 2 vào hàng đợi chạy và chạy quá trình 2. CPU 0 đặt quá trình 4 vào hàng đợi chạy và chạy quá trình 4.
- Time slot 9: Quá trình được tải lên từ tệp p1s với PID là 6 và ưu tiên là 15. CPU 2 đặt quá trình 5 vào hàng đợi chạy và chạy quá trình 6.
- Time slot 10: CPU 3 đặt quá trình 2 vào hàng đợi chạy và CPU 1 xử lý xong quá trình 3. CPU 0 đặt quá trình 4 vào hàng đợi chạy, CPU 1 chạy quá trình 5, CPU 3 chạy quá trình 2 và CPU 0 chạy quá trình 4. Kiểm tra sự trống trơn của thanh ghi 0 và in ra thông tin của thanh ghi.
- ...
- Time slot 14: CPU 0 xử lý xong quá trình 5, CPU 1 đặt quá trình 7 vào hàng đợi chạy và chạy quá trình 7, CPU 0 chạy quá trình 1. Xảy ra lỗi đọc nhớ khi thực hiện quá trình.
- Time slot 15: Xảy ra lỗi ghi nhớ khi thực hiện quá trình.
- Time slot 16: Quá trình được tải lên từ tệp s1 với PID là 8 và ưu tiên là 0. CPU 1 đặt quá trình 7 vào hàng đợi chạy và chạy quá trình 8. CPU 0 đặt quá trình 1 vào hàng đợi chạy và chạy quá trình 7.
- Time slot 17: CPU 3 xử lý xong quá trình 4, CPU 3 đặt quá trình 1 vào hàng đợi chạy và chạy quá trình 1. CPU 2 đặt quá trình 6 vào hàng đợi chạy và chạy quá trình 6.
- Time slot 18: CPU 0 đặt quá trình 7 vào hàng đợi chạy và chạy quá trình 7. CPU 1 đặt quá trình 8 vào hàng đợi chạy và chạy quá trình 8.
- Time slot 19: CPU 3 xử lý xong quá trình 1, CPU 3 đặt quá trình 1 vào hàng đợi chạy và chạy quá trình 1. Xảy ra lỗi đọc nhớ khi thực hiện quá trình.
- Time slot 20: CPU 2 xử lý xong quá trình 6 và CPU 2 dừng lại. CPU 1 đặt quá trình 8 vào hàng đợi chạy và chạy quá trình 8. CPU 0 đặt quá trình 7 vào hàng đợi chạy và chạy quá trình 7. Kiểm tra thanh ghi 4 có trống không và in ra thông tin của thanh ghi.
- ...
- Time slot 24: CPU 0 đặt quá trình 7 vào hàng đợi chạy và chạy quá trình 7.
- Time slot 25 - 27: CPU 0 đặt quá trình 7 vào hàng đợi chạy và chạy quá trình 7.



### 3.4 Question

#### Question

What will happen if the multi-core system has each CPU core can be run in a different context, and each core has its own MMU and its part of the core (the TLB)? In modern CPU, 2-level TLBs are common now, what is the impact of these new memory hardware configurations to our translation schemes?

#### Answer

Nếu một hệ thống đa nhân có mỗi lõi CPU có thể chạy trong ngữ cảnh khác nhau, và mỗi lõi có MMU riêng và một phần của lõi (TLB), điều này sẽ ảnh hưởng đến cách chúng ta thực hiện các chương trình dịch và cấu hình phần cứng bộ nhớ.

