VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEM (CO2017)

Assignment

Simple Operating System

Class: **L07**
Advisor: **Hoàng Lê Hải Thanh**

Students: Bùi Thế Kỷ Cương - 2210412
Nguyễn Minh Triết - 2213608
Phùng Xương Cận - 2210348
Nguyễn Duy Thức - 2213426

HO CHI MINH CITY, MAY 2024

MỤC LỤC

1 Danh sách thành viên

No.	Fullname	Student ID	Problems	Percentage of work
1	Bùi Thế Kỷ Cương	2210412	Memory Management	25%
2	Nguyễn Minh Triết	2213608	Memory Management	25%
3	Phùng Xương Cận	2210348	Memory Management	25%
4	Nguyễn Duy Thức	2213426	Scheduler + Viết báo cáo	25%

2 Lời nói đầu

3 Scheduler

3.1 Câu hỏi lí thuyết

Câu hỏi 1: "What is the advantage of the scheduling strategy used in this assignment in comparison with other scheduling algorithms you have learned?"
Dịch: Ưu điểm của chiến thuật định thời được dùng trong bài tập lớn này là gì, so sánh nó với những giải thuật định thời mà bạn đã được học?
Trả lời: Những giải thuật đã được học là:

a) **First Come First Service (FCFS):**

- **Ưu điểm:**
  - Đơn giản, quá trình nào đến trước sẽ thực thi trước
- **Nhược điểm:**
  - Nếu một quá trình có thời gian thực thi dài thì làm cho những quá trình phía sau phải đợi CPU và hiệu quả kém.

b) **Shortest Job First (SJF):**

- **Ưu điểm:**
  - Những process có thời gian thực thi ngắn (short process) sẽ thực hiện trước, những process có thời gian thực thi dài hơn sẽ chạy sau.
  - Điều này giúp cho thông lượng tăng lên vì nhiều process có thể thực hiện trong thời gian ngắn hơn tổng thời gian thực hiện của chúng.
- **Nhược điểm:**
  - Phải dự đoán trước thời gian thực thi của các quá trình. Điều này khó thể làm được.
  - Những process có thời gian hiện thực lâu hơn sẽ phải đợi một khoảng thời gian, thậm chí không được thực hiện (starvation).

c) **Round Robin (RR:**

- **Ưu điểm:**
  - Mỗi process được thực thi một khoảng thời gian nhất định.
  - Tránh tình trạng starvation.
- **Nhược điểm:**
  - Thông lượng phụ thuộc vào thời gian mỗi quá trình được hiện thực.
  - Nếu thời gian quá dài thì giống như FIFO.
  - Nếu thời gian quá nhỏ thì sẽ gây ra chuyển ngữ cảnh (context switch liên tục).

d) **Priority Scheduling:**

- **Ưu điểm:**
  - Process có độ ưu tiên cao, quan trọng được thực thi trước.
  - Hỗ trợ tốt trong thời gian thực.
- **Nhược điểm:**
  - Những process có độ ưu tiên thấp sẽ bị starvation.
  - Không tối ưu cho mọi trường hợp.

e) **Multilevel Feedback Queue:**

- **Ưu điểm:**
  - Ưu tiên cho process mức cao.
  - Một quá trình có thể di chuyển giữa các hàng đợi.
- **Nhược điểm:**
  - Nguy cơ starvation cho các process có mức thấp.
  - Phức tạp trong triển khai.

Giải thuật định thời được sử dụng trong bài tập lớn này là Multilevel Queue (MLQ). Đây là một thuật toán trong đó các quá trình được phân ở các mức khác nhau, mỗi nhóm được quản lý bằng một hàng đợi riêng biệt với mức độ ưu tiên khác nhau. Sau đây là ưu điểm và nhước điểm của MLQ:

- **Ưu điểm:**
  - Dễ dàng quản lý, những quá trình ở mức cao sẽ được thực thi trước.
  - Những quá trình cùng một mức độ ưu tiên sẽ được công bằng trong việc thực thi.
- **Nhược điểm:**
  - Dễ xảy ra hiện tượng starvation cho những quá trình ở mức thấp.
  - Khó khăn trong việc phân loại quá trình.

3.2 Code những phần yêu cầu

3.2.1 Scheduler

a) *Trong queue.c:*

- *enqueue():* Đưa một process mới vào cuối queue (hàng đợi).
- *dequeue():* Lấy ra một process ở vị trí đầu tiên của queue (hàng đợi).

