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

CO2017 - HỆ ĐIỀU HÀNH

Assignment

SIMPLE OPERATING SYSTEM

Nhóm 5 - L08

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

TP. HỒ CHÍ MINH, THÁNG 4/2023

Mục lục

1 Bảng phân chia công việc

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

2 Lời mở đầu

2.1 Tổng quan bài tập lớn

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

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

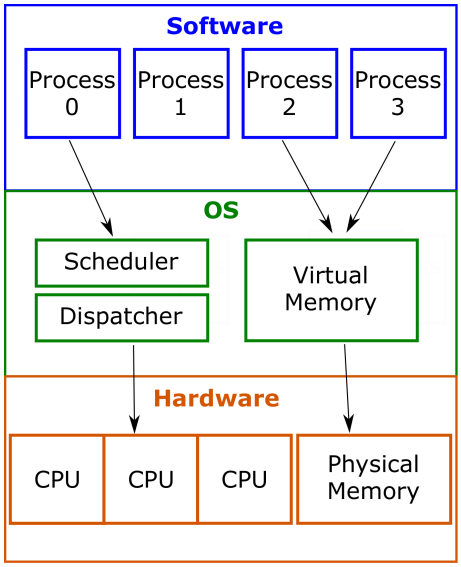
- Bộ nhớ ảo (Virtual Memory Engine): cô lập khoảng không gian bộ nhớ của mỗi quá trình với nhau. Mặc dù RAM được chia sẻ bởi nhiều quá trình (multi processeses) nhưng mỗi quá trình đều không biết được sự tồn tại của quá trình khác. Điều đó cho phép mỗi quá trình sỡ hữu không gian bộ nhớ ảo riêng cho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

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

2.2 Mục tiêu và kết quả cần đạt được

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

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



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

3 Yêu cầu của đề bài

3.1 Source code:

Header files:

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

Source files:

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

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

Make file
Input: các mẫu input cho sẵn được sử dụng cho việc đánh giá
Output: các mẫu output để đối chiếu kết quả với các input

3.2 Processes

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

```
// From include/common.h
struct pcb_t {
    uint32_t pid;
    uint32_t priority;
    uint32_t code_seg_t * code;
    addr_t regs;
    uint32_t pc;
    struct seg_table_t * seg_table;
    uint32_t bp;
}
```

Ý nghĩa:

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

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

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

  `alloc [size] [reg]`

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

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

```
free [reg]
```

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

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

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

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

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

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

3.3 Tạo một process như thế nào?

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

```
[priority] [N = number of instructions]
instruction 0
instruction 1
...
instruction N-1
```

3.4 Chạy mô phỏng như thế nào?

Để hiện thực quá trình mô phỏng, chúng ta phải mô tả cấu hình của phần cứng và môi trường mà ta mô phỏng. Cấu hình được tạo theo chuẩn:
- Time slide là khoảng thời gian mà quá trình được phép chạy.
- N là số CPU có sẵn
- M là số quá trình sẽ được chạy

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

4 Hiện thực code và trả lời câu hỏi

4.1 Scheduler

4.1.1 Hoàn thiện code

4.1.1.a Bổ sung thêm atribute cho struct queue_t

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

Listing 1: Source code queue_t struct

```
struct queue_t
{
  struct pcb_t *proc[MAX_QUEUE_SIZE];
  int size;
#ifdef MLQ_SCHED
  int slot_cpu_can_use; //use to limit the number of cpu each queue can use
#endif
};
```

4.1.1.b Hiện thực code trong file queue.c

Hiện thức queue cơ bản theo nguyên tắc FIFO. Chúng ta có $\mathbf{MAX\_QUEUE\_SIZE = 10}$ được khai báo trong file queue.h, vậy nên mỗi queue không chứa quá 10 process.

Listing 2: Source code enqueue and dequeue function

```
void enqueue(struct queue_t *q, struct pcb_t *proc)
{
        if (q == NULL || proc == NULL)
                return; // check for null pointers

        if (q->size >= MAX_QUEUE_SIZE)
        {
                printf("Queue is full!\n");
                return;
        }
        q->proc[q->size] = proc; // add process to end of array
        q->size++;             // increment queue size
}

struct pcb_t *dequeue(struct queue_t *q)
{
        if (q == NULL || q->size == 0)
                return NULL; // check for empty queue

        struct pcb_t *first_proc = q->proc[0];

        // remove first process from queue
        for (int i = 0; i < q->size - 1; i++)
        {
```

```
                q->proc[i] = q->proc[i + 1];
        }
        q->size--; // decrement queue size
        return first_proc;
}
```

4.1.1.c Hiện thực code trong file sched.c

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

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

4.1.1.d Bổ sung thêm code cho init_scheduler()

Listing 3: Source code init_scheduler function

```
#ifdef MLQ_SCHED
  int i;

  for (i = 0; i < MAX_PRIO; i++)
  {
    mlq_ready_queue[i].size = 0;
    // init number of cpu each queue can use maximally
    mlq_ready_queue[i].slot_cpu_can_use = MAX_PRIO - i;
  }

#endif
```

4.1.1.e Hàm add_mlq_proc()

.

Listing 4: Source code add_mlq_proc function

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

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

4.1.1.f Hàm get_mlq_proc()

.

Listing 5: Source code get_mlq_proc function

```
struct pcb_t *get_mlq_proc(void)
{
  struct pcb_t *proc = NULL;
  unsigned long curr_prio = 0, max_prio = MAX_PRIO;
  while (curr_prio < max_prio)
  {
    if (!empty(&mlq_ready_queue[curr_prio]) &&
        mlq_ready_queue[curr_prio].slot_cpu_can_use > 0)
    {
      pthread_mutex_lock(&queue_lock);
      proc = dequeue(&mlq_ready_queue[curr_prio]);
      if (proc != NULL)
      {
        mlq_ready_queue[curr_prio].slot_cpu_can_use--; //decrease slot_cpu_can_use
      }
      pthread_mutex_unlock(&queue_lock);
      break;
    }
    else
    {
      curr_prio++;
    }
  }
  return proc;
}
```

4.1.1.g Hàm put_mlq_proc()

Listing 6: Source code put_mlq_proc function

```
void put_mlq_proc(struct pcb_t *proc)
{
  pthread_mutex_lock(&queue_lock);
  enqueue(&mlq_ready_queue[proc->prio], proc);
  mlq_ready_queue[proc->prio].slot_cpu_can_use++; //increase slot_cpu_can_use
  pthread_mutex_unlock(&queue_lock);
}
}
```

4.1.1.h Hàm finish_proc()

.

Listing 7: Source code finish_proc function

```
void finish_proc(struct pcb_t **proc)
{
  pthread_mutex_lock(&queue_lock);
  mlq_ready_queue[(*proc)->prio].slot_cpu_can_use++;
  pthread_mutex_unlock(&queue_lock);
  free(*proc);
}
```

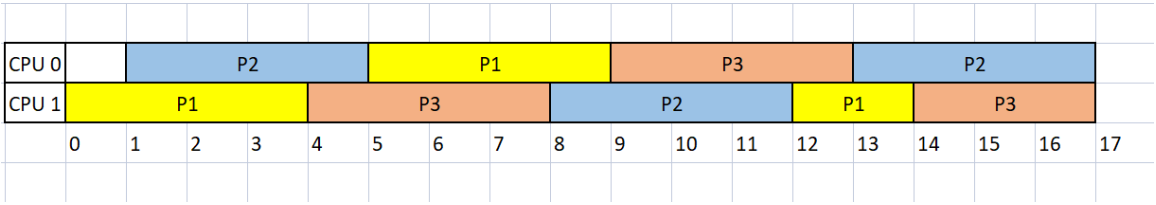
4.1.2 Output

4.1.2.a Trường hợp input sched

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

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

Biểu đồ Gantt và kết quả chạy testcase sched



Hình 2: Biểu đồ gantt khi chạy testcase sched

Listing 8: Output of sched

```
mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os sched
Time slot  0
ld_routine
       Loaded a process at input/proc/p1s, PID: 1 PRIO: 0
Time slot  1
       Loaded a process at input/proc/p2s, PID: 2 PRIO: 0
       CPU 1: Dispatched process 1
Time slot  2
       Loaded a process at input/proc/p3s, PID: 3 PRIO: 0
       CPU 0: Dispatched process 2
Time slot  3
Time slot  4
```

