ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN

MÔN HỌC: Hệ Điều Hành

Đề tài: Simple Operating System

Nhóm 12 - L08 - HK232

Giảng viên hướng dẫn lý thuyết: Lê Thanh Vân
Giảng viên hướng dẫn thí nghiệm: Nguyễn Đặng Anh Khoa
Sinh viên: Hồ Quốc Khương - 2211709
Phạm Gia Trí - 2213656
Phạm Duy Hưng - 2211379
Trần Đại Việt - 2213951
Nguyễn Lê Gia Kiệt - 2211761

HO CHI MINH CITY, MAY 2024

Danh sách sinh viên và phân công công việc

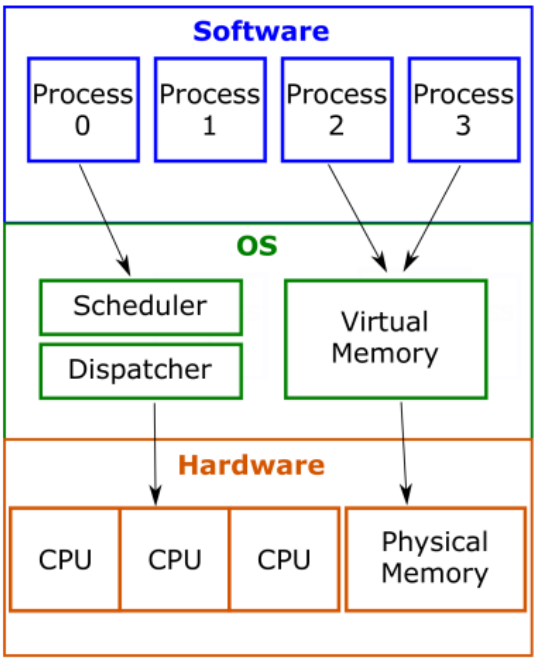
STT	Họ và tên	MSSV	Công việc	Đóng góp
1	Hồ Quốc Khương	2211709	- Memory Management	100%
2	Phạm Gia Trí	2213656	- Scheduler	100%
3	Phạm Duy Hưng	2211379	- TLB	100%
4	Nguyễn Lê Gia Kiệt	2211761	- TLB	100%
5	Trần Đại Việt	2213951	- Memory Management	100%

Contents

1 Giới thiệu đề tài

Mục tiêu của đề tài bài tập lớn này là mô phỏng một hệ điều hành đơn giản nhằm giúp chúng ta hiểu rõ các khái niệm cơ bản về lập lịch (scheduling), đồng bộ (synchronization), và quản lý bộ nhớ (memory management). Hình 1 minh họa kiến trúc tổng quát của hệ điều hành mà chúng ta sẽ triển khai. Hệ điều hành này cần quản lý hai tài nguyên ảo chính: CPU và RAM, thông qua hai thành phần cốt lõi:

- **Bộ lập lịch và bộ điều phối (Scheduler và Dispatcher)**: Xác định quá trình nào (process) được phép chạy trên CPU nào.
- **Bộ máy bộ nhớ ảo (Virtual Memory Engine - VME)**: Cách ly không gian bộ nhớ của từng quá trình với nhau. RAM vật lý được chia sẻ bởi nhiều quá trình nhưng mỗi quá trình không biết đến sự tồn tại của các quá trình khác. Điều này được thực hiện bằng cách cho mỗi quá trình có không gian bộ nhớ ảo riêng và bộ máy bộ nhớ ảo sẽ ánh xạ và dịch các địa chỉ ảo do các quá trình cung cấp thành các địa chỉ vật lý tương ứng.



Hình 1: Cái nhìn tổng quan về các mô-đun chính trong bài tập này

Thông qua các mô-đun này, hệ điều hành cho phép nhiều quá trình do người dùng tạo ra chia sẻ và sử dụng các tài nguyên tính toán ảo. Do đó, trong bài tập này, chúng ta tập trung vào việc triển khai bộ lập lịch/điều phối và bộ máy bộ nhớ ảo.

2 Cơ sở lý thuyết

2.1 Scheduler

Thuật toán lập lịch là một khái niệm quan trọng trong hệ điều hành, được sử dụng để xác định thứ tự các tiến trình sẽ được CPU thực thi. Trong báo cáo sẽ nói về thuật toán lập lịch hàng đợi đa cấp (multi level queue)
Thuật toán lập lịch hàng đợi đa cấp (multi level queue) chia hàng đợi Ready thành nhiều cấp hoặc tầng, mỗi cấp có mức độ ưu tiên khác nhau. Sau đó, mức độ thích hợp sẽ được chỉ định cho các quy trình dựa trên đặc điểm của chúng, chẳng hạn như mức độ ưu tiên, yêu cầu bộ nhớ và mức sử dụng CPU.

2.2 Memory Management Unit

Bộ nhớ vật lý (Physical Memory) là một tài nguyên hữu hạn nên ta cần bộ nhớ ảo (Virtual Memory) để có thể dễ dàng tăng được lượng tác vụ có thể xử lý đồng thời tăng hiệu suất của hệ điều hành thông qua Virtual Memory Engine (VME).

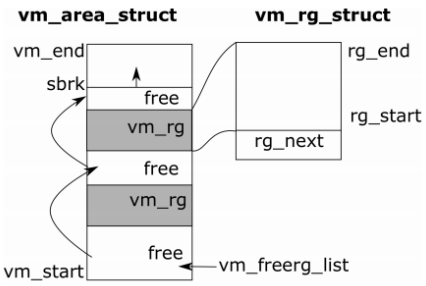
Virtual Memory Engine (VME) giúp cách ly không gian nhớ của từng process. RAM vật lý được chia sẻ bởi nhiều process nhưng mỗi process không biết đến sự tồn tại của nhau. Điều này được thực hiện bằng cách để cho từng process có riêng một không gian bộ nhớ ảo và VME sẽ ánh xạ lại và giải mã các địa chỉ ảo của các process với các địa chỉ vật lý tương ứng

Để hiện thực được cơ chế trên, ta cần nắm vững kiến thức về bộ nhớ ảo và bộ nhớ vật lý cũng như là cơ chế ánh xạ giữa 2 bộ nhớ đó qua phương pháp phân trang (paging).

2.2.1 Virtual Memory

Khi một process được thực thi bởi CPU, process đó sẽ được cấp phát một vùng nhớ trên bộ nhớ vật lý. Theo cơ chế phân trang(paging), bộ nhớ vật lý được chia thành các frame bằng nhau và bộ nhớ luận lý được chia thành các page bằng nhau ánh xạ với nhau để cung cấp vùng nhớ cho process (page size = frame size).

