Mục lục

Phân chia công việc

STT	Thành viên	MSSV	Công việc	Mức độ hoàn thành
1				100%
2				100%
3				100%

Danh sách hình ảnh

1 Scheduler

1.1 Trả lời câu hỏi

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

Chúng ta sẽ khái quát sơ các giải thuật lập lịch đã được học:

- Round Robin:

  + Ưu điểm: Làm tăng tính tương tác của hệ thống, không xảy ra hiện tượng starvation (các tiến trình phải đợi rất lâu trước khi được thực hiện).

  + Nhược điểm: có thể tốn nhiều thời gian để thay đổi trạng thái tiến trình, giảm hiệu suất CPU nếu quantum ngắn hoặc trở nên giống FCFS nếu quantum dài

- First-come First-served:

  + Ưu điểm: dễ hiểu, dễ thực hiện.

  + Nhược điểm: Thời gian đợi trung bình dài.

Trước khi đi vào trả lời câu hỏi, ta cần làm rõ khái niệm của giải thuật lập lịch cho các tiến trình bằng hàng đợi ưu tiên. Đây là một giải thuật lựa chọn process được lấy ra trong hàng đợi phụ thuộc vào độ ưu tiên của nó. Độ ưu tiên được thể hiện bằng một giá trị theo hai kiểu: giá trị ưu tiên càng lớn thì độ ưu tiên càng lớn và ngược lại hoặc giá trị ưu tiên càng nhỏ thì độ ưu tiên càng lớn và ngược lại.

Độ ưu tiên của một tiến trình được khởi tạo ngay lúc tiến trình đó được hình thành và có thể thay đổi trong quá trình thực thi, chúng ta hãy cùng đi qua khái quát về những yếu tố ảnh hưởng đến độ ưu tiên đó:

- Yêu cầu về bộ nhớ của tiến trình
- Yêu cầu về thời gian thực thi của tiến
- Loại tiến trình
- Tỉ lệ giữa thời gian yêu cầu I/O và thời gian tính toán trong CPU

Từ đây, ta có thể thấy được những lợi ích của việc lập lịch bằng hàng đợi ưu tiên so với các giải thuật khác đó là giúp chúng ta dễ can thiệp vào thứ tự thực hiện của các process hơn dựa vào điều chỉnh độ ưu tiên từ đó đạt được các lợi ích cụ thể sau:

Sử dụng bộ nhớ hiệu quả: bằng việc để cho yêu cầu về bộ nhớ chi phối đến độ ưu tiên của quá trình, chúng ta có thể uu tiên cho những tiến trình yêu cầu ít bộ nhớ làm trước, giúp giảm gánh nặng cho bộ nhớ vật lý, đặc biệt đối với các thiết bị nhỏ gọn, yêu cầu thiết kế đơn giản thì điều này đặc biệt quan trọng.

Tăng tốc độ xử lí: bằng việc để cho yêu cầu về thời gian thực thi của tiến trình chi phối đến độ ưu tiên của tiến trình, các tiến trình có burst time nhỏ sẽ được làm trước, làm thời gian chờ đợi của các tiến trình được giảm xuống. Giảm thời gian chờ trung bình của hệ thống.

Chuyên môn hóa hệ thống: mỗi hệ thống sẽ có những ưu tiên về việc thực hiện các loại tiến trình khác nhau là khác nhau. Ví dụ, đối với hệ thống dành cho user thì sẽ ưu tiên các tiến trình có tỉ lệ I/O cao để tăng tính tương tác, đối với các hệ thống cho thiết bị nhúng với giới hạn về phần cứng sẽ ưu tiên các tiến trình chiếm ít bộ nhớ... Những điều này có thể hiện thực bằng cách điều chỉnh các yếu tố ảnh hưởng đến độ ưu tiên của tiến trình.

1.2 Multi level queue (MLQ)

1.2.1 Khái niệm

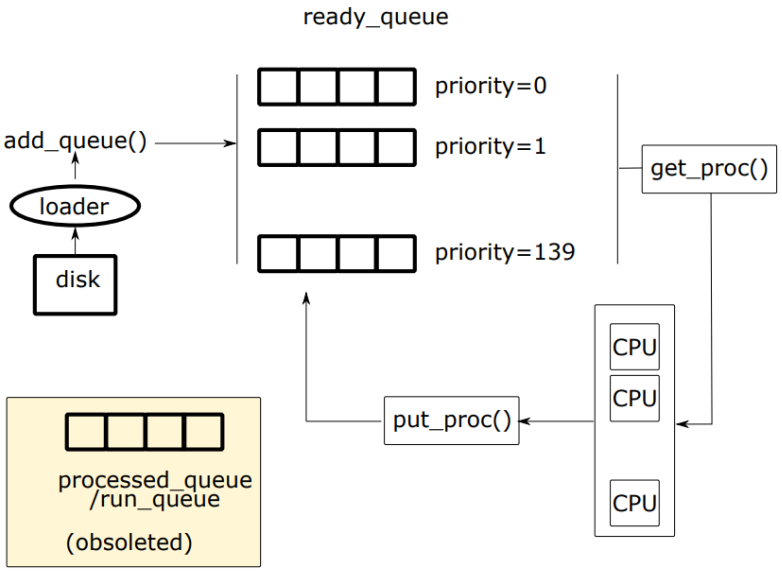
Multi level queue (MLQ) là một hệ thống lập lịch cho các tiến trình mà ở đó có nhiều hàng đợi dùng để chứa các tiến trình và mỗi hàng đợi có thể có các giải thuật lập lịch khác nhau hoặc giống nhau.

Các đặc điểm của multi level queue(MLQ):

- Nhiều hàng đợi: MLQ được chia làm nhiều hàng đợi, mỗi hàng đợi có độ ưu tiên khác nhau. Những hàng đợi có độ ưu tiên cao sẽ được đặt ở mức cao hơn các hàng đợi khác.
- Độ ưu tiên: Độ ưu tiên được khởi tạo cho các process dựa vào loại, đặc điểm và tầm quan trọng của nó.
- Cơ chế phản hồi: Cơ chế phản hồi có thể được hiện thực giúp điều chỉnh độ ưu tiên của một process dựa vào những biểu hiện của nó. Giúp giảm starvation của các process.
- Giải thuật lập lịch: Mỗi hàng đợi có thể được áp dụng một giải thuật lập lịch khác nhau, giúp đáp ứng tốt hơn yêu cầu đặc trưng của các process trong hàng đợi đó.

- Sự ưu tiên: được cho phép trong MLQ, tức là một process có độ ưu tiên thấp hơn dù đang chiếm dụng CPU vẫn phải nhường chỗ cho process có độ ưu tiên cao hơn nó vừa vào hàng đợi.