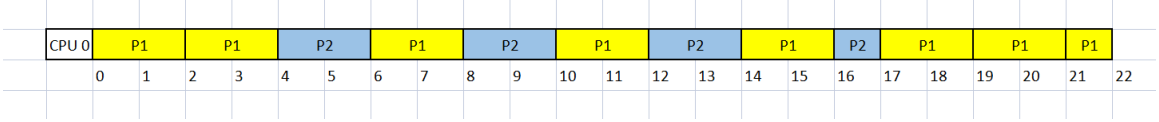
```
Time slot  5
       CPU 1: Put process 1 to run queue
       CPU 1: Dispatched process 3
Time slot  6
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 1
Time slot  7
Time slot  8
Time slot  9
       CPU 1: Put process 3 to run queue
       CPU 1: Dispatched process 2
Time slot 10
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 3
Time slot 11
Time slot 12
Time slot 13
       CPU 1: Put process 2 to run queue
       CPU 1: Dispatched process 1
Time slot 14
       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 2
Time slot 15
       CPU 1: Processed 1 has finished
       CPU 1: Dispatched process 3
Time slot 16
Time slot 17
Time slot 18
       CPU 1: Processed 3 has finished
       CPU 1 stopped
       CPU 0: Processed 2 has finished
       CPU 0 stopped
```

4.1.2.b Trường hợp input sched_0

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

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

Biểu đồ Gantt và kết quả chạy testcase sched_0



Hình 3: Biểu đồ gantt khi chạy testcase sched_0

Listing 9: Output of sched_0

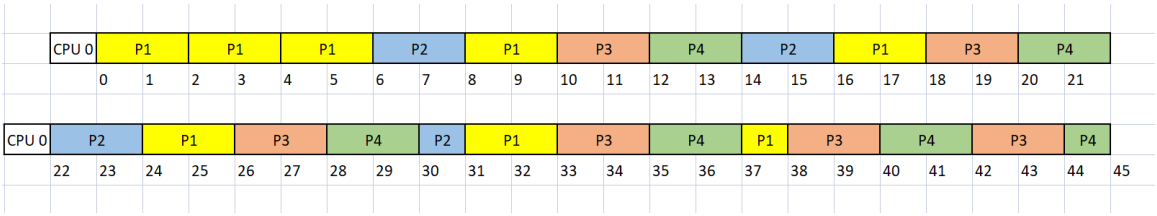
```
mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os sched_0
Time slot  0
ld_routine
       Loaded a process at input/proc/s0, PID: 1 PRIO: 0
       CPU 0: Dispatched process 1
Time slot  1
Time slot  2
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 1
Time slot  3
       Loaded a process at input/proc/s1, PID: 2 PRIO: 0
Time slot  4
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 2
Time slot  5
Time slot  6
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 1
Time slot  7
Time slot  8
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 2
Time slot  9
Time slot 10
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 1
Time slot 11
Time slot 12
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 2
Time slot 13
Time slot 14
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 1
Time slot 15
Time slot 16
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 2
Time slot 17
       CPU 0: Processed 2 has finished
       CPU 0: Dispatched process 1
Time slot 18
Time slot 19
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 1
Time slot 20
Time slot 21
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 1
Time slot 22
       CPU 0: Processed 1 has finished
       CPU 0 stopped
```

4.1.2.c Trường hợp input sched_1

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

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

Biểu đồ Gantt và kết quả chạy testcase sched_1



Hình 4: Biểu đồ gantt khi chạy testcase sched_1

Listing 10: Output of sched_1

```
mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os sched_1
Time slot  0
ld_routine
       Loaded a process at input/proc/s0, PID: 1 PRIO: 0
       CPU 0: Dispatched process 1
Time slot  1
Time slot  2
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 1
Time slot  3
Time slot  4
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 1
       Loaded a process at input/proc/s1, PID: 2 PRIO: 0
Time slot  5
Time slot  6
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 2
       Loaded a process at input/proc/s2, PID: 3 PRIO: 0
Time slot  7
       Loaded a process at input/proc/s3, PID: 4 PRIO: 0
Time slot  8
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 1
Time slot  9
Time slot 10
       CPU 0: Put process 1 to run queue
```

```
       CPU 0: Dispatched process 3
Time slot 11
Time slot 12
       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 4
Time slot 13
Time slot 14
       CPU 0: Put process 4 to run queue
       CPU 0: Dispatched process 2
Time slot 15
Time slot 16
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 1
Time slot 17
Time slot 18
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 3
Time slot 19
Time slot 20
       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 4
Time slot 21
Time slot 22
       CPU 0: Put process 4 to run queue
       CPU 0: Dispatched process 2
Time slot 23
Time slot 24
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 1
Time slot 25
Time slot 26
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 3
Time slot 27
Time slot 28
       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 4
Time slot 29
Time slot 30
       CPU 0: Put process 4 to run queue
       CPU 0: Dispatched process 2
Time slot 31
       CPU 0: Processed 2 has finished
       CPU 0: Dispatched process 1
Time slot 32
Time slot 33
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 3
Time slot 34
Time slot 35
       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 4
Time slot 36
```

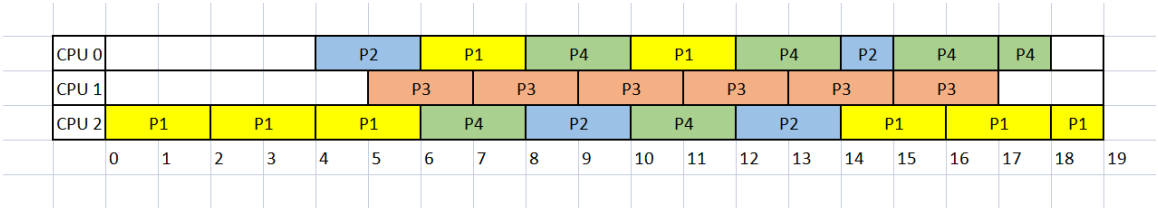
```
Time slot 37
       CPU 0: Put process 4 to run queue
       CPU 0: Dispatched process 1
Time slot 38
       CPU 0: Processed 1 has finished
       CPU 0: Dispatched process 3
Time slot 39
Time slot 40
       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 4
Time slot 41
Time slot 42
       CPU 0: Put process 4 to run queue
       CPU 0: Dispatched process 3
Time slot 43
Time slot 44
       CPU 0: Processed 3 has finished
       CPU 0: Dispatched process 4
Time slot 45
       CPU 0: Processed 4 has finished
       CPU 0 stopped
```

4.1.2.d Trường hợp input sched_full

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

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

Biểu đồ Gantt và kết quả chạy testcase sched_full



Hình 5: Biểu đồ gantt khi chạy testcase sched_full

Listing 11: Output of sched_full

```
mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os sched_full
Time slot  0
ld_routine
       Loaded a process at input/proc/s0, PID: 1 PRIO: 1
Time slot  1
       CPU 1: Dispatched process 1
Time slot  2
Time slot  3
       CPU 1: Put process 1 to run queue
       CPU 1: Dispatched process 1
Time slot  4
       Loaded a process at input/proc/s1, PID: 2 PRIO: 1
       CPU 0: Dispatched process 2
Time slot  5
       Loaded a process at input/proc/s2, PID: 3 PRIO: 2
       CPU 1: Put process 1 to run queue
       CPU 1: Dispatched process 1
       CPU 2: Dispatched process 3
Time slot  6
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 2
       Loaded a process at input/proc/s3, PID: 4 PRIO: 1
       CPU 1: Put process 1 to run queue
       CPU 1: Dispatched process 4
Time slot  7
       CPU 2: Put process 3 to run queue
       CPU 2: Dispatched process 3
Time slot  8
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 1
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 2
Time slot  9
       CPU 2: Put process 3 to run queue
       CPU 2: Dispatched process 3
Time slot 10
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 4
       CPU 1: Put process 2 to run queue
       CPU 1: Dispatched process 1
Time slot 11
       CPU 2: Put process 3 to run queue
       CPU 2: Dispatched process 3
Time slot 12
       CPU 0: Put process 4 to run queue
       CPU 0: Dispatched process 2
       CPU 1: Put process 1 to run queue
       CPU 1: Dispatched process 4
       CPU 0: Processed 2 has finished
       CPU 0: Dispatched process 1
Time slot 13
       CPU 2: Put process 3 to run queue
       CPU 2: Dispatched process 3
```

```
Time slot 14
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 4
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 1
Time slot 15
       CPU 2: Put process 3 to run queue
       CPU 2: Dispatched process 3
Time slot 16
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 4
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 1
Time slot 17
       CPU 2: Processed 3 has finished
       CPU 2 stopped
       CPU 0: Processed 1 has finished
       CPU 0 stopped
Time slot 18
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 4
Time slot 19
       CPU 1: Processed 4 has finished
       CPU 1 stopped
Time slot 20
```