Sau đây là code:

```c
void enqueue(struct queue_t * q, struct pcb_t * proc)
{
    if (q->size == MAX_QUEUE_SIZE)
    {
        perror("There␣are␣no␣available␣slots␣in␣this␣queue");
        return;
    }
    else
    {
        q->proc[q->size++] = proc;
    }
}

struct pcb_t * dequeue(struct queue_t * q)
{
        /* TODO: return a pcb whose prioprity is the highest
         * in the queue [q] and remember to remove it from q
         * */
    if (q->size == 0)
    {
        perror("There␣are␣no␣process␣in␣this␣queue");
        return NULL;
    }
    struct pcb_t * proc_out = q->proc[0];
    if (q->size == 1)
    {
        q->proc[0] = NULL;
        q->size--;
        return proc_out;
    }
    else
    {
        memmove(&(q->proc[0]), &(q->proc[1]), (q->size - 1) * sizeof(
            proc_out));
        q->proc[q->size] = NULL;
        q->size--;
        return proc_out;
    }
}
```

b) *Trong sched.c:*

get_ mlq_ proc(): Lấy một process từ PRIORITY (ready_queue).

```c
struct pcb_t *get_mlq_proc(void)
{
    struct pcb_t *proc = NULL;

    /*TODO: get a process from PRIORITY [ready_queue].
     * Remember to use lock to protect the queue.
     * */
    pthread_mutex_lock(&queue_lock);

    for (int i = 0; i < MAX_PRIO; i++)
    {
        if (!empty(&mlq_ready_queue[i]))
        {
            if (mlq_ready_queue[i].timeslot != 0)
            {
                proc = dequeue(&mlq_ready_queue[i]);
                mlq_ready_queue[i].timeslot--;
                break;
            }
        }
    }

    if (proc == NULL)
    {
        for (int i = 0; i < MAX_PRIO; i++)
        {
            mlq_ready_queue[i].timeslot = MAX_PRIO - i;
        }
        // run the algo again to get the proc after reset the time slot of
            every queue
        for (int i = 0; i < MAX_PRIO; i++)
        {
            if (!empty(&mlq_ready_queue[i]))
            {
                if (mlq_ready_queue[i].timeslot != 0)
                {
                    proc = dequeue(&mlq_ready_queue[i]);
                    mlq_ready_queue[i].timeslot--;
                    break;
                }
            }
        }
```

```
42      }
43      pthread_mutex_unlock(&queue_lock);
44      return proc;
45  }
```

3.2.2 Biểu đồ Gantt

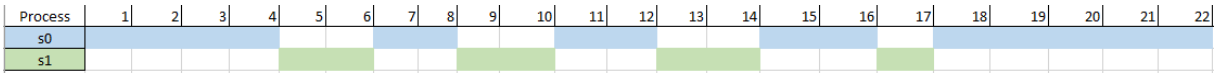
a) **Thực thi file input sched_0**

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

```
1  2 1 2
2  0 s0
3  4 s1
```

Process s0 có 15 lệnh. *Process s1* có 7 lệnh. Mỗi lệnh cần 1 đơn vị thời gian để thực thi. Ở ví dụ này, các process có cùng độ ưu tiên

- - Process s0 thực thi tại thời điểm 1, sau 2 đơn vị thời gian nó sẽ bị tạm dừng và được đẩy vào run_queue. Do ban đầu chỉ có process s0 nên ready_queue hiện tại đang trống, process s0 được đẩy qua ready_queue để tiếp tục thực thi.
- - Tại thời điểm thứ 5, process s1 được nạp vào để thực thi, tới thời điểm thứ 7 thì process s1 được đẩy vào run_queue, sau đó cả 2 process s0 và s1 được đẩy qua ready_queue, do s1 có độ ưu tiên cao hơn nên s1 tiếp tục được thực thi tại thời điểm 7. Cả 2 process tiếp tục thực thi như vậy cho đến khi cả hai thực thi xong.



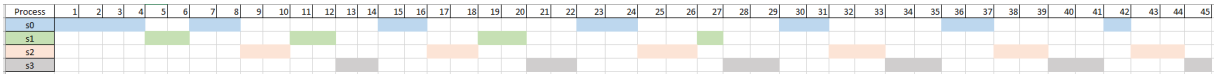
Hình 1: Biểu đồ Gantt của sched_0

b) **Thực thi file input sched_1**

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

```
1  2 1 4
2  0 s0
3  4 s1
4  6 s2
5  7 s3
```

Process s0 có 15 lệnh. *Process s1* có 7 lệnh. *Process s2* có 12 lệnh. *Process s3* có 11 lệnh. Mỗi lệnh cần 1 đơn vị thời gian để thực thi. Quá trình thực thi tương tự như thực thi file input sched_0. Sau đây là biểu đồ Gantt.



