Linux • Dec. 10, 2023
Linux • Dec. 10, 2023

# Simple operating system

# OUR TEAM GROUPS

Đoàn Thị Huế

Huỳnh Hữu Tín

Nguyễn Anh Tấn

# STRUCTURE

1. SCHEDULER

2. MEMORY MANAGEMENT

# SCHEDULER

- Sử dụng giải thuật Multilevel - Queue(MLQ):
  - Mỗi process với cùng độ ưu tiên sẽ được đặt vào cùng một hàng đợi ready-queue.
  - MLQ policy: CPU thực thi mỗi process theo RR-style. CPU sẽ chọn process có thứ tự ưu tiên cao nhất. Khi hàng đợi P thực hiện đủ số lần slot (slot = MAX_PRIORITY - prio) thì tạm dừng, process được đưa vào cuối hàng đợi với độ ưu tiên như cũ, chuyển thực thi cho hàng đợi Q có ưu tiên thấp hơn. Khi các hàng đợi có độ ưu tiên thấp đã thực hiện hết số lần slot cho phép sẽ cấp lại slot và quay lại lần lặp kế tiếp với P là hàng đợi có độ ưu tiên cao được thực thi trước với số lần slot được tính.

# SCHEDULER

- **Hiện thực**

```
struct queue_t {
    struct pcb_t * proc[MAX_QUEUE_SIZE];
    int size;

    int slot;
};
```

# SCHEDULER

- **Hiện thực**

```c
void init_scheduler(void)
{
#ifdef MLQ_SCHED
    int i;

    for (i = 0; i < MAX_PRIO; i++)
    {
        mlq_ready_queue[i].size = 0;
        mlq_ready_queue[i].slot = MAX_PRIO - i;
    }

#endif
    ready_queue.size = 0;
    run_queue.size = 0;
    pthread_mutex_init(&queue_lock, NULL);
}
```

```c
void resetSlot() {
    for (int i = 0; i < MAX_PRIO; ++i) {
        mlq_ready_queue[i].slot = MAX_PRIO - i;
    }
}
```

# SCHEDULER

- **Hiện thực**

```c
void enqueue(struct queue_t * q, struct pcb_t * proc) {
    /* TODO: put a new process to queue [q] */
    q->proc[q->size] = proc;
    q->size += 1;
}

struct pcb_t * dequeue(struct queue_t * q) {
    /* TODO: return a pcb whose prioprity is the highest
        * in the queue [q] and remember to remove it from q
        * */
    if(empty(q)) return NULL;

    // If the queue is not empty ...
    struct pcb_t *temp = q->proc[0];

    for(int i = 0; i < q->size - 1; i++) {
        q->proc[i] = q->proc[i + 1];
    }

    q->proc[q->size - 1] = NULL;
    q->size -= 1;

    return temp;
}
```

# SCHEDULER

- **Hiện thực**

```c
struct pcb_t *get_mlq_proc(void) {
    struct pcb_t * proc = NULL;
    /*TODO: get a process from PRIORITY [ready_queue].
     * Remember to use lock to protect the queue.
     * */
    if (queue_empty()) return NULL;


    int prio_select = -1;
    for (prio_select = 0; prio_select < MAX_PRIO; ++prio_select){
        pthread_mutex_lock(&queue_lock);
        int flag_empty = empty(&mlq_ready_queue[prio_select]);
        if (flag_empty) {
            if (prio_select == MAX_PRIO - 1) resetSlot();
            pthread_mutex_unlock(&queue_lock);
            continue;
        } else pthread_mutex_unlock(&queue_lock);

        if (mlq_ready_queue[prio_select].slot > 0) {
            pthread_mutex_lock(&queue_lock);
            proc = dequeue(&mlq_ready_queue[prio_select]);
            mlq_ready_queue[prio_select].slot--;
            pthread_mutex_unlock(&queue_lock);
            break;
        } else {
            if(prio_select == MAX_PRIO - 1) {
                pthread_mutex_lock(&queue_lock);
                resetSlot();
                prio_select = 0;
                pthread_mutex_unlock(&queue_lock);
            }
        }
    }

    return proc;
}
```

- **Input: input/my_ched_o**

```
3 1 6
1048576 16777216 0 0 0
0 sched_proc_1 138
1 sched_proc_2 138
2 sched_proc_3 135
3 sched_proc_4 134
4 sched_proc_5 139
5 sched_proc_6 139
```

| proc_1 | proc_2 | proc_3 | proc_4 | proc_5 | proc_6 |
|---|---|---|---|---|---|
| 10 6<br>calc<br>calc<br>calc<br>calc<br>calc<br>calc | 2 5<br>calc<br>calc<br>calc<br>calc<br>calc<br>calc | 2 4<br>calc<br>calc<br>calc<br>calc | 3 15<br>calc<br>.....<br>calc<br>calc<br>calc<br>calc<br>calc<br>calc | 2 5<br>calc<br>calc<br>calc<br>calc<br>calc<br>calc | 2 6<br>calc<br>calc<br>calc<br>calc<br>calc<br>calc |