1. Tăng cường Song song hóa: Với mỗi lõi có MMU và TLB riêng, các hoạt động dịch và bộ nhớ có thể được thực hiện độc lập trên từng lõi. Điều này cho phép tăng cường song song hóa và giảm sự cạnh tranh về tài nguyên dịch giữa các lõi khác nhau. Mỗi lõi có thể xử lý các bản dịch bộ nhớ riêng mà không phải dựa vào các cấu trúc dịch chung hoặc phối hợp với các lõi khác.
2. Giảm chia sẻ TLB: Trong một hệ thống nơi mỗi lõi có TLB riêng, các mục TLB của một lõi không được chia sẻ với các lõi khác. Điều này giảm nhu cầu thường xuyên về việc vô hiệu hóa và xóa các mục TLB cần thiết trong một hệ thống TLB chia sẻ, trong đó các mục TLB cần được đồng bộ hóa trên nhiều lõi. Kết quả là, việc mất TLB và chi phí dịch có thể được giảm thiểu.
3. Tăng dung lượng TLB: Trong một hệ thống đa nhân với mỗi lõi có TLB riêng của mình, dung lượng TLB tổng thể của hệ thống tăng lên. Điều này bởi vì mỗi lõi có TLB riêng của mình và TLB của các lõi khác nhau có thể lưu trữ các bản dịch riêng biệt. Điều này có thể giúp giảm việc mất TLB và cải thiện hiệu suất tổng thể của hệ thống.
4. Phức tạp trong tính nhất quán: Việc có MMU và TLB riêng biệt cho mỗi lõi đưa ra sự phức tạp và thách thức bổ sung trong việc duy trì tính nhất quán bộ nhớ trên các lõi. Khi một lõi sửa đổi một trang bộ nhớ, các lõi khác cần nhận biết sự thay đổi để đảm bảo tính nhất quán dữ liệu. Các giao thức nhất quán như MESI (Modified, Exclusive, Shared, Invalid) hoặc MOESI (Modified, Owned, Exclusive, Shared, Invalid) thường được sử dụng để quản lý tính nhất quán bộ nhớ cache trong hệ thống đa nhân. Những giao thức này cần được mở rộng hoặc sửa đổi để xử lý các yêu cầu tính nhất quán của MMU và TLB riêng biệt.
5. Cô lập bộ nhớ: Với mỗi lõi có MMU riêng, việc thực hiện cách cô lập và bảo vệ bộ nhớ giữa các lõi khác nhau trở nên dễ dàng hơn. Mỗi lõi có thể có không gian địa chỉ ảo và quyền truy cập bộ nhớ riêng của mình, cho phép cô lập mạnh hơn giữa các quy trình chạy trên các lõi khác. Như vậy, việc có MMU và TLB riêng biệt cho mỗi lõi trong một hệ thống đa nhân mang lại những lợi ích như tăng cường song song hóa, giảm sự chia sẻ TLB, tăng dung lượng TLB và cải thiện cô lập bộ nhớ. Tuy nhiên, điều này cũng làm tăng độ phức tạp trong việc quản lý tính nhất quán bộ nhớ và yêu cầu sửa đổi các giao thức nhất quán để đảm bảo tính nhất quán dữ liệu giữa các lõi.

## 4 Put All Together

### Question

What will happen if the synchronization is not handled in your simple OS? Illustrate the problem of your simple OS by example if you have any. Note: You need to run two versions of your simple OS: the program with/without synchronization, then observe their performance based on demo results and explain their differences.

Nhóm sử dụng `struct timer_id_t` để quản lí việc đồng bộ quá trình load các process và chờ timeslot:

```
struct timer_id_t {
    int done;
    int fsh;
    int cpu_wait;
    int time_wait;
    pthread_cond_t event_cond;
    pthread_mutex_t event_lock;
    pthread_cond_t timer_cond;
    pthread_mutex_t timer_lock;
};
```

Ở hàm `cpu_routine` có 2 vòng lặp `while(timer_id->time_wait == 0)` dùng để chờ timeslot đến trước khi in ra, và `while(timer_id->cpu_wait == 0)` để chờ việc load process vào:

```
static void * cpu_routine(void * args) {
    struct timer_id_t * timer_id = ((struct cpu_args*)args)->timer_id;
    int id = ((struct cpu_args*)args)->id;
    /* Check for new process in ready queue */
    int time_left = 0;
    struct pcb_t * proc = NULL;
    while (1) {
        /* Check the status of current process */
        while(timer_id->time_wait==0);

        while(timer_id->cpu_wait==0); //wait for load

        if (proc == NULL && !done) {
            /* No process is running, the we load new process from
             * ready queue */
            proc = get_proc();
        }
    }
}
```

Ở hàm `ld_routine` có một vòng lặp `while(timer_id->time_wait == 0)` để chờ timeslot trước khi in ra:

```
static void * ld_routine(void * args) {
#ifndef MM_PAGING
#ifndef CPU_TLB
    struct memphy_struct* tlb = ((struct mmpaging_ld_args *)args)->tlb;
#endif
    struct memphy_struct* mram = ((struct mmpaging_ld_args *)args)->mram;
    struct memphy_struct* msdp = ((struct mmpaging_ld_args *)args)->msdp;
    struct memphy_struct* active_msdp = ((struct mmpaging_ld_args *)args)->active_msdp;
    struct timer_id_t * timer_id = ((struct mmpaging_ld_args *)args)->timer_id;
#else
    struct timer_id_t * timer_id = (struct timer_id_t*)args;
#endif
    int i = 0;
    printf("ld_routine\n");
    fprintf(output_file, "ld_routine\n");
    while (i < num_processes) {
        while(timer_id->time_wait == 0);
        struct pcb_t * proc = load(ld_processes.path[i]);
#ifndef MLQ_SCHED
        proc->prio = ld_processes.prio[i];
#endif
    }
}
```