1.2.2 Giải thuật



Hình 1: Cơ chế vận hành của MLQ

Từ hình trên, ta có thể thấy ready_queue (nơi chứa các process chuẩn bị được đưa vào CPU để thực thi) được chia làm nhiều hàng đợi khác nhau, mỗi hàng đợi có chỉ số priority bằng với số thứ tự của hàng đợi đó (bắt đầu từ 0 và kết thúc ở 139). Hàm *add_queue()* được dùng để đẩy process từ loader vào ready queue. Hàm *get_proc()* dùng để lấy tiến trình từ ready queue vào CPU để thực thi. Hàm put_proc() dùng để đẩy tiến trình chưa thực thi xong (vì một số lý do) vào lại ready queue.

Chúng ta hãy cùng đi vào chi tiết giải thuật của MLQ:

Đầu tiên, loader sẽ tạo ra tiến trình từ chương trình tương ứng và gán PCB cho nó, trong PCB có một trường giá trị là prio, hàm add_queue() sẽ dựa vào giá trị prio này mà đẩy process vào hàng đợi có chỉ số priority tương ứng (tiến trình có prio = 1 sẽ được đẩy vào hàng đợi thứ 2 (priority=1)).

Tiếp theo, hàm get_proc() sẽ lấy process từ các hàng đợi theo cơ chế như sau:

- Lần lượt đi từ hàng đợi có độ ưu tiên cao đến hàng đợi có độ ưu tiên thấp nhất, khi đang duyệt hàng đợi có độ ưu tiên thấp hơn mà hàng đợi có độ ưu tiên cao có thêm process thì vẫn tiếp tục duyệt tuần tự như cũ (không preemptive).
- Ở mỗi hàng đợi sẽ có số lần lấy process là slot = MAX PRIO - priority (được miêu tả ở Hình 2). Nếu một hàng đợi đã được sử dụng hết slot thì sẽ dời lại công việc cho lần duyệt trong tương lai và đi đến hàng đợi tiếp theo.

```
prio = 0        |         1          | ....    | MAX_PRIO - 1
slot = MAX_PRIO |   MAX_PRIO - 1     | ....    |      1
```

Hình 2: Ví dụ về slot trong Linux với MAX PRIO = 140, prio = 0..(MAX PRIO - 1)

- Ở đây, mỗi hàng đợi sẽ được lập lịch theo cơ chế Round Robin, vì thế nên mỗi process nếu vẫn chưa hoàn thành sau khi chạy hết quantum trong môi trường đang chạy thì sẽ được đẩy vào lại ready queue. Nhiệm vụ đẩy process vào lại ready queue do hàm *put_proc()* thực hiện. Do chỉ số *prio* trong PCB của tiến trình không bị thay đổi trong thực thi nên nó sẽ lại được đẩy vào hàng đợi ban đầu.

1.2.3 Hiện thực

a) Hàm enqueue() và dequeue()

Ở đây mỗi hàng đợi trong ready queue có cấu trúc dạng mảng nên hàm enqueue() và dequeue() có nhiệm vụ lần lượt là đẩy 1 tiến trình vào một mảng và lấy một tiến trình ra khỏi mảng.

enqueue():

- Thêm tiến trình vào vị trí cuối của hàng đợi
- Tăng kích thước của hàng đợi lên một

```
void enqueue(struct queue_t *q, struct pcb_t *proc)
{
    q->proc[q->size] = proc;
    q->size++;
}
```

dequeue():

- Nếu queue rỗng thì trả về NULL
- Vì các process trong 1 hàng đợi của MLQ có priority giống nhau nên chỉ cần trả về phần tử đầu tiên trong mảng
- Sau đó dịch trái tất cả các phần tử, xóa phần tử cuối ra khỏi hàng đợi và giảm kích thước trong mảng đi 1

```
struct pcb_t *dequeue(struct queue_t *q)
{
    if (empty(q))
    {
        return NULL;
    }
    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--;
    return temp;
}
```

b) Get_mlq_proc()

Đầu tiên, ta sẽ khai báo mảng queue_slot[], phần tử thứ i của mảng sẽ lưu slot của queue thứ i của MLQ (số lần truy cập của từng queue) theo công thức slot = MAX_PRIO - prio. Ngoài ra, ta tạo thêm môt biến là prio với giá trị khởi tạo là 0 để giữ vị trí của hàng đợi đang được duyệt trong ready queue.

Tiếp theo, ta tiến hành duyệt từng queue trong mlq_ready_queue từ vị trí prio đến cuối. Nếu lần đầu tiên tìm được queue không rỗng và slot $>$ 0 thì dequeue queue đó để thực thi. Cùng với đó sẽ giảm slot đi 1.

Nếu quá trình duyệt ở trên không lấy ra được process nào (ta nhận biết bằng một biến cờ) thì sẽ một lần nữa duyệt từ đầu đến cuối tất cả hàng đợi trong ready queue với mục đích như cũ.

Nếu lần duyệt thứ hai vẫn không lấy ra được process thì sẽ trả về NULL và gán prio = 0.

Khi giảm slot đi 1 sau khi đã lấy tiến trình ra khỏi hàng đợi, ta cần phải lưu ý rằng nếu slot đã giảm về 0 thì phải phục hồi slot về giá trị MAX_PRIO - prio như ban đầu và tăng prio lên 1 (queue hiện tại hết lượt truy cập, nhường tài nguyên cho queue tiếp theo) và nếu prio tăng đến MAX_PRIO (đã duyệt xong queue cuối cùng trong ready queue) thì lại cập nhật prio=0.

Khởi tạo MLQ:

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

Hiện thực hàm get_mlq_proc():

```c
int prio = 0;
struct pcb_t *get_mlq_proc(void)
{
    struct pcb_t *proc = NULL;
    bool flag = false;
    pthread_mutex_lock(&queue_lock);
    for (int i = prio; i < MAX_PRIO; i++)
    {
        if (!empty(&mlq_ready_queue[i]) && queue_slot[i] > 0)
        {
            prio = i;
            flag = true;
            break;
        }
    }
    if (!flag)
    {
        for (int i = 0; i < prio; i++)
        {
            if (!empty(&mlq_ready_queue[i]) && queue_slot[i] > 0)
            {
                prio = i;
                flag = true;
                break;
            }
        }
        if (!flag)
        {
            prio = 0;
            pthread_mutex_unlock(&queue_lock);
            return NULL;
        }
    }
    proc = dequeue(&mlq_ready_queue[prio]);
    queue_slot[prio]--;
    if (queue_slot[prio] == 0)
    {
        queue_slot[prio] = MAX_PRIO - prio;
        prio++;
        if (prio == MAX_PRIO)
        {
            prio = 0;
        }
    }
    pthread_mutex_unlock(&queue_lock);
    return proc;
}
```