- **Output: output/my_ched_o**

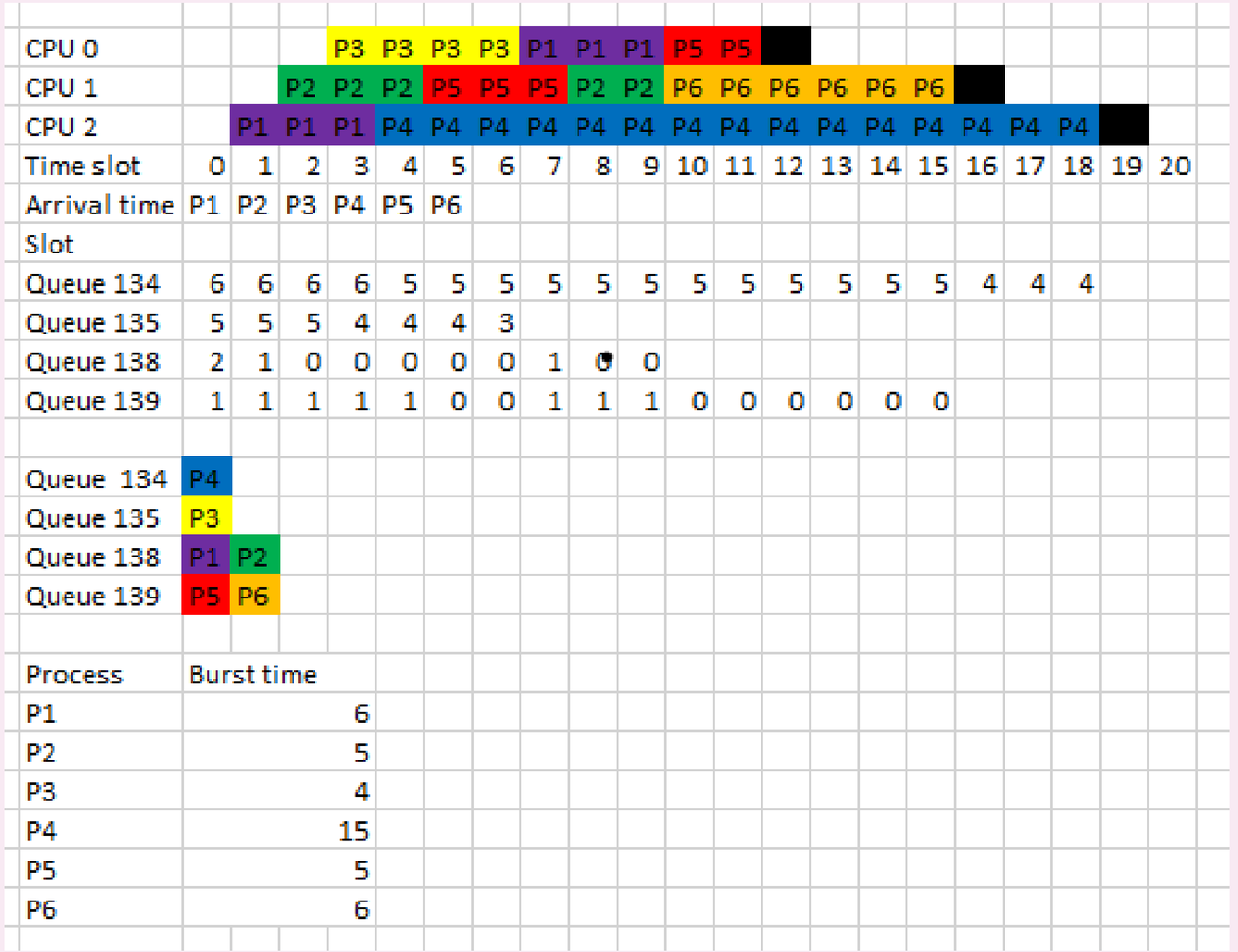
| CPU | P1 | P1 | P1 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P3 | P3 | P3 | P3 | P2 | P2 | P2 | P5 | P5 | P5 | P1 | P1 | P1 | P2 | P2 | P6 | P6 | P6 | P5 | P5 | P6 | P6 | P6 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Time slot | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 |
| Arrrival | P1 | P2 | P3 | P4 | P5 | P6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Slot | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Queue 134 | 6 | 6 | 6 | 6 | 5 | 5 | 5 | 4 | 4 | 4 | 3 | 3 | 3 | 2 | 2 | 2 | 1 | 1 | 1 | | | | | | | | | | | | | | | | | | | | | | | | |
| Queue 135 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 4 | 4 | 4 | 3 | | | | | | | | | | | | | | | | | | | | |
| Queue 138 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | | | | | | | | |
| Queue 139 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| | | |
|---|---|---|
| Queue 134 | P4 | |
| Queue 135 | P3 | |
| Queue 138 | P1 | P2 |
| Queue 139 | P5 | P6 |