Hình 2: Biểu đồ Gantt của sched_1

4 Memory Management

4.1 Câu hỏi lí thuyết

Câu hỏi 2: "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?"
Dịch: Trong hệ điều hành đơn giản này, chúng ta hiện thực một thiết kế phân đoạn bộ nhớ hoặc vùng nhớ trong khai báo mã nguồn. Ưu điểm của đề xuất thiết kế nhiều phân đoạn là gì?
Trả lời:

- **Không xảy ra hiện tượng phân mảnh nội:** Không giống như một số kỹ thuật quản lý bộ nhớ khác, phân đoạn tránh phân mảnh nội. Mỗi phân đoạn có thể có kích thước thay đổi, cho phép sử dụng bộ nhớ hiệu quả hơn.
- **Hiệu quả về không gian:** Bảng phân đoạn tiêu tốn ít không gian hơn so với bảng trang được sử dụng trong phân trang. Điều này dẫn đến sự biểu diễn nhỏ gọn hơn của các phân đoạn bộ nhớ.
- **Cải thiện việc sử dụng CPU:** Vì các mô-đun (phân đoạn) hoàn chỉnh được tải vào bộ nhớ cùng một lúc nên việc phân đoạn sẽ tăng cường việc sử dụng CPU. Điều này đặc biệt có lợi khi thực hiện các chương trình có.

Câu hỏi 3: "What will happen if we divide the address to more than 2-levels in the paging memory management system?"
Dịch: Điều gì sẽ xảy ra nếu chung ta chia địa chỉ thành hơn 2 cấp trong hệ thống quản lý bộ nhớ phân trang?
Trả lời:

- **Giảm chi phí bộ nhớ:** Trong hệ thống phân trang đa cấp, chúng ta tổ chức các bảng trang thành một cấu trúc phân cấp. Mỗi cấp độ chứa ít mục hơn, dẫn đến sử dụng bộ nhớ nhỏ hơn để lưu trữ bảng trang. Bảng trang ngoài cùng (thường là cấp độ đầu tiên) nằm trong bộ nhớ chính, trong khi các bảng trang khác được đưa vào khi cần thiết. Điều này giúp giảm lãng phí bộ nhớ vì chúng ta không cần giữ đồng thời tất cả các bảng trang trong bộ nhớ.
- **Tra cứu bảng nhanh hơn:** Với ít mục nhập hơn cho mỗi cấp độ, thời gian cần thiết để thực hiện tra cứu bảng trang sẽ giảm đi. Mỗi mục trong bảng trang (ngoại trừ cấp độ cuối cùng) chứa địa chỉ cơ sở của bảng trang cấp độ tiếp theo. Để dịch một địa chỉ logic thành địa chỉ vật lý, chúng tôi tuân theo một chuỗi truy cập bộ nhớ thông qua các cấp độ này.
- **Dịch địa chỉ theo từng giai đoạn:** Phân trang đa cấp cho phép dịch địa chỉ hiệu quả bằng cách

thực hiện dịch theo từng giai đoạn. Cách tiếp cận này cho phép phân giải địa chỉ nhanh hơn và hiệu quả hơn, cuối cùng giảm thời gian cần thiết để truy cập bộ nhớ.

- **Khả năng thích ứng:** Nếu kích thước bảng trang vượt quá kích thước mong muốn, chúng ta có thể tạo các cấp độ bổ sung để chứa không gian địa chỉ lớn hơn.Tính linh hoạt này đảm bảo rằng hệ thống có thể xử lý các yêu cầu bộ nhớ khác nhau.

Câu hỏi 4: "What is the advantage and disadvantage of segmentation with paging?"
Dịch: Ưu điểm và nhược điểm của phân đoạn với phân trang là gì?
Trả lời:

- **Advantage** (Ưu điểm):
  - Kích thước bảng trang được giảm xuống vì các trang chỉ hiển thị cho dữ liệu của các phân đoạn, do đó giảm yêu cầu bộ nhớ.
  - Hạn chế được hiện tượng phân mảnh ngoại bằng cách phân trang trong mỗi đoạn.
  - Vì không cần phải hoán đổi toàn bộ phân đoạn nên việc hoán đổi bộ nhớ ảo trở nên dễ dàng hơn.
- **Disadvantage** (Nhược điểm):
  - Dễ xảy ra hiện tượng phân mảnh nội.
  - Bắt buộc phải có phần cứng bổ sung.