1.2.4 Kết quả

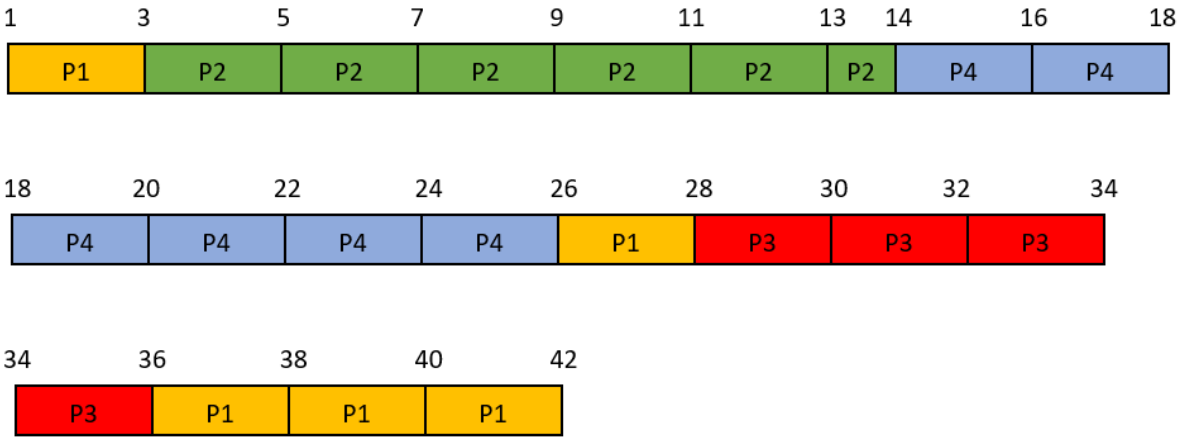
Ta xét input sau:

```
2   1   4
1   p0  139
2   s3  39
4   m1  0
6   m2  120
```

Ở đây, chúng ta đã cài đặt môi trường với:
- Time slice = 2
- Số CPU thực thi = 1
- Số process = 4

Arrival time và priority của các process:
- Process 1 (P1) có arrival time = 1, priority = 139
- Process 2 (P2) có arrival time = 2, priority = 39
- Process 3 (P3) có arrival time = 4, priority = 0
- Process 4 (P4) có arrival time = 6, priority = 120

Từ đây, ta biết được vị trí hàng đợi mà các process được đẩy vào như sau:
- mlq_ready_queue[0]: P3 — slot: MAX_PRIO - prio = 140 - 0 = 140
- mlq_ready_queue[39]: P2 — slot: MAX_PRIO - prio = 140 - 39 = 101
- mlq_ready_queue[120]: P4 — slot: MAX_PRIO - prio = 140 - 120 = 20
- mlq_ready_queue[139]: P1 — slot: MAX_PRIO - prio = 140 - 139 = 1



Hình 3: Sơ đồ Gantt của ví dụ trên

Output sau quá trình thực thi:

```
Time slot 0
    Loaded a process at input/proc/p0, PID: 1 PRIO: 139
Time slot 1
    CPU 0: Dispatched process 1
Time slot 2
    Loaded a process at input/proc/s3, PID: 2 PRIO: 39
Time slot 3
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 2
Time slot 4
    Loaded a process at input/proc/m1, PID: 3 PRIO: 0
Time slot 5
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot 6
    Loaded a process at input/proc/m2, PID: 4 PRIO: 120
```

```
Time slot 7
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot 8
Time slot 9
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot 10
Time slot 11
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot 12
Time slot 13
    CPU 0: Put process 2 to run queue
    CPU 0: Dispatched process 2
Time slot 14
    CPU 0: Process 2 has finished
    CPU 0: Dispatched process 4
Time slot 15
Time slot 16
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 4
Time slot 17
Time slot 18
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 4
Time slot 19
Time slot 20
    CPU 0: Put process 4 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 4
Time slot 23
Time slot 24
    CPU 0: Put process 4 to run queue
    CPU 0: Dispatched process 4
Time slot 25
Time slot 26
    CPU 0: Process 4 has finished
    CPU 0: Dispatched process 1
Time slot 27
Time slot 28
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 3
Time slot 29
Time slot 30
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 3
Time slot 31
Time slot 32
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 3
Time slot 33
Time slot 34
    CPU 0: Put process 3 to run queue
    CPU 0: Dispatched process 3
Time slot 35
Time slot 36
    CPU 0: Process 3 has finished
    CPU 0: Dispatched process 1
```

```
Time slot 37
Time slot 38
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 39
Time slot 40
    CPU 0: Put process 1 to run queue
    CPU 0: Dispatched process 1
Time slot 41
Time slot 42
    CPU 0: Process 1 has finished
    CPU 0: stopped
```

1.2.5 Nhận xét kết quả

Vì chỉ có một CPU thực thi nên ta có thể dễ dàng kiểm chứng thứ tự thực hiện các process. Từ output trên ta có thể thấy thứ tự thực hiện các tiến trình đúng như lý thuyết đã nêu trước đó.

2 Memory Management

2.1 Trả lời câu hỏi

2.1.1 Question 2.1

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 hệ điều hành, chương trình và dữ liệu được lưu trữ trong bộ nhớ. Nhưng không phải toàn bộ bộ nhớ đều cần được yêu cầu sử dụng cho mỗi chương trình. Vì vậy, bộ nhớ được chia thành nhiều phân đoạn khác nhau để phục vụ cho mục đích khác nhau.

Lợi thế của việc chia bộ nhớ thành các phân đoạn:

- Không có phân mảnh nội.
- Bảng phân đoạn tiêu thụ ít không gian hơn so với Bảng trang trong phân trang.
- Người dùng được chỉ định kích thước phân đoạn, trong khi trong phân trang, phần cứng xác định kích thước trang.
- Tính linh hoạt: Phân đoạn cung cấp mức độ linh hoạt cao hơn so với phân trang. Các phân đoạn có thể có kích thước thay đổi và các quy trình có thể được thiết kế để có nhiều phân đoạn, cho phép phân bổ bộ nhớ chi tiết hơn.
- Chia sẻ: Phân đoạn cho phép chia sẻ các phân đoạn bộ nhớ giữa các quy trình. Điều này có thể hữu ích cho giao tiếp giữa các quá trình hoặc để chia sẻ thư viện mã.
- Bảo vệ: Phân đoạn cung cấp mức độ bảo vệ giữa các phân đoạn, ngăn không cho một quá trình truy cập hoặc sửa đổi phân đoạn bộ nhớ của quá trình khác. Điều này có thể giúp tăng tính bảo mật và ổn định của hệ thống. Ví dụ, chương trình không được phép ghi đè vào mã máy của chính nó trong Code segment, nhưng có thể ghi vào Data segment.

2.1.2 Question 2.2

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

Khi kích thước của bảng trang nhỏ hơn kích thước của một Frame thì chúng ta không cần lo lắng vì chúng ta có thể đặt trực tiếp bảng trang vào một frame của bộ nhớ chính. Như vậy chúng ta có thể truy cập trực tiếp vào bảng trang. Nhưng nếu kích thước của bảng trang lớn hơn kích thước của Frame, chúng ta cần phải sử dụng phân trang đa cấp (Multilevel Paging).