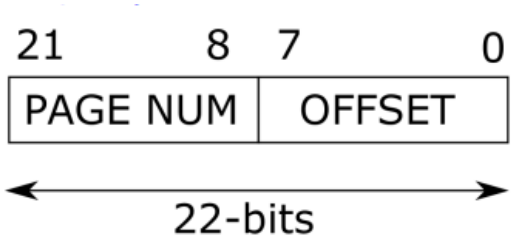
Trong mỗi process sẽ có một con trỏ trỏ đến địa chỉ của memory map chứa các virtual memory area. Các khu vực này được sắp xếp liên tục trong [vm_start, vm_end]. Tuy nhiên, khu vực thực chất có thể sử dụng được lại bị giới hạn bởi điểm đầu tại sbrk. Các virtual memory area sẽ được chia thành các trang với kích thước bằng với kích thước của 1 frame trong RAM và được ánh xạ tới các frame trong RAM. Trong mỗi virtual memory area chứa các virtual memory region là các vùng nhớ ảo gồm region đang được sử dụng và region đang trống (có thể không liền kề với nhau và có kích thước khác nhau).



Hình 2: Cấu trúc virtual memory area và virtual memory region.

Trong không gian bộ nhớ ảo, địa chỉ được chia thành hai phần :

- Page number : là chỉ số trang, dùng để ánh xạ đến frame tương ứng trong bộ nhớ vật lý.
- Page offset : là vị trí của địa chỉ trong trang, khi kết hợp với Page number sẽ tạo ra địa chỉ bộ nhớ vật lý.



Hình 3: Cấu trúc địa chỉ CPU.

Trong bài tập lớn, ta dùng 22-bit CPU để biểu diễn địa chỉ CPU, trong đó ta dùng 14-bit để biểu diễn page number và 8-bit để biểu diễn offset, với kích thước của 1 page là 256 byte và kích thước của 1 page entry là 4 byte.

2.2.2 Memory Physical

Bộ nhớ vật lý được chia thành các khối nhỏ có kích thước cố định được gọi là frame. Không gian địa chỉ bộ nhớ vật lý gồm 2 phần:

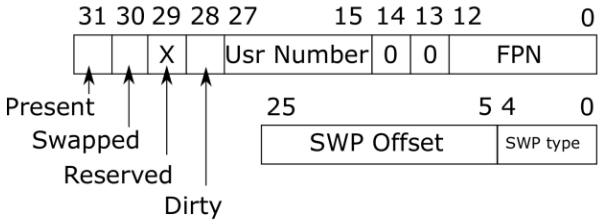
- RAM: Bộ nhớ chính, có thể được truy xuất trực tiếp bởi CPU.
- SWAP: Bộ nhớ thứ cấp, không được truy xuất trực tiếp bởi CPU, các process muốn được thực thi cần phải được load lên RAM, kích thước lớn hơn RAM và số lượng nhiều (trong bài tập lớn dùng 4 SWAP). SWAP thường được dùng khi RAM không đủ để chứa tất cả dữ liệu cần thiết cho các process đang sử dụng.

Trong thực tế, bộ nhớ RAM là hữu hạn nên không đủ lớn cho mọi process. Do đó, cơ chế swapping được sử dụng để chuyển frame giữa RAM và SWAP, giúp cho hệ thống thực thi các chương trình có kích thước lớn hơn RAM. Cơ chế swapping in di chuyển nội dung của frame từ SWAP (MEMSWP) vào các frame của RAM (MEMRAM) khi cần truy cập nội dung để process thực thi tương ứng. Ngược lại, cơ chế swapping out sẽ di chuyển nội dung từ MEMRAM sang MEMSWP, để có được MEMRAM trống do kích thước của SWAP thường đủ lớn.

Cơ chế này hỗ trợ được lượng process lớn hơn so với dung lượng RAM cho phép, bằng cách dùng SWAP để lưu trữ dữ liệu khi không còn đủ không gian trong RAM. Tuy nhiên, cơ chế này có thể làm giảm hiệu suất do tốc độ truy cập vào SWAP thường chậm hơn so với RAM.

2.2.3 Phương pháp phân trang Paging-based

Mỗi process đều có một bảng phân trang Page Table, cấu trúc page table cho phép userspace process tìm ra physical frame mà mỗi trang ảo đều có và được ánh xạ tới.



Hình 4: Page Table Entry Format.

Mỗi bảng phân trang chứa các Page Table Entries (PTEs), mỗi PTE thường bao gồm một giá trị 32-bit cho mỗi trang ảo, chứa các thông tin liên quan đến việc ánh xạ giữa trang ảo và địa chỉ vật lý trong bộ nhớ. Cấu trúc của PTE tùy thuộc vào kiến trúc của hệ điều hành và cách thức triển khai.

```
 * Bits 0-12  page frame number (FPN) if present
 * Bits 13-14 zero if present
 * Bits 15-27 user-defined numbering if present
 * Bits 0-4   swap type if swapped
 * Bits 5-25  swap offset if swapped
 * Bit  28    dirty
 * Bits 29    reserved
 * Bit  30    swapped
 * Bit  31    presented
```

2.3 TLB

Bộ đệm tra cứu dịch thuật (Translation Lookaside Buffer - TLB) là một thành phần quan trọng trong các hệ điều hành hiện đại, đóng vai trò như bộ nhớ đệm giúp tăng tốc quá trình dịch địa chỉ ảo sang địa chỉ vật lý. Nó làm giảm độ trễ truy cập bộ nhớ bằng cách lưu trữ ánh xạ địa chỉ được sử dụng gần đây và cho phép tra cứu nhanh chóng. Khi một chương trình yêu cầu quyền truy cập bộ nhớ, TLB sẽ được tư vấn trước tiên, dẫn đến việc truy xuất dữ liệu nhanh hơn nếu xảy ra kết quả trùng khớp (lần truy cập TLB).

Ưu điểm và Nhược điểm của TLB

- Ưu điểm của TLB
  1. Truy cập bộ nhớ nhanh hơn
  2. Giảm độ trễ
  3. Sử dụng hiệu quả bộ nhớ ảo
  4. Tiêu thụ điện năng thấp hơn
  5. Truy cập bảng trang được thu nhỏ
  6. Cải thiện khả năng đáp ứng của hệ thống
- Nhược điểm của TLB
  1. Kích thước giới hạn, Chi phí quản lý, Phần cứng phức tạp
  2. TLB bỏ lỡ hình phạt

3 Schedule

3.1 Trình bày giải thuật

1. Kiểm tra tất cả hàng đợi xem có rỗng không. Nếu rỗng, trả về một `NULL` process.
2. Sau khi đã xác định có ít nhất một hàng đợi không rỗng, tiến hành lấy một process theo cách sau:
   - Thuật toán duyệt qua tất cả các hàng đợi từ độ ưu tiên cao đến độ ưu tiên thấp.
   - Nếu phát hiện có một hàng đợi i không rỗng và số `slot[i]` > 0, lập tức lấy process trong hàng đợi và `break`. Để chương trình con trả về process đó. (Mục đích là lấy độ ưu tiên cao nhất và còn sử dụng được)
   - Nếu ($i ==$ `MAX_PRIO` $- 1$) lúc này đã duyệt hết vẫn không tìm thấy process phù hợp:
     - Kiểm tra xem hàng đợi có rỗng không. Nếu rỗng thì trả về một `NULL` process.
     - Cấp lại một lượt mới cho các hàng đợi (`slot[i]` $=$ `MAX_PRIO` $- i$). Và chạy vòng lặp lại từ đầu để tìm process phù hợp.