Câu hỏi 5: "What will happen if the multi-core system has each CPU core can be run in a different context, and each core has its own MMU and its part of the core (the TLB)? In modern CPU, 2-level TLBs are common now, what is the impact of these new memory hardware configurations to our translation schemes?"
Dịch: Điều gì sẽ xảy ra nếu hệ thống đa lõi có mỗi lõi CPU có thể chạy trong một bối cảnh khác nhau và mỗi lõi có MMU riêng và một phần lõi (TLB) của nó ? Trong CPU hiện đại, TLB 2 cấp hiện nay rất phổ biến. Tác động của các cấu hình phần cứng bộ nhớ mới này đối với sơ đồ dịch thuật của chúng ta là gì?
Trả lời:

- **Resource isolation** (Cách ly tài nguyên):
  - MMU và TLB của mỗi lõi hoạt động độc lập, đảm bảo cách ly tài nguyên. Điều này có nghĩa là lỗi TLB của một lõi (core) hoặc việc di chuyển trong bảng trang không ảnh hưởng trực tiếp đến các lõi khác.
  - Sự cô lập là rất quan trọng để có thể dự đoán được hiệu suất, đặc biệt là trong môi trường multi-tenant hoặc dịch vụ đám mây (cloud) nơi các ứng dụng khác nhau chạy đồng thời.

- **Giảm chi phí chia sẻ TLB:**
  - Với các TLB riêng biệt cho từng lõi, sẽ có ít sự tranh chấp hơn đối với các mục TLB được chia sẻ. TLB bị thiếu trên một lõi không làm mất hiệu lực các mục trong TLB của lõi khác.
  - Điều này làm giảm chi phí liên quan đến việc chia sẻ TLB và cải thiện hiệu suất hệ thống tổng thể.
- **Cải thiện sự kết hợp bộ đệm:**
  - Các TLB riêng biệt góp phần kết hợp bộ đệm tốt hơn. Khi một lõi sửa đổi một mục trong bảng trang, nó không ảnh hưởng đến TLB của các lõi khác.
  - Ngược lại, TLB dùng chung sẽ yêu cầu các cơ chế đồng bộ hóa bổ sung để duy trì tính nhất quán giữa các lõi.
- **Hiệu quả của bảng trang Walker (Walker page table):**
  - Bộ đi bảng trang - Page Table Walker (PTW) là thành phần quan trọng của MMU. Trong hệ thống đa lõi (multicore system), PTW có thể được chia sẻ hoặc dành riêng cho mỗi lõi.
  - Nếu PTW được chia sẻ, băng thông của chúng sẽ trở thành tài nguyên được chia sẻ. Truy cập bộ nhớ bùng nổ từ NPU được hưởng lợi từ việc chia sẻ băng thông PTW giữa các lõi.
  - Tuy nhiên, nếu PTW được dành riêng cho mỗi lõi thì mỗi lõi có thể thực hiện dịch địa chỉ một cách độc lập, giảm tranh chấp.
- **Tác động đến khả năng mở rộng:**
  - Khi số lượng lõi tăng lên, tác động của việc chia sẻ tài nguyên trở nên rõ rệt hơn. Khả năng mở rộng phụ thuộc vào việc sử dụng tài nguyên hiệu quả.
  - Việc ánh xạ các mô hình không đồng nhất (Mapping heterogeneous models) tới nhiều NPU (hoặc lõi) đòi hỏi phải xem xét cẩn thận các tài nguyên dùng chung để tránh tắc nghẽn.
- **Sự cân bằng giữa độ phức tạp và thiết kế:** Trong khi các MMU và TLB riêng biệt tăng cường sự cách ly tài nguyên, chúng cũng gây ra sự phức tạp trong thiết kế.

Câu hỏi 6: "What will happen if the synchronization is not handled in your simple OS? Illustrate the problem of your simple OS by example if you have any. Note: You need to run two versions of your simple OS: the program with/without synchronization, then observe their performance based on demo results and explain their differences."

Dịch: Điều gì sẽ xảy ra nếu việc đồng bộ hóa không được xử lý trong hệ điều hành đơn giản của bạn? Hãy minh họa vấn đề của hệ điều hành đơn giản của bạn bằng ví dụ nếu có. Lưu ý: Bạn cần chạy hai phiên bản HĐH đơn giản của mình: chương trình có/không có đồng bộ hóa, sau đó quan sát hiệu suất của chúng dựa trên kết quả demo và giải thích sự khác biệt của chúng.

Trả lời:

Nếu việc đồng bộ hóa không được xử lý trong một hệ điều hành đơn giản, nó có thể dẫn đến nhiều vấn đề khác nhau, bao gồm các điều kiện tương tranh, dữ liệu không nhất quán và xảy ra Deadlock.
Xét một ví dụ đơn giản để minh họa những vấn đề này. Giả sử hệ điều hành đơn giản cho phép nhiều quy trình truy cập và sửa đổi đồng thời một biến được chia sẻ mà không có bất kỳ cơ chế đồng bộ hóa nào:
Process A và Process B là hai tiến trình đồng thời tăng và giảm số lượng của một biến đếm (count) chung. Cả hai tiếng trình đều bắt đầu với biến đếm được khởi tạo bằng 0.

