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

OPERATING SYSTEM (CO2018)

Assignment

"Simple Operating System HK242"

Instructor(s): Hoàng Lê Hải Thanh

Students: Nguyễn Xuân Huy Hoàng - 2311070 *(Group TN01 - Team 11, **Leader**)*
Trần Vĩnh Dũng - 2310574 *(Group TN01 - Team 11)*
Lê Quang Dũng - 2310548 *(Group TN01 - Team 11)*
Bùi Trọng Hiển - 2310993 *(Group TN01 - Team 11)*
Ngô Quang Hiệu - 2311015 *(Group TN01 - Team 11)*

HO CHI MINH CITY, APRIL 2025

Contents

List of Symbols

$\mathbb{N}$ Set of natural numbers

$\mathbb{R}$ Set of real numbers

$\mathbb{R}^+$ Set of positive real numbers

List of Acronyms

ODE (First-Order) Ordinary Differential Equation

IVP Initial-Value Problem

LTE Local Truncation Error

DS Dynamical System

Fig. Figure

Tab. Table

Sys. System of Equations

Eq. Equation

e.g. For Example

i.e. That Is

List of Figures

List of Tables

Member list & Workload

No.	Fullname	Student ID	Problems	% done
1	Nguyễn Xuân Huy Hoàng	2311070	- Gõ LaTeX	100%
2	Trần Vĩnh Dũng	2310574	- Hiện thực `Memory Management`	100%
3	Lê Quang Dũng	2310548	- Hiện thực `system call`	100%
4	Bùi Trọng Hiển	2310993	- Hiện thực `Memory Management`	100%
5	Ngô Quang Hiệu	2311015	- Hiện thực `schedule`	100%

Table 1: Member list & workload

1 Giới thiệu

Bài tập này mô phỏng một hệ điều hành đơn giản với ba chức năng chính:

- Bộ lập lịch (`Scheduler`): Cài đặt bộ lập lịch sử dụng `MLQ (Multilevel Queue)`.
- Quản lý bộ nhớ (`Memory Management`): Cấp phát bộ nhớ từ bộ nhớ ảo đến bộ nhớ vật lý sử dụng phân trang (`paging`).
- `System Call`: Hoàn thiện `killall command`. Xác định tiến trình cần huỷ bỏ để chấm dứt, tương tự như lệnh `killall` thông thường.

Việc triển khai các chức năng quan trọng cũng như kết quả kiểm thử sẽ được mô tả trong các phần tiếp theo.

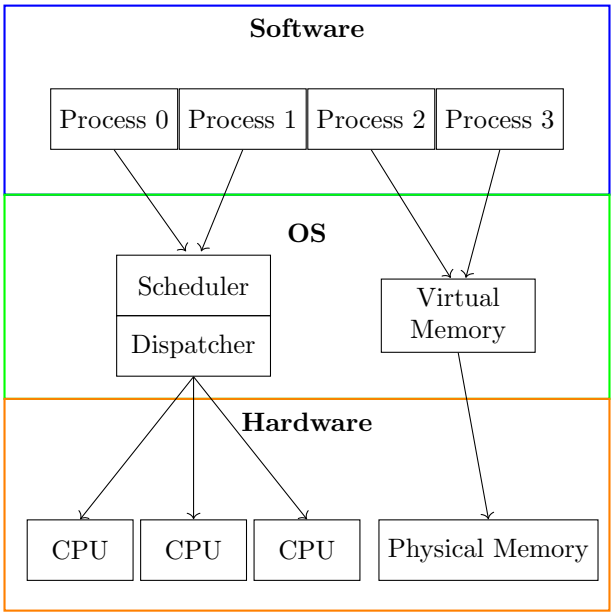


Figure 1: Tổng quan về các mô-đun chính trong bài tập lớn

2 Lý thuyết

2.1 Giới thiệu về lập lịch trong hệ điều hành

Lập lịch là một trong những thành phần quan trọng của hệ điều hành, giúp quản lý và phân phối tài nguyên `CPU` cho các tiến trình một cách hiệu quả. Hệ điều hành hiện đại thường hỗ trợ đa xử lý, tức là có thể chạy đồng thời nhiều tiến trình trên nhiều bộ xử lý. Để làm được điều này, hệ điều hành sử dụng các thuật toán lập lịch để quyết định tiến trình nào sẽ được thực thi khi một `CPU` trở nên sẵn sàng.

Một trong những phương pháp lập lịch phổ biến là sử dụng hàng đợi sẵn sàng (`Ready Queue`). Mỗi tiến trình mới được tạo ra sẽ được đưa vào hàng đợi này để chờ `CPU`. Khi `CPU` trống, hệ điều hành sẽ chọn một tiến trình từ hàng đợi để thực thi. Quá trình này được điều phối bởi bộ lập lịch (`Scheduler`). Ở bài tập lớn này, nhóm sẽ thực hiện thuật toán `Multilevel Queue (MLQ)` cho việc điều phối tiến trình.

2.1.1 Tổng quan về MLQ

Thuật toán `Multilevel Queue (MLQ)` là một phương pháp lập lịch tiên tiến được sử dụng trong các hệ điều hành như `Linux`. Thuật toán này chia các tiến trình thành nhiều hàng đợi khác nhau, mỗi hàng đợi có một mức độ ưu tiên cố định. Tiến trình sẽ được xếp vào hàng đợi phù hợp dựa trên mức độ ưu tiên của nó (`prio`).

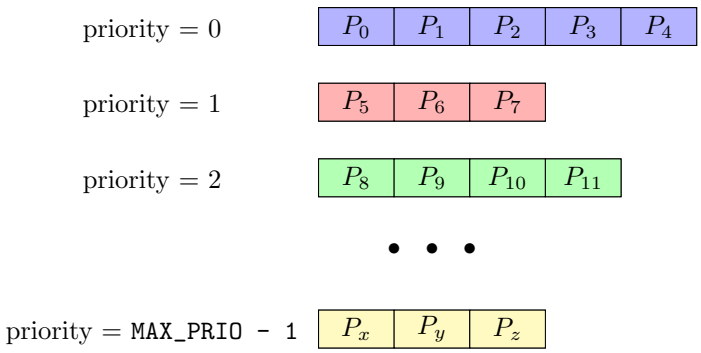


Figure 2: Các hàng đợi khác nhau tương ứng với mỗi độ ưu tiên

Hệ thống chứa một số mức ưu tiên tối đa được định nghĩa bởi hằng số `MAX_PRIO`. Ví dụ, trong `Linux`, giá trị `MAX_PRIO` có thể là 140, với mức độ ưu tiên `prio` chạy từ 0 đến `MAX_PRIO - 1`. Mỗi mức độ ưu tiên sẽ tương ứng với một thời lượng `CPU` nhất định, được tính bằng công thức:

```
slot = (MAX_PRIO - prio)
```
(1)

Điều này có nghĩa là các tiến trình có mức độ ưu tiên cao hơn (`prio` nhỏ hơn) sẽ có nhiều thời gian `CPU` hơn so với các tiến trình có mức độ ưu tiên thấp.

2.1.2 Cách thức hoạt động của thuật toán MLQ

1. **Tạo tiến trình mới:** Khi một chương trình mới được nạp vào hệ thống, bộ tải (`Loader`) sẽ tạo một tiến trình mới và gán cho nó một `PCB` (`Process Control Block`).
2. **Xếp tiến trình vào hàng đợi sẵn sàng:** Bộ tải sẽ sao chép mã chương trình vào vùng nhớ mã lệnh của tiến trình, sau đó đưa `PCB` của tiến trình vào hàng đợi sẵn sàng có mức độ ưu tiên tương ứng với `prio` của tiến trình.
3. **Lập lịch theo MLQ:** `CPU` chạy các tiến trình theo cơ chế vòng tròn (`Round Robin`) trong từng hàng đợi. Mỗi tiến trình được phép chạy trong một khoảng thời gian cố định (`time slice`) hay (`quantum`). Nếu thời gian này hết, tiến trình sẽ bị đưa trở lại hàng đợi ưu tiên tương ứng, và `CPU` sẽ chọn một tiến trình khác để thực thi.
4. **Chuyển đổi tiến trình:** Khi một tiến trình sử dụng hết `slot` của nó, hệ thống sẽ chuyển tài nguyên `CPU` cho tiến trình tiếp theo trong hàng đợi ưu tiên. Nếu một tiến trình chưa hoàn thành công việc của mình, nó sẽ phải chờ đến lượt tiếp theo.

2.1.3 Đặc điểm của MLQ

- **Hỗ trợ nhiều mức độ ưu tiên**: Mỗi hàng đợi có một mức độ ưu tiên cố định, giúp hệ thống phân bổ tài nguyên hợp lý.
- **Cân bằng tải giữa các tiến trình**: Cơ chế `slot` đảm bảo mỗi tiến trình có cơ hội sử dụng `CPU` dựa trên mức độ ưu tiên của nó.
- **Triển khai đơn giản nhưng hiệu quả**: `MLQ` cung cấp một phương pháp dễ hiểu nhưng mạnh mẽ để quản lý tiến trình trong hệ điều hành.

2.2 Tổng quan về quản lý bộ nhớ

Quản lý bộ nhớ là một thành phần quan trọng của hệ điều hành, đảm bảo rằng các tiến trình có thể truy cập tài nguyên bộ nhớ một cách hiệu quả mà không xung đột với nhau. Các chức năng chính của quản lý bộ nhớ bao gồm:

- **Phân bổ và giải phóng bộ nhớ**: Hệ điều hành cấp phát không gian bộ nhớ cho tiến trình khi nó được tạo ra và giải phóng bộ nhớ khi tiến trình kết thúc.
- **Bảo vệ bộ nhớ**: Ngăn chặn một tiến trình truy cập vùng nhớ của tiến trình khác nếu không được phép.
- **Tối ưu hóa hiệu suất**: Giảm thiểu phân mảnh bộ nhớ và tối ưu tốc độ truy cập.

Hệ điều hành sử dụng các phương pháp như phân đoạn (`segmentation`), phân trang (`paging`), bộ nhớ ảo (`virtual memory`) để quản lý bộ nhớ hiệu quả. Cụ thể trong bài tập lớn này, phân trang (paging) kết hợp bộ nhớ ảo (virtual memory) sẽ được sử dụng để quản lý bộ nhớ hiệu quả.

2.2.1 Khái niệm về bộ nhớ ảo

Bộ nhớ ảo là một kỹ thuật giúp hệ điều hành cấp phát bộ nhớ lớn hơn bộ nhớ vật lý thực tế bằng cách sử dụng một phần không gian lưu trữ trên đĩa cứng. Các ưu điểm của bộ nhớ ảo:

- Tách biệt không gian bộ nhớ giữa các tiến trình, tránh xung đột.
- Hỗ trợ thực thi các chương trình lớn hơn bộ nhớ RAM thực tế.
- Cung cấp cơ chế bảo vệ bộ nhớ thông qua ánh xạ địa chỉ ảo sang địa chỉ vật lý.