3.2 Trả lời câu hỏi

Question: What is the advantage of the scheduling strategy used in this assignment in comparison with other scheduling algorithms you have learned?
Answer Giải thuật định thời trong bài tập lớn trên tỏ ra có ưu điểm hơn so với các giải thuật định thời khác như:

- **Chi phí lập lịch thấp:** Vì các quy trình được gán vĩnh viễn cho các hàng đợi tương ứng của chúng nên chi phí lập lịch thấp, vì người lập lịch chỉ cần chọn hàng đợi thích hợp để thực thi.
- **Phân bổ thời gian CPU hiệu quả:** Đảm bảo rằng các quy trình có mức ưu tiên cao hơn được thực thi kịp thời, trong khi vẫn cho phép các quy trình có mức ưu tiên thấp hơn thực thi khi CPU không hoạt động. Điều này đảm bảo sử dụng tối ưu thời gian CPU.
- **Tính công bằng:** Thuật toán lập lịch cung cấp sự phân bổ hợp lý thời gian CPU cho các loại quy trình khác nhau, dựa trên mức độ ưu tiên và yêu cầu của chúng. Điều này hạn chế được tình trạng *starvation* giữa các hàng đợi có độ ưu tiên khác nhau và đảm bảo phân bổ thời gian CPU công bằng.
- **Tính linh hoạt:** Thuật toán lập lịch có thể được tùy chỉnh để đáp ứng các yêu cầu cụ thể của các loại quy trình khác nhau. Các thuật toán lập lịch khác nhau có thể được sử dụng cho mỗi hàng đợi, tùy thuộc vào yêu cầu của các tiến trình trong hàng đợi đó.
- **Ưu tiên:** Mức độ ưu tiên được chỉ định cho các quy trình dựa trên loại, đặc điểm và tầm quan trọng của chúng, điều này đảm bảo rằng các quy trình quan trọng được thực hiện kịp thời.
- **Quyền ưu tiên (Preemption):** Quyền ưu tiên được cho phép trong Lập lịch hàng đợi đa cấp, có nghĩa là các quy trình có mức độ ưu tiên cao hơn có thể giành quyền ưu tiên các quy trình có mức độ ưu tiên thấp hơn và CPU được phân bổ cho quy trình có mức độ ưu tiên cao hơn. Điều này giúp đảm bảo rằng các quy trình có mức độ ưu tiên cao được thực hiện kịp thời.

3.3 Giải thích kết quả

3.3.1 Bài toán 1 CPU

Input

Output

```
viet@viet-VirtualBox:~/Desktop/Group12-L08$ ./os test2
Time slot   0
ld_routine
        Loaded a process at input/proc/m0s, PID: 1 PRIO: 138
Time slot   1
        CPU 0: Dispatched process  1
        Loaded a process at input/proc/s3, PID: 2 PRIO: 139
Time slot   2
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
        Loaded a process at input/proc/p3s, PID: 3 PRIO: 137
Time slot   3
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  3
Time slot   4
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  3
Time slot   5
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  3
Time slot   6
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  2
Time slot   7
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot   8
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  3
Time slot   9
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  3
Time slot  10
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  1
Time slot  11
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  12
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot  13
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
```

```
Time slot  14
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  3
Time slot  15
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  3
Time slot  16
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  1
TLB miss at write region=1 offset=20 value=102
Dumping Memory Content (up to 1048576 bytes ):
Byte with content not found.
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Free_entries : 255
Time slot  17
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
TLB miss at write region=2 offset=1000 value=1
Dumping Memory Content (up to 1048576 bytes ):
BYTE 000000a4: 102
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Free_entries : 254
Time slot  18
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot  19
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  20
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  3
Time slot  21
        CPU 0: Processed  3 has finished
        CPU 0: Dispatched process  1
TLB miss at write region=0 offset=0 value=0
Dumping Memory Content (up to 1048576 bytes ):
BYTE 000000b0: 1
print_pgtbl: 0 - 512
00000000: c0000000
00000004: 80000000
Free entries : 253
```

```
Time slot   22
        CPU 0: Processed  1 has finished
        CPU 0: Dispatched process  2
Time slot   23
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   24
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   25
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   26
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   27
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   28
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   29
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   30
        CPU 0: Processed  2 has finished
        CPU 0 stopped
HIT = 0
MISS = 3
HIT/MISS= 0.000000
```

Sơ đồ Gantt của kết quả

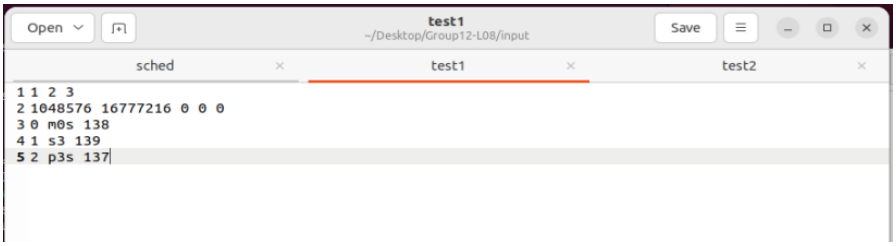
Giải thích kết quả

Ở bài toán này, nhóm chọn `MAX_PRIO` là 140 và `time slice` = 1 nên số slot của mỗi P1, P2, P3 lần lượt là 2, 1, 3 và mỗi lần thực hiện mất 1 đơn vị thời gian.

- Ở `time slot 0`, P1 được nạp vào hàng đợi và tại `time slot 1` P1 được nạp vào CPU.
- Sau đó tại `time slot 3`, P3 được nạp vào do P3 có độ ưu tiên cao hơn nên được ưu tiên thực hiện trước.
- Do đã thực hiện hết tài nguyên ở P1 và P3 nên P2 được chọn vào CPU ở `time slot` thứ 6.
- Sau khi thực hiện xong P2, bắt đầu một chu kỳ mới và thực hiện theo thứ tự P3, P1, P2 theo thứ tự độ ưu tiên với tỷ lệ chiếm tài nguyên là 3, 2, 1.
- Tại thời điểm 20, P3 hết tài nguyên nên được chuyển tiếp cho P1.
- Tại thời điểm 21, P1 hết tài nguyên nên được chuyển tiếp cho P2.
- CPU thực hiện P2 đến khi kết thúc chương trình.