- Process A thực thi đoạn mã sau: count = count + 1
- Process B thực thi đoạn mã sau: count = count - 1

Nếu không sử dụng đồng bộ hóa, chuỗi sự kiện sau có thể xảy ra:

- Process A đọc giá trị hiện tại của số đếm (0) vào thanh ghi.
- Process B đọc giá trị hiện tại của số đếm (0) vào thanh ghi.
- Process A tăng giá trị thanh ghi cục bộ của nó lên 1 (0 + 1 = 1).
- Process B giảm giá trị thanh ghi cục bộ của nó đi 1 (0 - 1 = -1).
- Process A ghi lại giá trị cập nhật của nó (1) vào biến đếm.
- Process B ghi lại giá trị cập nhật của nó (-1) vào biến đếm.

Trong trường hợp này, cả hai quy trình đã thực hiện đồng thời các hoạt động của chúng mà không có bất kỳ sự đồng bộ hóa nào. Kết quả cuối cùng của giá trị biến đếm count không chính xác (là -1 thay vì 0). Vấn đề này được gọi là tình trang chạy đua, trong đó đầu ra của một chương trình phụ thuộc vào thời gian và sự xen kẽ của các hoạt động đồng thời.
Ngoài ra, nếu không có cơ chế đồng bộ phù hợp sẽ có nguy cơ bị Deadlock (Khóa chết). Khóa chết có thể xảy ra khi hai hoặc nhiều tiến trình đang chờ đợi lẫn nhau vô thời hạn để giải phóng tài nguyên. Nếu có các tài nguyên được chia sẻ mà các quy trình cần quyền truy cập độc quyền mà không có sự đồng bộ hóa thích hợp thì có thể xảy ra tình huống trong có nhiều quy trình đang chờ một tài nguyên sẽ không bao giờ được giải phóng.