4.1.3 Trả lời câu hỏi

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

- Priority Feedback Queue (PFQ) có ý tưởng giống với các giải thuật Round Robin (RR), Priority Scheduling, và Multilevel Queue nên PFQ thừa hưởng các lợi thế sau:
  - Priority Scheduling: sử dụng độ ưu tiên khi lựa chọn process giúp các process quan trọng hơn được xử lý trước
  - Round Robin: sử dụng quantum time giúp các process đều được thực thi luân phiên và tránh tình trạng 'starvation' khi một process có độ ưu tiên thấp có thể sẽ không được thực thi.
  - Multilevel Queue: sử dụng nhiều hàng đợi (cụ thể trong PFQ là ready queue và run queue) giúp lựa chọn hiệu quả hơn.
- Tuy nhiên, ta có thể thấy các giải thuật trên tồn tại một số nhược điểm:
  - Priority Scheduling: một process có priority thấp có thể phải đợi vô thời hạn.
  - Round Robin: thời gian đợi trung bình lớn. Nếu như một process cần được thực thi trước (priority cao) phải đợi lâu thì hệ thống sẽ không đươc vận hành tốt.
  - Multilevel Queue: Giải thuật có độ phức tạp cao, ngoài ra Process ở level thấp nhất có khả năng không được thực thi hoặc phải đợi thời gian dài do không được cung cấp tài nguyên.
- Khi áp dụng nhiều ý tưởng trên cho giải thuật, PFQ sẽ khắc phục nhược điểm của các giải thuật gốc:
  - Giả sử ta có một process q0 với priority rất cao và thời gian thực thi quá lớn. Một process mới q1 được thêm vào. Nếu chỉ có một queue, mặc dù q1 có priority cao nhưng không bằng q0, sau một hay nhiều quantum time thì q1 vẫn phải đợi rất lâu do thời gian thực thi của q0 quá lớn. Ở đây, PFQ sử dụng ready_queue để giữ những process đợi được đưa vào thực thi và run_queue để giữ những process đã được thực thi sau một quantum_time. Cơ chế này giúp q1 sau khi được thêm vào ready_queue, q0 thực thi sau quantum time sẽ được đưa vào run_queue. Lúc này, giải thuật lựa chọn process có priority cao nhất trong ready_queue để thực thi, nếu q1 có priority cao nhất thì sẽ được lựa chọn. Điều này tương tự với q2,q3,q4.. với các priority khác nhau. Như vậy, giải thuật giúp tất cả process được thực thi luân phiên nhưng vẫn ưu tiên ưu tiên process có priority cao hơn và đảm bảo không process nào chờ đợi vô hạn.

4.2 Memory Management

4.2.1 Hoàn thiện code

4.2.1.a Bổ sung attribute cho vm_rg_struct

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

Listing 12: Source code vm_rg_struct struct

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

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

4.2.1.b Bổ sung attribute cho vm_rg_struct

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

Listing 13: Source code framephy_struct struct

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

4.2.1.c Hoàn thiện ALLOC

Kiểm tra danh sách free region của area hiện tại, nếu có thì region mới lấy từ danh sách free region, nếu không mở rộng area hiện tại.
Lưu ý: Bộ nhớ cấp phát theo paging nên size sẽ được chỉnh sửa theo kích thước page

Listing 14: Source code __alloc() function

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

```
  }

  pthread_mutex_lock(&vm_lock);

  if (get_free_vmrg_area(caller, vmaid, size, &rgnode) == 0)
  {

    caller->mm->symrgtbl[rgid].rg_start = rgnode.rg_start;
    caller->mm->symrgtbl[rgid].rg_end = rgnode.rg_end;
    caller->mm->symrgtbl[rgid].is_allocated = 1;

    *alloc_addr = rgnode.rg_start;

    pthread_mutex_unlock(&vm_lock);
    return 0;
  }

  /* TODO get_free_vmrg_area FAILED handle the region management (Fig.6)*/

  /*Attempt to increate limit to get space */
  struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, vmaid);
  int old_sbrk;

  old_sbrk = cur_vma->sbrk;

  /* TODO INCREASE THE LIMIT
   */
  inc_vma_limit(caller, vmaid, size);

  /*Successful increase limit */
  caller->mm->symrgtbl[rgid].rg_start = old_sbrk;
  caller->mm->symrgtbl[rgid].rg_end = old_sbrk + size;
  caller->mm->symrgtbl[rgid].is_allocated = 1;

  *alloc_addr = old_sbrk;


  pthread_mutex_unlock(&vm_lock);
  return 0;
}
```

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

Listing 15: Source code inc_vma_limit() fucntion

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

```
  struct vm_rg_struct *newrg = malloc(sizeof(struct vm_rg_struct));
  int inc_amt = PAGING_PAGE_ALIGNSZ(inc_sz);
  int incnumpage = inc_amt / PAGING_PAGESZ;
  struct vm_rg_struct *area = get_vm_area_node_at_brk(caller, vmaid, inc_sz, inc_amt);
  struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, vmaid);

  int old_end = cur_vma->vm_end;

  /*Validate overlap of obtained region */
  if (validate_overlap_vm_area(caller, vmaid, area->rg_start, area->rg_end) < 0)
  {
    fprintf(stderr, "error: overlap when allocated memory region");
    return -1; /*Overlap and failed allocation */
  }

  /* The obtained vm area (only)
   * now will be alloc real ram region */
  cur_vma->vm_end += inc_sz;
  cur_vma->sbrk += inc_sz; // Update the sbrk field to reflect the new vm_end value
  if (vm_map_ram(caller, area->rg_start, area->rg_end,
                 old_end, incnumpage, newrg) < 0)
    return -1; /* Map the memory to MEMRAM */

  // assign new region to new memory of vm area
  struct vm_rg_struct *new_rg = malloc(sizeof(struct vm_rg_struct));
  new_rg->rg_start = old_end;
  new_rg->rg_end = cur_vma->sbrk;

  return 0;
}
```

Listing 16: Source code vm_map_ram() fucntion

```
/*
 * vm_map_ram - do the mapping all vm are to ram storage device
 * @caller   : caller
 * @astart   : vm area start
 * @aend     : vm area end
 * @mapstart : start mapping point
 * @incpgnum : number of mapped page
 * @ret_rg   : returned region
 */
int vm_map_ram(struct pcb_t *caller, int astart, int aend, int mapstart, int incpgnum,
    struct vm_rg_struct *ret_rg)
{
  struct framephy_struct *frm_lst = NULL;
  int ret_alloc;

  /*@bksysnet: author provides a feasible solution of getting frames
   *FATAL logic in here, wrong behaviour if we have not enough page
   *i.e. we request 1000 frames meanwhile our RAM has size of 3 frames
   *Don't try to perform that case in this simple work, it will result
```

```
   *in endless procedure of swap-off to get frame and we have not provide
   *duplicate control mechanism, keep it simple
   */
  ret_alloc = alloc_pages_range(caller, incpgnum, &frm_lst);

  if (ret_alloc < 0 && ret_alloc != -3000)
    return -1;

  /* Out of memory */
  if (ret_alloc == -3000)
  {
#ifdef MMDBG
    printf("OOM: vm_map_ram out of memory \n");
#endif
    return -1;
  }

  /* it leaves the case of memory is enough but half in ram, half in swap
   * do the swaping all to swapper to get the all in ram */
  vmap_page_range(caller, mapstart, incpgnum, frm_lst, ret_rg);

  return 0;
}
```

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

Listing 17: Source code alloc_pages_range() fucntion

```
/*
 * alloc_pages_range - allocate req_pgnum of frame in ram
 * @caller   : caller
 * @req_pgnum : request page num
 * @frm_lst  : frame list
 */

int alloc_pages_range(struct pcb_t *caller, int req_pgnum, struct framephy_struct
    **frm_lst)
{
  int pgit, fpn;
  struct framephy_struct *newfp_str = NULL;

  for (pgit = 0; pgit < req_pgnum; pgit++)
  {
    newfp_str = (struct framephy_struct *)malloc(sizeof(struct framephy_struct));
    if (MEMPHY_get_freefp(caller->mram, &fpn) == 0)
    {
      newfp_str->fpn = fpn;
      newfp_str->owner = caller->mm;
    }
    else
    { // ERROR CODE of obtaining somes but not enough frames
      int vicpgn, swpfpn;
```