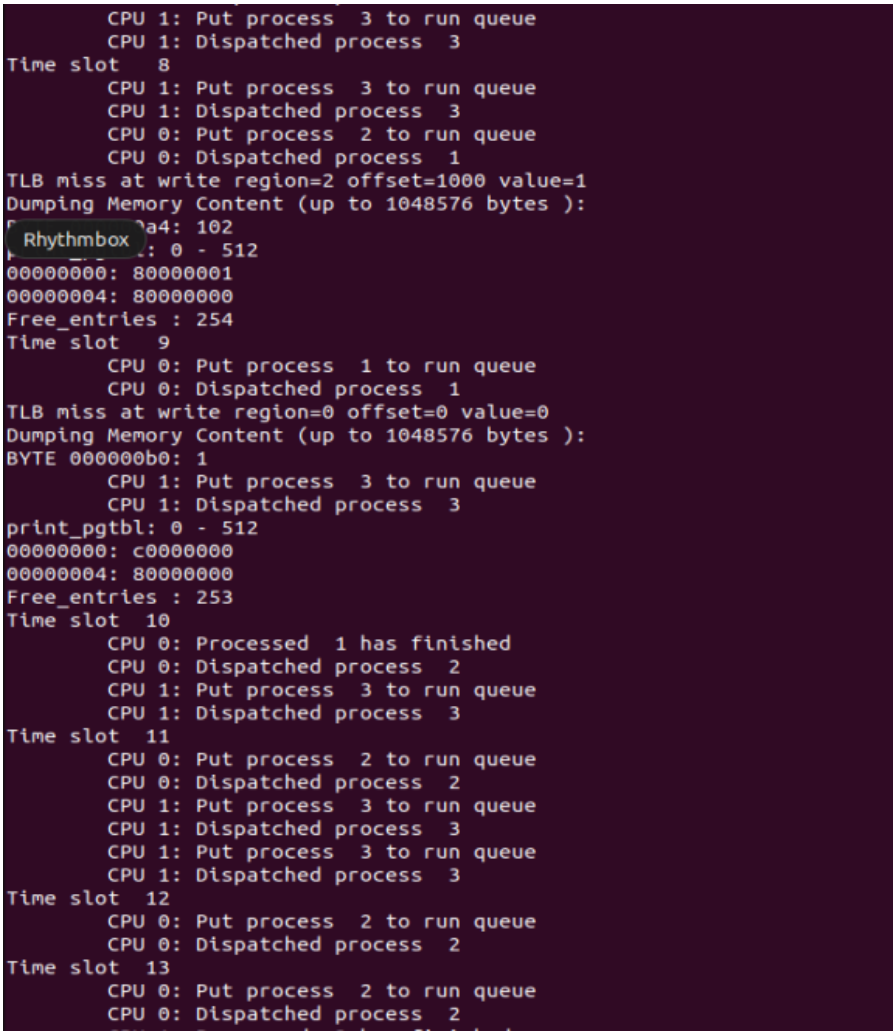
3.3.2 Bài toán 2 CPU

Input



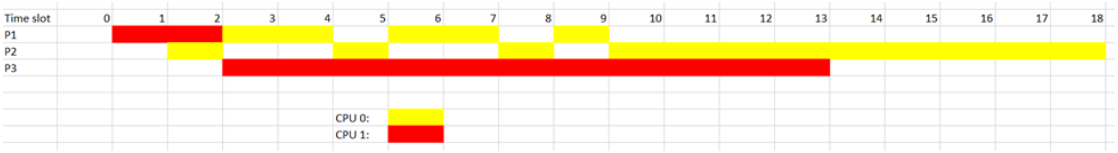
Output

```
viet@viet-VirtualBox:~/Desktop/Group12-L08$ ./os test1
Time slot   0
ld_routine
        Loaded a process at input/proc/m0s, PID: 1 PRIO: 138
Time slot   1
        Loaded a process at input/proc/s3, PID: 2 PRIO: 139
        CPU 1: Dispatched process  1
        CPU 0: Dispatched process  2
Time slot   2
        CPU 0: Put process  2 to run queue
        Loaded a process at input/proc/p3s, PID: 3 PRIO: 137
        CPU 0: Dispatched process  2
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  3
Time slot   3
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  1
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
Time slot   4
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
Time slot   5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
Time slot   6
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
TLB miss at write region=1 offset=20 value=102
Dumping Memory Content (up to 1048576 bytes ):
Byte with content not found.
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Free_entries : 255
Time slot   7
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
```


CPU 1: Put process 3 to run queue
CPU 1: Dispatched process 3
Time slot 8
CPU 1: Put process 3 to run queue
CPU 1: Dispatched process 3
CPU 0: Put process 2 to run queue
CPU 0: Dispatched process 1
TLB miss at write region=2 offset=1000 value=1
Dumping Memory Content (up to 1048576 bytes):
Rhythmbox
: 0 - 512
00000000: 80000001
00000004: 80000000
Free_entries : 254
Time slot 9
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 1
TLB miss at write region=0 offset=0 value=0
Dumping Memory Content (up to 1048576 bytes):
BYTE 000000b0: 1
CPU 1: Put process 3 to run queue
CPU 1: Dispatched process 3
print_pgtbl: 0 - 512
00000000: c0000000
00000004: 80000000
Free_entries : 253
Time slot 10
CPU 0: Processed 1 has finished
CPU 0: Dispatched process 2
CPU 1: Put process 3 to run queue
CPU 1: Dispatched process 3
Time slot 11
CPU 0: Put process 2 to run queue
CPU 0: Dispatched process 2
CPU 1: Put process 3 to run queue
CPU 1: Dispatched process 3
CPU 1: Put process 3 to run queue
CPU 1: Dispatched process 3
Time slot 12
CPU 0: Put process 2 to run queue
CPU 0: Dispatched process 2
Time slot 13
CPU 0: Put process 2 to run queue
CPU 0: Dispatched process 2

```
        CPU 0: Dispatched process  2
        CPU 1: Processed  3 has finished
        CPU 1 stopped
Time slot  14
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  15
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  16
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  17
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  18
        CPU 0: Processed  2 has finished
        CPU 0 stopped
HIT = 0
MISS = 3
HIT/MISS= 0.000000
```

Sơ đồ Gantt của kết quả



Giải thích kết quả

Ở bài toán này, nhóm chọn `MAX_PRIO` là 140 và `time slice` = 1 nên số slot của mỗi P1, P2, P3 lần lượt là 2, 1, 3 và mỗi lần thực hiện mất 1 đơn vị thời gian.