2.2.2 Quá trình ánh xạ bộ nhớ ảo sang bộ nhớ vật lý

Bộ nhớ ảo hoạt động bằng cách chia không gian địa chỉ thành các khung trang (`page frame`) trong bộ nhớ vật lý và các trang (`page`) trong bộ nhớ ảo. Quá trình ánh xạ này được thực hiện thông qua **bảng trang (page table)**, giúp chuyển đổi địa chỉ ảo thành địa chỉ vật lý.

Các bước ánh xạ bộ nhớ ảo sang bộ nhớ vật lý:

1. **Tiến trình truy cập địa chỉ ảo**: Tiến trình sử dụng địa chỉ ảo để đọc hoặc ghi vào vùng nhớ.
2. **Tra cứu bảng trang**: Hệ điều hành tìm kiếm khung trang vật lý tương ứng.
3. **Kiểm tra bộ nhớ cache (`TLB - Translation Lookaside Buffer`)**: Nếu địa chỉ ảo đã được ánh xạ trước đó, nó có thể được lưu trong bộ nhớ đệm TLB để tăng tốc độ truy cập (Phần này bài tập lớn không đề cập và mô tả nên sẽ không thiết kế).
4. **Cấp phát hoặc tải trang**: Nếu trang đã nằm trong bộ nhớ vật lý, hệ thống sẽ truy xuất trực tiếp. Nếu không, hệ điều hành tải trang từ đĩa cứng vào `RAM`.
5. **Cập nhật bảng trang**: Khi một trang mới được tải vào bộ nhớ vật lý, hệ điều hành cập nhật bảng trang.

2.2.3 Kỹ thuật Quản lý Bộ nhớ Ảo

Hệ điều hành sử dụng các kỹ thuật sau để quản lý bộ nhớ ảo hiệu quả:

- **Phân trang theo yêu cầu (Demand Paging)**: Chỉ tải trang vào bộ nhớ khi cần thiết để giảm lượng RAM sử dụng.
- **Hoán đổi trang (Page Replacement)**: Khi bộ nhớ vật lý đầy, hệ điều hành loại bỏ một trang cũ để nhường chỗ cho trang mới. Các thuật toán phổ biến gồm:
  - FIFO (First-In-First-Out)
  - LRU (Least Recently Used)
  - Clock Algorithm
- **Bộ nhớ ảo theo phân đoạn (Segmentation + Paging)**: Kết hợp hai kỹ thuật để tối ưu hóa quản lý bộ nhớ.

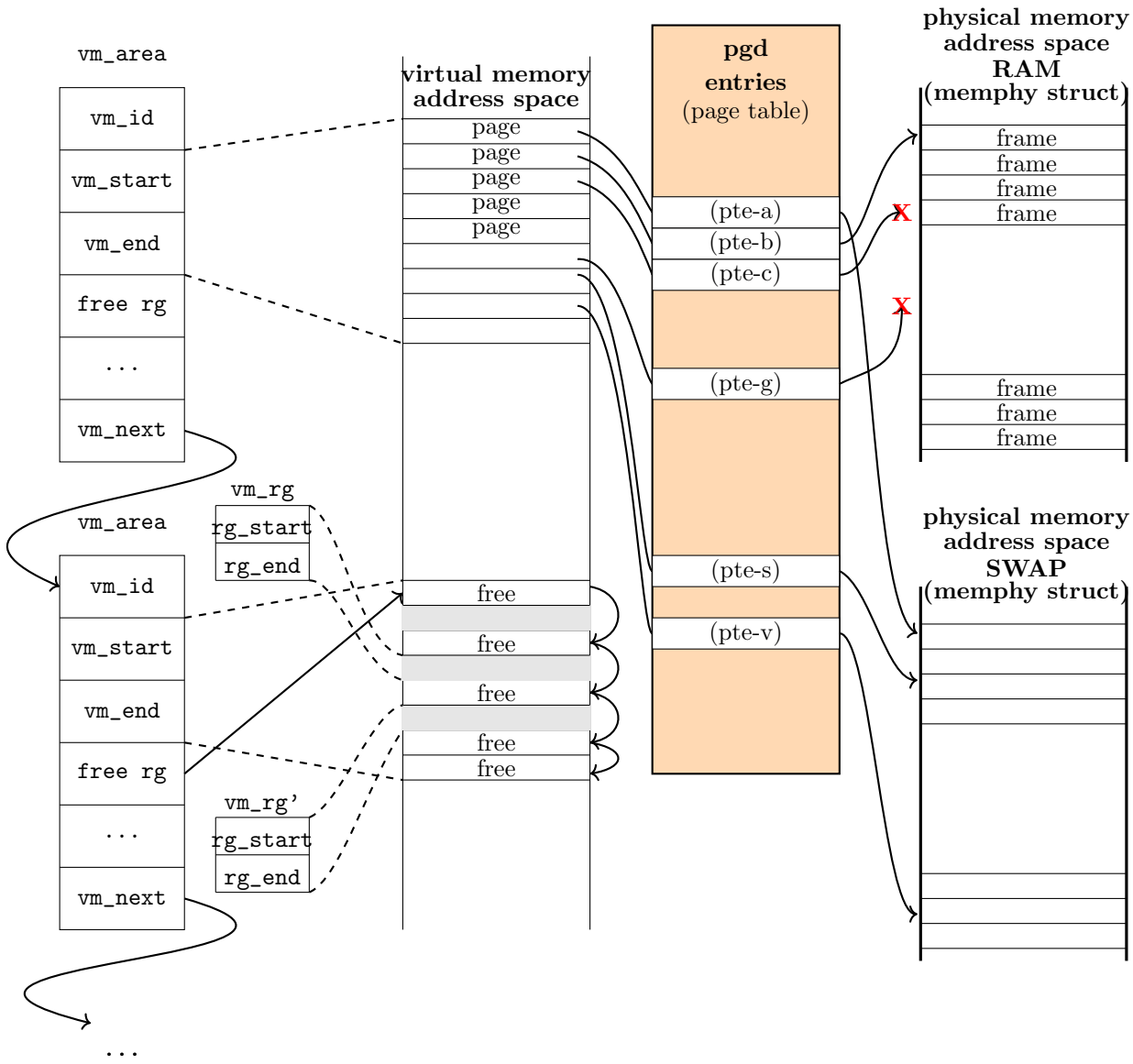


Figure 3: Quản lý bộ nhớ sử dụng phân trang và bộ nhớ ảo

2.3 Sơ lược về System Call

System call (lời gọi hệ thống) là giao diện cốt lõi giữa các chương trình ứng dụng và nhân của hệ điều hành (kernel). Thông qua system call, chương trình người dùng có thể yêu cầu kernel thực hiện các chức năng đặc quyền như quản lý tiến trình, bộ nhớ, hoặc thiết bị ngoại vi. Trong bài tập lớn này, cần hiện thực phần system call `killall`, một lệnh phổ biến dùng để **kết thúc tất cả tiến trình có tên trùng khớp**.

2.3.1 Đặc điểm của System Call

- **Không gọi trực tiếp:** System call thường không được gọi trực tiếp, mà thông qua các *wrapper function* trong thư viện hệ thống (như `libstd`).
- **Wrapper function:** Là lớp bọc mỏng, thực hiện việc sao chép đối số vào các thanh ghi thích hợp và gọi system call. Đôi khi wrapper cũng xử lý kiểm tra lỗi hoặc chuẩn bị dữ liệu.
- **Cơ chế thực thi:** Khi một system call được gọi, chương trình sẽ chuyển quyền điều khiển sang kernel thông qua một ngắt (ví dụ: `int 0x80` trong x86) hoặc lệnh đặc biệt như `syscall`, `svc`,...

2.3.2 Phân loại system call (theo chức năng):

- Quản lý tiến trình (*process control*)
- Quản lý bộ nhớ (*memory management*)
- Quản lý thiết bị (*device management*)
- Quản lý thông tin (*information maintenance*)
- Giao tiếp liên tiến trình (*inter-process communication*)

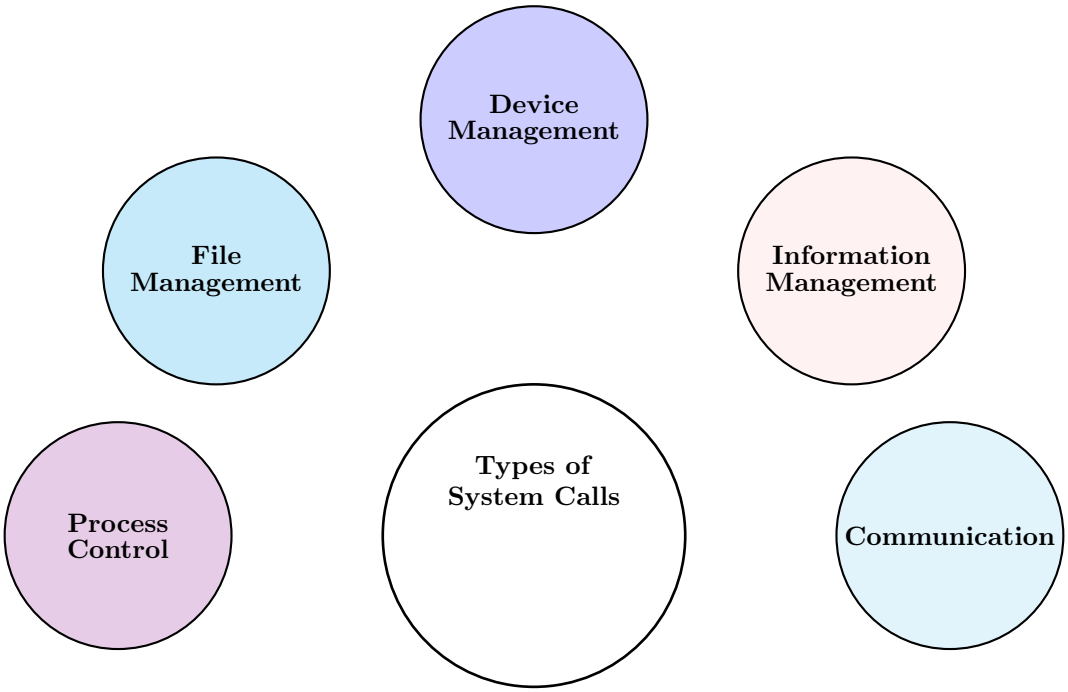


Figure 4: Phân loại system call

3 Phần trả lời câu hỏi

3.1 Lập lịch (scheduler)

3.1.1 What are the advantages of using the scheduling algorithm described in this assignment compared to other scheduling algorithms you have learned?

☐ First-Come, First-Served Scheduling (FCFS)

- First-Come, First-Served (FCFS) là một giải thuật lập lịch đơn giản và non-preemptive, trong đó các tiến trình được thực thi theo thứ tự đến trước, sử dụng hàng đợi kiểu FIFO. FCFS dễ hiện thực và đảm bảo sự công bằng theo thời gian đến, tuy nhiên nó gặp phải hiện tượng convoy effect, khi các tiến trình ngắn phải chờ tiến trình dài hoàn thành, làm giảm hiệu suất trong các hệ thống chia sẻ thời gian. Bên cạnh đó, FCFS không có cơ chế ưu tiên, dẫn đến việc các tiến trình quan trọng hay cần thực hiện ngay có thể bị trì hoãn lâu.
- Ngược lại, Multi-Level Queue (MLQ) cung cấp tính linh hoạt cao hơn bằng cách cho phép thực thi dựa trên mức độ ưu tiên, giúp ngăn chặn hiệu quả convoy effect và cải thiện hiệu suất cho các tiến trình quan trọng.