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

Listing 18: Source code vmap_page_range() fucntion

```
1  /*
2   * vmap_page_range - map a range of page at aligned address
3   */
4  int vmap_page_range(struct pcb_t *caller,      // process call
5                      int addr,                  // start address which is aligned to
                           pagesz
6                      int pgnum,                 // num of mapping page
7                      struct framephy_struct *frames, // list of the mapped frames
8                      struct vm_rg_struct *ret_rg) // return mapped region, the real mapped
                           fp
9  {                                              // no guarantee all given pages are
       mapped
10   struct framephy_struct *fpit = frames;       // Start with the first frame in the list
11   int pgit;
12   int pgn = PAGING_PGN(addr);
13
14   ret_rg->rg_end = ret_rg->rg_start = addr; // at least the very first space is usable
15
16   /* TODO map range of frame to address space
17    *      [addr to addr + pgnum*PAGING_PAGESZ
18    *      in page table caller->mm->pgd[]
```

```
   */

  for (pgit = 0; pgit < pgnum; pgit++)
  {
    // Update the page table entry at the given addressS
    pte_set_fpn(&caller->mm->pgd[pgn + pgit], fpit->fpn);

    // Update the region boundaries
    ret_rg->rg_end += PAGING_PAGESZ;

    /* Tracking for later page replacement activities (if needed)
     * Enqueue new usage page */
    enlist_pgn_node(&caller->mm->fifo_pgn, pgn + pgit);
    enlist_fpn_node(&caller->mram->used_fp_list, fpit->fpn, caller->mm, pgn + pgit,
        caller);

    fpit->p_owner = caller;
    fpit->pte_id= pgn + pgit;

    // Move to the next frame in the list
    fpit = fpit->fp_next;
  }

  return 0;
}
```

4.2.1.d Hoàn thiện FREE

Thay đổi atribute is_allocate thành giá trị 0 và thêm region và freerg_list.

Listing 19: Source code __free() function

```
/*__free - remove a region memory
 *@caller: caller
 *@vmaid: ID vm area to alloc memory region
 *@rgid: memory region ID (used to identify variable in symbole table)
 *@size: allocated size
 *
 */
int __free(struct pcb_t *caller, int vmaid, int rgid)
{
  if (rgid < 0 || rgid > PAGING_MAX_SYMTBL_SZ)
  {
    return -1;
  }
  /* TODO: Manage the collect freed region to freerg_list */

  // Get the free region
  struct vm_rg_struct *free_rg = get_symrg_byid(caller->mm, rgid);

  if (free_rg->is_allocated != 1)
  {
```

```
    printf("Unable to delocated memory region %d\n", rgid);
    return -1;
  }

  if (free_rg->rg_start == free_rg->rg_end)
  {
    return -1;
  }

  pthread_mutex_lock(&vm_lock);

  struct vm_rg_struct *rgnode = malloc(sizeof(struct vm_rg_struct));

  // Assign the rgnode with appropriate values
  rgnode->rg_start = free_rg->rg_start;
  rgnode->rg_end = free_rg->rg_end;
  rgnode->rg_next = NULL;

  free_rg->is_allocated = 0;

  /*enlist the obsoleted memory region */
  enlist_vm_freerg_list(caller->mm, rgnode);

  pthread_mutex_unlock(&vm_lock);

  return 0;
}
```

4.2.1.e Hoàn thiện READ/WRITE

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

Listing 20: Source code __read() function

```
/*__read - read value in region memory
 *@caller: caller
 *@vmaid: ID vm area to alloc memory region
 *@offset: offset to acess in memory region
 *@rgid: memory region ID (used to identify variable in symbole table)
 *@size: allocated size
 *
 */
int __read(struct pcb_t *caller, int vmaid, int rgid, int offset, BYTE *data)
{

  struct vm_rg_struct *currg = get_symrg_byid(caller->mm, rgid);

  if (!currg->is_allocated)
  {
    printf("read region=%d offset=%d\n", rgid, offset);
    printf("Access violation reading location: memory region %d\n", rgid);
```

```
    return -1;
  }

  struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, vmaid);

  if (currg == NULL || cur_vma == NULL) /* Invalid memory identify */
  {
    return -1;
  }

  pthread_mutex_lock(&vm_lock);

  pg_getval(caller->mm, currg->rg_start + offset, data, caller);

  pthread_mutex_unlock(&vm_lock);

  return 0;
}
```

Listing 21: Source code __write() function

```
/*__write - write a region memory
 *@caller: caller
 *@vmaid: ID vm area to alloc memory region
 *@offset: offset to acess in memory region
 *@rgid: memory region ID (used to identify variable in symbole table)
 *@size: allocated size
 *
 */
int __write(struct pcb_t *caller, int vmaid, int rgid, int offset, BYTE value)
{

  struct vm_rg_struct *currg = get_symrg_byid(caller->mm, rgid);

  if (!currg->is_allocated)
  {
    printf("Access violation writing location: memory region %d\n", rgid);
    return -1;
  }

  struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, vmaid);

  if (currg == NULL || cur_vma == NULL) /* Invalid memory identify */
  {
    return -1;
  }

  pthread_mutex_lock(&vm_lock);

  pg_setval(caller->mm, currg->rg_start + offset, value, caller);

  pthread_mutex_unlock(&vm_lock);
```

```
  return 0;
}
```

Listing 22: Source code pg_getval() function

```
/*pg_getval - read value at given offset
 *@mm: memory region
 *@addr: virtual address to acess
 *@value: value
 *
 */
int pg_getval(struct mm_struct *mm, int addr, BYTE *data, struct pcb_t *caller)
{
  int pgn = PAGING_PGN(addr);
  int off = PAGING_OFFST(addr);
  int fpn;

  /* Get the page to MEMRAM, swap from MEMSWAP if needed */
  if (pg_getpage(mm, pgn, &fpn, caller) != 0)
    return -1; /* invalid page access */

  int phyaddr = (fpn << PAGING_ADDR_FPN_LOBIT) + off;

  MEMPHY_read(caller->mram, phyaddr, data);

  return 0;
}
```

Listing 23: Source code pg_setval() function

```
/*pg_setval - write value to given offset
 *@mm: memory region
 *@addr: virtual address to acess
 *@value: value
 *
 */
int pg_setval(struct mm_struct *mm, int addr, BYTE value, struct pcb_t *caller)
{
  int pgn = PAGING_PGN(addr);
  int off = PAGING_OFFST(addr);
  int fpn;

  /* Get the page to MEMRAM, swap from MEMSWAP if needed */
  if (pg_getpage(mm, pgn, &fpn, caller) != 0)
    return -1; /* invalid page access */

  int phyaddr = (fpn << PAGING_ADDR_FPN_LOBIT) + off;

  MEMPHY_write(caller->mram, phyaddr, value);

  return 0;
```

```
}
```

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

Listing 24: Source code pg_getpage() function

```
/*pg_getpage - get the page in ram
 *@mm: memory region
 *@pagenum: PGN
 *@framenum: return FPN
 *@caller: caller
 *
 */
int pg_getpage(struct mm_struct *mm, int pgn, int *fpn, struct pcb_t *caller)
{
  uint32_t pte = mm->pgd[pgn];

  if (!PAGING_PAGE_PRESENT(pte))
  { /* Page is not online, make it actively living */
    int vicpgn, swpfpn;
    int vicfpn;
    uint32_t vicpte;

    int tgtfpn = PAGING_SWP(pte); // the target frame storing our variable

    /* TODO: Play with your paging theory here */
    /* Find victim page */
    struct framephy_struct *victim_fp = (struct framephy_struct *)malloc(sizeof(struct
        framephy_struct));
    if (find_victim_page(caller, &victim_fp) == -1)
      return -1;

    vicpgn = victim_fp->pte_id;

    vicpte = victim_fp->owner->pgd[vicpgn];
    vicfpn = PAGING_FPN(vicpte);

    /* Get free frame in MEMSWP */
    if (MEMPHY_get_freefp(caller->active_mswp, &swpfpn) == -1)
      return -1;

    /* Do swap frame from MEMRAM to MEMSWP and vice versa*/
    /* Copy victim frame to swap */
    __swap_cp_page(victim_fp->p_owner->mram, vicfpn, caller->active_mswp, swpfpn);
    /* Copy target frame from swap to mem */
    __swap_cp_page(caller->active_mswp, tgtfpn, caller->mram, vicfpn);

    /* Update page table */
    pte_set_swap(&vicpte, 0, swpfpn);

    /* Update its online status of the target page */
```

```
    pte_set_fpn(&pte, vicfpn);

    *fpn = PAGING_FPN(pte);

    enlist_pgn_node(&caller->mm->fifo_pgn, pgn);

    enlist_fpn_node(&caller->mram->used_fp_list, *fpn, caller->mm, pgn, caller);
  }

  *fpn = PAGING_FPN(pte);

  return 0;
}
```