- **Input: input/my_ched_1**

```
3 3 6
1048576 16777216 0 0 0
0 sched_proc_1 138
1 sched_proc_2 138
2 sched_proc_3 135
3 sched_proc_4 134
4 sched_proc_5 139
5 sched_proc_6 139
```

- **Output: output/my_ched_1**



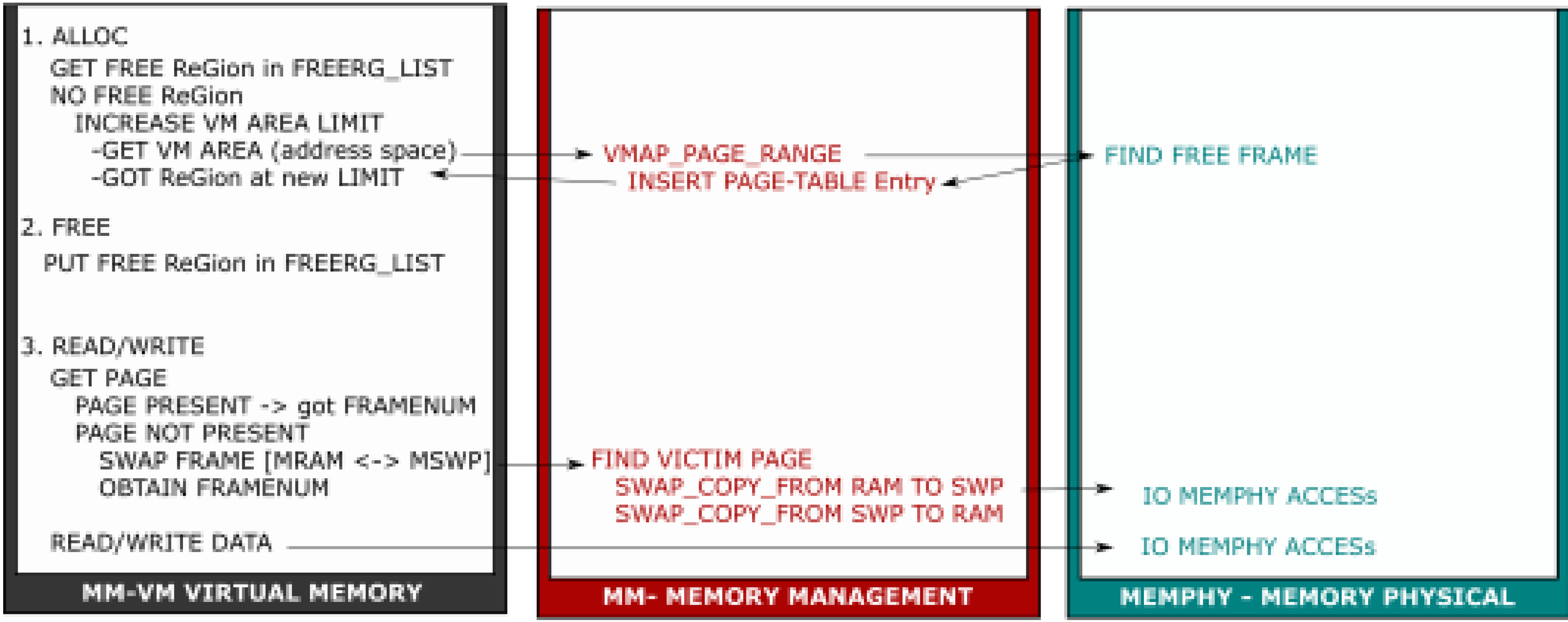
| | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CPU 0 | | | | P3 | P3 | P3 | P3 | P1 | P1 | P1 | P5 | P5 | | | | | | | | | |
| CPU 1 | | | P2 | P2 | P2 | P5 | P5 | P5 | P2 | P2 | P6 | P6 | P6 | P6 | P6 | P6 | | | | | |
| CPU 2 | | P1 | P1 | P1 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | |
| Time slot | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| Arrival time | P1 | P2 | P3 | P4 | P5 | P6 | | | | | | | | | | | | | | | |
| Slot | | | | | | | | | | | | | | | | | | | | | |
| Queue 134 | 6 | 6 | 6 | 6 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 4 | 4 | 4 | | |
| Queue 135 | 5 | 5 | 5 | 4 | 4 | 4 | 3 | | | | | | | | | | | | | | |
| Queue 138 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | | | | | | | | | | | |
| Queue 139 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | |

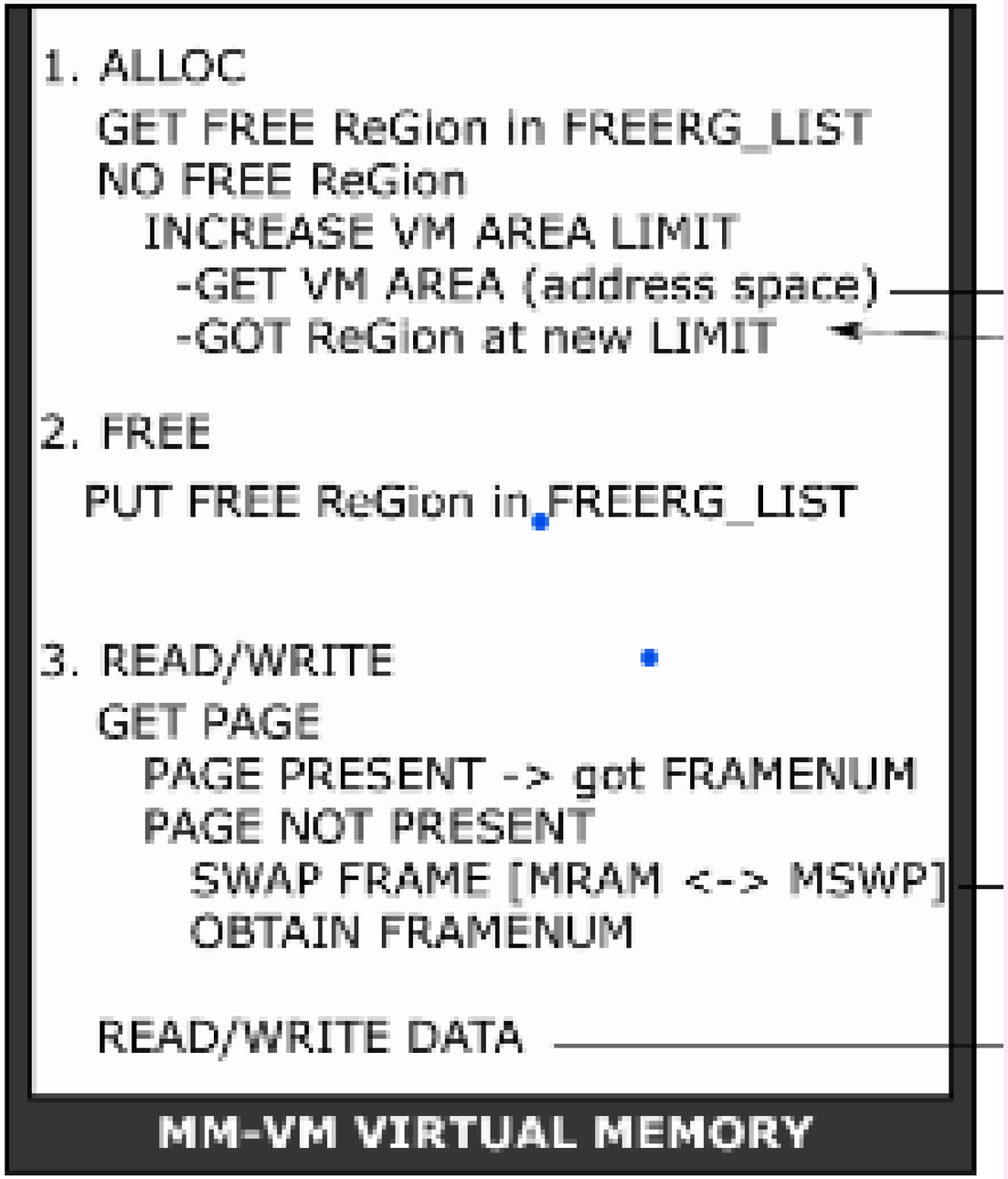
| | | |
|---|---|---|
| Queue 134 | P4 | |
| Queue 135 | P3 | |
| Queue 138 | P1 | P2 |
| Queue 139 | P5 | P6 |

| Process | Burst time |
|---|---|
| P1 | 6 |
| P2 | 5 |
| P3 | 4 |
| P4 | 15 |
| P5 | 5 |
| P6 | 6 |

# MEMORY MANAGEMENT

## PAGING-BASED MEMORY MANAGEMENT MODULES






## Không gian địa chỉ bộ nhớ ảo bao gồm:

- Memory Area (mỗi area đại diện cho các segment của 1 process: data segment, code segment, stack segment, heap segment,...). Trong Assignment, ta chỉ sử dụng 1 memory area tương ứng vmaid = 0.
- Mỗi Memory Area bao gồm các Memory Region (region đang được sử dụng và region đang free).