☐ Shortest-Job-First Scheduling (SJF)

- Shortest Job First (SJF) là giải thuật lập lịch chọn tiến trình có thời gian xử lý (burst time) ngắn nhất để thực thi, có thể preemptive (gọi là Shortest Remaining Time First - SRTF) hoặc là non-preemptive). SJF giúp giảm thời gian chờ trung bình, nhưng gặp phải vấn đề starvation, khi các tiến trình dài có thể không bao giờ được cấp CPU nếu liên tục có tiến trình ngắn xuất hiện. Ngoài ra, SJF khó triển khai trong thực tế do việc ước lượng thời gian xử lý là không chắc chắn.
- So với Multi-Level Queue (MLQ), SJF thiếu cơ chế thực thi dựa trên ưu tiên và không đảm bảo công bằng, trong khi MLQ giúp các tiến trình dài vẫn có cơ hội được xử lý và phù hợp hơn với môi trường đa người dùng. Dù SJF tối ưu hiệu suất xử lý, MLQ lại thực tế hơn cho các hệ thống yêu cầu phân phối tài nguyên cân bằng.

☐ Round-Robin Scheduling (RR)

- Round Robin (RR) là giải thuật lập lịch cấp cho mỗi tiến trình một khoảng thời gian cố định (gọi là quantum) trước khi bị thu hồi CPU và đưa xuống cuối hàng đợi, đảm bảo tính công bằng thông qua cơ chế lập lịch có thu hồi. Cách tiếp cận này ngăn chặn hiện tượng starvation, rất phù hợp với hệ thống chia sẻ thời gian (time-sharing) và hệ thống tương tác (interactive systems), đồng thời công bằng hơn so với FCFS và Priority Scheduling trong môi trường đa người dùng. Tuy nhiên, RR gặp nhược điểm là có quá nhiều lần chuyển đổi ngữ cảnh (context switching), gây giảm hiệu quả sử dụng CPU. Hiệu suất của RR còn phụ thuộc vào giá trị quantum - nếu quá nhỏ thì chuyển đổi ngữ cảnh xảy ra thường xuyên; nếu quá lớn thì lại giống như FCFS.
- So với Multi-Level Queue (MLQ), RR thực thi nghiêm ngặt theo lát thời gian và đối xử công bằng với tất cả tiến trình, trong khi MLQ cân bằng giữa ưu tiên và công bằng bằng cách cho phép tiến trình ưu tiên cao hơn sử dụng CPU nhiều hơn. RR lý tưởng cho các hệ thống tương tác, còn MLQ hiệu quả hơn trong các tải công việc định hướng theo mức độ ưu tiên.

☐ Priority Scheduling

- Lập lịch Priority Scheduling gán cho mỗi tiến trình một mức độ ưu tiên, với CPU sẽ thực thi tiến trình có ưu tiên cao nhất trước. Giải thuật này có thể là preemptive hoặc non-preemptive, giúp xử lý hiệu quả các tác vụ quan trọng như trong hệ thống thời gian thực. Ngoài ra, nó có thể được tối ưu cho các loại tải công việc khác nhau, chẳng hạn như tiến trình thiên về CPU hoặc tiến trình thiên về I/O. Tuy nhiên, Priority Scheduling gặp phải vấn đề đói tài nguyên (starvation), khi các tiến trình có ưu tiên thấp có thể không bao giờ được cấp CPU, do đó cần một cơ chế già hóa (aging) để ngăn chặn việc bị trì hoãn vô thời hạn.

- So với `Multi-Level Queue (MLQ) scheduling`, `Priority Scheduling` sử dụng một hàng đợi duy nhất với các mức độ ưu tiên khác nhau, trong khi `MLQ` tổ chức các tiến trình thành nhiều hàng đợi và giới hạn thời gian mà mỗi hàng đợi có thể sử dụng `CPU` nhằm tránh tình trạng đói tài nguyên. Mặc dù `Priority Scheduling` đơn giản hơn, `MLQ` lại phù hợp hơn cho các hệ thống đa người dùng nhờ cách tiếp cận có cấu trúc trong việc phân bổ tài nguyên.

☐ `Multilevel Feedback Queue Scheduling (MLFQ)`

- Lập lịch `Multi-Level Feedback Queue (MLFQ)` tương tự như `Multi-Level Queue (MLQ)` nhưng cho phép các tiến trình di chuyển giữa các hàng đợi dựa trên lịch sử thực thi. Nếu một tiến trình sử dụng quá nhiều thời gian `CPU`, nó sẽ bị chuyển xuống hàng đợi có mức ưu tiên thấp hơn, trong khi các tiến trình chờ quá lâu sẽ được chuyển lên hàng đợi có mức ưu tiên cao hơn thông qua cơ chế `aging`. Cách tiếp cận này khiến `MLFQ` trở nên rất linh hoạt và thích nghi tốt với các loại tải công việc khác nhau, hiệu quả trong việc ngăn chặn đói tài nguyên và cân bằng giữa các tiến trình dài và ngắn.
- Tuy nhiên, `MLFQ` phức tạp hơn `MLQ` trong việc triển khai và đòi hỏi phải tinh chỉnh các tham số cẩn thận để xác định tốc độ chuyển hàng đợi của các tiến trình. So với `MLQ`, nơi các hàng đợi là cố định, `MLFQ` điều chỉnh mức ưu tiên một cách linh hoạt, giúp giảm hiện tượng đói tài nguyên hiệu quả hơn. Mặc dù `MLFQ` mang lại sự linh hoạt cao hơn, `MLQ` lại dễ hiện thực hơn và có thể phù hợp hơn với các hệ thống đơn giản.

☐ **Kết luận:** `Multi-Level Queue (MLQ)` là một thuật toán hiệu quả trong việc quản lý các loại tiến trình khác nhau bằng cách tổ chức chúng vào các hàng đợi có mức ưu tiên cố định. Thuật toán này đảm bảo phân bổ tài nguyên `CPU` hợp lý đồng thời duy trì trình tự thực thi có cấu trúc dựa trên mức độ ưu tiên.

- Quản lý hiệu quả các loại tiến trình đa dạng (hệ thống (`system`), tương tác (`interactive`), xử lý lô (`batch`), nền (`background`)).
- Ưu tiên các tiến trình quan trọng bằng cách gán chúng vào các hàng đợi có mức ưu tiên cao hơn.
- Ngăn ngừa hiện tượng đói tài nguyên (*starvation*) và `convey effect` bằng cách giới hạn thời gian sử dụng `CPU`.
- Dễ triển khai hơn so với `Multi-Level Feedback Queue (MLFQ)`, giúp đơn giản hóa quá trình vận hành.

3.2 Quản lý bộ nhớ (Memory management)

3.2.1 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?

☐ Thiết kế nhiều vùng nhớ (`multiple segments`) mang lại các lợi ích sau:

- **Phân tách trách nhiệm rõ ràng:** Mỗi vùng nhớ đảm nhận vai trò riêng biệt như `code`, `heap`, `stack` → dễ dàng trong việc quản lý.
- **Bảo vệ bộ nhớ tốt hơn:** Có thể thiết lập quyền truy cập khác nhau cho từng vùng (`read-only` cho `code`, `read-write` cho `heap`, ...).
- **Cấp phát hiệu quả hơn:** Các vùng nhớ như `heap` và `stack` có thể mở rộng độc lập → giảm phân mảnh bộ nhớ (`memory fragmentation`).
- **Đơn giản hóa việc quản lý tiến trình:** Theo dõi vùng nhớ riêng biệt giúp dễ cấp phát, thu hồi và swap.
- **Tối ưu hiệu năng:** Dễ dàng tối ưu từng vùng dựa trên tính chất sử dụng (có sự khác biệt giữa các thuật toán trên các vùng nhớ khác nhau như `stack`, `heap`, ...).

☐ Tổng kết: Thiết kế phân vùng giúp hệ điều hành mô phỏng quản lý bộ nhớ một cách rõ ràng, hiệu quả và an toàn hơn.

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

☐ Ưu điểm:

- **Tăng khả năng tổ chức bộ nhớ:** Khi sử dụng nhiều cấp trong bảng trang, hệ thống có thể tổ chức bộ nhớ hiệu quả hơn, đặc biệt là đối với bộ nhớ lớn, giúp giảm thiểu lượng bộ nhớ yêu cầu cho bảng trang.
- **Tiết kiệm bộ nhớ:** Không cần giữ toàn bộ bảng trang lớn trong `RAM`. Chỉ cần giữ những bảng con đang sử dụng.
- **Tổ chức hợp lý hơn cho không gian địa chỉ lớn:** Ví dụ `32-bit` hoặc `64-bit`.

☐ Nhược điểm:

- **Chi phí tra cứu địa chỉ cao hơn:** Mặc dù bảng trang nhiều cấp giúp tiết kiệm bộ nhớ, nhưng nó có thể làm tăng độ phức tạp khi tra cứu địa chỉ. Cần phải thực hiện nhiều lần tra cứu trong các bảng trang để tìm ra địa chỉ vật lý cuối cùng, dẫn đến tăng độ trễ (`latency`) trong việc chuyển đổi địa chỉ từ ảo sang vật lý.
- **Phức tạp hóa hệ thống:** Cần thêm `logic` để xử lý các cấp bảng, khó khăn trong việc hiện thực thuật toán cho hệ thống.

3.2.3 What are the advantages and disadvantages of segmentation with paging?

☐ Ưu điểm:

- **Tách biệt và quản lý vùng nhớ theo ngữ nghĩa:**
  - Vì mỗi segment phản ánh một thành phần logic trong chương trình, và chỉ cho phép quy định quyền truy cập khác nhau cho các đoạn (VD: `only-read` cho `code segment`, `read-write` cho `data segment`, ...)
  - Bên cạnh đó, nhu cầu cấp phát bộ nhớ của các thành phần sẽ không ảnh hưởng đến các thành phần khác (VD: khi cần cấp phát cho `heap`, chỉ cần tăng con trỏ `srbk` của `heap segment`, không ảnh hưởng đến `stack` hay `code`)
- **Giảm thiểu phân mảnh ngoài (external fragmentation):** Do `paging` sử dụng khung trang kích thước cố định, nên vùng nhớ vật lý được sử dụng hiệu quả hơn.
- **Hỗ trợ không gian địa chỉ lớn:** Việc kết hợp giữa phân đoạn (`segmentation`) với phân trang (`paging`) giúp chia nhỏ không gian địa chỉ lớn, dễ dàng trong việc quản lý và debug (đặc biệt hiệu quả với các hệ thống `32-bit`, `64-bit`).
- **Hỗ trợ swapping mượt mà hơn:** Trang nào không dùng có thể bị `swap` ra đĩa, không cần `swap` cả đoạn.