Lợi thế khi sử dụng Multilevel Paging:

- Giảm chi phí bộ nhớ: Phân trang đa mức có thể giúp giảm chi phí bộ nhớ liên quan đến bảng trang. Điều này là do mỗi mức chứa ít mục nhập hơn, nghĩa là cần ít bộ nhớ hơn để lưu trữ bảng trang.
- Tra cứu bảng trang nhanh hơn: Với số lượng mục nhỏ hơn trên mỗi cấp, sẽ mất ít thời gian hơn để thực hiện tra cứu bảng trang. Điều này có thể dẫn đến hiệu suất hệ thống tổng thể nhanh hơn.
- Tính linh hoạt: Phân trang đa cấp cung cấp tính linh hoạt cao hơn về cách tổ chức không gian bộ nhớ. Điều này có thể đặc biệt hữu ích trong các hệ thống có yêu cầu bộ nhớ khác nhau, vì nó cho phép điều chỉnh bảng trang để đáp ứng các nhu cầu thay đổi.

Hạn chế của Multilevel Paging:

- Tăng độ phức tạp: Phân trang đa cấp tăng thêm độ phức tạp cho hệ thống quản lý bộ nhớ, điều này có thể gây khó khăn hơn cho việc thiết kế, triển khai và gỡ lỗi.
- Tăng chi phí: Mặc dù phân trang đa cấp có thể giảm chi phí bộ nhớ liên quan đến bảng trang, nhưng nó cũng có thể tăng chi phí liên quan đến tra cứu bảng trang. Điều này là do mỗi cấp độ phải được duyệt qua để tìm mục nhập bảng trang mong muốn.
- Phân mảnh: Phân trang đa cấp có thể dẫn đến phân mảnh không gian bộ nhớ, điều này có thể làm giảm hiệu suất tổng thể của hệ thống. Điều này là do các mục trong bảng trang có thể không liền kề nhau, điều này có thể dẫn đến chi phí bổ sung khi truy cập bộ nhớ.

Nói chung, việc chia địa chỉ thành nhiều cấp đều có những mặt lợi và hạn chế riêng. Cần xem xét nhu cầu cũng như mục tiêu hiện thực để lựa chọn số cấp phân chia địa chỉ vùng nhớ phù hợp.

2.1.3 Question 2.3

What is the advantage and disadvantage of segmentation with paging?

Phân đoạn (segmentation) kết hợp với phân trang (paging) còn được gọi là segmented paging hay paged segmentation là một phương pháp quản lý bộ nhớ trong hệ điều hành kết hợp cả ưu điểm của phân đoạn và phân trang.

Ưu điểm

- Tổ chức bộ nhớ linh hoạt: Phân đoạn cho phép tổ chức bộ nhớ linh hoạt và thuận tiện thành các phân đoạn hợp lý, có thể đại diện cho các phần khác nhau của chương trình (ví dụ: mã, dữ liệu, ngăn xếp). Điều này cho phép quản lý hiệu quả tài nguyên bộ nhớ và hỗ trợ lập trình mô-đun.
- Sử dụng bộ nhớ hiệu quả: Phân trang trong mỗi phân đoạn cho phép sử dụng bộ nhớ hiệu quả. Bằng cách chia các phân đoạn thành các trang có kích thước cố định, việc phân bổ bộ nhớ có thể được thực hiện trên cơ sở trang, giảm phân mảnh nội bộ. Điều này dẫn đến việc sử dụng bộ nhớ tổng thể tốt hơn so với các sơ đồ phân bổ bộ nhớ trong các khối có kích thước cố định.
- Bảo vệ và chia sẻ: Phân đoạn cung cấp sự bảo vệ giữa các phân đoạn, cho phép mỗi phân đoạn có quyền truy cập và quyền hạn riêng. Điều này giúp đảm bảo an toàn dữ liệu và ngăn chặn truy cập trái phép. Ngoài ra, có thể dễ dàng chia sẻ các trang bộ nhớ giữa các phân đoạn khác nhau, thúc đẩy khả năng sử dụng lại mã và chia sẻ bộ nhớ hiệu quả.
- Hỗ trợ bộ nhớ ảo: Sự kết hợp giữa phân đoạn và phân trang cho phép triển khai các hệ thống bộ nhớ ảo. Các phân đoạn và trang có thể được ánh xạ tới bộ lưu trữ đĩa, cho phép không gian địa chỉ logic lớn hơn bộ nhớ vật lý. Điều này tạo điều kiện thuận lợi cho việc quản lý bộ nhớ hiệu quả, vì chỉ những phần bắt buộc của chương trình mới cần được tải vào bộ nhớ vật lý tại bất kỳ thời điểm nào.

Nhược điểm

- Tăng độ phức tạp: Phân đoạn với phân trang giới thiệu độ phức tạp bổ sung so với các sơ đồ phân đoạn hoặc phân trang riêng lẻ. Đơn vị quản lý bộ nhớ (MMU) cần duy trì bảng phân đoạn và bảng trang, yêu cầu hỗ trợ phần cứng bổ sung và tăng chi phí dịch địa chỉ.
- Phân mảnh: Mặc dù phân trang giúp giảm phân mảnh bên trong các phân đoạn, phân mảnh bên ngoài vẫn có thể xảy ra do các phân đoạn và trang có kích thước thay đổi. Theo thời gian, khi các phân đoạn và trang được phân bổ và giải phóng, các khoảng trống của bộ nhớ trống có thể bị phân tán, dẫn đến lãng phí không gian bộ nhớ.
- Tăng chi phí: Sự kết hợp giữa phân đoạn và phân trang đưa ra chi phí bổ sung về các hoạt động quản lý bộ nhớ. MMU cần thực hiện dịch địa chỉ hai cấp, liên quan đến tra cứu trong bảng phân đoạn và bảng trang, có thể tăng thêm độ trễ cho truy cập bộ nhớ.

Nhìn chung, phân đoạn với phân trang cung cấp tính linh hoạt, sử dụng bộ nhớ hiệu quả, bảo vệ và hỗ trợ cho bộ nhớ ảo. Tuy nhiên, nó phải trả giá bằng việc tăng độ phức tạp, phân mảnh và chi phí hoạt động. Những sự đánh đổi này cần được xem xét cẩn thận khi thiết kế và triển khai các hệ thống quản lý bộ nhớ.