- Ở `time slot 0`, P1 được nạp vào hàng đợi và tại `time slot 1` P1 được nạp vào CPU 1.
- Ở `time slot 2`, P2 được nạp vào CPU0 còn trống. CPU tiếp tục thực hiện P1.
- Ở `time slot 3`, lúc này đã là một vòng mới. P3 và P1 được nạp vào CPU1 và CPU0 do P3 và P1 có độ ưu tiên cao.
- Tại `time slot 5`, P1 đã thực hiện hết 2 lượt của mình chuyển lượt cho P2. Sau khi kết thúc `time slot 5`, P1 đã thực hiện 2 lượt, P2 1 lượt và P3 3 lượt. Lúc này đã chuyển vào vòng mới. Nên `time slot 6` sẽ giống với trạng thái ở `time slot 3`.
- Ở `time slot 9`, P1 đã hoàn tất. Nên 2 CPU cùng thực hiện P2 và P3 đến khi kết thúc chương trình.

4 Memory Management Unit

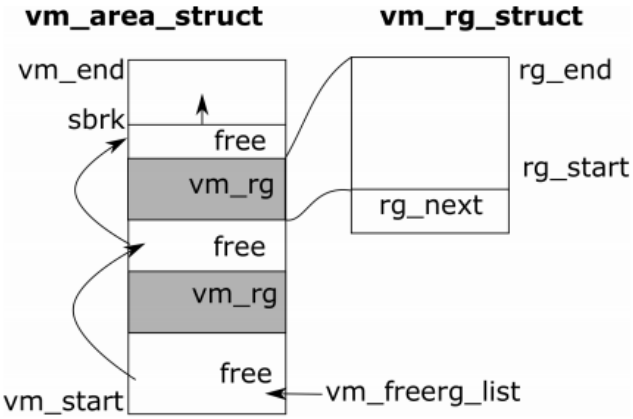
4.1 Trình bày giải thuật

Giải thuật của các thao tác (operation) trên virtual memory bao gồm:

- ALLOC: cấp phát 1 region memory.

  Giải thuật:

  - Tìm vùng nhớ trống có kích thước theo yêu cầu của process, nếu tìm thấy thì cấp phát cho process. Nếu không tìm thấy vùng nhớ trống phù hợp, nó sẽ tăng giá trị sbrk để tăng giới hạn vùng nhớ và cấp phát. Nếu tồn tại vùng nhớ thừa sau khi cấp phát, tạo cấu trúc vùng trống cho vùng nhớ thừa đó và thêm vào danh sách các vùng nhớ trống.
  - Hàm sử dụng khóa mutex để ngăn các threads khác truy cập khi process đang thực thi.



Hình 5: Minh họa cho thao tác alloc.

- FREE: giải phóng 1 region memory.

  Giải thuật:

  - Đầu tiên, hàm kiểm tra tính hợp lệ của region id (rgid), lấy thông tin và kiểm tra vùng nhớ cần giải phóng. Sau đó, tạo một region node mới để lưu trữ thông tin về vùng nhớ sắp giải phóng. Thiết lập giá trị start và end của region node mới bằng 0 để để đánh dấu vùng nhớ này đã được giải phóng. Cuối cùng, thêm region node vào danh sách các vùng nhớ trống.
  - Hàm sử dụng khóa mutex để ngăn các threads khác truy cập khi process đang thực thi.

- READ: đọc 1 byte giá trị tại 1 ô nhớ trong region memory.

  Giải thuật:

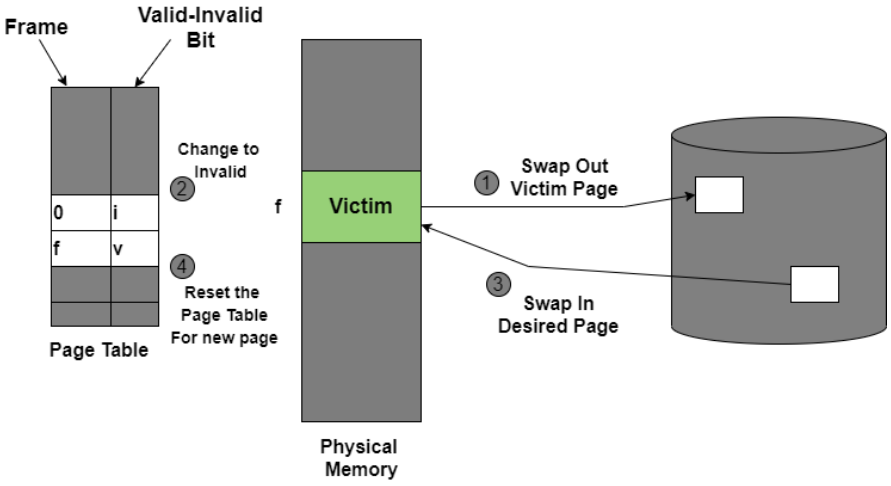
  - Lấy thông tin vùng nhớ cần đọc và kiểm tra tính hợp lệ của vùng nhớ. Đọc giá trị từ bộ nhớ thông qua địa chỉ được tính toán dựa trên địa chỉ bắt đầu của vùng nhớ và offset.
  - Có thêm khóa mutex để ngăn các threads khác truy cập khi process đang thực thi.

- WRITE: ghi 1 byte giá trị vào 1 ô nhớ trong region memory.

  Giải thuật:

  - Lấy thông tin vùng nhớ cần ghi và kiểm tra tính hợp lệ của vùng nhớ. Tính toán địa chỉ cần ghi dựa trên địa chỉ bắt đầu của vùng nhớ và offset, sau đó ghi giá trị vào địa chỉ đã tính toán.
  - Có thêm khóa mutex để ngăn các threads khác truy cập khi process đang thực thi.

Giải thuật phân trang: (được hiện thực trong hàm pg_getpage trong file mm-vm.c) được sử dụng để lấy thông tin về trang từ bộ nhớ vật lý (RAM) thông qua bảng phân trang của một tiến trình cụ thể. Đầu tiên, truy cập vào mục nhập ứng với trang mong muốn trong bảng phân trang để xem trang này có được load vào bộ nhớ vật lý chưa. Nếu chưa, chúng ta tiến hành tìm một victim_page trong bộ nhớ vật lý theo cơ chế giải thuật thay trang FIFO và tiến hành swap với trang đang cần load vào bộ nhớ vật lý để tiến hành thực thi (trang này nằm trong bộ nhớ swap). Thiết lập frame number và page number mới cho entry của bảng trang, đồng thời đưa page này vào danh sách quản lý thay trang.



Hình 6: Minh họa cho quá trình thay trang.

Ngoài các giải thuật chính, còn có một số hàm hiện thực để hỗ trợ các giải thuật :

- Hàm find_victim_page được dùng để tìm trang thay thế bằng thuật toán FIFO và giải phóng vùng nhớ mà trang đó giữ trong bộ nhớ ảo để có thể tiến hành thay trang ở các bước tiếp theo.
- Hàm validate_overlap_vm_area được dùng để kiểm tra xem khu vực vùng nhớ mới mở rộng có xung đột không gian bộ nhớ với các khu vực vùng nhớ khác trong tiến trình không.
- Hàm MEMPHY_dump dùng vòng lặp for để in ra nội của cấu trúc bộ nhớ vật lý ra màn hình. Nếu tìm thấy block có giá trị khác 0, hàm sẽ in địa chỉ của block đó dưới định dạng của số thập lục phân và giá trị của block đó dưới dạng số thập phân. Nếu không tìm thấy thì hàm in ra lời nhắn không tìm thấy.