☐ Nhược điểm:

- **Phức tạp trong quản lý bộ nhớ:**
  - Việc hiện thực phân trang kết hợp phân đoạn cần đảm bảo sự duy trì cho các cấu trúc dữ liệu như `Segment Table`, `Page Table`.
  - Với mỗi lần tra cứu, phải duyệt qua nhiều bước như:
    - [`segment selector`] → `Segment Table` → `base address`
    - [`offset`] → `Page Table` → `frame number`
    - `Cuối cùng` → `Physical Memory`

    ⇒ Dẫn đến việc tăng `overhead`, ảnh hưởng đến hiệu năng.
- **Chậm hơn khi tra địa chỉ:**
  - Do cần nhiều lần tra bảng trang, nếu như sử dụng paging nhiều cấp, có thể gia tăng độ trễ (`latency`).
  - Cách khắc phục: thường dùng thêm các `TLB` (`Translation Lookaside Buffer`) để `cache` các kết quả khi ánh xạ bộ nhớ.
- **Tốn thêm bộ nhớ cho các bảng quản lý:**
  - Mỗi `segment` cần có một `table` riêng → Tốn thêm `RAM` cho các `metadata`
  - Nếu như chương trình có nhiều `segment` nhỏ → Tăng `overhead` quản lý.

3.3 Lời gọi hệ thống (System Call)

3.3.1 What is the mechanism to pass a complex argument to a system call using the limited registers?

☐ Các `system call` (lời gọi hệ thống) thường được truyền qua một số thanh ghi giới hạn. Ví dụ như trong kiến trúc `x86_64 Linux`, các thanh ghi như `rdi`, `rsi`, `rdx`, `r10`, `r8`, `r9` thường được dùng để truyền tối đa 6 tham số đầu tiên.

☐ Khi tham số cần truyền là phức tạp (ví dụ: một cấu trúc `struct` lớn, một mảng, hoặc chuỗi dài...), thì không thể truyền trực tiếp toàn bộ nội dung thông qua các thanh ghi.

☐ Do đó cần phải có một phương thức truyền tham số khác, cụ thể là truyền theo con trỏ hay nói cách khác là truyền địa chỉ của dữ liệu vào thanh ghi, khi đó chỉ cần một thanh ghi để truyền cho một `struct` lớn.

☐ Cách hoạt động cụ thể như sau:

- Từ `user space`, chương trình chuẩn bị dữ liệu (ví dụ: `struct`) trong vùng nhớ của nó.
- Địa chỉ của dữ liệu đó được truyền vào một trong các thanh ghi khi gọi `syscall`.
- Trong `kernel`, dùng hàm `copy_from_user()` để sao chép dữ liệu từ vùng nhớ của `user` sang `kernel space`. Kiểm tra con trỏ không `NULL`, không vượt quá không gian địa chỉ người dùng. Bảo vệ trang (`page fault`) và quyền truy cập (`read/write`).
- Sau đó, `kernel` mới sử dụng dữ liệu đó để xử lý.

☐ Những ưu điểm khi truyền con trỏ.

- **Tiết kiệm tài nguyên**: chỉ cần 1 thanh ghi để truyền con trỏ.
- **An toàn**: `kernel` có thể kiểm tra hợp lệ, tránh lỗi hoặc tấn công từ `user space`.
- **Linh hoạt**: truyền `struct`, chuỗi, mảng, danh sách... đều được chỉ bằng con trỏ.

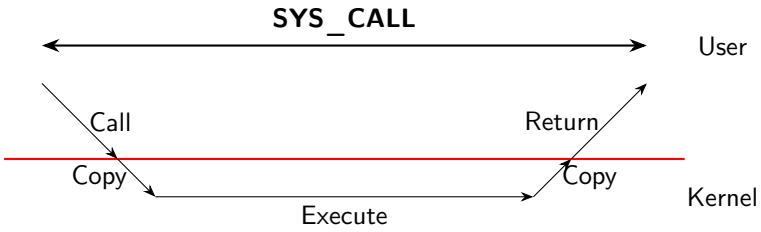


Figure 5: Kernel - User

3.3.2 What happens if the syscall job implementation takes too long execution time?

☐ Khi một syscall bị thực thi quá thì sẽ gây ra các hệ quả sau:

- Hầu hết mã trong nhân, đặc biệt là trong ngữ cảnh `syscall`, được thực thi không thể bị gián đoạn một cách tùy ý. Điều này có nghĩa là: khi `syscall` đang chạy, `kernel` không thể tự ý tạm dừng để cho tiến trình khác chạy; các tiến trình khác, kể cả có mức ưu tiên cao, có thể không được cấp `CPU`; trên hệ thống một lõi, hệ thống có thể treo hoàn toàn; trên hệ thống đa lõi, lõi `CPU` đang thực thi `syscall` sẽ bị chiếm dụng.
- Nếu `syscall` giữ một `spinlock` hoặc `mutex`, nó sẽ chặn mọi đoạn mã khác đang cần cùng tài nguyên đó. Nếu `syscall` bị treo trong khi đang giữ khóa, các `syscall` hoặc tiến trình khác cũng có thể bị treo theo, dẫn đến `deadlock` toàn hệ thống.
- Hệ thống phản hồi chậm: Các ứng dụng người dùng bị treo vì chúng đang đợi `syscall` hoàn thành, giao diện người dùng (`UI`) bị chậm hoặc không phản hồi, đặc biệt nếu `syscall` là một phần trong thao tác quan trọng (như thao tác với tệp, xử lý tiến trình...), các tiến trình thời gian thực có thể bị trễ, điều này cực kỳ nguy hiểm trong hệ thống nhúng hoặc điều khiển thời gian thực.

3.4 Put It All Together

3.4.1 What happens if the synchronization is not handled in your Simple OS? Illustrate the problem of your simple OS (assignment outputs) by example if you have any.

Nếu như đồng bộ không được xử lý trong bài tập lớn này. Có nhiều trường hợp nghiêm trọng có thể xảy ra, điển hình là tranh đoạt điều khiển (`race condition`).

☐ Trường hợp không sử dụng cơ chế đồng bộ (`mutex_lock`) trong `sched.c` cụ thể là ở các hàm `get_mlq_proc`, `put_mlq_proc`, `add_mlq_proc`.

- Input sử dụng:

Thông tin đầu vào

- `Time slide: 1`
- `Number of CPU: 4`
- `Number of processes: 8`
- `RAM size: 1048576 bytes`
- `SWAP size 1: 16777216 bytes`
- `SWAP size 2: 0 bytes`
- `SWAP size 3: 0 bytes`
- `SWAP size 4: 0 bytes`

Process	Arrival Time	Priority
p0s (p1)	0	0
p0s (p2)	0	0
p0s (p3)	0	0
p0s (p4)	0	0
p0s (p5)	0	0
p0s (p6)	0	0
p0s (p7)	0	0
p0s (p8)	0	0

- Ví dụ minh hoạ:

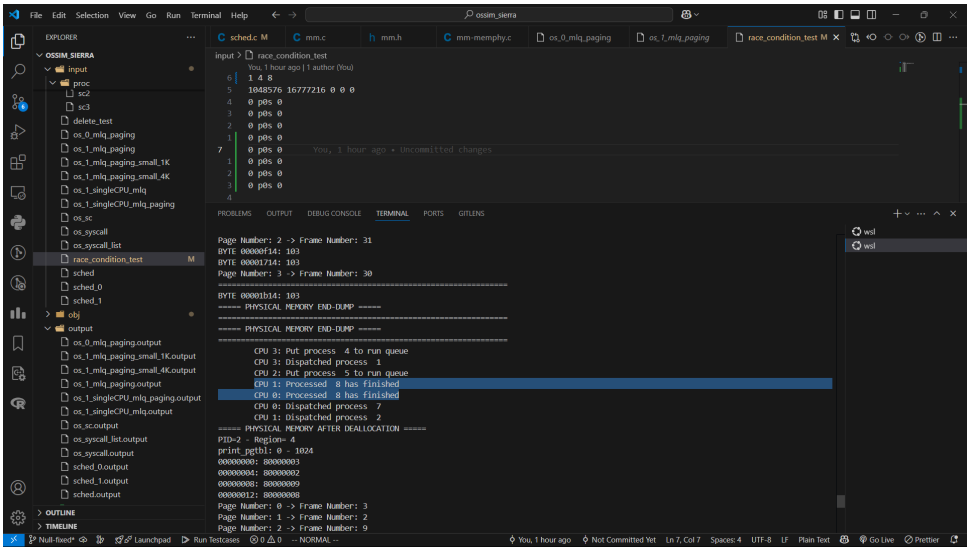


Figure 6: Double free or corruption

```
...
CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  8
BYTE 00000314: 103
BYTE 00000714: 100
BYTE 00000b14: 103
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  8
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=8 - Region= 4
print_pgtbl: 0 - 1024
00000000: 8000001b
00000004: 8000001a
00000008: 8000001f
00000012: 8000001e
Page Number: 0 -> Frame Number: 27
Page Number: 1 -> Frame Number: 26
Page Number: 2 -> Frame Number: 31
BYTE 00000f14: 103
BYTE 00001714: 103
Page Number: 3 -> Frame Number: 30
================================================================
BYTE 00001b14: 103
===== PHYSICAL MEMORY END-DUMP =====
================================================================
===== PHYSICAL MEMORY END-DUMP =====
================================================================
        CPU 3: Put process  4 to run queue
        CPU 3: Dispatched process  1
        CPU 2: Put process  5 to run queue
        CPU 1: Processed  8 has finished
        CPU 0: Processed  8 has finished
                CPU 0: Dispatched process  7
        CPU 1: Dispatched process  2
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=2 - Region= 4
print_pgtbl: 0 - 1024
00000000: 80000003
00000004: 80000002
00000008: 80000009
00000012: 80000008
Page Number: 0 -> Frame Number: 3
Page Number: 1 -> Frame Number: 2
Page Number: 2 -> Frame Number: 9
Page Number: 3 -> Frame Number: 8
================================================================
Time slot  23
        CPU 2: Dispatched process  7
        CPU 3: Processed  1 has finished
        CPU 3: Dispatched process  4
        CPU 0: Processed  7 has finished
        CPU 0: Dispatched process  5
===== PHYSICAL MEMORY AFTER READING =====
read region=3 offset=20 value=103
print_pgtbl: 0 - 1024
00000000: 8000000f
00000004: 8000000e
00000008: 80000015
00000012: 80000014
        CPU 1: Put process  2 to run queue
Page Number: 0 -> Frame Number: 15
Page Number: 1 -> Frame Number: 14
Page Number: 2 -> Frame Number: 21
Page Number: 3 -> Frame Number: 20
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 103
BYTE 00000314: 103
Time slot  24
        CPU 2: Processed  7 has finished
        CPU 1: Dispatched process  2
        CPU 2 stopped
BYTE 00000714: 100
BYTE 00000b14: 103
BYTE 00000f14: 103
BYTE 00001714: 103
BYTE 00001b14: 103
===== PHYSICAL MEMORY END-DUMP =====
================================================================
        CPU 3: Put process  4 to run queue
        CPU 0: Put process  5 to run queue
Time slot  25
        CPU 0: Dispatched process  5
        CPU 3: Dispatched process  4
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=4 - Region= 4
print_pgtbl: 0 - 1024
00000000: 8000000b
00000004: 8000000a
00000008: 80000011
00000012: 80000010
Page Number: 0 -> Frame Number: 11
Page Number: 1 -> Frame Number: 10
Page Number: 2 -> Frame Number: 17
Page Number: 3 -> Frame Number: 16
================================================================
        CPU 1: Processed  2 has finished
        CPU 1 stopped
        CPU 3: Put process  4 to run queue
        CPU 0: Put process  5 to run queue
        CPU 0: Dispatched process  5
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=5 - Region= 4
print_pgtbl: 0 - 1024
00000000: 8000000f
00000004: 8000000e
00000008: 80000015
00000012: 80000014
Time slot  26
        CPU 3: Dispatched process  4
Page Number: 0 -> Frame Number: 15
Page Number: 1 -> Frame Number: 14
Page Number: 2 -> Frame Number: 21
Page Number: 3 -> Frame Number: 20
================================================================
        CPU 3: Processed  4 has finished
        CPU 3 stopped
        CPU 0: Put process  5 to run queue
        CPU 0: Dispatched process  5
Time slot  27
Time slot  28
        CPU 0: Processed  5 has finished
        CPU 0 stopped
double free or corruption (out)
Aborted (core dumped)
```