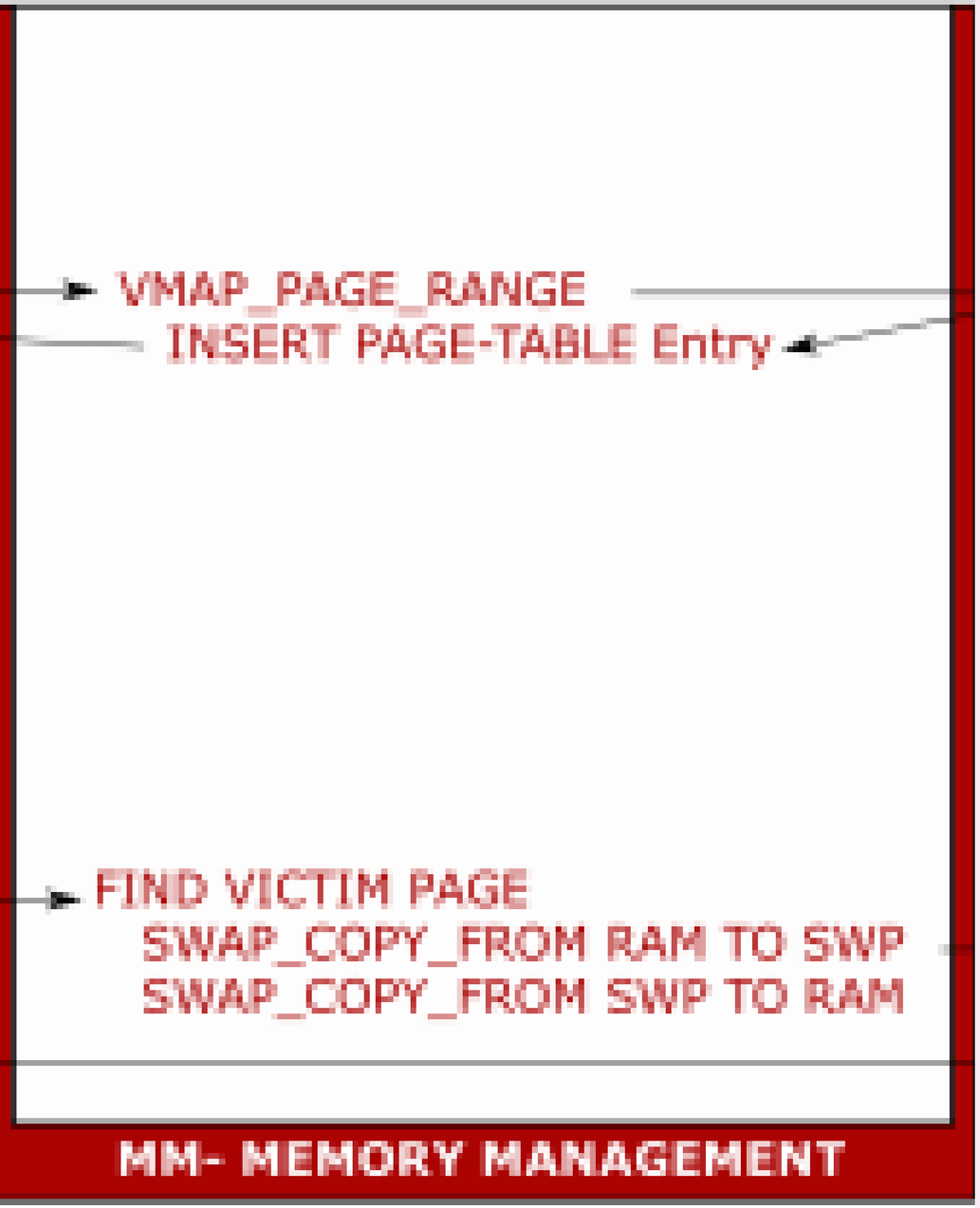
## Lợi ích của việc thiết kế multiple segments:

- Tổ chức bộ nhớ tốt hơn, dễ dàng quản lý và truy xuất dữ liệu: Stack segment quản lý các local variables và các function calls, Heap segment quản lý việc cấp phát động, Code segment quản lý các câu lệnh được thực thi,...
- Giảm context switching: Khi CPU switch giữa các processes hoặc threads, chỉ những segments cần thiết mới được restore trở lại CPU để tiếp tục thực thi. Điều này hiệu quả và nhanh hơn thay vì restore toàn bộ không gian địa chỉ.

## Các operations trên virtual memory bao gồm:

- ALLOC: cấp phát 1 region memory.
- FREE: giải phóng 1 region memory.
- READ: đọc 1 byte giá trị tại 1 ô nhớ trong region memory.
- WRITE: ghi 1 byte giá trị vào 1 ô nhớ trong region memory.