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

Listing 25: Source code find_victim_page() function

```
/*find_victim_page - find victim page
 *@caller: caller
 *@pgn: return physical page
 *
 */
int find_victim_page(struct pcb_t *caller, struct framephy_struct **re_fp)
{
  struct framephy_struct *fp_q = caller->mram->used_fp_list;
  struct framephy_struct *prev = NULL;

  /* TODO: Implement the theorical mechanism to find the victim page */

  if (fp_q == NULL)
  {
    // The FIFO queue is empty, which should not happen
    return -1;
  }

  // Traverse to the end of the FIFO queue to find the oldest page
  while (fp_q->fp_next != NULL)
  {
    prev = fp_q;
    fp_q = fp_q->fp_next;
  }

  // Get the page number of the oldest page
  *re_fp = fp_q;

  // Remove the oldest page from the FIFO queue
  if (prev != NULL)
  {
    prev->fp_next = NULL;
  }
  else
```

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

4.2.1.f Các hàm kết xuất dữ liệu

Hiển thị table entries của process

Listing 26: Source code print_pgtbl() function

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

Hiển thị physical memory.

Listing 27: Source code MEMPHY_dump() function

```
int MEMPHY_dump(struct memphy_struct *mp)
{
   /*TODO dump memphy contnt mp->storage
    *     for tracing the memory content
    */
   printf("Memory Dump:\n");
   printf("-----------\n");

   if (mp == NULL || mp->storage == NULL)
   {
      printf("Invalid memory structure.\n");
      return -1;
   }

   for (int i = 0; i < mp->maxsz; ++i)
   {
      if (mp->storage[i] != 0)
      {
         printf("BYTE %08x: %d\n", i, mp->storage[i]);
      }
   }

   printf("\n");
   return 0;
}
```

4.2.2 Output

4.2.2.a os_0_mlq_paging

Listing 28: output of os_0_mlq_paging

```
mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os
    os_0_mlq_paging
Time slot  0
ld_routine
       Loaded a process at input/proc/p0s, PID: 1 PRIO: 0
       CPU 1: Dispatched process 1
Time slot  1
process 1 alloc region 0 size 300
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Memory Dump:
-----------

Time slot  2
       Loaded a process at input/proc/p1s, PID: 2 PRIO: 15
```

```
process 1 alloc region 4 size 300
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------
       CPU 0: Dispatched process 2

Time slot  3
       Loaded a process at input/proc/p1s, PID: 3 PRIO: 0
process 1 free region 0
Time slot  4
       Loaded a process at input/proc/p1s, PID: 4 PRIO: 0
process 1 alloc region 1 size 100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------

Time slot  5
write region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------
BYTE 00000114: 100

Time slot  6
       CPU 1: Put process 1 to run queue
       CPU 1: Dispatched process 3
Time slot  7
Time slot  8
       CPU 0: Put process 2 to run queue
```

```
       CPU 0: Dispatched process 4
Time slot  9
Time slot 10
Time slot 11
Time slot 12
       CPU 1: Put process 3 to run queue
       CPU 1: Dispatched process 1
read region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------
BYTE 00000114: 100

Time slot 13
write region=2 offset=20 value=102
access violation writing location: memory region 2
Memory Dump:
-----------
BYTE 00000114: 100

Time slot 14
       CPU 0: Put process 4 to run queue
       CPU 0: Dispatched process 3
read region=2 offset=20
access violation reading location: memory region 2
Time slot 15
write region=3 offset=20 value=103
access violation writing location: memory region 3
Memory Dump:
-----------
BYTE 00000114: 100

Time slot 16
       CPU 1: Processed 1 has finished
       CPU 1: Dispatched process 4
Time slot 17
Time slot 18
       CPU 0: Processed 3 has finished
       CPU 0: Dispatched process 2
Time slot 19
Time slot 20
       CPU 1: Processed 4 has finished
       CPU 1 stopped
Time slot 21
Time slot 22
```

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

4.2.2.b os_1_mlq_paging

Listing 29: output of os_1_mlq_paging

```
mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os
    os_1_mlq_paging
Time slot  0
ld_routine
Time slot  1
       Loaded a process at input/proc/p0s, PID: 1 PRIO: 130
       CPU 0: Dispatched process 1
Time slot  2
       Loaded a process at input/proc/s3, PID: 2 PRIO: 39
process 1 alloc region 0 size 300
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Memory Dump:
-----------

Time slot  3
       CPU 2: Dispatched process 2
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 1
process 1 alloc region 4 size 300
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------

Time slot  4
       Loaded a process at input/proc/m1s, PID: 3 PRIO: 15
process 1 free region 0
       CPU 3: Dispatched process 3
process 3 alloc region 0 size 300
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Page Number: 0 -> Frame Number: 5
```

```
Page Number: 1 -> Frame Number: 4
Memory Dump:
-----------
Time slot  5
       CPU 2: Put process 2 to run queue
       CPU 2: Dispatched process 2
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 1
process 1 alloc region 1 size 100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------


       Loaded a process at input/proc/s2, PID: 4 PRIO: 120
Time slot  6
       CPU 1: Dispatched process 4
write region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------
BYTE 00000114: 100
process 3 alloc region 1 size 100
print_pgtbl: 0 - 768
00000000: 80000005
00000004: 80000004
00000008: 80000006
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
Page Number: 2 -> Frame Number: 6
Memory Dump:
-----------
BYTE 00000114: 100


       Loaded a process at input/proc/m0s, PID: 5 PRIO: 120
Time slot  7
```

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

```
Time slot 10
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 5
process 5 free region 0
process 3 free region 1
Time slot 11
       CPU 3: Put process 6 to run queue
       CPU 3: Dispatched process 6
       CPU 2: Put process 3 to run queue
       CPU 2: Dispatched process 3
Unable to delocated memory region 1
process 3 free region 1
process 5 alloc region 2 size 100
print_pgtbl: 0 - 768
00000000: 80000008
00000004: 80000007
00000008: 80000009
Page Number: 0 -> Frame Number: 8
Page Number: 1 -> Frame Number: 7
Page Number: 2 -> Frame Number: 9
Memory Dump:
-----------
BYTE 00000114: 100
       Loaded a process at input/proc/s0, PID: 7 PRIO: 38

       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 7
Time slot 12
Unable to delocated memory region 1
process 3 free region 1
       CPU 1: Put process 5 to run queue
       CPU 1: Dispatched process 2
Time slot 13
       CPU 3: Put process 6 to run queue
       CPU 3: Dispatched process 6
       CPU 2: Processed 3 has finished
       CPU 2: Dispatched process 4
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
Time slot 14
       CPU 1: Put process 2 to run queue
       CPU 1: Dispatched process 2
Time slot 15
       CPU 3: Put process 6 to run queue
       CPU 3: Dispatched process 6
       CPU 2: Put process 4 to run queue
       CPU 2: Dispatched process 5
write region=1 offset=20 value=102
print_pgtbl: 0 - 768
00000000: 80000008
00000004: 80000007
00000008: 80000009
Page Number: 0 -> Frame Number: 8
```

```
Page Number: 1 -> Frame Number: 7
Page Number: 2 -> Frame Number: 9
Memory Dump:
-----------
BYTE 00000114: 100
BYTE 00000914: 102
       CPU 1: Processed 2 has finished
       CPU 1: Dispatched process 4

       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
Time slot 16
       Loaded a process at input/proc/s1, PID: 8 PRIO: 0
write region=2 offset=1000 value=1
access violation writing location: memory region 2
Time slot 17
       CPU 3: Put process 6 to run queue
       CPU 3: Dispatched process 8
       CPU 2: Put process 5 to run queue
       CPU 2: Dispatched process 6
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 5
write region=2 offset=1000 value=1
access violation writing location: memory region 2
Time slot 18
       CPU 1: Processed 5 has finished
       CPU 1: Dispatched process 4
       CPU 3: Put process 8 to run queue
       CPU 3: Dispatched process 8
Time slot 19
       CPU 2: Processed 6 has finished
       CPU 2: Dispatched process 1
read region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------
BYTE 00000114: 100
BYTE 00000914: 102
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7

Time slot 20
       CPU 1: Put process 4 to run queue
```

```
       CPU 1: Dispatched process 4
write region=2 offset=20 value=102
access violation writing location: memory region 2
Memory Dump:
-----------
BYTE 00000114: 100
BYTE 00000914: 102

       CPU 3: Put process 8 to run queue
       CPU 3: Dispatched process 8
Time slot 21
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
       CPU 2: Put process 1 to run queue
       CPU 2: Dispatched process 1
read region=2 offset=20
access violation reading location: memory region 2
Time slot 22
       CPU 1: Processed 4 has finished
       CPU 1 stopped
write region=3 offset=20 value=103
access violation writing location: memory region 3
Memory Dump:
-----------
BYTE 00000114: 100
BYTE 00000914: 102

       CPU 3: Put process 8 to run queue
       CPU 3: Dispatched process 8
Time slot 23
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
       CPU 2: Processed 1 has finished
       CPU 2 stopped
       CPU 3: Processed 8 has finished
       CPU 3 stopped
Time slot 24
Time slot 25
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
Time slot 26
       CPU 0: Processed 7 has finished
       CPU 0 stopped
```