- Hiện tượng
  - Hai `CPU` (cụ thể là `CPU 1` và `CPU 0`) cùng chọn `process 8` từ hàng đợi (do không có `mutex` bảo vệ hàng đợi tiến trình).
  - Cả hai `CPU` cùng chạy `process 8`.
  - Khi `process` kết thúc, cả hai `CPU` cùng gọi `free(proc)` cho `process 8`.
  - Gây ra lỗi `double free or corruption`.
- Phân loại lỗi đồng bộ:
  - `Race Condition`
  - `Data Inconsistency`
- Giải thích: Hai `CPU` tranh nhau dùng chung một tài nguyên (`queue` và `PCB` của *process 8*) mà không có cơ chế `mutual exclusion`, dẫn đến thao tác không đồng bộ $\longrightarrow$ `race condition` xảy ra. Kết quả là trạng thái dữ liệu (ví dụ: tiến trình đã bị huỷ chưa?) trở nên không nhất quán giữa các `CPU`.

☐ Trường hợp không sử dụng cơ chế đồng bộ (`mutex_lock`) trong `libmem.c` cụ thể là ở hàm `__alloc` và (`mutex_lock`) trong `sched.c` cụ thể là ở các hàm `get_mlq_proc`, `put_mlq_proc`, `add_mlq_proc`.

- Input sử dụng:

Thông tin đầu vào

- Time slide: 1
- Number of CPU: 4
- Number of processes: 8
- RAM size: 1048576 bytes
- SWAP size 1: 16777216 bytes
- SWAP size 2: 0 bytes
- SWAP size 3: 0 bytes
- SWAP size 4: 0 bytes

Process	Arrival Time	Priority
p0s (p1)	0	0
p0s (p2)	0	0
p0s (p3)	0	0
p0s (p4)	0	0
p0s (p5)	0	0
p0s (p6)	0	0
p0s (p7)	0	0
p0s (p8)	0	0

- Ví dụ minh hoạ:

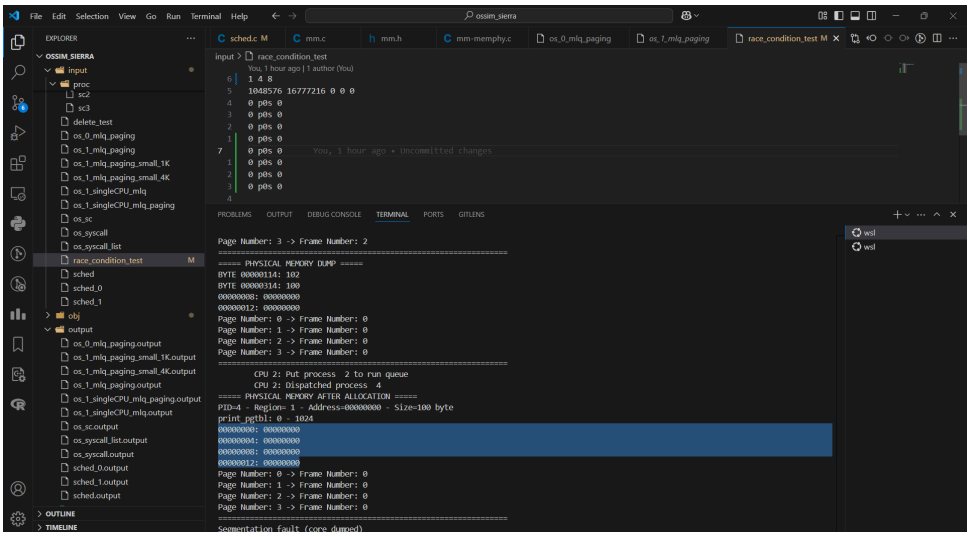


Figure 7: Segment fault

```
...
write region=2 offset=20 value=102
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000001
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 1
Page Number: 3 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 102
BYTE 00000314: 100
00000008: 00000000
00000012: 00000000
Page Number: 0 -> Frame Number: 0
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 0
Page Number: 3 -> Frame Number: 0
================================================================
```

```
================================================================
       CPU 2: Put process  2 to run queue
       CPU 2: Dispatched process  4
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=4 - Region= 1 - Address=00000000 - Size=100 byte
print_pgtbl: 0 - 1024
00000000: 00000000
00000004: 00000000
00000008: 00000000
00000012: 00000000
Page Number: 0 -> Frame Number: 0
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 0
Page Number: 3 -> Frame Number: 0
================================================================
Segmentation fault (core dumped)
```

- Hiện tượng
  - Hai trang trong không gian ảo của một `process` hoặc hai `process` khác nhau cùng được ánh xạ vào cùng một `frame` vật lý (do thiếu lock bảo vệ khi phân phối `frame`).
  - Khi một trong hai trang bị huỷ (`unmap`), `frame` bị trả lại.
  - Trang còn lại vẫn giữ ánh xạ → access vùng bộ nhớ đã bị free gây ra lỗi `Segmentation fault`.
- Phân loại lỗi đồng bộ:

- Race Condition
- Memory corruption / use-after-free
- Data Inconsistency

◦ Giải thích: Tình huống này là race condition trong memory allocator, do hai CPU cùng lúc truy cập và ghi đè thông tin ánh xạ mà không đồng bộ. Kết quả là dữ liệu ánh xạ (page table entry) và frame table bị sai hay inconsistency và cuối cùng là segmentation fault.

4 Giải thích một số output

4.1 Lập lịch (scheduler)

4.1.1 sched (Sử dụng 2 CPU)

☐ **Input**

Thông tin đầu vào

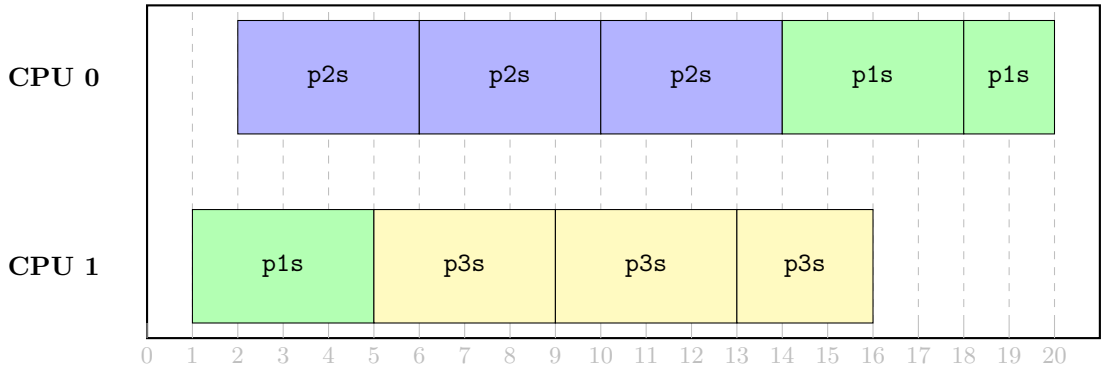
- Time slide: 4
- Number of CPU: 2
- Number of processes: 3

Process	Arrival Time	Priority
p1s	0	0
p2s	1	0
p3s	2	0

☐ **Output**

```
Time slot   0
ld_routine
        Loaded a process at input/proc/p1s, PID: 1 PRIO: 1
Time slot   1
        CPU 1: Dispatched process  1
        Loaded a process at input/proc/p2s, PID: 2 PRIO: 0
Time slot   2
        CPU 0: Dispatched process  2
        Loaded a process at input/proc/p3s, PID: 3 PRIO: 0
Time slot   3
Time slot   4
Time slot   5
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  3
Time slot   6
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   7
Time slot   8
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
Time slot   9
Time slot  10
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  11
Time slot  12
Time slot  13
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
Time slot  14
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  1
Time slot  15
        CPU 1: Processed  3 has finished
Time slot  16
        CPU 1 stopped
Time slot  17
Time slot  18
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  19
Time slot  20
        CPU 0: Processed  1 has finished
        CPU 0 stopped
```

☐ **Biểu đồ Gantt**



☐ **Giải thích**

- Giai đoạn khởi tạo
  - **Time slot 0:** Tiến trình `p1` được nạp vào hệ thống với `PID 1`, độ ưu tiên (`PRIO`) là 1.
  - **Time slot 1:** `CPU 1` bắt đầu thực thi **p1**. Đồng thời, `p2` được nạp với `PID 2`, `PRIO 0`.
  - **Time slot 2:** `CPU 0` bắt đầu thực thi `p2`. `p3` được nạp vào hệ thống với `PID 3`, `PRIO 0`.
- Thực thi và chuyển đổi CPU
  - **Time slot 5:** `CPU 1` chuyển từ tiến trình `p1` sang `p3`.
  - **Time slot 6:** `CPU 0` tiếp tục thực thi `p2`.
  - **Time slot 8:** `CPU 1` tiếp tục thực thi `p3` (thực tế do xử lý đồng thời, điều này đáng lẽ diễn ra ở `Time slot 9`).
  - **Time slot 10**: `CPU 0` tiếp tục thực thi `p2`.
  - **Time slot 13**: `CPU 1` tiếp tục thực thi `p3`.
- Hoàn tất tiến trình
  - **Time slot 14:** `p2` trên `CPU 0` kết thúc. `CPU 0` chuyển sang thực thi `p1`.
  - **Time slot 15:** `p3` trên `CPU 1` kết thúc (do thực thi đồng thời, điều này đáng lẽ diễn ra ở `Time slot 16`).
  - **Time slot 16:** `CPU 1` dừng lại do không còn tác vụ nào.
  - **Time slot 20:** `p1` trên `CPU 0` kết thúc, và `CPU 0` cũng dừng lại.
- Kết quả cuối cùng
  - Các tiến trình `p1`, `p2` và `p3` đều đã hoàn thành.
  - `CPU 1` dừng sớm hơn `CPU 0` do có ít tác vụ hơn.
  - **Tổng thời gian thực thi**: 20 đơn vị thời gian.