4.2 Trả lời câu hỏi

Câu hỏi 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?

Trả lời: Trong hệ điều hành này, việc thiết kế nhiều memory segments hoặc memory areas mang lợi nhiều lợi ích như:

- Giảm thiểu công việc của hệ thống quản lý bộ nhớ: Mỗi segments, vùng nhớ đều có riêng 1 hệ thống cấp phát và hủy trong chính segment đó, giúp cho hệ thống quản lý bộ nhớ không cần phải can thiệp quá nhiều vào công việc của từng segment và vùng nhớ đó.
- Việc mỗi segment, vùng nhớ có các thuộc tính và quyền truy cập riêng sẽ ngăn chặn truy cập không hợp pháp hoặc thay đổi dữ liệu.
- Tối ưu hóa hiệu suất nhờ việc chia nhỏ không gian bộ nhớ thành nhiều phân đoạn hoặc khu vực nhỏ giúp giảm thiểu lãng phí bộ nhớ và tăng hiệu suất của hệ thống.

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

Trả lời: Với các hệ thống quản lý bộ nhớ phân trang từ 2 cấp trở lên (còn gọi là phân trang đa cấp), các mục (entry) của bảng trang cấp cao hơn sẽ ánh xạ đến các mục của bảng trang cấp thấp hơn, riêng các mục của bảng trang cấp cuối cùng sẽ ánh xạ đến vùng nhớ tương ứng trong bộ nhớ thực. Khi đó, phân trang đa cấp sẽ mang lại những ưu điểm và nhược điểm sau:

Ưu điểm:

- **Giảm chi phí bộ nhớ**: Phân trang đa cấp có thể giúp giảm chi phí bộ nhớ liên quan đến bảng trang. Điều này là do mỗi cấp độ chứa ít mục nhập hơn, có nghĩa là cần ít bộ nhớ hơn để lưu trữ bảng trang.
- **Tra cứu bảng trang nhanh hơn**: Với số lượng mục nhập trên mỗi cấp độ nhỏ hơn, việc thực hiện tra cứu bảng trang sẽ mất ít thời gian hơn. Điều này có thể dẫn đến hiệu suất hệ thống tổng thể nhanh hơn.
- **Quản lý linh hoạt bộ nhớ**: Phân trang đa cấp mang lại sự 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 bảng trang được điều chỉnh để đáp ứng nhu cầu thay đổi.

Nhược điểm:

- **Độ phức tạp tăng lên**: Phân trang đa cấp làm 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 việc tra cứu bảng trang. Điều này là do mỗi cấp độ phải được duyệt qua để tìm mục nhập bảng trang mong muốn.
- **Xảy ra 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 tốn thêm chi phí khi truy cập bộ nhớ.

Câu hỏi 3: What is the advantage and disadvantage of segmentation with paging ?
Trả lời: Ưu điểm và nhược điểm của phân đoạn theo phân trang là:
Ưu điểm:

- **Tiết kiệm không gian bộ nhớ**: Page table chỉ dùng thông tin các page trong các phân đoạn riêng biệt và kích thước của page table có thể được xác định dựa vào kích thước của các phân đoạn. Điều này giúp kích thước của page table giảm đi và tiết kiệm không gian bộ nhớ so với single-level paging.
- **Tránh phân mảnh ngoài (external fragmentation)**: Việc kết hợp giữa phân đoạn với phân trang giúp duy trì được tính liên tục của không gian bộ nhớ, tránh được hiện tượng phân mảnh ngoài.
- **Quản lý bộ nhớ hiệu quả**: Cho phép kích thước các phân đoạn thay đổi và có thể chuyển đổi trang giữa bộ nhớ chính và thứ cấp để việc quản lý bộ nhớ tốt hơn, bảo mật hơn.

Nhược điểm:

- **Tạo ra phân mảnh nội (internal fragmentation)**: Page có kích thước cố định nên có thể gây ra phân mảnh nội trong các phân đoạn.
- **Độ phức tạp cao hơn**: Cần xác định kích thước của phân đoạn, số lượng page cần cấp phát và phải quản lý cả phân đoạn, các page trong từng phân đoạn, đòi hỏi thuật toán và cấu trúc dữ liệu phức tạp hơn.
- **Tăng thời gian truy cập bộ nhớ**: Khi số lượng các phân đoạn tăng lên dẫn đến yêu cầu lưu trữ bảng phân trang và truy cập đến các thành phần trong page table diễn ra liên tục.

4.3 Giải thích kết quả

Input: os_0_mlq_paging

```
viet@viet-VirtualBox:~/Desktop/Group12-L08$ ./os os_0_mlq_paging
Time slot   0
ld_routine
        Loaded a process at input/proc/p0s, PID: 1 PRIO: 0
Time slot   1
        CPU 1: Dispatched process  1
Time slot   2
        Loaded a process at input/proc/p1s, PID: 2 PRIO: 15
Time slot   3
        CPU 0: Dispatched process  2
        Loaded a process at input/proc/p1s, PID: 3 PRIO: 0
Time slot   4
        Loaded a process at input/proc/p1s, PID: 4 PRIO: 0
Time slot   5
Time slot   6
TLB miss at write region=1 offset=20 value=100
Dumping Memory Content (up to 1048576 bytes ):
Byte with content not found.
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Free_entries : 255
Time slot   7
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  3
Time slot   8
Time slot   9
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  4
Time slot  10
Time slot  11
Time slot  12
Time slot  13
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  1
TLB hit at read region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Time slot  14
TLB miss at write region=3 offset=20 value=103
```

```
Dumping Memory Content (up to 1048576 bytes ):
BYTE 000000a4: 100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Free_entries : 254
Time slot  15
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  3
TLB hit at read region=3 offset=20 value=103
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Time slot  16
Time slot  17
        CPU 1: Processed  1 has finished
        CPU 1: Dispatched process  4
Time slot  18
Time slot  19
        CPU 0: Processed  3 has finished
        CPU 0: Dispatched process  2
Time slot  20
Time slot  21
        CPU 1: Processed  4 has finished
        CPU 1 stopped
Time slot  22
Time slot  23
        CPU 0: Processed  2 has finished
        CPU 0 stopped
HIT = 2
MISS = 2
HIT/MISS= 1.000000
viet@viet-VirtualBox:~/Desktop/Group12-L08$
```

Giải thích kết quả os_0_mlq_paging:

- Khi có thao tác truy cập vào vùng nhớ vật lý (ở đây là write) thì hàm MEMPHY_dump sẽ in ra nội dung trong của cấu trúc vùng nhớ vật lý trước khi có thao tác đó ra màn hình. Ở đây do trước đó chưa có thao tác write nào vào vùng nhớ nên chưa có nội dung nào được ghi. Do vậy hàm in ra : Byte with content not found. Sau đó in ra nội dung bảng trang của bộ nhớ ảo ra màn hình với cột bên trái là địa chỉ page number(được tính bằng byte) và cột bên phải là giá trị các frame tương ứng. Có thể ghi lại bảng trang như sau để dễ quan sát hơn:

  Page Number: 0 -> Frame Number: 1

  Page Number: 1 -> Frame Number: 0

  Page Number: 2 -> Frame Number: 3

  Page Number: 3 -> Frame Number: 2

```
TLB miss at write region=1 offset=20 value=100
Dumping Memory Content (up to 1048576 bytes ):
Byte with content not found.
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Free_entries : 255
```

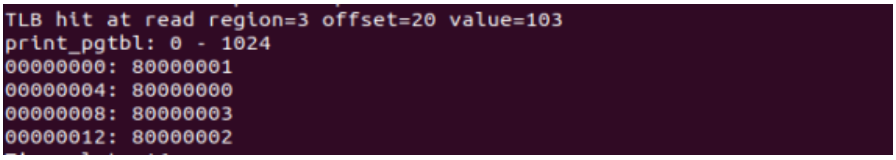
- Ở bước tiếp theo, do lệnh read đã đọc nội dung từ dữ liệu đã load lên TLB(được trình bày ở phần TLB), nên không truy cập vào vùng nhớ vật lý. Do vậy phần, hàm MEMPHY_dump không được in ra mà chỉ in ra nội dung bảng trang bộ nhớ ảo.

```
TLB hit at read region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
```

- Khi có thao tác write truy cập vào vùng nhớ vật lý tiếp theo, hàm MEMPHY_dump lúc này sẽ in ra địa chỉ và nội dung được ghi vào của thao tác write trước đó (BYTE 000000a4: 100). Và vẫn in lại nội dung bảng trang của bộ nhớ ảo ra màn hình.

```
TLB miss at write region=3 offset=20 value=103
Dumping Memory Content (up to 1048576 bytes ):
BYTE 000000a4: 100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Free_entries : 254
```

- Tương tự như lệnh read ở trên, phần này đọc nội dung đã có trong TLB nên hàm MEMPHY_dump không in ra. Do đó chỉ in ra bảng trang.



5 TLB

5.1 Trình bày giải thuật

1. **Kiểm tra TLB:** Khi CPU cần truy cập một địa chỉ bộ nhớ ảo, nó gửi yêu cầu đến TLB để kiểm tra xem địa chỉ đó đã được ánh xạ hay chưa.
2. **Tìm kiếm trong TLB:** TLB kiểm tra xem địa chỉ bộ nhớ ảo có tồn tại trong bộ nhớ cache của nó không. Nếu có, TLB trả về địa chỉ tương ứng trong bộ nhớ vật lý.
3. **Nếu không tìm thấy trong TLB:** Trong trường hợp địa chỉ bộ nhớ ảo không được tìm thấy trong TLB, CPU truy cập bảng trang để lấy mục nhập bảng trang tương ứng với địa chỉ ảo.
4. **Cập nhật TLB:** Sau khi CPU đã nhận được địa chỉ vật lý tương ứng từ MMU, nó sẽ cập nhật TLB với bản ghi mới này. Điều này giúp tăng tốc độ truy cập trong các truy cập sau, vì không cần phải sử dụng bộ dịch mỗi lần.
5. **Xử lý xung đột TLB:** Trong trường hợp TLB đã đầy, một cơ chế thay thế sẽ được áp dụng để chọn ra bản ghi nào sẽ bị loại bỏ để nhường chỗ cho bản ghi mới. (Ở đây nhóm hiện thực cơ chế FIFO)

5.2 Trả lời câu hỏi

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?
Trong một hệ thống đa lõi, nếu mỗi lõi CPU có thể chạy trong một ngữ cảnh khác nhau, và mỗi lõi có MMU riêng cũng như phần của lõi (TLB), điều này sẽ ảnh hưởng đến các phương pháp dịch địa chỉ như sau:

- **MMU Riêng Biệt:** Mỗi lõi CPU có MMU riêng sẽ cho phép chúng xử lý các bảng trang địa chỉ vật lý một cách độc lập. Điều này giúp tăng hiệu suất khi chạy đa tiến trình, vì mỗi lõi có thể chuyển đổi ngữ cảnh mà không ảnh hưởng đến các lõi khác.
- **TLB Đa Cấp:** Với TLB đa cấp, việc tra cứu địa chỉ được tối ưu hóa hơn. TLB cấp 1 (L1) thường nhỏ và nhanh, chứa các mục tra cứu gần đây nhất, trong khi TLB cấp 2 (L2) lớn hơn và chứa một phạm vi rộng hơn các mục tra cứu. Điều này giảm thiểu số lần truy cập bảng trang, từ đó giảm độ trễ và tăng hiệu suất tổng thể.
- **Hiệu Suất Cao:** Khi mỗi lõi có TLB riêng, việc tra cứu địa chỉ có thể được thực hiện song song, giảm thiểu xung đột và tăng tốc độ xử lý.

- **Quản Lý Bộ Nhớ:** Các cấu hình phần cứng mới này đòi hỏi hệ điều hành phải quản lý bộ nhớ một cách thông minh hơn, đặc biệt là trong việc đồng bộ hóa các bảng trang và TLB giữa các lõi khi chạy cùng một tiến trình trên nhiều lõi.

Tóm lại, việc mỗi lõi CPU có MMU và TLB riêng biệt cùng với sự xuất hiện của TLB đa cấp trong CPU hiện đại làm tăng hiệu suất xử lý và tối ưu hóa việc dịch địa chỉ trong hệ thống đa lõi. Tuy nhiên, nó cũng đặt ra thách thức mới cho hệ điều hành trong việc quản lý bộ nhớ một cách hiệu quả.