4.2.2.c os_1_mlq_paging_small_1K

Listing 30: output of os_1_mlq_paging_small_1K

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

```
ld_routine
Time slot  1
       Loaded a process at input/proc/p0s, PID: 1 PRIO: 130
       CPU 1: Dispatched process 1
Time slot  2
       Loaded a process at input/proc/s3, PID: 2 PRIO: 39
process 1 alloc region 0 size 300
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Memory Dump:
-----------

Time slot  3
       CPU 2: Dispatched process 2
       CPU 1: Put process 1 to run queue
       CPU 1: Dispatched process 1
process 1 alloc region 4 size 300
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------

Time slot  4
process 1 free region 0
       Loaded a process at input/proc/m1s, PID: 3 PRIO: 15
Time slot  5
       CPU 2: Put process 2 to run queue
       CPU 2: Dispatched process 3
process 3 alloc region 0 size 300
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
Memory Dump:
-----------

       CPU 0: Dispatched process 2
       CPU 1: Put process 1 to run queue
       CPU 1: Dispatched process 1
process 1 alloc region 1 size 100
print_pgtbl: 0 - 1024
00000000: 80000001
```

```
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------

Time slot  6
process 3 alloc region 1 size 100
print_pgtbl: 0 - 768
00000000: 80000005
00000004: 80000004
00000008: 80000006
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
Page Number: 2 -> Frame Number: 6
Memory Dump:
-----------

write region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------
BYTE 00000114: 100

       Loaded a process at input/proc/s2, PID: 4 PRIO: 120
Time slot  7
       CPU 3: Dispatched process 4
       CPU 2: Put process 3 to run queue
       CPU 2: Dispatched process 3
process 3 free region 0
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 2
       CPU 1: Put process 1 to run queue
       CPU 1: Dispatched process 1
read region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
```

```
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------
BYTE 00000114: 100

       Loaded a process at input/proc/m0s, PID: 5 PRIO: 120
Time slot  8
process 3 alloc region 2 size 100
print_pgtbl: 0 - 768
00000000: 80000005
00000004: 80000004
00000008: 80000006
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
Page Number: 2 -> Frame Number: 6
Memory Dump:
-----------
BYTE 00000114: 100

write region=2 offset=20 value=102
access violation writing location: memory region 2
Memory Dump:
-----------
BYTE 00000114: 100

Time slot  9
       CPU 3: Put process 4 to run queue
       CPU 3: Dispatched process 5
process 5 alloc region 0 size 300
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000007
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 7
Memory Dump:
-----------
BYTE 00000114: 100

       CPU 2: Put process 3 to run queue
       CPU 2: Dispatched process 3
process 3 free region 2
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 2
       CPU 1: Put process 1 to run queue
       CPU 1: Dispatched process 4
       Loaded a process at input/proc/p1s, PID: 6 PRIO: 15
Time slot 10
process 5 alloc region 1 size 100
print_pgtbl: 0 - 768
00000000: 80000001
00000004: 80000007
```

```
00000008: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 7
Page Number: 2 -> Frame Number: 0
Memory Dump:
-----------
BYTE 00000114: 100

process 3 free region 1
Time slot 11
       CPU 3: Put process 5 to run queue
       CPU 3: Dispatched process 6
       CPU 2: Put process 3 to run queue
       CPU 2: Dispatched process 3
Unable to delocated memory region 1
process 3 free region 1
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 2
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 5
process 5 free region 0
       Loaded a process at input/proc/s0, PID: 7 PRIO: 38
Time slot 12
Unable to delocated memory region 1
process 3 free region 1
process 5 alloc region 2 size 100
print_pgtbl: 0 - 768
00000000: 80000001
00000004: 80000007
00000008: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 7
Page Number: 2 -> Frame Number: 0
Memory Dump:
-----------
BYTE 00000114: 100

Time slot 13
       CPU 3: Put process 6 to run queue
       CPU 3: Dispatched process 6
       CPU 2: Processed 3 has finished
       CPU 2: Dispatched process 7
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 2
       CPU 1: Put process 5 to run queue
       CPU 1: Dispatched process 4
Time slot 14
       CPU 0: Processed 2 has finished
       CPU 0: Dispatched process 5
write region=1 offset=20 value=102
print_pgtbl: 0 - 768
00000000: 80000001
00000004: 80000007
```

```
00000008: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 7
Page Number: 2 -> Frame Number: 0
Memory Dump:
-----------
BYTE 00000014: 102
BYTE 00000114: 100

Time slot 15
       CPU 3: Put process 6 to run queue
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 4
       CPU 3: Dispatched process 6
       CPU 2: Put process 7 to run queue
       CPU 2: Dispatched process 7
write region=2 offset=1000 value=1
access violation writing location: memory region 2
Time slot 16
       CPU 0: Put process 5 to run queue
       CPU 0: Dispatched process 5
write region=2 offset=1000 value=1
access violation writing location: memory region 2
       Loaded a process at input/proc/s1, PID: 8 PRIO: 0
Time slot 17
       CPU 2: Put process 7 to run queue
       CPU 2: Dispatched process 8
       CPU 3: Put process 6 to run queue
       CPU 3: Dispatched process 6
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 7
       CPU 0: Processed 5 has finished
       CPU 0: Dispatched process 4
Time slot 18
Time slot 19
       CPU 2: Put process 8 to run queue
       CPU 2: Dispatched process 8
       CPU 0: Put process 4 to run queue
       CPU 0: Dispatched process 4
       CPU 1: Put process 7 to run queue
       CPU 1: Dispatched process 7
       CPU 3: Put process 6 to run queue
       CPU 3: Dispatched process 6
Time slot 20
Time slot 21
       CPU 0: Processed 4 has finished
       CPU 0: Dispatched process 1
read region=2 offset=20
access violation reading location: memory region 2
       CPU 1: Put process 7 to run queue
       CPU 1: Dispatched process 7
       CPU 3: Processed 6 has finished
       CPU 3 stopped
```

```
       CPU 2: Put process 8 to run queue
       CPU 2: Dispatched process 8
Time slot 22
write region=3 offset=20 value=103
access violation writing location: memory region 3
Memory Dump:
-----------
BYTE 00000014: 102
BYTE 00000114: 100

Time slot 23
       CPU 0: Processed 1 has finished
       CPU 0 stopped
       CPU 1: Put process 7 to run queue
       CPU 1: Dispatched process 7
       CPU 2: Put process 8 to run queue
       CPU 2: Dispatched process 8
Time slot 24
       CPU 2: Processed 8 has finished
       CPU 2 stopped
Time slot 25
       CPU 1: Put process 7 to run queue
       CPU 1: Dispatched process 7
Time slot 26
Time slot 27
       CPU 1: Put process 7 to run queue
       CPU 1: Dispatched process 7
Time slot 28
       CPU 1: Processed 7 has finished
       CPU 1 stopped
```