2.2 Hệ thống quản lý bộ nhớ theo cơ chế paging

2.2.1 Ánh xạ bộ nhớ ảo trong mỗi process

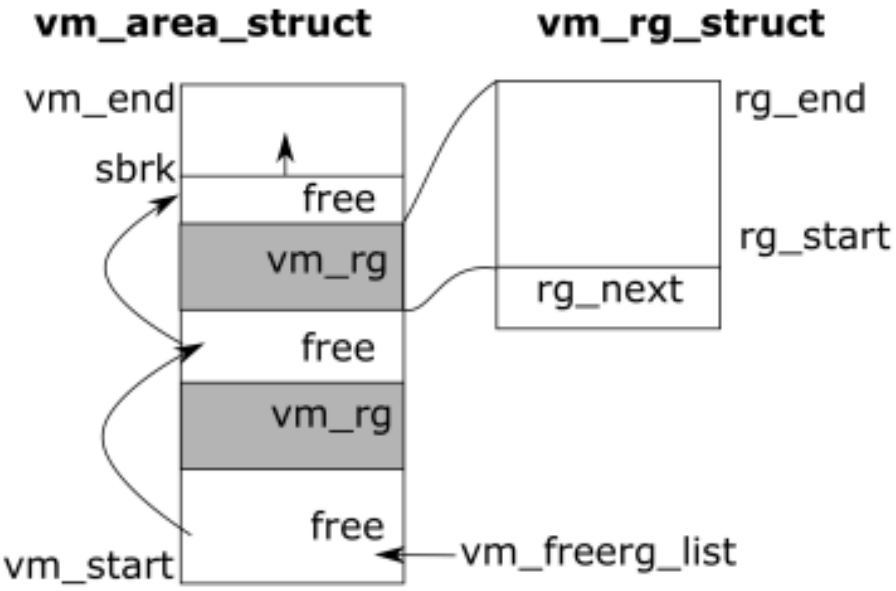
Không gian bộ nhớ ảo được tổ chức dưới dạng ánh xạ bộ nhớ cho từng quy trình PCB. Từ góc nhìn của process, địa chỉ ảo bao gồm nhiều vùng vm areas (liên tục). Trong thực tế, mỗi khu vực có thể hoạt động như code, stack hoặc heap. Do đó, process sẽ giữ trong PCB của nó một con trỏ đến các vùng nhớ liền kề. Một số cấu trúc cần lưu ý trong phần này gồm có Memory Area, Memory Region, Memory Mapping, CPU Address. Tất cả cấu trúc trên và một vài cấu trúc hỗ trợ Virtual Memory khác nằm trong module mm-vm.c.

2.2.2 Hệ thống bộ nhớ vật lý

Tất cả các quy trình sở hữu ánh xạ bộ nhớ riêng biệt của chúng, nhưng tất cả ánh xạ đều nhắm mục tiêu đến một thiết bị vật lý đơn lẻ. có hai các loại thiết bị là RAM và SWAP.

Các thiết bị **RAM**, thuộc hệ thống con bộ nhớ chính, có thể được truy cập trực tiếp từ địa chỉ CPU bus, tức là, có thể được đọc/ghi bằng các lệnh của CPU.

Trong khi đó, **SWAP** chỉ là một thiết bị bộ nhớ phụ và tất cả các thao tác dữ liệu được lưu trữ của nó phải được thực hiện bằng cách di chuyển chúng vào bộ nhớ chính.



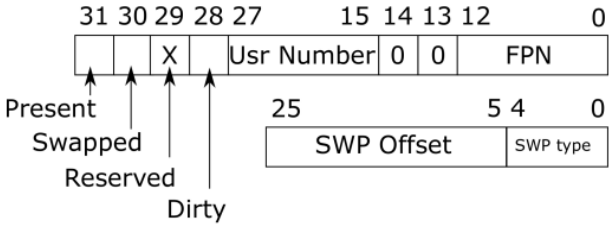
Hình 4: Cấu trúc vm_area và vm_region

Do không có quyền truy cập trực tiếp từ CPU nên hệ thống thường trang bị một SWAP lớn với chi phí nhỏ và thậm chí có nhiều hơn một thiết bị.

Các cấu trúc Framephy Struct, Memphy Struct và một vài cấu trúc khác hỗ trợ Physical Memory nằm trong module *mm-memphy.c*.

2.2.3 Cơ chế phiên dịch địa chỉ

Trong phiên bản này, nhóm phát triển một hệ thống single paging tận dụng một thiết bị RAM và một SWAP. Theo đúng yêu cầu của đề, nhóm chỉ tập trung sử dụng phân đoạn đầu tiên và là phân đoạn duy nhất của vm_area (với vmaid = 0). Cơ chế phiên dịch có thể diễn giải như sau:



Hình 5: Cơ chế phiên dịch địa chỉ trong page table

Page Table: Cho phép một process xác định physical frame mà mỗi virtual page ánh xạ tới. Nó chứa một giá trị 32 bit cho mỗi virtual page. Đối với mỗi entry, số phân trang có thể có một khung được liên kết trong MEMRAM hoặc MEMSWP hoặc có thể có giá trị null, chức năng của từng bit dữ liệu của Page Table Entry được minh họa trong Hình 9.

Memory Swapping: Một vùng bộ nhớ có thể không được sử dụng hết dung lượng lưu trữ giới hạn của nó. Điều đó có nghĩa là có những không gian lưu trữ không được ánh xạ tới MEMRAM. Việc swapping có thể giúp di chuyển nội dung của khung vật lý giữa MEMRAM và MEMSWAP. Swapping là cơ chế thực hiện việc sao chép nội dung của khung từ bên ngoài vào bộ nhớ chính ram. Ngược lại, swapping out cố gắng di chuyển nội dung của khung trong MEMRAM sang MEMSWAP. Trong bối cảnh điển hình, việc hoán đổi giúp chúng tôi có được khung RAM trống do kích thước của thiết bị SWAP thường đủ lớn.

Các thực thi cơ bản trong hệ thống: Hệ thống quản lý bộ nhớ dựa trên cơ chế Paging có các thực thi như ALLOC, FREE, READ, WRITE. Chi tiết về các thực thi này và cấu trúc liên quan nằm

trong module *mm.c*.

2.3 Hiện thực

Trong phần này, nhóm đã sử dụng kiến thức được học để hiện thực các hàm và lệnh được yêu cầu trong phần TODO của các file nguồn (mm-vm.c, mm.c, mm-memphy.c).

- mm.c:
  - Ánh xạ một vùng frames đến một không gian địa chỉ cụ thể trong bảng trang.
  - Phân bổ một số lượng frames cụ thể lên RAM.
- mm-vm.c:
  - Tăng giới hạn vùng nhớ thực khả dụng
  - Giải phóng vùng nhớ.
  - Truy xuất một frame cụ thể trên RAM bằng số trang.
  - Tìm victim page.
  - Kiểm tra một vùng nhớ có bị chồng chéo hay không.
- mm-memphy.c:
  - Xuất nội dung trên bộ nhớ vật lý để theo dõi.

2.4 Trạng thái của RAM

Để thể hiện các trạng thái của RAM một cách rõ ràng hơn, nhóm đã tiến hành tạo ra một process tên p_student_test và file thực thi riêng tên os_student_test. Các lệnh process được gọi như sau:

```
1 7
alloc 128 0
alloc 128 1
write 100 0 10
write 15 1 10
read 2 20 9
free 0
free 1
```

Tiến hành gọi những lệnh này trên hệ thống với config 1048576 16777216 0 0 0 và timeslice=6, ta thu được output như sau:

```
Time slot  0
ld_routine
       Loaded a process at input/proc/p_student_test, PID: 1 PRIO: 0
Time slot  1
       CPU 0: Dispatched process 1
Time slot  2
Time slot  3
write region=0 offset=10 value=100
print_pgtbl: 0 - 512
00000000: c0000000
00000004: c0000020
ADDRESS | VALUE
Time slot  4
write region=1 offset=10 value=15
print_pgtbl: 0 - 512
00000000: c0000000
00000004: c0000020
ADDRESS | VALUE
0000000a: 100
Time slot  5
read region=2 offset=20 value=0
print_pgtbl: 0 - 512
00000000: c0000000
00000004: c0000020
ADDRESS | VALUE
0000000a: 100
```

```
0000008a: 15
Time slot  6
Time slot  7
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 1
Time slot  8
       CPU 0: Processed 1 has finished
       CPU 0 stopped
```

Theo đó

- Sau khi hai lệnh alloc đầu tiên được thực thi, một vùng nhớ trên RAM được cấp phát cho process 1. Lúc này, do chưa có dữ liệu nào được khởi tạo, nên MEMPHY DUMP chưa in ra bất kỳ thông tin gì.
- Tiếp đến, khi hai lệnh write được thực thi, các ô nhớ tương ứng lần lượt được ghi giá trị và được in ra màn hình thông qua MEMPHY DUMP.
- Cuối cùng, khi giải phóng hai vùng nhớ đã tạo bằng lệnh free, process trả lại quyền điều khiển các ô nhớ cho hệ thống. Lúc này, danh sách vùng nhớ được giải phóng sẽ tăng lên và được in ra màn hình. Nội dung ô nhớ không bị thay đổi.