5.3 Giải thích kết quả

Input: os_0_mlq_paging

```
viet@viet-VirtualBox:~/Desktop/Group12-L08$ ./os os_0_mlq_paging
Time slot   0
ld_routine
        Loaded a process at input/proc/p0s, PID: 1 PRIO: 0
Time slot   1
        CPU 1: Dispatched process  1
Time slot   2
        Loaded a process at input/proc/p1s, PID: 2 PRIO: 15
Time slot   3
        CPU 0: Dispatched process  2
        Loaded a process at input/proc/p1s, PID: 3 PRIO: 0
Time slot   4
        Loaded a process at input/proc/p1s, PID: 4 PRIO: 0
Time slot   5
Time slot   6
TLB miss at write region=1 offset=20 value=100
Dumping Memory Content (up to 1048576 bytes ):
Byte with content not found.
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Free_entries : 255
Time slot   7
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  3
Time slot   8
Time slot   9
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  4
Time slot  10
Time slot  11
Time slot  12
Time slot  13
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  1
TLB hit at read region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Time slot  14
TLB miss at write region=3 offset=20 value=103
```

```
Dumping Memory Content (up to 1048576 bytes ):
BYTE 000000a4: 100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Free_entries : 254
Time slot  15
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  3
TLB hit at read region=3 offset=20 value=103
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Time slot  16
Time slot  17
        CPU 1: Processed  1 has finished
        CPU 1: Dispatched process  4
Time slot  18
Time slot  19
        CPU 0: Processed  3 has finished
        CPU 0: Dispatched process  2
Time slot  20
Time slot  21
        CPU 1: Processed  4 has finished
        CPU 1 stopped
Time slot  22
Time slot  23
        CPU 0: Processed  2 has finished
        CPU 0 stopped
HIT = 2
MISS = 2
HIT/MISS= 1.000000
viet@viet-VirtualBox:~/Desktop/Group12-L08$
```

Giải thích kết quả os_0_mlq_paging:

- Khi chúng ta nhận lệnh `write` từ input, chúng ta truy cập để xem địa chỉ có `region 1 offset 20 value 100` ở bộ nhớ ảo đã tồn tại trong TLB cache chưa. Trong trường hợp này là chưa, vậy nên TLB miss. Sau đó, chúng ta cập nhật TLB bằng cách dùng CPU truy cập bảng trang để lấy mục nhập bảng trang tương ứng với địa chỉ ảo. Vì khởi tạo mặc định của TLB có 256 entries nên ở đây chúng ta đã dùng 1 entry.

```
TLB miss at write region=1 offset=20 value=100
Dumping Memory Content (up to 1048576 bytes ):
Byte with content not found.
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Free_entries : 255
```

- Ở bước tiếp theo, khi chúng ta nhận lệnh `read` từ cùng một `region, offset` và `value`, lúc này TLB hit trả về địa chỉ tương ứng trong bộ nhớ vật lý giúp truy cập nhanh hơn.

```
TLB hit at read region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
```

- Khi chúng ta nhận lệnh `write` từ input, chúng ta truy cập để xem địa chỉ có `region 3 offset 20 value 103` ở bộ nhớ ảo đã tồn tại trong TLB cache chưa. Trong trường hợp này là chưa, vậy nên TLB miss. Sau đó, chúng ta cập nhật TLB bằng cách dùng CPU truy cập bảng trang để lấy mục nhập bảng trang tương ứng với địa chỉ ảo. Vì khởi tạo mặc định của TLB có 256 entries, chúng ta đã dùng 2 entries nên còn 254 entries.

```
TLB miss at write region=3 offset=20 value=103
Dumping Memory Content (up to 1048576 bytes ):
BYTE 000000a4: 100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Free_entries : 254
```

- Ở bước tiếp theo, khi chúng ta nhận lệnh `read` từ cùng một `region, offset` và `value`, lúc này TLB hit trả về địa chỉ tương ứng trong bộ nhớ vật lý giúp truy cập nhanh hơn.

```
TLB hit at read region=3 offset=20 value=103
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
```

- Cuối cùng, ta thống kê số lần hit, miss và tỉ lệ hit/miss.

```
HIT = 2
MISS = 2
HIT/MISS= 1.000000
```

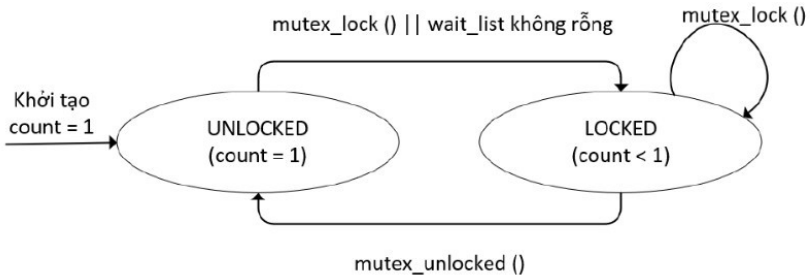
6 Put it all together

Question: What will happen if the synchronization is not handled in your simple OS? Illustrate the problem of your simple OS by example if you have any.
Answer
Nếu việc đồng bộ hóa không được xử lý đúng cách, nó có thể dẫn đến các vấn đề như:

- **Data Race:** Xảy ra khi các process hoặc thread cùng thực hiện đọc và ghi lên một khối dữ liệu, dẫn đến tính không nhất quán của dữ liệu.
- **Race Condition:** Là một tình huống không mong muốn xảy ra khi một thiết bị hoặc hệ thống cố gắng thực hiện hai hoặc nhiều hoạt động cùng một lúc, nhưng do tính chất của thiết bị hoặc hệ thống, các hoạt động phải được thực hiện theo thứ tự đúng đắn để được thực hiện một cách chính xác.
- **Deadlock:** Deadlock trong hệ điều hành là tình huống mà hai hoặc nhiều tiến trình hoặc luồng không thể tiếp tục vì mỗi cái đang chờ đợi cái khác thả ra tài nguyên. Nói cách khác, đó là một trạng thái mà một nhóm các tiến trình bị mắc kẹt và không thể tiến triển. Deadlock thường xảy ra trong các hệ thống nơi nhiều tiến trình cạnh tranh cho tài nguyên hạn chế như thời gian CPU, bộ nhớ hoặc thiết bị nhập hoặc xuất.

Vì vậy để hoàn thiện bài tập lớn này, nhóm em chủ yếu sử dụng kĩ thuật Mutex Lock:

- Mutex lock là một cấu trúc dữ liệu, được Linux kernel xây dựng theo nguyên tắc mutual exclusion, dùng để ngăn chặn race condition xảy ra trên các cấu trúc dữ liệu khác. Nói nôm na, mutex lock đảm bảo rằng: tại một thời điểm bất kì, chỉ có tối đa một thread truy cập vào critical resource.



7 DEMO

Sau đây là đường dẫn của phần video báo cáo của nhóm chúng em:

- Phần báo cáo report:
  https://drive.google.com/file/d/163CUyUUaHty3_FNzdZvEi-APcxhYw9PD/view?usp=sharing
- Phần demo code:
  https://drive.google.com/file/d/1X0lMtVEQwc9rw0OYuzQOaEcaicFZPcye/view?usp=sharing

Link drive tổng hợp video báo cáo:
https://drive.google.com/drive/u/1/folders/15Dob9lXrMWt8VSuIcNwW0EATKoteKxXy

References

[1] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne. (2018). *Operating System Concepts*, Tenth edition.

[2] David A. Patterson, John L. Hennessy. (2014). *Computer Organization And Design*, Fifth edition.