4.2.2.d os_1_mlq_paging_small_4K

Listing 31: output of os_1_mlq_paging_small_4K

```
mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os
    os_1_mlq_paging_small_4K
Time slot  0
ld_routine
Time slot  1
       Loaded a process at input/proc/p0s, PID: 1 PRIO: 130
Time slot  2
       CPU 2: Dispatched process 1
       Loaded a process at input/proc/s3, PID: 2 PRIO: 39
Time slot  3
process 1 alloc region 0 size 300
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
```

```
Memory Dump:
-----------

       CPU 3: Dispatched process 2
Time slot  4
       CPU 2: Put process 1 to run queue
       CPU 2: Dispatched process 1
process 1 alloc region 4 size 300
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------

       Loaded a process at input/proc/m1s, PID: 3 PRIO: 15
Time slot  5
process 1 free region 0
       CPU 0: Dispatched process 3
process 3 alloc region 0 size 300
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
Memory Dump:
-----------

       CPU 3: Put process 2 to run queue
       CPU 3: Dispatched process 2
Time slot  6
       CPU 2: Put process 1 to run queue
       CPU 2: Dispatched process 1
process 1 alloc region 1 size 100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------

process 3 alloc region 1 size 100
print_pgtbl: 0 - 768
```

```
00000000: 80000005
00000004: 80000004
00000008: 80000006
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
Page Number: 2 -> Frame Number: 6
Memory Dump:
-----------

       Loaded a process at input/proc/s2, PID: 4 PRIO: 120
Time slot  7
write region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------
BYTE 00000114: 100

       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 3
process 3 free region 0
       CPU 1: Dispatched process 4
       Loaded a process at input/proc/m0s, PID: 5 PRIO: 120
       CPU 3: Put process 2 to run queue
       CPU 3: Dispatched process 2
Time slot  8
       CPU 2: Put process 1 to run queue
       CPU 2: Dispatched process 5
process 5 alloc region 0 size 300
print_pgtbl: 0 - 512
00000000: 80000008
00000004: 80000007
Page Number: 0 -> Frame Number: 8
Page Number: 1 -> Frame Number: 7
Memory Dump:
-----------
BYTE 00000114: 100

process 3 alloc region 2 size 100
print_pgtbl: 0 - 768
00000000: 80000005
00000004: 80000004
00000008: 80000006
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
Page Number: 2 -> Frame Number: 6
```

```
Memory Dump:
-----------
BYTE 00000114: 100

Time slot  9
process 5 alloc region 1 size 100
print_pgtbl: 0 - 768
00000000: 80000008
00000004: 80000007
00000008: 80000009
Page Number: 0 -> Frame Number: 8
Page Number: 1 -> Frame Number: 7
Page Number: 2 -> Frame Number: 9
Memory Dump:
-----------
BYTE 00000114: 100

       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 3
process 3 free region 2
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 4
       CPU 3: Put process 2 to run queue
       CPU 3: Dispatched process 2
       Loaded a process at input/proc/p1s, PID: 6 PRIO: 15
Time slot 10
       CPU 2: Put process 5 to run queue
       CPU 2: Dispatched process 6
process 3 free region 1
Time slot 11
       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 3
Unable to delocated memory region 1
process 3 free region 1
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 5
process 5 free region 0
       CPU 3: Put process 2 to run queue
       CPU 3: Dispatched process 2
       Loaded a process at input/proc/s0, PID: 7 PRIO: 38
Time slot 12
       CPU 2: Put process 6 to run queue
       CPU 2: Dispatched process 6
Unable to delocated memory region 1
process 3 free region 1
process 5 alloc region 2 size 100
print_pgtbl: 0 - 768
00000000: 80000008
00000004: 80000007
00000008: 80000009
Page Number: 0 -> Frame Number: 8
Page Number: 1 -> Frame Number: 7
Page Number: 2 -> Frame Number: 9
```

```
Memory Dump:
-----------
BYTE 00000114: 100

Time slot 13
       CPU 0: Processed 3 has finished
       CPU 0: Dispatched process 7
       CPU 3: Put process 2 to run queue
       CPU 3: Dispatched process 2
       CPU 1: Put process 5 to run queue
       CPU 1: Dispatched process 4
Time slot 14
       CPU 2: Put process 6 to run queue
       CPU 2: Dispatched process 6
       CPU 3: Processed 2 has finished
       CPU 3: Dispatched process 5
write region=1 offset=20 value=102
print_pgtbl: 0 - 768
00000000: 80000008
00000004: 80000007
00000008: 80000009
Page Number: 0 -> Frame Number: 8
Page Number: 1 -> Frame Number: 7
Page Number: 2 -> Frame Number: 9
Memory Dump:
-----------
BYTE 00000114: 100
BYTE 00000914: 102

Time slot 15
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
write region=2 offset=1000 value=1
access violation writing location: memory region 2
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 4
Time slot 16
       CPU 2: Put process 6 to run queue
       CPU 2: Dispatched process 6
       CPU 3: Put process 5 to run queue
       CPU 3: Dispatched process 5
write region=2 offset=1000 value=1
access violation writing location: memory region 2
       Loaded a process at input/proc/s1, PID: 8 PRIO: 0
Time slot 17
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 8
       CPU 3: Processed 5 has finished
       CPU 3: Dispatched process 7
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 4
Time slot 18
       CPU 2: Put process 6 to run queue
```

```
       CPU 2: Dispatched process 6
Time slot 19
       CPU 0: Put process 8 to run queue
       CPU 0: Dispatched process 8
       CPU 3: Put process 7 to run queue
       CPU 3: Dispatched process 7
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 4
Time slot 20
       CPU 2: Processed 6 has finished
       CPU 2: Dispatched process 1
read region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------
BYTE 00000114: 100
BYTE 00000914: 102

Time slot 21
write region=2 offset=20 value=102
access violation writing location: memory region 2
Memory Dump:
-----------
BYTE 00000114: 100
BYTE 00000914: 102

       CPU 0: Put process 8 to run queue
       CPU 0: Dispatched process 8
       CPU 3: Put process 7 to run queue
       CPU 3: Dispatched process 7
       CPU 1: Processed 4 has finished
       CPU 1 stopped
Time slot 22
       CPU 2: Put process 1 to run queue
       CPU 2: Dispatched process 1
read region=2 offset=20
access violation reading location: memory region 2
Time slot 23
write region=3 offset=20 value=103
access violation writing location: memory region 3
Memory Dump:
-----------
BYTE 00000114: 100
BYTE 00000914: 102

```

```
       CPU 0: Put process 8 to run queue
       CPU 0: Dispatched process 8
       CPU 3: Put process 7 to run queue
       CPU 3: Dispatched process 7
Time slot 24
       CPU 2: Processed 1 has finished
       CPU 2 stopped
       CPU 0: Processed 8 has finished
       CPU 0 stopped
Time slot 25
       CPU 3: Put process 7 to run queue
       CPU 3: Dispatched process 7
Time slot 26
Time slot 27
       CPU 3: Put process 7 to run queue
       CPU 3: Dispatched process 7
Time slot 28
       CPU 3: Processed 7 has finished
       CPU 3 stopped
```