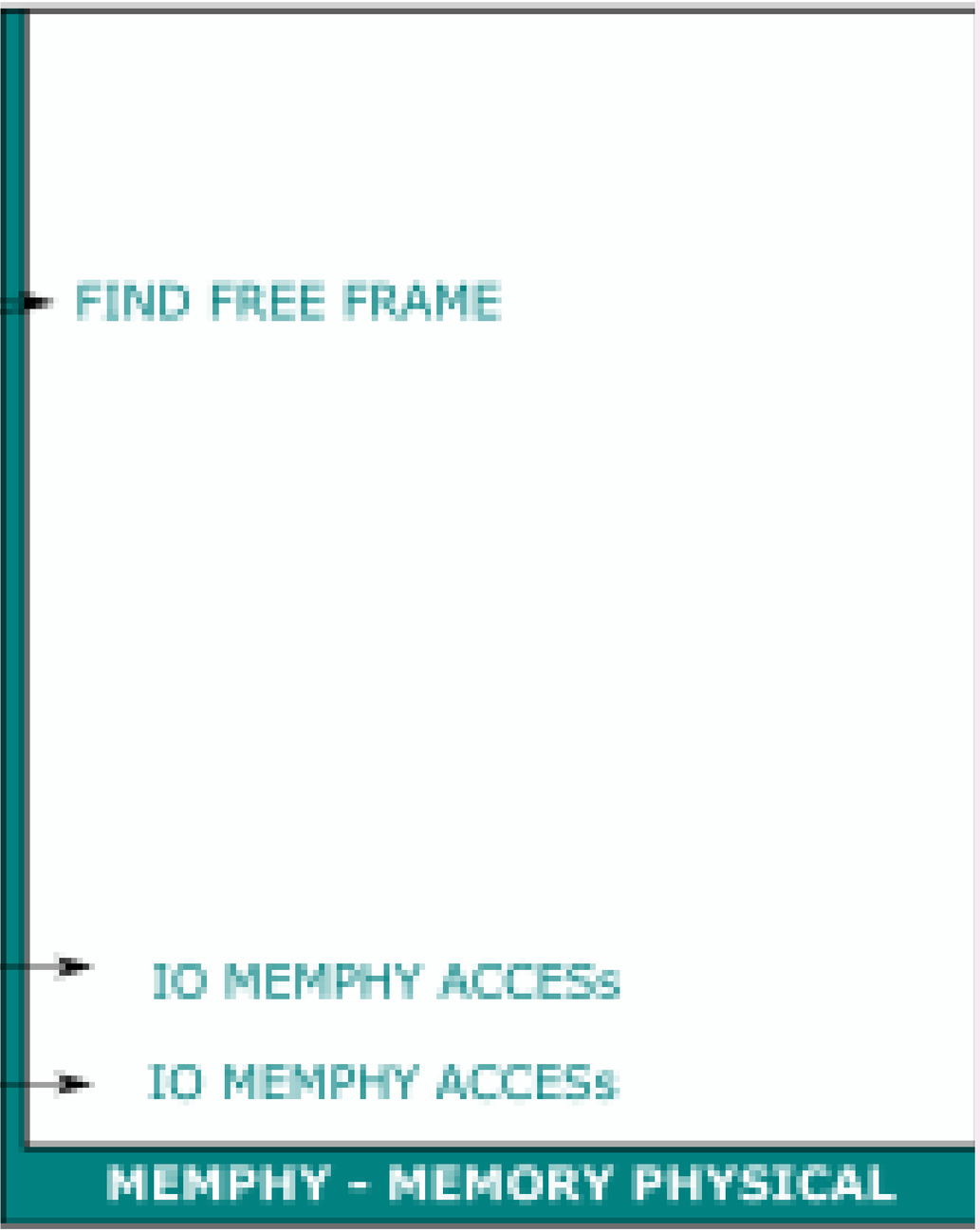
# MM - MEMORY MANAGEMENT




Mỗi process có 1 module Memory Management với vai trò mapping virtual address sang physical address, cụ thể:

- Giữ địa chỉ page table directory của process. Page table sẽ ánh xạ mỗi logical address sang page table entry.
- Quản lý địa chỉ các virtual memory area (segment) của process.
- Quản lý các memory region đang được sử dụng của process.
- Quản lý danh sách các page được map với frame trong physical memory theo cơ chế FIFO.

Memory Management quản lý logical address theo cơ chế paging (mỗi page là một không gian địa chỉ với kích thước cố định). Cấu trúc logical address 22 bit trong assignment được chia thành 2 mức, bao gồm 14 bit page number và 8 bit page offset.




- Không gian địa chỉ bộ nhớ vật lý bao gồm:

- RAM: Bộ nhớ chính, có thể được truy xuất trực tiếp bởi CPU.
- SWAP: Bộ nhớ thứ cấp, không được truy xuất trực tiếp bởi CPU, kích thước lớn hơn RAM và số lượng nhiều (assignment dùng 4 SWAP).

- Các opearions trên physical memory bao gồm:

- READ: directly đối với RAM và sequentially đối với SWAP.
- WRITE: directly đối với RAM và sequentially đối với SWAP.