4.2 Trạng thái của RAM

Dưới đây là kết quả của quá trình sau mỗi lệnh cấp phát, giải phóng, đọc và ghi lệnh trong chương trình của 1 testcase.
Input: /input/ex_syn_mem
Output: /output/ex_syn_mem.output

```
1 Time slot   0
2 ld_routine
3         Loaded a process at input/proc/a0, PID: 1 PRIO: 1
```

```
        CPU 2: Dispatched process  1
---------------------------------------
Process 1 ALLOC CALL | SIZE = 200

>>>>>Alloc Case>>>>>No free region. #RGID: 0
[Page mapping]  PID #1: Frame:0 PTE:80000000    PGN:0
>>>>>Done>>>>> #RGID: 0
---------------------------------------
Process 1 Allocated Region list
Region id 0 : Range = [0 - 200]
---------------------------------------
Process 1 Free Region list
Range = [200 - 256], freespace = 56
---------------------------------------
VMA id 1 : start = 0, end = 256, sbrk = 256
----------- RAM mapping status -----------
Number of mapped frames:        1
Number of remaining frames:     1
---------------------------------------
-----------------LRU LIST-----------------
[0]
---------------------------------------

Time slot   1
---------------------------------------
Process 1 ALLOC CALL | SIZE = 100

>>>>>Alloc Case>>>>>No free region. #RGID: 4
[Page mapping]  PID #1: Frame:1 PTE:80000001    PGN:1
>>>>>Done>>>>> #RGID: 4
---------------------------------------
Process 1 Allocated Region list
Region id 0 : Range = [0 - 200]
Region id 4 : Range = [256 - 356]
---------------------------------------
Process 1 Free Region list
Range = [200 - 256], freespace = 56
Range = [356 - 512], freespace = 156
---------------------------------------
VMA id 1 : start = 0, end = 512, sbrk = 512
----------- RAM mapping status -----------
Number of mapped frames:        2
```

```
Number of remaining frames:     0
-----------------------------------------
-----------------LRU LIST----------------
[0] -> [1]
-----------------------------------------

        Loaded a process at input/proc/m1, PID: 2 PRIO: 0
Time slot   2
        CPU 4: Dispatched process  2
        CPU 2: Put process  1 to run queue
        CPU 2: Dispatched process  1
-----------------------------------------
Process 2 ALLOC CALL | SIZE = 200

>>>>>Alloc Case>>>>>No free region. #RGID: 2
[Page Replacement]      PID #2: Victim:0        PTE:80000000
[After Swap]    PID #2: Victim:0        PTE:40000000
[Page mapping]  PID #2: Frame:0 PTE:80000000    PGN:0
>>>>>Done>>>>> #RGID: 2
-----------------------------------------
Process 2 Allocated Region list
Region id 2 : Range = [0 - 200]
-----------------------------------------
Process 2 Free Region list
Range = [200 - 256], freespace = 56
-----------------------------------------
VMA id 1 : start = 0, end = 256, sbrk = 256
----------- RAM mapping status -----------
Number of mapped frames:        2
Number of remaining frames:     0
-----------------------------------------
-----------------LRU LIST----------------
[1] -> [0]
-----------------------------------------

-----------------------------------------
Process 1 ALLOC CALL | SIZE = 56

>>>>>Alloc Case>>>>> GET FREE RG in FREERG LIST. #RGID: 1

FOUND A FREE region to alloc. REMEMBER to check LRU for update.
-----------------------------------------
```

```
Process 1 Allocated Region list
Region id 0 : Range = [0 - 200]
Region id 1 : Range = [200 - 256]
Region id 4 : Range = [256 - 356]
----------------------------------------
Process 1 Free Region list
Range = [356 - 512], freespace = 156
----------------------------------------
VMA id 1 : start = 0, end = 512, sbrk = 512
----------- RAM mapping status -----------
Number of mapped frames:        2
Number of remaining frames:     0
----------------------------------------
-----------------LRU LIST-----------------
[1] -> [0]
----------------------------------------

Time slot   3
----------------------------------------
Process 2 ALLOC CALL | SIZE = 200

>>>>>Alloc Case>>>>>No free region. #RGID: 5
[Page Replacement]      PID #2: Victim:1        PTE:80000001
[After Swap]    PID #2: Victim:1        PTE:40000020
[Page mapping]  PID #2: Frame:1 PTE:80000001    PGN:1
>>>>>Done>>>>> #RGID: 5
----------------------------------------
Process 2 Allocated Region list
Region id 2 : Range = [0 - 200]
Region id 5 : Range = [256 - 456]
----------------------------------------
Process 2 Free Region list
Range = [200 - 256], freespace = 56
Range = [456 - 512], freespace = 56
----------------------------------------
VMA id 1 : start = 0, end = 512, sbrk = 512
----------- RAM mapping status -----------
Number of mapped frames:        2
Number of remaining frames:     0
----------------------------------------
-----------------LRU LIST-----------------
[0] -> [1]
```

```
----------------------------------------

----------------------------------------
Process 1 WRITE CALL
Process 1 write region=4 offset=10 value=121
[Page Replacement]      PID #1: Victim:0        PTE:80000000    Target:1
[After Swap]    PID #1: Victim:0        PTE:40000040    Target:1
print_pgtbl: 0 - 512
00000000: 40000000
00000004: 80000000

Print content of RAM (only print nonzero value)
---------------------------------
Address 0x0000000a: 121
---------------------------------
-----------------LRU LIST-----------------
[1] -> [0]
----------------------------------------

Time slot   4
        CPU 4: Processed  2 has finished
        CPU 4 stopped
        CPU 2: Put process  1 to run queue
        CPU 2: Dispatched process  1
----------------------------------------
Process 1 READ CALL
Process 1 read region=4 offset=10 value=121
print_pgtbl: 0 - 512
00000000: 40000000
00000004: 80000000

Print content of RAM (only print nonzero value)
---------------------------------
Address 0x0000000a: 121
---------------------------------
-----------------LRU LIST-----------------
[1] -> [0]
----------------------------------------

Time slot   5
----------------------------------------
Process 1 FREE CALL | Region id 0 after free: [0,0]
```

```
Region id 1 : Range = [200 - 256]
Region id 4 : Range = [256 - 356]
----------------------------------------
Process 1 Free Region list
Range = [356 - 512], freespace = 156
Range = [0 - 200], freespace = 200
----------------------------------------
Time slot   6
        CPU 2: Processed  1 has finished
        CPU 2 stopped
Time slot   7
        CPU 3 stopped
        CPU 1 stopped
        CPU 5 stopped
        CPU 0 stopped
```

4.3 Giải thích

Input: /input/ex_syn_mem

```
2 6 2
512 16777216 0 0 0
0 a0 1
1 m1 0
```

Lưu ý, các tiến trình nằm trong /input/proc Giải thích Ouput:

Ta sử dụng thuật toán thay thế trang LRU (Least Recently Used). Nó hiển thị các sự kiện xảy ra trong hệ thống theo từng khoảng thời gian (time slot).

Giải thích chi tiết:

Hệ thống khởi tạo với 2 khung trang (frames) trong bộ nhớ vật lý (RAM).

Tại time slot 0, tiến trình 1 (PID 1) được tải và yêu cầu cấp phát 200 đơn vị bộ nhớ. Vì không có vùng trống, hệ thống ánh xạ trang đầu tiên (frame 0) cho tiến trình này.

Tại time slot 1, tiến trình 1 yêu cầu cấp phát thêm 100 đơn vị bộ nhớ. Hệ thống ánh xạ trang thứ hai (frame 1) cho tiến trình này.