4.2.2.e os_1_mlq_paging_small_4K

Listing 32: output of os_1_mlq_paging_small_4K

```
mihon@Ubuntu:~/OS/OS-Assignment/ossim_source_code_part2_hk231_paging$ ./os
    os_1_mlq_paging_small_4K
Time slot  0
ld_routine
Time slot  1
       Loaded a process at input/proc/p0s, PID: 1 PRIO: 130
Time slot  2
       CPU 2: Dispatched process 1
       Loaded a process at input/proc/s3, PID: 2 PRIO: 39
Time slot  3
process 1 alloc region 0 size 300
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Memory Dump:
-----------

       CPU 3: Dispatched process 2
Time slot  4
       CPU 2: Put process 1 to run queue
       CPU 2: Dispatched process 1
process 1 alloc region 4 size 300
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
```

```
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------

       Loaded a process at input/proc/m1s, PID: 3 PRIO: 15
Time slot  5
process 1 free region 0
       CPU 0: Dispatched process 3
process 3 alloc region 0 size 300
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
Memory Dump:
-----------

       CPU 3: Put process 2 to run queue
       CPU 3: Dispatched process 2
Time slot  6
       CPU 2: Put process 1 to run queue
       CPU 2: Dispatched process 1
process 1 alloc region 1 size 100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------

process 3 alloc region 1 size 100
print_pgtbl: 0 - 768
00000000: 80000005
00000004: 80000004
00000008: 80000006
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
Page Number: 2 -> Frame Number: 6
Memory Dump:
-----------

       Loaded a process at input/proc/s2, PID: 4 PRIO: 120
Time slot  7
```

```
write region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------
BYTE 00000114: 100

       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 3
process 3 free region 0
       CPU 1: Dispatched process 4
       Loaded a process at input/proc/m0s, PID: 5 PRIO: 120
       CPU 3: Put process 2 to run queue
       CPU 3: Dispatched process 2
Time slot  8
       CPU 2: Put process 1 to run queue
       CPU 2: Dispatched process 5
process 5 alloc region 0 size 300
print_pgtbl: 0 - 512
00000000: 80000008
00000004: 80000007
Page Number: 0 -> Frame Number: 8
Page Number: 1 -> Frame Number: 7
Memory Dump:
-----------
BYTE 00000114: 100

process 3 alloc region 2 size 100
print_pgtbl: 0 - 768
00000000: 80000005
00000004: 80000004
00000008: 80000006
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
Page Number: 2 -> Frame Number: 6
Memory Dump:
-----------
BYTE 00000114: 100

Time slot  9
process 5 alloc region 1 size 100
print_pgtbl: 0 - 768
00000000: 80000008
00000004: 80000007
00000008: 80000009
Page Number: 0 -> Frame Number: 8
```

```
Page Number: 1 -> Frame Number: 7
Page Number: 2 -> Frame Number: 9
Memory Dump:
-----------
BYTE 00000114: 100

       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 3
process 3 free region 2
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 4
       CPU 3: Put process 2 to run queue
       CPU 3: Dispatched process 2
       Loaded a process at input/proc/p1s, PID: 6 PRIO: 15
Time slot 10
       CPU 2: Put process 5 to run queue
       CPU 2: Dispatched process 6
process 3 free region 1
Time slot 11
       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 3
Unable to delocated memory region 1
process 3 free region 1
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 5
process 5 free region 0
       CPU 3: Put process 2 to run queue
       CPU 3: Dispatched process 2
       Loaded a process at input/proc/s0, PID: 7 PRIO: 38
Time slot 12
       CPU 2: Put process 6 to run queue
       CPU 2: Dispatched process 6
Unable to delocated memory region 1
process 3 free region 1
process 5 alloc region 2 size 100
print_pgtbl: 0 - 768
00000000: 80000008
00000004: 80000007
00000008: 80000009
Page Number: 0 -> Frame Number: 8
Page Number: 1 -> Frame Number: 7
Page Number: 2 -> Frame Number: 9
Memory Dump:
-----------
BYTE 00000114: 100

Time slot 13
       CPU 0: Processed 3 has finished
       CPU 0: Dispatched process 7
       CPU 3: Put process 2 to run queue
       CPU 3: Dispatched process 2
       CPU 1: Put process 5 to run queue
       CPU 1: Dispatched process 4
```

```
Time slot 14
       CPU 2: Put process 6 to run queue
       CPU 2: Dispatched process 6
       CPU 3: Processed 2 has finished
       CPU 3: Dispatched process 5
write region=1 offset=20 value=102
print_pgtbl: 0 - 768
00000000: 80000008
00000004: 80000007
00000008: 80000009
Page Number: 0 -> Frame Number: 8
Page Number: 1 -> Frame Number: 7
Page Number: 2 -> Frame Number: 9
Memory Dump:
-----------
BYTE 00000114: 100
BYTE 00000914: 102

Time slot 15
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
write region=2 offset=1000 value=1
access violation writing location: memory region 2
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 4
Time slot 16
       CPU 2: Put process 6 to run queue
       CPU 2: Dispatched process 6
       CPU 3: Put process 5 to run queue
       CPU 3: Dispatched process 5
write region=2 offset=1000 value=1
access violation writing location: memory region 2
       Loaded a process at input/proc/s1, PID: 8 PRIO: 0
Time slot 17
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 8
       CPU 3: Processed 5 has finished
       CPU 3: Dispatched process 7
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 4
Time slot 18
       CPU 2: Put process 6 to run queue
       CPU 2: Dispatched process 6
Time slot 19
       CPU 0: Put process 8 to run queue
       CPU 0: Dispatched process 8
       CPU 3: Put process 7 to run queue
       CPU 3: Dispatched process 7
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 4
Time slot 20
       CPU 2: Processed 6 has finished
       CPU 2: Dispatched process 1
```

```
read region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
Memory Dump:
-----------
BYTE 00000114: 100
BYTE 00000914: 102

Time slot 21
write region=2 offset=20 value=102
access violation writing location: memory region 2
Memory Dump:
-----------
BYTE 00000114: 100
BYTE 00000914: 102

       CPU 0: Put process 8 to run queue
       CPU 0: Dispatched process 8
       CPU 3: Put process 7 to run queue
       CPU 3: Dispatched process 7
       CPU 1: Processed 4 has finished
       CPU 1 stopped
Time slot 22
       CPU 2: Put process 1 to run queue
       CPU 2: Dispatched process 1
read region=2 offset=20
access violation reading location: memory region 2
Time slot 23
write region=3 offset=20 value=103
access violation writing location: memory region 3
Memory Dump:
-----------
BYTE 00000114: 100
BYTE 00000914: 102

       CPU 0: Put process 8 to run queue
       CPU 0: Dispatched process 8
       CPU 3: Put process 7 to run queue
       CPU 3: Dispatched process 7
Time slot 24
       CPU 2: Processed 1 has finished
       CPU 2 stopped
       CPU 0: Processed 8 has finished
       CPU 0 stopped
Time slot 25
       CPU 3: Put process 7 to run queue
```

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

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

4.2.3 Trả lời câu hỏi

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

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

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

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

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

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

Ưu điểm của giải thuật:

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

Hạn chế của giải thuật:

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

4.3 Synchronization

4.3.1 Hoàn thiện code

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

Listing 33: Synchronization for scheduler

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

Listing 34: Synchronization for virtual memory

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

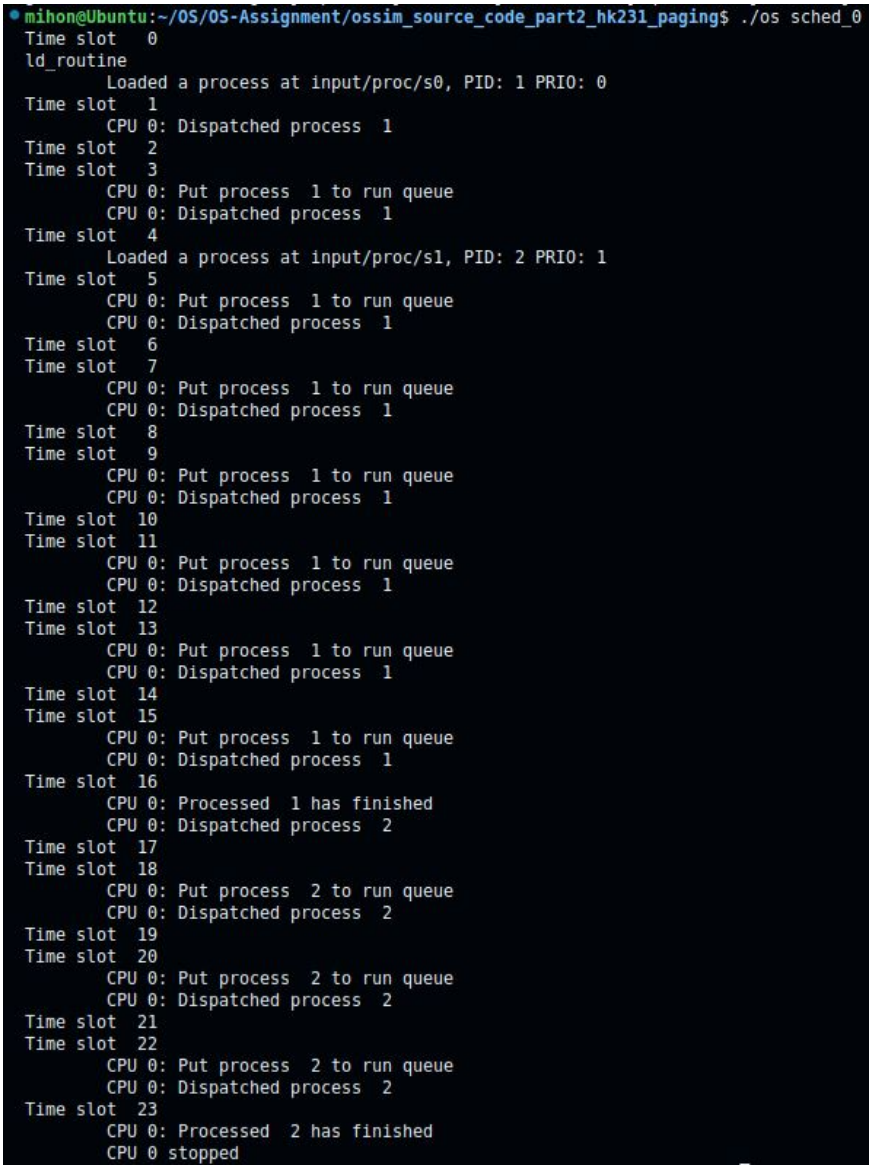
Listing 35: Example of synchronization

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

4.3.2 Trả lời câu hỏi

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

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



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