Hàm `wait_time()` cho phép các hoạt động khác tiếp tục làm việc sau khi đã in ra timeslot:

```
static void * timer_routine(void * args) {
    while (!timer_stop) {
        printf("Time slot %3lu\n", current_time());
        fprintf(output_file,"Time slot %3lu\n", current_time());

        struct timer_id_container_t * temp;
        wait_time();
        int fsh = 0;
        int event = 0;
```

```
1 void wait_cpu(){
2     struct timer_id_container_t * temp;
3     for (temp = dev_list; temp != NULL; temp = temp->next){
4         temp->id.cpu_wait = 1;
5     }
6 }
7 void wait_time(){
8     struct timer_id_container_t * temp;
9     for (temp = dev_list; temp != NULL; temp = temp->next){
10        temp->id.time_wait = 1;
11    }
12 }
13 void unlock_cpu(){
14     struct timer_id_container_t * temp;
15     for (temp = dev_list; temp != NULL; temp = temp->next){
16        temp->id.cpu_wait = 0;
17    }
18 }
```

**Sự khác biệt khi có và không có cơ chế đồng bộ:** Khi không có cơ chế đồng bộ, các thông tin *Loaded process*, *Put process* và *Dispatched process* không in ra đúng vị trí timeslot tương ứng của chúng. Cơ chế đồng bộ giúp các tiến trình được thực hiện theo đúng trình tự logic, in ra thông tin tại đúng vị trí timeslot tương ứng, ngăn ngừa race condition, giúp các tiến trình truy cập và sử dụng tài nguyên chung một cách an toàn, tránh tình trạng cạnh tranh gây ra lỗi và giúp tăng hiệu suất chương trình, khi các tiến trình được đồng bộ hóa chính xác, chương trình sẽ hoạt động ổn định và đạt hiệu suất cao hơn.

```

Time slot  0
ld_routine
    Loaded a process at input/proc/s2, PID: 1 PRIO: 1
    CPU 1: Dispatched process 1
Time slot  1
    Loaded a process at input/proc/s3, PID: 2 PRIO: 1
    CPU 0: Dispatched process 2
Time slot  2
    Loaded a process at input/proc/s1, PID: 3 PRIO: 1
    CPU 1: Put process 1 to run queue
    CPU 0: Dispatched process 3
Time slot  3
    Loaded a process at input/proc/s0, PID: 4 PRIO: 0
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 4
Time slot  4
    CPU 1: Put process 3 to run queue
    CPU 1: Dispatched process 2
Time slot  5
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 4
Time slot  6
    CPU 1: Put process 2 to run queue
    CPU 1: Dispatched process 2
Time slot  7
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 4
Time slot  8
    CPU 1: Put process 2 to run queue
    CPU 1: Dispatched process 2
Time slot  9
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 4
Time slot 10
    CPU 1: Put process 2 to run queue
    CPU 1: Dispatched process 2
Time slot 11
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 4
Time slot 12
    CPU 1: Put process 2 to run queue
    CPU 1: Dispatched process 2
Time slot 13
    CPU 1: Processed 2 has finished
    CPU 1: Dispatched process 1
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 4
Time slot 14
Time slot 15
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 4
    CPU 1: Put process 1 to run queue
    CPU 1: Dispatched process 3