3 Put It All Together

3.1 Trả lời câu hỏi

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.

Nếu trong hệ điều hành đơn giản của chúng ta không có xử lí đồng bộ thì sẽ xảy ra những kết quả sau đây:

- Race condition: xảy ra khi có 2 hoặc nhiều tiến trình cùng truy cập vào 1 lượng tài nguyên nhất định, và có ít nhất 1 tiến trình thao tác ghi/cập nhật dữ liệu trong tài nguyên đó. Nếu không xử lí đồng bộ thì sẽ có thể gây ra khả năng một trong những tiến trình đọc phải dữ liệu cũ, không được cập nhật kịp thời, dẫn tới việc sai sót khi tính toán, gây thất thoát cho người dùng. Ngoài ra, khi nhiều tiến trình đồng thời cùng thực hiện thao tác ghi dữ liệu lên cùng một tài nguyên, có thể xảy ra trường hợp dữ liệu bị hư hỏng hoặc thêm bớt ngoài ý muốn.
- Deadlock: khi chúng ta xử lí đồng bộ bằng lock thì ta phải để ý tới thao tác mở đóng khoá cho phù hợp, tránh trường hợp nhiều tiến trình cùng đợi tài nguyên vô tận, không có điểm dừng, khiến hệ thống bị treo.
- Performance: Bằng cách sử dụng các cơ chế đồng bộ, CPU sẽ tránh bị hao tốn cycle rỗi hoặc bị tiến trình khác chiếm lấy nhưng không để làm gì (do bị tranh chấp tài nguyên hoặc thực hiện quá lâu dẫn tới các tiến trình khác không được thực thi), nhờ đó tăng hiệu năng của hệ thống và cải thiện độ phản hồi với người dùng.

3.2 Kết quả hiện thực

3.2.1 os_1_mlq_paging

- Input

```
2 4 8
1048576 16777216 0 0 0
1 p0s 130
2 s3 39
4 m1s 15
6 s2 120
7 m0s 120
9 p1s 15
11 s0 38
16 s1 0
```

- Output

```
Time slot  0
ld_routine
       Loaded a process at input/proc/p0s, PID: 1 PRIO: 130
Time slot  1
       CPU 3: Dispatched process 1
Time slot  2
       Loaded a process at input/proc/s3, PID: 2 PRIO: 39
Time slot  3
       CPU 1: Dispatched process 2
       Loaded a process at input/proc/m1s, PID: 3 PRIO: 15
       CPU 3: Put process 1 to run queue
       CPU 3: Dispatched process 1
       CPU 0: Dispatched process 3
Time slot  4
       CPU 1: Put process 2 to run queue
       CPU 1: Dispatched process 2
Time slot  5
       Loaded a process at input/proc/s2, PID: 4 PRIO: 120
       CPU 3: Put process 1 to run queue
       CPU 3: Dispatched process 4
Time slot  6
       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 1
       CPU 2: Dispatched process 3
Time slot  7
       Loaded a process at input/proc/m0s, PID: 5 PRIO: 120
write region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: c0000000
00000004: c0000020
00000008: c0000040
00000012: c0000060
ADDRESS | VALUE
       CPU 1: Put process 2 to run queue
       CPU 1: Dispatched process 2
       CPU 3: Put process 4 to run queue
       CPU 0: Put process 1 to run queue
       CPU 3: Dispatched process 5
       CPU 2: Put process 3 to run queue
Time slot  8
       CPU 2: Dispatched process 1
read region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: c0000000
00000004: c0000020
00000008: c0000040
00000012: c0000060
ADDRESS | VALUE
00000098: 100
       CPU 0: Dispatched process 4
       Loaded a process at input/proc/p1s, PID: 6 PRIO: 15
write region=2 offset=20 value=102
print_pgtbl: 0 - 1024
00000000: c0000000
00000004: c0000020
00000008: c0000040
00000012: c0000060
ADDRESS | VALUE
00000098: 100
       CPU 1: Put process 2 to run queue
       CPU 1: Dispatched process 3
Time slot  9
```

```
       CPU 3: Put process 5 to run queue
       CPU 3: Dispatched process 6
Time slot 10
       CPU 0: Put process 4 to run queue
       CPU 0: Dispatched process 2
       CPU 2: Put process 1 to run queue
       CPU 2: Dispatched process 5
       Loaded a process at input/proc/s0, PID: 7 PRIO: 38
Time slot 11
       CPU 1: Put process 3 to run queue
       CPU 1: Dispatched process 4
       CPU 2: Put process 5 to run queue
       CPU 2: Dispatched process 5
Time slot 12
write region=1 offset=20 value=102
print_pgtbl: 0 - 512
00000000: c00000c0
00000004: c00000e0
ADDRESS | VALUE
00000014: 102
00000098: 100
       CPU 3: Put process 6 to run queue
       CPU 3: Dispatched process 1
read region=2 offset=20 value=102
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 3
print_pgtbl: 0 - 1024
00000000: c0000000
00000004: c0000020
00000008: c0000040
00000012: c0000060
ADDRESS | VALUE
00000014: 102
00000098: 100
write region=3 offset=20 value=103
print_pgtbl: 0 - 1024
write region=2 offset=1000 value=1
print_pgtbl: 0 - 512
00000000: c00000c0
00000004: c00000e0
ADDRESS | VALUE
       CPU 1: Put process 4 to run queue
       CPU 1: Dispatched process 6
00000014: 102
00000098: 100
000000a4: 102
Time slot 13
00000000: c0000000
00000004: c0000020
00000008: c0000040
00000012: c0000060
ADDRESS | VALUE
00000014: 102
00000098: 100
000000a4: 102
       CPU 3: Processed 1 has finished
Time slot 14
       CPU 2: Put process 5 to run queue
       CPU 0: Processed 3 has finished
       CPU 0: Dispatched process 2
       CPU 2: Dispatched process 4
       CPU 3: Dispatched process 7
```

```
       CPU 1: Put process 6 to run queue
       CPU 1: Dispatched process 5
write region=0 offset=0 value=0
print_pgtbl: 0 - 512
00000000: c0000100
00000004: c00000e0
ADDRESS | VALUE
00000014: 103
00000098: 100
000000a4: 102
000000b0: 1
Time slot 15
       Loaded a process at input/proc/s1, PID: 8 PRIO: 0
       CPU 2: Put process 4 to run queue
       CPU 2: Dispatched process 4
       CPU 3: Put process 7 to run queue
       CPU 3: Dispatched process 8
Time slot 16
       CPU 1: Processed 5 has finished
       CPU 1: Dispatched process 6
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 7
Time slot 17
       CPU 3: Put process 8 to run queue
       CPU 3: Dispatched process 2
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 8
Time slot 18
       CPU 2: Put process 4 to run queue
       CPU 2: Dispatched process 7
       CPU 1: Put process 6 to run queue
       CPU 1: Dispatched process 4
       CPU 3: Processed 2 has finished
       CPU 3: Dispatched process 6
Time slot 19
       CPU 2: Put process 7 to run queue
       CPU 2: Dispatched process 7
Time slot 20
       CPU 0: Put process 8 to run queue
       CPU 0: Dispatched process 8
       CPU 1: Processed 4 has finished
       CPU 1 stopped
       CPU 3: Put process 6 to run queue
Time slot 21
       CPU 3: Dispatched process 6
Time slot 22
       CPU 0: Put process 8 to run queue
       CPU 0: Dispatched process 8
       CPU 2: Put process 7 to run queue
       CPU 2: Dispatched process 7
       CPU 3: Processed 6 has finished
       CPU 0: Processed 8 has finished
Time slot 23
       CPU 0 stopped
       CPU 3 stopped
Time slot 24
       CPU 2: Put process 7 to run queue
       CPU 2: Dispatched process 7
Time slot 25
Time slot 26
       CPU 2: Put process 7 to run queue
       CPU 2: Dispatched process 7
```

```
Time slot 27
Time slot 28
       CPU 2: Put process 7 to run queue
       CPU 2: Dispatched process 7
Time slot 29
       CPU 2: Processed 7 has finished
       CPU 2 stopped
```