4.1.2 sched_0 (Sử dụng CPU 0)

☐ **Input**

Thông tin đầu vào

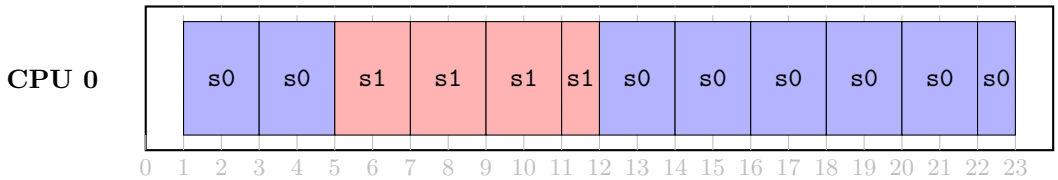
- `Time slide: 2`
- `Number of CPU: 1`
- `Number of processes: 2`

Process	Arrival Time	Priority
s0	0	4
s1	4	0

☐ **Output**

```
Time slot   0
ld_routine
        Loaded a process at input/proc/s0, PID: 1 PRIO: 4
Time slot   1
        CPU 0: Dispatched process  1
Time slot   2
Time slot   3
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot   4
        Loaded a process at input/proc/s1, PID: 2 PRIO: 0
Time slot   5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   6
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
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  1
Time slot  13
Time slot  14
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  15
Time slot  16
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  17
Time slot  18
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  19
Time slot  20
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  21
Time slot  22
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  23
        CPU 0: Processed  1 has finished
        CPU 0 stopped
```

□ Biểu đồ Gantt



□ Giải thích

- Giai đoạn khởi tạo
  - **Time slot 0:** Tiến trình `s0` (`PID 1`, `PRIO 4`) được nạp vào hệ thống.
  - **Time slot 1:** `CPU 0` bắt đầu thực thi tiến trình `s0`.
- Giai đoạn thực thi và lập lịch

- **Time slot 3:** s0 được đưa lại vào hàng đợi và được cấp phát lại.
- **Time slot 4:** Tiến trình s1 (PID 2, PRIO 0) được nạp vào hệ thống.
- **Time slot 5:** s0 được đưa lại vào hàng đợi, CPU 0 chuyển sang thực thi s1.
- **Time slots 7, 9, 11:** s1 được xếp lại vào hàng đợi và cấp phát lại nhiều lần.

◦ Hoàn tất tiến trình

- **Time slot 12:** s1 hoàn tất thực thi, CPU 0 chuyển sang thực thi s0.
- **Time slots 14, 16, 18, 20, 22:** s0 liên tục được đưa lại vào hàng đợi và thực thi.

◦ Kết quả cuối cùng

- **Time slot 23:** s0 hoàn tất thực thi.
- CPU 0 dừng hoạt động vì không còn tiến trình nào trong hệ thống.

4.1.3 sched_1 (Sử dụng CPU 1)

☐ **Input**

Thông tin đầu vào

◦ Time slide: 2

◦ Number of CPU: 1

◦ Number of processes: 4

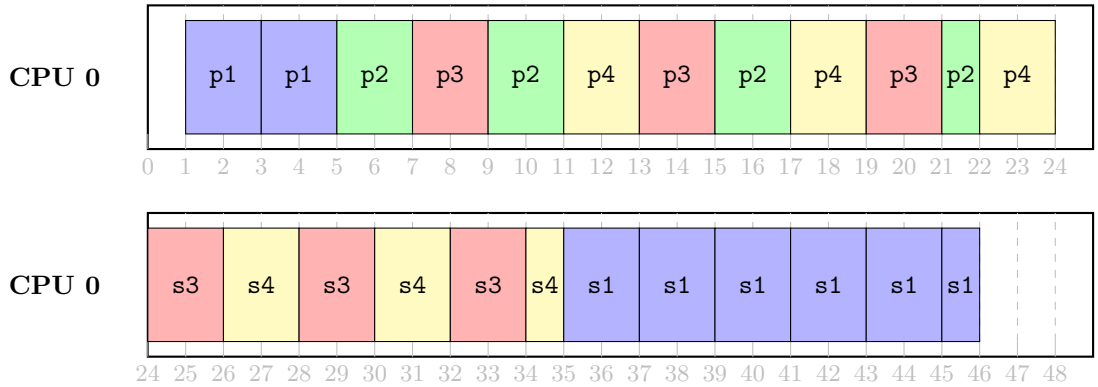
Process	Arrival Time	Priority
s0	0	4
s1	4	0
s2	6	0
s3	7	0

☐ **Output**

```
Time slot   0
ld_routine
        Loaded a process at input/proc/s0, PID: 1 PRIO: 4
Time slot   1
        CPU 0: Dispatched process  1
Time slot   2
Time slot   3
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot   4
        Loaded a process at input/proc/s1, PID: 2 PRIO: 0
Time slot   5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
        Loaded a process at input/proc/s2, PID: 3 PRIO: 0
Time slot   6
Time slot   7
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
        Loaded a process at input/proc/s3, PID: 4 PRIO: 0
Time slot   8
Time slot   9
        CPU 0: Put process  3 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  4
Time slot  12
Time slot  13
        CPU 0: Put process  4 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  2
Time slot  16
Time slot  17
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  4
Time slot  18
Time slot  19
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  3
Time slot  20
Time slot  21
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  2
Time slot  22
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  4
Time slot  23
Time slot  24
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  3
Time slot  25
Time slot  26
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  27
Time slot  28
        CPU 0: Put process  4 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  4
Time slot  31
Time slot  32
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  3
Time slot  33
Time slot  34
        CPU 0: Processed  3 has finished
        CPU 0: Dispatched process  4
Time slot  35
        CPU 0: Processed  4 has finished
        CPU 0: Dispatched process  1
Time slot  36
Time slot  37
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  38
Time slot  39
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  40
Time slot  41
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  42
Time slot  43
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  44
Time slot  45
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  46
        CPU 0: Processed  1 has finished
        CPU 0 stopped
```

☐ **Biểu đồ Gantt**



☐ **Giải thích**

- Giai đoạn khởi tạo
  - **Time slot 0:** Tiến trình `s0` (`PID 1`, `PRIO 4`) được nạp vào hệ thống.
  - **Time slot 1:** `CPU 0` bắt đầu thực thi tiến trình `s0`.
  - **Time slot 4:** Tiến trình `s1` (`PID 2`, `PRIO 0`) được nạp vào hệ thống.
  - **Time slot 5:** Tiến trình `s2` (`PID 3`, PRIO 0) được nạp vào hệ thống.
  - **Time slot 7:** Tiến trình `s3` (`PID 4`, `PRIO 0`) được nạp vào hệ thống.
- Giai đoạn thực thi và lập lịch
  - Các **Time slot 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 24, 26, 28, 30, 32:** Các tiến trình lần lượt được đưa lại vào hàng đợi và được phân phối lại theo vòng tròn (`round-robin`).
  - **Time slot 22:** Tiến trình `s2` hoàn tất thực thi.
  - **Time slot 34:** Tiến trình `s3` hoàn tất thực thi.
  - **Time slot 35:** Tiến trình `s4` hoàn tất thực thi, `CPU 0` chuyển sang tiến trình `s0`.
  - **Time slot 46:** Tiến trình `s0` hoàn tất thực thi.
- Kết quả cuối cùng
  - Tất cả các tiến trình đã hoàn tất thực thi.
  - `CPU 0` dừng lại sau khi xử lý xong toàn bộ tiến trình.

4.2 Quản lý bộ nhớ (Memory Management)

4.2.1 os_0_mlq_paging (Sử dụng 2 CPU)

☐ **Input**

Thông tin đầu vào

- `Time slide: 6`
- `Number of CPU: 2`
- `Number of processes: 4`
- `RAM size: 1048576 bytes`
- `SWAP size 1: 16777216 bytes`
- `SWAP size 2: 0 bytes`
- `SWAP size 3: 0 bytes`
- `SWAP size 4: 0 bytes`

Process	Arrival Time	Priority
p0s (p1)	0	0
p1s_1 (p2)	2	15
p1s_2 (p3)	4	0
p1s_3 (p4)	6	0

Thông tin process

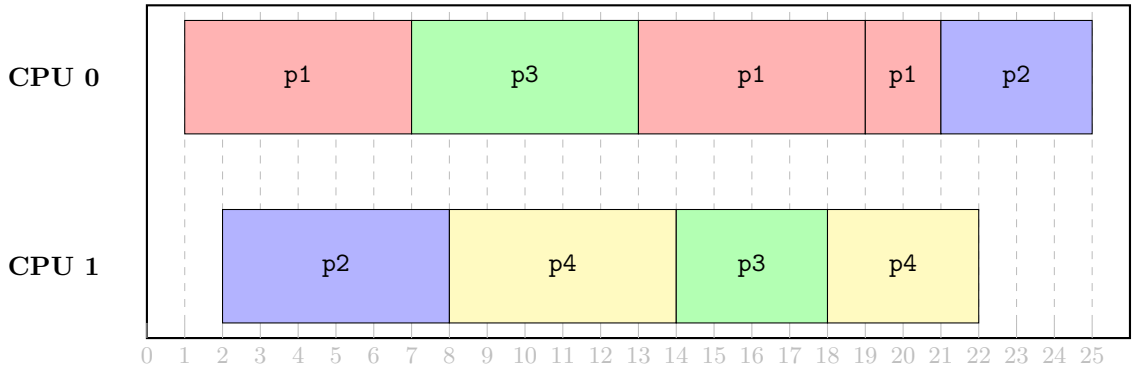
Process	p1	p2	p3	p4
Process priority and Numbers of codes	1 14	1 10	1 10	1 10
Code	calc alloc 300 0 alloc 300 4 free 0 alloc 100 1 write 100 1 20 read 1 20 20 write 102 2 20 read 2 20 20 write 103 3 20 read 3 20 20 calc free 4 calc	calc calc calc calc calc calc calc calc calc calc	calc calc calc calc calc calc calc calc calc calc	calc calc calc calc calc calc calc calc calc calc