# ALLOC

- int find_victim_page(struct mm_struct *mm, int *retpgn)

Tìm một "victim" page trong virtual memory:

- Lấy một page từ cuối hàng đợi fifo_pgn mà mm đang quản lý, lưu số thứ tự của page đó vào retpgn.

- int alloc_pages_range(struct pcb_t *caller, int req_pgnum, struct framephy_struct **frm_lst)

Cấp phát các frame trên RAM và lưu trong danh sách frm_lst:

- Lặp qua req_pgnum lần. Ở mỗi bước lặp, kiểm tra có thể cấp phát 1 frame trong RAM. Nếu được, thêm frame vừa được cấp phát vào đầu danh sách frm_lst. Nếu không, thực thi bước kế tiếp.
- Tìm một frame vicfpn trong RAM và thay thế nó bằng một free frame swpfpn trong SWAP. Việc lựa chọn một vicfpn cần tìm một "victim" page vicpgn trong virtual memory bằng hàm find_victim_page.
- Nếu việc tìm vicpgn hoặc lấy một swpfpn trong SWAP thất bại thì giải phóng toàn bộ các frame đã được cấp phát trước đó và trả về lỗi.
- Swap vicfpn trong RAM với swpfpn trong SWAP.
- Set page table entry tương ứng với vicpgn bằng hàm pte_set_swap, thêm swpfpn vào đầu danh sách frm_lst.

# ALLOC

- int vmap_page_range(struct pcb_t *caller, int addr, int pgnum, struct framephy_struct *frames, struct vm_rg_struct *ret_rg)

Mapping các page tại địa chỉ addr với các frame trong danh sách frames:

- Duyệt qua các page, set frame page table entry tương ứng với các page bằng hàm pte_set_fpn. Sau đó chèn các page đó theo thứ tự vào đầu danh sách fifo_pgn.

- int vm_map_ram(struct pcb_t *caller, int astart, int aend, int mapstart, int incpgnum, struct vm_rg_struct *ret_rg)

Mapping virtual memory có kích thước aend - astart với RAM:

- Cấp phát trên RAM số lượng frame bằng với số page incpgnum trên virtual memory bằng hàm alloc_pages_range, lưu các frame sau khi cấp phát thành một danh sách frm_lst.
- Nếu cấp phát thành công, map các page tại địa chỉ mapstart với các frame trong danh sách frm_lst bằng hàm vmap_page_range.

# ALLOC

- int inc_vma_limit(struct pcb_t *caller, int vmaid, int inc_sz)

Tăng kích thước vùng nhớ của vmaid lên một khoảng bằng inc_sz:

- Kiểm tra vmaid nếu tăng kích thước có bị tràn qua các vmaid lân cận. Nếu có, trả về lỗi (return -1), ngược lại thực thi bước kế tiếp.
- Dịch con trỏ sbrk và vm_end của vmaid lên thêm 1 khoảng bằng inc_sz.
- Tiến hành mapping virtual memory vừa được cấp phát sang physical memory trên RAM bằng hàm vm_map_ram.

- int __alloc(struct pcb_t *caller, int vmaid, int rgid, int size, int *alloc_addr)

Cấp phát cho region rgid một vùng nhớ với kích thước size, sử dụng khóa mutex để bảo vệ virtual memory:

- Nếu kích thước vùng nhớ trống của area vmaid >= size thì cấp phát thành công.
- Nếu không đủ vùng nhớ, tăng kích thước của area vmaid lên một khoảng bằng số lượng page vừa đủ chứa size bằng hàm inc_vma_limit.
- Sau khi tăng kích thước thành công, cấp phát size cho rgid.
- Thêm fragment do paging gây ra sau khi cấp phát (nếu có) vào danh sách vm_freerg_list của mm.

- int pgalloc(struct pcb_t *proc, uint32_t size, uint32_t reg_index)

Thực thi hàm __alloc với vmaid = 0 và ghi kết quả vào file output.

# FREE

- int __free(struct pcb_t *caller, int vmaid, int rgid)

Giải phóng vùng nhớ rgid, sử dụng khóa mutex để bảo vệ virtual memory:

- Nếu rgid < 0 hoặc rgid > PAGING_MAX_SYMTBL_SZ hoặc size rgid = 0, trả về lỗi (return -1).
- Gán kích thước của vùng nhớ rgid bằng 0, thêm vùng nhớ vừa được giải phóng vào danh sách vm_freerg_list mà mm quản lý.

- int pgfree_data(struct pcb_t *proc, uint32_t reg_index)

- Thực thi hàm __free với vmaid = 0 và ghi kết quả vào file output.