3.2.2 os_1_mlq_paging_small_4K

- Input

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

- Output

```
Time slot  0
ld_routine
       Loaded a process at input/proc/p0s, PID: 1 PRIO: 130
       CPU 2: Dispatched process 1
Time slot  1
       Loaded a process at input/proc/s3, PID: 2 PRIO: 39
       CPU 1: Dispatched process 2
Time slot  2
       CPU 2: Put process 1 to run queue
       CPU 2: Dispatched process 1
Time slot  3
       Loaded a process at input/proc/m1s, PID: 3 PRIO: 15
       CPU 3: Dispatched process 3
Time slot  4
       CPU 1: Put process 2 to run queue
       CPU 1: Dispatched process 2
Time slot  5
       CPU 2: Put process 1 to run queue
       CPU 2: Dispatched process 1
       Loaded a process at input/proc/s2, PID: 4 PRIO: 120
       CPU 3: Put process 3 to run queue
       CPU 3: Dispatched process 3
Time slot  6
write region=1 offset=20 value=100
       CPU 0: Dispatched process 4
print_pgtbl: 0 - 1024
00000000: c0000000
00000004: c0000020
00000008: c0000040
00000012: c0000060
ADDRESS | VALUE
       CPU 1: Put process 2 to run queue
       CPU 1: Dispatched process 2
       Loaded a process at input/proc/m0s, PID: 5 PRIO: 120
Time slot  7
       CPU 2: Put process 1 to run queue
       CPU 2: Dispatched process 5
```

```
       CPU 3: Put process 3 to run queue
       CPU 0: Put process 4 to run queue
       CPU 3: Dispatched process 1
read region=1 offset=20 value=100
Time slot  8
       CPU 0: Dispatched process 3
print_pgtbl: 0 - 1024
00000000: c0000000
00000004: c0000020
00000008: c0000040
00000012: c0000060
ADDRESS | VALUE
00000098: 100
       CPU 1: Put process 2 to run queue
       CPU 1: Dispatched process 2
       Loaded a process at input/proc/p1s, PID: 6 PRIO: 15
Time slot  9
write region=2 offset=20 value=102
print_pgtbl: 0 - 1024
00000000: c0000000
00000004: c0000020
00000008: c0000040
00000012: c0000060
ADDRESS | VALUE
00000098: 100
       CPU 2: Put process 5 to run queue
       CPU 2: Dispatched process 4
       CPU 3: Put process 1 to run queue
       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 5
       CPU 3: Dispatched process 1
read region=2 offset=20 value=102
print_pgtbl: 0 - 1024
00000000: c0000000
00000004: c0000020
00000008: c0000040
00000012: c0000060
ADDRESS | VALUE
00000014: 102
00000098: 100
Time slot 10
       CPU 1: Put process 2 to run queue
       CPU 1: Dispatched process 6
       Loaded a process at input/proc/s0, PID: 7 PRIO: 38
       CPU 2: Put process 4 to run queue
       CPU 2: Dispatched process 3
write region=3 offset=20 value=103
print_pgtbl: 0 - 1024
00000000: c0000000
00000004: c0000020
00000008: c0000040
00000012: c0000060
ADDRESS | VALUE
00000014: 102
00000098: 100
Time slot 11
       CPU 3: Processed 1 has finished
       CPU 3: Dispatched process 7
       CPU 1: Put process 6 to run queue
       CPU 1: Dispatched process 2
Time slot 12
       CPU 0: Put process 5 to run queue
```

```
       CPU 0: Dispatched process 4
       CPU 2: Processed 3 has finished
       CPU 2: Dispatched process 5
write region=1 offset=20 value=102
print_pgtbl: 0 - 512
00000000: c00000c0
00000004: c00000e0
ADDRESS | VALUE
Time slot 13
00000014: 103
00000098: 100
       CPU 3: Put process 7 to run queue
Time slot 14
       CPU 1: Put process 2 to run queue
       CPU 1: Dispatched process 7
       CPU 3: Dispatched process 6
       CPU 0: Put process 4 to run queue
       CPU 0: Dispatched process 2
write region=2 offset=1000 value=1
print_pgtbl: 0 - 512
00000000: c00000c0
00000004: c00000e0
ADDRESS | VALUE
00000014: 103
00000098: 100
000000a4: 102
       CPU 2: Put process 5 to run queue
       CPU 0: Processed 2 has finished
       CPU 0: Dispatched process 5
write region=0 offset=0 value=0
print_pgtbl: 0 - 512
00000000: c0000100
00000004: c00000e0
ADDRESS | VALUE
00000014: 103
00000098: 100
000000a4: 102
000000b0: 1
Time slot 15
       CPU 2: Dispatched process 4
       Loaded a process at input/proc/s1, PID: 8 PRIO: 0
Time slot 16
       CPU 3: Put process 6 to run queue
       CPU 3: Dispatched process 8
       CPU 1: Put process 7 to run queue
       CPU 1: Dispatched process 6
       CPU 0: Processed 5 has finished
       CPU 0: Dispatched process 7
       CPU 2: Put process 4 to run queue
       CPU 2: Dispatched process 4
Time slot 17
       CPU 3: Put process 8 to run queue
       CPU 3: Dispatched process 8
       CPU 1: Put process 6 to run queue
       CPU 1: Dispatched process 6
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
Time slot 18
       CPU 2: Put process 4 to run queue
       CPU 2: Dispatched process 4
Time slot 19
       CPU 3: Put process 8 to run queue
```

```
       CPU 3: Dispatched process 8
Time slot 20
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
       CPU 1: Put process 6 to run queue
       CPU 1: Dispatched process 6
Time slot 21
       CPU 2: Processed 4 has finished
       CPU 2 stopped
       CPU 1: Processed 6 has finished
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
       CPU 1 stopped
       CPU 3: Put process 8 to run queue
       CPU 3: Dispatched process 8
Time slot 22
       CPU 3: Processed 8 has finished
       CPU 3 stopped
Time slot 23
Time slot 24
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
Time slot 25
Time slot 26
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
Time slot 27
       CPU 0: Processed 7 has finished
       CPU 0 stopped
```