□ Output

```
Time slot   0
ld_routine
        Loaded a process at input/proc/p0s, PID: 1 PRIO: 0
Time slot   1
        CPU 0: Dispatched process  1
Time slot   2
        Loaded a process at input/proc/p1s, PID: 2 PRIO: 15
        CPU 1: Dispatched process  2
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=1 - Region= 0 - Address=00000000 - Size=300 byte
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
Time slot   3
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=1 - Region= 4 - Address=0000012c - Size=300 byte
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
Time slot   4
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=1 - Region= 0
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
        Loaded a process at input/proc/p1s, PID: 3 PRIO: 0
Time slot   5
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=1 - Region= 1 - Address=00000000 - Size=100 byte
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
        Loaded a process at input/proc/p1s, PID: 4 PRIO: 0
Time slot   6
===== PHYSICAL MEMORY AFTER WRITING =====
write region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 100
===== PHYSICAL MEMORY END-DUMP =====
================================================================
Time slot   7
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  3
Time slot   8
        CPU 1: Put process  2 to run queue
        CPU 1: Dispatched process  4
Time slot   9
Time slot  10
Time slot  11
Time slot  12
Time slot  13
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  1
===== PHYSICAL MEMORY AFTER READING =====
read region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 100
===== PHYSICAL MEMORY END-DUMP =====
================================================================
Time slot  14
===== PHYSICAL MEMORY AFTER WRITING =====
write region=2 offset=20 value=102
print\_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 102
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  3
===== PHYSICAL MEMORY END-DUMP =====
================================================================
Time slot  15
===== PHYSICAL MEMORY AFTER READING =====
read region=2 offset=20 value=102
print\_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 102
===== PHYSICAL MEMORY END-DUMP =====
================================================================
```

```
Time slot  16
===== PHYSICAL MEMORY AFTER WRITING =====
write region=3 offset=20 value=103
print\_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 103
===== PHYSICAL MEMORY END-DUMP =====
================================================================
Time slot  17
===== PHYSICAL MEMORY AFTER READING =====
read region=3 offset=20 value=103
print\_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 103
===== PHYSICAL MEMORY END-DUMP =====
================================================================
Time slot  18
        CPU 1: Processed  3 has finished
        CPU 1: Dispatched process  4
Time slot  19
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=1 - Region= 4
print\_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
Time slot  20
Time slot  21
        CPU 0: Processed  1 has finished
        CPU 0: Dispatched process  2
        CPU 1: Processed  4 has finished
        CPU 1 stopped
Time slot  22
Time slot  23
Time slot  24
        CPU 0: Processed  2 has finished
        CPU 0 stopped
Time slot  25
```

☐ Biểu đồ gantt



☐ Giải thích

- Giai đoạn khởi tạo
  - **Time slot 0:** Tiến trình `p1` được nạp vào hệ thống với `PID=1`, độ ưu tiên `PRIO=0`.
  - **Time slot 1:** `CPU 0` bắt đầu thực thi `p1`.
  - **Time slot 2:** Tiến trình `p2` được nạp với `PID=2`, `PRIO=15`; `CPU 1` bắt đầu thực thi `p2`.
  - **Time slot 4:** Tiến trình `p3` được nạp với `PID=3`, `PRIO=0`.
  - **Time slot 5:** Tiến trình `p4` được nạp với `PID=4`, `PRIO=0`.
- Thực thi và chuyển đổi CPU
  - **Time slot 7:** `CPU 0` chuyển từ `p1` sang `p3`.
  - **Time slot 8:** `CPU 1` chuyển từ `p2` sang `p4`.
  - **Time slot 13:** `CPU 0` chuyển từ `p3` sang `p1`.
- Quản lý bộ nhớ
  - **Time slot 2:** Cấp phát cho `PID=1` vùng 0 (địa chỉ `0x00000000`, kích thước 300 byte).
  - **Time slot 3:** Cấp phát cho `PID=1` vùng 4 (địa chỉ `0x0000012C`, kích thước 300 byte).
  - **Time slot 4:** Giải phóng cho `PID=1` vùng 0, kích thước 300 byte.
  - **Time slot 5:** Cấp phát cho `PID=1` vùng 1 (địa chỉ `0x00000000`, kích thước 100 byte).
  - **Time slot 19:** Giải phóng cho `PID=1` vùng 4, kích thước 300 byte.
- Truy xuất đọc ghi cho vùng nhớ
  - **Time slot 6:** Ghi (write) lên vùng 1, offset=20, giá trị=100. Xuất ra `BYTE 00000114: 100`.
  - **Time slot 13:** Đọc (read) vùng 1, offset=20, thu được 100. Xuất ra `BYTE 00000114: 100`.

- **Time slot 14:** Ghi vùng 2, offset=20, giá trị=102. Xuất ra `BYTE 00000114: 102`.
- **Time slot 15:** Đọc vùng 2, offset=20, thu được 102. Xuất ra `BYTE 00000114: 102`.
- **Time slot 16:** Ghi vùng 3, offset=20, giá trị=103. Xuất ra `BYTE 00000114: 103`.
- **Time slot 17:** Đọc vùng 3, offset=20, thu được 103. Xuất ra `BYTE 00000114: 103`.

◦ Hoàn tất tiến trình

- **Time slot 18:** `p3` trên `CPU 1` hoàn thành; `CPU 1` tiếp tục thực thi `p4`.
- **Time slot 19:** `p1` trên `CPU 0` hoàn thành; `CPU 0` chuyển sang `p2`. Đồng thời `p4` trên `CPU 1` hoàn thành và `CPU 1` dừng.
- **Time slot 24:** `p2` trên `CPU 0` hoàn thành và `CPU 0` dừng.

◦ Kết quả cuối cùng

- Các tiến trình `p1`, `p2`, `p3` và `p4` đều đã hoàn thành.
- `CPU 1` dừng sớm hơn `CPU 0` do ít tác vụ hơn.
- **Tổng thời gian thực thi**: 24 đơn vị thời gian.

4.2.2 os_1_mlq_paging_small_1K (Sử dụng 4 CPU)

☐ **Input**

Thông tin đầu vào

◦ `Time slide: 2`

◦ `Number of CPU: 4`

◦ `Number of processes: 8`

◦ `RAM size: 2048 bytes`

◦ `SWAP size 1: 16777216 bytes`

◦ `SWAP size 2: 0 bytes`

◦ `SWAP size 3: 0 bytes`

◦ `SWAP size 4: 0 bytes`

Process	Arrival Time	Priority
p0s (p1)	1	130
s3 (p2)	2	39
m1s (p3)	4	15
s2 (p4)	6	120
m0s (p5)	7	120
p1s (p6)	9	15
s0 (p7)	11	38
s1 (p8)	16	0

Thông tin process

Process	p1	p2	p3	p4
Process priority and Numbers of codes	1 14	7 11	1 6	20 12
Code	calc alloc 300 0 alloc 300 4 free 0 alloc 100 1 write 100 1 20 read 1 20 20 write 102 2 20 read 2 20 20 write 103 3 20 read 3 20 20 calc free 4 calc	calc calc calc calc calc calc calc calc calc calc calc	alloc 300 0 alloc 100 1 free 0 alloc 100 2 free 2 free 1	calc calc calc calc calc calc calc calc calc calc calc calc calc

Thông tin process

Process	p5	p6	p7	p8
Process priority and Numbers of codes	1 6	1 10	12 15	20 7
Code	alloc 300 0 alloc 100 1 free 0 alloc 100 2 write 102 1 20 write 1 2 1000	calc calc calc calc calc calc calc calc calc calc	calc calc calc calc calc calc calc calc calc calc calc calc calc calc calc	calc calc calc calc calc calc calc