# READ

- int pg_getpage(struct mm_struct *mm, int pgn, int *fpn, struct pcb_t *caller)

Lấy frame fpn tương ứng với page pgn:

- Nếu frame online (frame đang được lưu trên RAM ), trả về fpn. Nếu frame offline (frame đang được lưu trên SWAP), thực thi bước kế tiếp.
- Tìm 1 "victim" page vicpgn trong virtual memory bằng hàm find_victim_page và lấy 1 free frame swpfpn từ SWAP. Nếu thành công, thực thi bước kế tiếp.
- Swap vicfpn tương ứng vicpgn trong RAM với swpfpn trong SWAP.
- Swap tgtfpn tương ứng pgn trong SWAP với swpfpn trong RAM.
- Set swap page table entry tương ứng với vicpgn bằng hàm pte_set_swap.
- Set frame page table entry tương ứng pgn bằng hàm pte_set_fpn.
- Thêm pgn vào danh sách fifo_pgn của mm, trả về fpn.

# READ

- int pg_getval(struct mm_struct *mm, int addr, BYTE *data, struct pcb_t *caller)

Đọc giá trị của ô nhớ tại địa chỉ addr và lưu vào data:

- Lấy page number và page offset từ addr.
- Lấy frame number trong RAM hoặc SWAP tương ứng với page number trong virtual memory bằng hàm pg_getpage.
- Nếu thành công, kết hợp frame number và page offset trả về physical memory. Đọc giá trị của địa chỉ vật lý này trong bộ nhớ vật lý bằng hàm MEMPHY_read.

- int __read(struct pcb_t *caller, int vmaid, int rgid, int offset, BYTE *data)

Thực thi hàm pg_getval để đọc 1 byte giá trị ô nhớ ở vị trí offset trong vùng nhớ rgid vào data, sử dụng khóa mutex để bảo vệ virtual memory.

- int pgread(struct pcb_t *proc, uint32_t source, uint32_t offset, uint32_t destination)

Thực thi hàm __read với vmaid = 0 và ghi kết quả vào file output.

# WRITE

- int pg_setval(struct mm_struct *mm, int addr, BYTE value, struct pcb_t *caller)

Ghi value vào ô nhớ tại địa chỉ addr:

- Lấy page number và page offset từ addr.
- Lấy frame number trong RAM hoặc SWAP tương ứng với page number trong virtual memory bằng hàm pg_getpage.
- Nếu thành công, kết hợp frame number và page offset trả về physical memory. Ghi value vào địa chỉ vật lý này trong bộ nhớ vật lý bằng hàm MEMPHY_write.

- int __write(struct pcb_t *caller, int vmaid, int rgid, int offset, BYTE value)

Thực thi hàm pg_setval để ghi 1 byte value vào ô nhớ ở vị trí offset trong vùng nhớ rgid, sử dụng khóa mutex để bảo vệ virtual memory.

- int pgwrite(struct pcb_t *proc, BYTE data, uint32_t destination, uint32_t offset)

Thực thi hàm __write với vmaid = 0 và ghi kết quả vào file output.

- **Input: input/os_1_singleCPU_mlq**

os_1_singleCPU_mlq:

```
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
```

s4

```
20 7
calc
.....
calc
```

s3

```
7 11
calc
...
calc
```

m1s

```
1 8
alloc 300
0
alloc 100 1
free 0
alloc 100 2
free 2
free 1
```

s2

```
20 12
calc
.....
calc
```

m0s

```
1 7
alloc 300
0
alloc 100 1
free 0
alloc 100 2
write 102 1
20
write 1 2
1000
```

m1s

```
1 8
alloc 300
0
alloc 100 1
free 0
alloc 100 2
free 2
free 1
```

- **Out put: Kết quả chi tiết trong /output/os_1_singleCPU_mlq.output**

**SHOW RAM:**

- Time slot 6: CPU thực thi lệnh alloc 300 0 của process m1s

```
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=3 - Region=0 - Address=00000000 - Size=300 byte
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
```

- Time slot 7: CPU thực thi lệnh alloc 100 1 của process m1s

```
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=3 - Region=1 - Address=0000012c - Size=100 byte
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
```

**SHOW RAM:**

- Time slot 8: CPU thực thi lệnh free 0 của process m1s

```
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=5 - Region=2 - Address=00000000 - Size=100 byte
print_pgtbl: 0 - 512
00000000: 80000003
00000004: 80000002
Page Number: 0 -> Frame Number: 3
Page Number: 1 -> Frame Number: 2
================================================================
```

- Time slot 9: CPU thực thi lệnh alloc 100 2 của process m1s

```
===== PHYSICAL MEMORY AFTER WRITING =====
write region=1 offset=20 value=102
print_pgtbl: 0 - 512
00000000: 80000003
00000004: 80000002
Page Number: 0 -> Frame Number: 3
Page Number: 1 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000240: 102
===== PHYSICAL MEMORY END-DUMP =====
================================================================
```

**SHOW RAM:**

- Time slot 55: CPU thực thi lệnh alloc 100 2 của process m0s

```
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=3 - Region=0
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
```

- Time slot 60: CPU thực thi lệnh write 102 1 20 của process m0s

```
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=3 - Region=1 - Address=0000012c - Size=100 byte
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
```

# Thank you !