3.2.3 os_1_singleCPU_mlq_paging

- Input

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

- Output

```
Time slot  0
ld_routine
       Loaded a process at input/proc/s4, PID: 1 PRIO: 4
Time slot  1
Time slot  2
       CPU 0: Dispatched process 1
       Loaded a process at input/proc/s3, PID: 2 PRIO: 3
Time slot  3
       Loaded a process at input/proc/m1s, PID: 3 PRIO: 2
Time slot  4
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 1
Time slot  5
       Loaded a process at input/proc/s2, PID: 4 PRIO: 3
       CPU 0: Put process 1 to run queue
```

```
       CPU 0: Dispatched process 1
Time slot  6
Time slot  7
       Loaded a process at input/proc/m0s, PID: 5 PRIO: 3
Time slot  8
       CPU 0: Put process 1 to run queue
       CPU 0: Dispatched process 1
       Loaded a process at input/proc/p1s, PID: 6 PRIO: 2
Time slot  9
       CPU 0: Processed 1 has finished
       CPU 0: Dispatched process 3
Time slot 10
       Loaded a process at input/proc/s0, PID: 7 PRIO: 1
       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 6
Time slot 11
Time slot 12
Time slot 13
       CPU 0: Put process 6 to run queue
       CPU 0: Dispatched process 3
Time slot 14
Time slot 15
       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 6
       Loaded a process at input/proc/s1, PID: 8 PRIO: 0
Time slot 16
Time slot 17
       CPU 0: Put process 6 to run queue
       CPU 0: Dispatched process 3
Time slot 18
Time slot 19
       CPU 0: Put process 3 to run queue
       CPU 0: Dispatched process 6
Time slot 20
Time slot 21
       CPU 0: Put process 6 to run queue
       CPU 0: Dispatched process 3
Time slot 22
Time slot 23
       CPU 0: Processed 3 has finished
       CPU 0: Dispatched process 6
Time slot 24
Time slot 25
       CPU 0: Put process 6 to run queue
       CPU 0: Dispatched process 6
Time slot 26
Time slot 27
       CPU 0: Processed 6 has finished
       CPU 0: Dispatched process 2
Time slot 28
Time slot 29
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 4
Time slot 30
Time slot 31
       CPU 0: Put process 4 to run queue
       CPU 0: Dispatched process 5
Time slot 32
Time slot 33
       CPU 0: Put process 5 to run queue
       CPU 0: Dispatched process 2
Time slot 34
```

```
Time slot 35
       CPU 0: Put process 2 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 5
Time slot 38
Time slot 39
       CPU 0: Put process 5 to run queue
       CPU 0: Dispatched process 2
Time slot 40
Time slot 41
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 4
Time slot 42
Time slot 43
       CPU 0: Put process 4 to run queue
       CPU 0: Dispatched process 5
write region=1 offset=20 value=102
print_pgtbl: 0 - 512
00000000: c0000040
00000004: c0000060
ADDRESS | VALUE
Time slot 44
write region=2 offset=1000 value=1
print_pgtbl: 0 - 512
00000000: c0000040
00000004: c0000060
ADDRESS | VALUE
000000a4: 102
Time slot 45
       CPU 0: Put process 5 to run queue
       CPU 0: Dispatched process 2
Time slot 46
Time slot 47
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 4
Time slot 48
Time slot 49
       CPU 0: Put process 4 to run queue
       CPU 0: Dispatched process 5
write region=0 offset=0 value=0
print_pgtbl: 0 - 512
00000000: c0000080
00000004: c0000060
ADDRESS | VALUE
000000a4: 102
000000b0: 1
Time slot 50
       CPU 0: Processed 5 has finished
       CPU 0: Dispatched process 2
Time slot 51
Time slot 52
       CPU 0: Put process 2 to run queue
       CPU 0: Dispatched process 4
Time slot 53
Time slot 54
       CPU 0: Put process 4 to run queue
       CPU 0: Dispatched process 2
Time slot 55
       CPU 0: Processed 2 has finished
```

```
       CPU 0: Dispatched process 4
Time slot 56
Time slot 57
       CPU 0: Processed 4 has finished
       CPU 0: Dispatched process 8
Time slot 58
Time slot 59
       CPU 0: Put process 8 to run queue
       CPU 0: Dispatched process 8
Time slot 60
Time slot 61
       CPU 0: Put process 8 to run queue
       CPU 0: Dispatched process 8
Time slot 62
Time slot 63
       CPU 0: Put process 8 to run queue
       CPU 0: Dispatched process 8
Time slot 64
       CPU 0: Processed 8 has finished
       CPU 0: Dispatched process 7
Time slot 65
Time slot 66
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
Time slot 67
Time slot 68
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
Time slot 69
Time slot 70
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
Time slot 71
Time slot 72
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
Time slot 73
Time slot 74
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
Time slot 75
Time slot 76
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
Time slot 77
Time slot 78
       CPU 0: Put process 7 to run queue
       CPU 0: Dispatched process 7
Time slot 79
       CPU 0: Processed 7 has finished
       CPU 0 stopped
```

4 Kết luận

Trong bài tập lớn này, nhóm đã hoàn thành mô phỏng một hệ điều hành đơn giản bao gồm scheduler (bộ định thời), memory management (hệ thống quản lý bộ nhớ) cũng như là synchronization (đồng bộ hóa). Từ đó, nhóm đã hiểu thêm về các phần lý thuyết dược học trên lớp và các buổi thí nghiệm.

References

[1] Abraham Silberschatz, Peter B. Galvin, Greg Gagne, *Operating System Concepts* 10th. 2018

[2] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 2015.

[3] Multilevel Queue (MLQ) CPU Scheduling (https://www.geeksforgeeks.org/multilevel-queue-mlq-cpu-scheduling)

[4] Segmentation in Operating System (https://www.geeksforgeeks.org/segmentation-in-operating-system/)

[5] Multilevel Paging in Operating System (https://www.geeksforgeeks.org/multilevel-paging-in-operating-system/)