```

(a) Khi có cơ chế đồng bộ

```

Time slot  0
ld_routine
    Loaded a process at input/proc/s2, PID: 1 PRIO: 1
    CPU 1: Dispatched process 1
    Loaded a process at input/proc/s3, PID: 2 PRIO: 1
Time slot  1
    CPU 1: Put process 1 to run queue
    CPU 1: Dispatched process 1
Time slot  2
    Loaded a process at input/proc/s1, PID: 3 PRIO: 1
    CPU 0: Dispatched process 2
Time slot  3
    Loaded a process at input/proc/s0, PID: 4 PRIO: 0
    CPU 1: Put process 1 to run queue
    CPU 1: Dispatched process 4
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot  4
    CPU 1: Put process 4 to run queue
    CPU 1: Dispatched process 4
Time slot  5
    CPU 1: Put process 4 to run queue
    CPU 1: Dispatched process 4
Time slot  6
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot  7
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 4
Time slot  8
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot  9
    CPU 1: Put process 4 to run queue
    CPU 1: Dispatched process 4
Time slot 10
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot 11
    CPU 1: Put process 4 to run queue
    CPU 1: Dispatched process 4
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot 12
Time slot 13
    CPU 0: Processed 2 has finished
    CPU 0: Dispatched process 3
    CPU 1: Put process 4 to run queue
    CPU 1: Dispatched process 4
Time slot 14
Time slot 15
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 1
    CPU 1: Put process 4 to run queue
    CPU 1: Dispatched process 4

```

(b) Khi không có cơ chế đồng bộ



## 5 Kết luận

Qua việc hoàn thành bài tập lớn này, chúng em đã có cơ hội tiến hành mô phỏng các thành phần chính trong một hệ điều hành đơn giản. Bài tập đã giúp chúng em nắm vững các khái niệm cơ bản về *scheduler* (bộ lập lịch), đồng bộ hóa và quản lý bộ nhớ. Chúng em đã thực hành với các module quan trọng: *scheduler*, *synchronization* (đồng bộ hóa) và cơ chế phân bổ bộ nhớ từ bộ nhớ ảo sang bộ nhớ vật lý (*Paging*), cũng như tìm hiểu sâu hơn về cách thức hoạt động của *TLB* (Translation Lookaside Buffer).

Kết quả của bài tập này là chúng em đã hiểu một phần nguyên tắc hoạt động của một hệ điều hành đơn giản. Nhờ việc thực hành và mô phỏng, chúng em đã trải nghiệm trực tiếp quá trình xử lý các tiến trình, ưu tiên và lập lịch chúng, cũng như quản lý bộ nhớ và đồng bộ hóa tài nguyên. Điều này đã giúp chúng em thấu hiểu cách mà các thành phần hệ điều hành tương tác với nhau và hoạt động như thế nào để đảm bảo sự ổn định và hiệu quả cho hệ thống.

Trong quá trình thực hiện bài tập, chúng em đã đạt được những kỹ năng quan trọng như phân tích, thiết kế và triển khai các thành phần hệ điều hành. Chúng em đã học cách phân tích yêu cầu và thiết kế các thuật toán *lập lịch* và *quản lý bộ nhớ* phù hợp cho hệ thống. Chúng em cũng đã rèn kỹ năng triển khai mã nguồn, khắc phục các vấn đề và tối ưu hóa hiệu suất. Qua việc làm việc với mã nguồn và các tệp tin liên quan, chúng em đã làm quen với quy trình phát triển phần mềm và biết cách làm việc trong môi trường hệ điều hành.

Tuy nhiên, chúng em nhận thấy rằng bài tập này chỉ giới hạn ở mức đơn giản và chưa thể trải nghiệm được toàn bộ khía cạnh của hệ điều hành. Hệ điều hành thực tế có nhiều tính năng phức tạp hơn và đa dạng hơn, bao gồm *quản lý tài nguyên mạng*, *hệ thống tập tin phân tán*, *bảo mật*, *giao diện người dùng*, và nhiều khía cạnh khác. Chúng em nhận thức rằng để nghiên cứu sâu hơn về hệ điều hành và đạt được kiến thức thực tiễn, chúng em cần dành thời gian và kiên nhẫn để tìm hiểu và thực hành trên các hệ điều hành phức tạp hơn.

Tổng kết lại, qua việc thực hiện bài tập lớn này, chúng em đã có được một cái nhìn tổng quan về cách hoạt động của một hệ điều hành đơn giản và đạt được những kỹ năng cơ bản trong việc triển khai các thành phần quan trọng của hệ điều hành. Chúng em hi vọng rằng kiến thức và kinh nghiệm từ bài tập này sẽ là nền tảng cho chúng em tiếp tục nghiên cứu và phát triển các hệ điều hành phức tạp hơn trong tương lai.



## 6 Tài liệu tham khảo

<https://www.youtube.com/watch?v=zeHrA0AbgxUt=4s>