Tại time slot 2, tiến trình 2 (PID 2) được tải và yêu cầu cấp phát 200 đơn vị bộ nhớ. Vì không có khung trang trống, hệ thống thực hiện thay thế trang theo thuật toán LRU, đẩy trang đầu tiên của tiến trình 1 ra khỏi bộ nhớ vật lý (swap out) và ánh xạ trang đầu tiên (frame 0) cho tiến trình 2.

Tiếp theo, tiến trình 1 yêu cầu cấp phát thêm 56 đơn vị bộ nhớ và được cấp phát từ vùng trống còn lại.

Tại time slot 3, tiến trình 2 yêu cầu cấp phát thêm 200 đơn vị bộ nhớ. Hệ thống thực hiện thay thế trang theo thuật toán LRU, đẩy trang thứ hai của tiến trình 1 ra khỏi bộ nhớ vật lý (swap out) và ánh xạ trang thứ hai (frame 1) cho tiến trình 2.

Tiếp theo, tiến trình 1 thực hiện ghi dữ liệu vào vùng nhớ đã được cấp phát. Hệ thống phải đọc trang tương ứng từ bộ nhớ swap (swap in) trước khi ghi dữ liệu.

Tại time slot 4, tiến trình 1 thực hiện đọc dữ liệu từ vùng nhớ đã được ghi trước đó.

Tại time slot 5, tiến trình 1 giải phóng vùng nhớ đã cấp phát ban đầu (region id 0).

Cuối cùng, tất cả các tiến trình đã hoàn thành và hệ thống dừng hoạt động.

Đoạn output này minh họa quá trình quản lý bộ nhớ ảo, bao gồm cấp phát bộ nhớ, thay thế trang theo thuật toán LRU, đọc/ghi dữ liệu và giải phóng bộ nhớ. Nó cũng hiển thị trạng thái của bảng ánh xạ trang (page table), danh sách vùng nhớ đã cấp phát và vùng nhớ trống, cũng như danh sách LRU để theo dõi trật tự sử dụng các trang.

Cụ thể:

Time slot 0:

Hệ thống tải một tiến trình từ input/proc/a0, với PID (Process ID) là 1 và độ ưu tiên (PRIO) là 1. CPU 2 được giao xử lý tiến trình 1. Tiến trình 1 thực hiện yêu cầu cấp phát vùng nhớ có kích thước 200 đơn vị. Vì không có vùng trống sẵn có, hệ thống ánh xạ khung trang (frame) 0 cho tiến trình 1 và cập nhật mục nhập bảng trang (PTE) tương ứng. Tiến trình 1 được cấp phát vùng nhớ có ID 0 với phạm vi [0 - 200]. Vùng trống còn lại cho tiến trình 1 là [200 - 256] với 56 đơn vị trống. Vùng nhớ ảo (VMA) ID 1 bao trùm từ 0 đến 256. Trạng thái ánh xạ RAM cho thấy có 1 khung trang được ánh xạ và 1 khung trang còn lại. Danh sách LRU chỉ có khung trang 0.

Time slot 1:

Tiến trình 1 thực hiện yêu cầu cấp phát thêm vùng nhớ có kích thước 100 đơn vị. Vì không có vùng trống sẵn có, hệ thống ánh xạ khung trang 1 cho tiến trình 1 và cập nhật mục nhập bảng trang (PTE) tương ứng. Tiến trình 1 được cấp phát vùng nhớ có ID 4 với phạm vi [256 - 356]. Vùng trống còn lại cho tiến trình 1 bao gồm [200 - 256] với 56 đơn vị trống và [356 - 512] với 156 đơn vị trống. Vùng nhớ ảo (VMA) ID 1 bao trùm từ 0 đến 512. Trạng thái ánh xạ RAM cho thấy có 2 khung trang được ánh xạ và không còn khung trang trống. Danh sách LRU được cập nhật với khung trang 0 trước khung trang 1. Hệ thống tải một tiến trình khác từ input/proc/m1, với PID là 2 và độ ưu tiên (PRIO) là 0.

Time slot 2:

CPU 4 được giao xử lý tiến trình 2. CPU 2 đưa tiến trình 1 vào hàng đợi chạy và giao xử lý tiến trình 1. Tiến trình 2 thực hiện yêu cầu cấp phát vùng nhớ có kích thước 200 đơn vị. Vì không có vùng trống sẵn có, xảy ra thay thế trang theo thuật toán LRU. Khung trang 0 của tiến trình 1 bị đẩy ra khỏi bộ nhớ vật lý (swap out) và khung trang 0 được ánh xạ cho tiến trình 2. Tiến trình 2 được cấp phát vùng nhớ có ID 2 với phạm vi [0 - 200]. Vùng trống còn lại cho tiến trình 2 là [200 - 256] với 56 đơn vị trống. Vùng nhớ ảo (VMA) ID 1 bao trùm từ 0 đến 256. Trạng thái ánh xạ RAM cho thấy có 2 khung trang được ánh xạ và không còn khung trang trống. Danh sách LRU được cập nhật với khung trang 1 trước khung trang 0. Tiến trình 1 thực hiện yêu cầu cấp phát thêm vùng nhớ có kích thước 56 đơn vị. Hệ thống tìm thấy vùng trống [200 - 256] trong danh sách vùng trống và cấp phát cho tiến trình 1 với ID 1. Danh sách vùng nhớ đã cấp phát của tiến trình 1 bao gồm ID 0 ([0 - 200]), ID 1 ([200 - 256]) và ID 4 ([256 - 356]). Vùng trống còn lại cho tiến trình 1 là [356 - 512] với 156 đơn vị trống. Vùng nhớ ảo (VMA) ID 1 bao trùm từ 0 đến 512. Trạng thái ánh xạ RAM vẫn giữ nguyên với 2 khung trang được ánh xạ và không còn

khung trang trống. Danh sách LRU không thay đổi.

Time slot 3:

Tiến trình 2 thực hiện yêu cầu cấp phát thêm vùng nhớ có kích thước 200 đơn vị. Vì không có vùng trống sẵn có, xảy ra thay thế trang theo thuật toán LRU. Khung trang 1 của tiến trình 1 bị đẩy ra khỏi bộ nhớ vật lý (swap out) và khung trang 1 được ánh xạ cho tiến trình 2. Tiến trình 2 được cấp phát vùng nhớ có ID 5 với phạm vi [256 - 456]. Vùng trống còn lại cho tiến trình 2 bao gồm [200 - 256] với 56 đơn vị trống và [456 - 512] với 56 đơn vị trống. Vùng nhớ ảo (VMA) ID 1 bao trùm từ 0 đến 512. Trạng thái ánh xạ RAM vẫn giữ nguyên với 2 khung trang được ánh xạ và không còn khung trang trống. Danh sách LRU được cập nhật với khung trang 0 trước khung trang 1. Tiến trình 1 thực hiện yêu cầu ghi dữ liệu vào vùng nhớ có ID 4 tại offset 10 với giá trị 121. Vì khung trang tương ứng không có trong bộ nhớ vật lý, hệ thống phải đọc khung trang 0 của tiến trình 1 từ bộ nhớ swap (swap in) và thay thế khung trang 1 của tiến trình 2. Bảng ánh xạ trang (page table) được cập nhật để phản ánh thay đổi này. Nội dung của RAM tại địa chỉ 0x0000000a được ghi với giá trị 121. Danh sách LRU được cập nhật với khung trang 1 trước khung trang 0.

Time slot 4:

Tiến trình 2 đã hoàn thành và CPU 4 dừng hoạt động. CPU 2 đưa tiến trình 1 vào hàng đợi chạy và giao xử lý tiến trình 1. Tiến trình 1 thực hiện yêu cầu đọc dữ liệu từ vùng nhớ có ID 4 tại offset 10. Bảng ánh xạ trang (page table) và nội dung của RAM tại địa chỉ 0x0000000a được in ra. Danh sách LRU không thay đổi.

Time slot 5:

Tiến trình 1 thực hiện yêu cầu giải phóng vùng nhớ có ID 0. Danh sách vùng nhớ đã cấp phát của tiến trình 1 bao gồm ID 1 ([200 - 256]) và ID 4 ([256 - 356]). Vùng trống còn lại cho tiến trình 1 bao gồm [356 - 512] với 156 đơn vị trống và [0 - 200] với 200 đơn vị trống.

Time slot 6:

Tiến trình 1 đã hoàn thành và CPU 2 dừng hoạt động.

Time slot 7:

Tất cả các CPU còn lại (CPU 3, CPU 1, CPU 5 và CPU 0) dừng hoạt động, cho thấy hệ thống đã hoàn thành tất cả các tác vụ. Đoạn output này mô phỏng hoạt động của hệ thống quản lý bộ nhớ ảo, bao gồm cấp phát vùng nhớ, thay thế trang theo thuật toán LRU, đọc/ghi dữ liệu và giải phóng vùng nhớ. Nó cũng hiển thị trạng thái của bảng ánh xạ trang, danh sách vùng nhớ đã cấp phát và vùng nhớ trống, cũng như danh sách LRU để theo dõi trật tự sử dụng các trang.

5 Source code

Source code: https://github.com/buithekys/os_assignmentHK232

Tài liệu tham khảo

[1] Slide môn học: CO2017 - Operating Systems

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

[3] Abhishek Bhattacharjee and Margaret Martonosi. *Characterizing the TLB Behavior of Emerging Parallel Workloads on Chip Multiprocessor*. Department of Electrical Engineering, Princeton University, 2011.