☐ Output

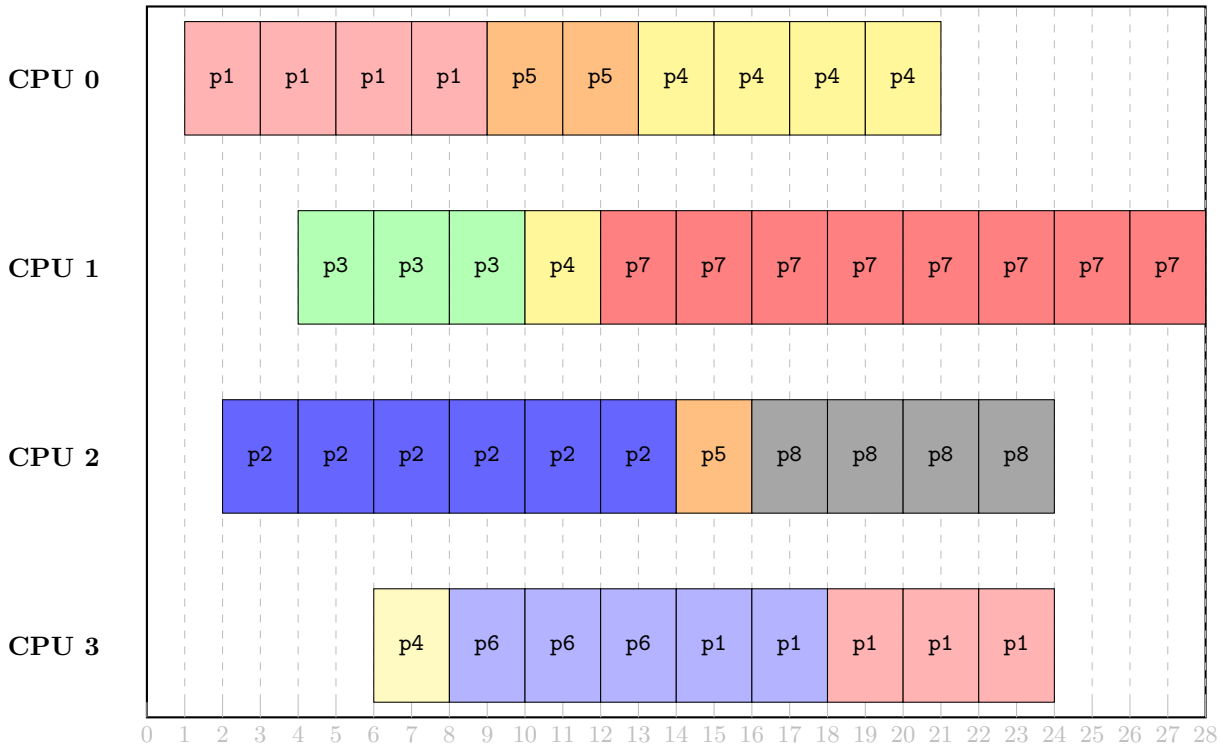
```
Time slot   0
ld_routine
Time slot   1
        Loaded a process at input/proc/p0s, PID: 1 PRIO: 130
        CPU 0: Dispatched process  1
Time slot   2
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=1 - Region= 0 - Address=00000000 - Size=300 byte
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
        Loaded a process at input/proc/s3, PID: 2 PRIO: 39
        CPU 2: Dispatched process  2
Time slot   3
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=1 - Region= 4 - Address=0000012c - Size=300 byte
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
Time slot   4
        Loaded a process at input/proc/m1s, PID: 3 PRIO: 15
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=1 - Region= 0
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=3 - Region= 0 - Address=00000000 - Size=300 byte
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
        CPU 2: Put process  2 to run queue
        CPU 2: Dispatched process  2
Time slot   5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=1 - Region= 1 - Address=00000000 - Size=100 byte
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=3 - Region= 1 - Address=0000012c - Size=100 byte
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
================================================================
        Loaded a process at input/proc/s2, PID: 4 PRIO: 120
        CPU 1: Put process  3 to run queue
Time slot   6
        CPU 1: Dispatched process  3
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=3 - Region= 0
===== PHYSICAL MEMORY AFTER WRITING =====
write region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 100

print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
================================================================
===== PHYSICAL MEMORY END-DUMP =====
================================================================
        CPU 3: Dispatched process  4
        CPU 2: Put process  2 to run queue
        CPU 2: Dispatched process  2
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=3 - Region= 2 - Address=00000000 - Size=100 byte
print_pgtbl: 0 - 512
    CPU 0: Put process  1 to run queue
===== PHYSICAL MEMORY AFTER READING =====
read region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 100

00000000: 80000005
00000004: 80000004
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
================================================================
Time slot   7
```

```
===== PHYSICAL MEMORY END-DUMP =====
        CPU 0: Dispatched process  1
================================================================
        Loaded a process at input/proc/m0s, PID: 5 PRIO: 120
Time slot   8
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
===== PHYSICAL MEMORY AFTER WRITING =====
write region=2 offset=20 value=102
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=3 - Region= 2
Page Number: 3 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 102
===== PHYSICAL MEMORY END-DUMP =====
================================================================
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
================================================================
        Loaded a process at input/proc/p1s, PID: 6 PRIO: 15
        CPU 3: Put process  4 to run queue
        CPU 3: Dispatched process  6
        CPU 2: Put process  2 to run queue
        CPU 2: Dispatched process  2
Time slot   9
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  5
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=3 - Region= 1
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Page Number: 0 -> Frame Number: 5
Page Number: 1 -> Frame Number: 4
================================================================
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=5 - Region= 0 - Address=00000000 - Size=300 byte
print_pgtbl: 0 - 512
00000000: 80000007
00000004: 80000006
Page Number: 0 -> Frame Number: 7
Page Number: 1 -> Frame Number: 6
================================================================
Time slot  10
        CPU 1: Processed  3 has finished
        CPU 1: Dispatched process  4
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=5 - Region= 1 - Address=0000012c - Size=100 byte
print_pgtbl: 0 - 512
00000000: 80000007
00000004: 80000006
Page Number: 0 -> Frame Number: 7
Page Number: 1 -> Frame Number: 6
================================================================
        CPU 2: Put process  2 to run queue
        CPU 3: Put process  6 to run queue
        CPU 2: Dispatched process  2
        CPU 3: Dispatched process  6
Time slot  11
        CPU 0: Put process  5 to run queue
        CPU 0: Dispatched process  5
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=5 - Region= 0
print_pgtbl: 0 - 512
00000000: 80000007
00000004: 80000006
Page Number: 0 -> Frame Number: 7
Page Number: 1 -> Frame Number: 6
================================================================
        Loaded a process at input/proc/s0, PID: 7 PRIO: 38
        CPU 1: Put process  4 to run queue
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=5 - Region= 2 - Address=00000000 - Size=100 byte
print_pgtbl: 0 - 512
00000000: 80000007
00000004: 80000006
Page Number: 0 -> Frame Number: 7
Page Number: 1 -> Frame Number: 6
================================================================
Time slot  12
        CPU 1: Dispatched process  7
        CPU 3: Put process  6 to run queue
        CPU 3: Dispatched process  6
        CPU 2: Put process  2 to run queue
        CPU 2: Dispatched process  2
Time slot  13
        CPU 0: Put process  5 to run queue
        CPU 0: Dispatched process  4
        CPU 2: Processed  2 has finished
Time slot  14
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
        CPU 2: Dispatched process  5
===== PHYSICAL MEMORY AFTER WRITING =====
write region=1 offset=20 value=102
print_pgtbl: 0 - 512
00000000: 80000007
00000004: 80000006
Page Number: 0 -> Frame Number: 7
Page Number: 1 -> Frame Number: 6
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 102
BYTE 00000640: 102
===== PHYSICAL MEMORY END-DUMP =====
================================================================
        CPU 3: Put process  6 to run queue
        CPU 3: Dispatched process  6
Time slot  15
===== PHYSICAL MEMORY AFTER WRITING =====
write region=2 offset=1000 value=1
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
print_pgtbl: 0 - 512
00000000: c0000000
00000004: 80000006
Page Number: 0 -> Frame Number: 0
Page Number: 1 -> Frame Number: 6
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 102
BYTE 00000640: 102
BYTE 000007e8: 1
===== PHYSICAL MEMORY END-DUMP =====
================================================================
        Loaded a process at input/proc/s1, PID: 8 PRIO: 0
        CPU 2: Processed  5 has finished
Time slot  16
        CPU 2: Dispatched process  8
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
        CPU 3: Put process  6 to run queue
        CPU 3: Dispatched process  6
Time slot  17
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
Time slot  18
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
        CPU 3: Processed  6 has finished
        CPU 3: Dispatched process  1
===== PHYSICAL MEMORY AFTER READING =====
read region=2 offset=20 value=102
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 102
        CPU 0: Put process  4 to run queue
Time slot  19
BYTE 00000640: 102
BYTE 000007e8: 1
===== PHYSICAL MEMORY END-DUMP =====
================================================================
        CPU 0: Dispatched process  4
===== PHYSICAL MEMORY AFTER WRITING =====
write region=3 offset=20 value=103
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 103
Time slot  20
        CPU 1: Put process  7 to run queue
BYTE 00000640: 102
BYTE 000007e8: 1
===== PHYSICAL MEMORY END-DUMP =====
================================================================
        CPU 1: Dispatched process  7
        CPU 3: Put process  1 to run queue
        CPU 0: Processed  4 has finished
        CPU 3: Dispatched process  1
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
Time slot  21
===== PHYSICAL MEMORY AFTER READING =====
read region=3 offset=20 value=103
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
================================================================
        CPU 0 stopped
===== PHYSICAL MEMORY DUMP =====
BYTE 00000114: 103
BYTE 00000640: 102
BYTE 000007e8: 1
===== PHYSICAL MEMORY END-DUMP =====
================================================================
Time slot  22
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
        CPU 3: Put process  1 to run queue
        CPU 3: Dispatched process  1
```

```
Time slot  23
        CPU 2: Processed  8 has finished
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=1 - Region= 4
        CPU 2 stopped
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
Page Number: 2 -> Frame Number: 3
Page Number: 3 -> Frame Number: 2
```

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

□ Biểu đồ Gantt



□ Giải thích

- Giai đoạn khởi tạo
  - **Time slot 1:** Tiến trình `p1` được nạp vào với `PID=1`, `PRIO=130`, chạy trên `CPU 0`.
  - **Time slot 2:** Tiến trình `p2` được nạp vào với `PID=2`, `PRIO=39`, chạy trên `CPU 2`.
  - **Time slot 4:** Tiến trình `p3` được nạp vào với `PID=3`, `PRIO=15`, chạy trên `CPU 2`.
  - **Time slot 6:** Tiến trình `p4` được nạp vào với `PID=4`, `PRIO=120`.
  - **Time slot 8:** Tiến trình `p6` được nạp vào với `PID=6`, `PRIO=15`.
  - **Time slot 9:** Tiến trình `p5` được nạp vào với `PID=5`, `PRIO=120`.
  - **Time slot 12:** Tiến trình `p7` được nạp vào với `PID=7`, `PRIO=38`.
  - **Time slot 16:** Tiến trình `p8` được nạp vào với `PID=8`, `PRIO=0`.
- Thực thi và chuyển đổi `CPU`
  - Tiến trình `PID=1` liên tục được đưa vào hàng đợi và lại được cấp CPU nhiều lần (**Time slot 1, 3, 5, 7, 18, 20, 22**).
  - Tương tự, tiến trình `PID=2` liên tục được đưa vào hàng đợi và lại được cấp CPU nhiều lần (**Time slot 2, 4, 6, 8, 10, 12, 14**).
  - Tương tự, tiến trình `PID=3` liên tục được đưa vào hàng đợi và lại được cấp CPU nhiều lần (**Time slot 4, 6, 8**).
  - Tương tự, tiến trình `PID=4` liên tục được đưa vào hàng đợi và lại được cấp CPU nhiều lần (**Time slot 6, 10, 13**).

- Tương tự, tiến trình PID=5 liên tục được đưa vào hàng đợi và lại được cấp CPU nhiều lần (**Time slot 9, 11, 14**).
- Tương tự, tiến trình PID=6 liên tục được đưa vào hàng đợi và lại được cấp CPU nhiều lần (**Time slot 8, 10, 12, 14, 16**).
- Tương tự, tiến trình PID=7 liên tục được đưa vào hàng đợi và lại được cấp CPU nhiều lần (**Time slot 12, 14, 16, 18, 20, 22, 24, 26**).
- Cuối cùng, tiến trình PID=8 liên tục được đưa vào hàng đợi và lại được cấp CPU nhiều lần (**Time slot 16, 18, 20, 22**).
- Các CPU xử lý song song với các tiến trình khác nhau, thể hiện tính đa luồng và chia tải.

◦ Quản lý bộ nhớ

- **Time slot 2:** Cấp phát cho PID=1 vùng 0, địa chỉ 0x00000000, kích thước 300 byte.
- **Time slot 3:** Cấp phát cho PID=1 vùng 4, địa chỉ 0x0000012c, kích thước 300 byte; cấp phát cho PID=3 vùng 0, địa chỉ 0x00000000, kích thước 300 byte.
- **Time slot 4:** Giải phóng cho PID=1 vùng 0. kích thước 300 byte.
- **Time slot 5:** Cấp phát cho PID=1 vùng 1, địa chỉ 0x00000000, kích thước 100 byte; Cấp phát cho PID=3 vùng 1, địa chỉ 0x0000012c, kích thước 100 byte.
- **Time slot 6:** Giải phóng cho PID=3 vùng 0, kích thước 300 byte; Cấp phát cho PID=3 vùng 2, kích thước 100 byte.
- **Time slot 8:** Giải phóng cho PID=3 vùng 1, kích thước 100 byte.
- **Time slot 9:** Giải phóng cho PID=3 vùng 2, kích thước 100 byte; Cấp phát cho PID=5 vùng 0 (địa chỉ 0x00000000, kích thước 300 byte).
- **Time slot 10:** Cấp phát cho PID=5 vùng 1 (địa chỉ 0x0000012C, kích thước 100 byte).
- **Time slot 11:** Giải phóng cho PID=5 vùng 0, kích thước 300 byte; Cấp phát cho PID=5 vùng 2 (địa chỉ 0x00000000, kích thước 100 byte).
- **Time slot 23:** Giải phóng cho PID=1 vùng 4.

◦ Truy xuất đọc ghi cho vùng nhớ

- **Time slot 6:** PID1 Ghi vào vùng 1, offset=20, giá trị = 100. Xuất ra BYTE 00000114: 100; PID=1 đọc vùng 1, offset=20 được giá trị 100. Xuất ra BYTE 00000114: 100.
- **Time slot 8:** PID=1 ghi vào vùng 2, offset=20, giá trị = 102. Xuất ra BYTE 00000114: 102.
- **Time slot 14:** PID=5 ghi vào vùng 1, offset=20, giá trị = 102. Xuất ra BYTE 00000640: 102.
- **Time slot 15:** PID=5 ghi vào vùng 2, offset=1000, giá trị = 1 (có swap page). Xuất ra BYTE 000007E8: 1.
- **Time slot 18:** PID=1 đọc vùng 2, offset=20 được giá trị 102.
- **Time slot 19:** PID=1 ghi vào vùng 3, offset=20, giá trị = 103. Xuất ra BYTE 00000114: 103.
- **Time slot 21:** PID=1 đọc vùng 3, offset=20 được giá trị 103. Xuất ra BYTE 00000114: 103. .

◦ Hoàn tất tiến trình

- **Time slot 10:** PID=3 thực thi và kết thúc.
- **Time slot 13:** PID=2 thực thi và kết thúc.
- **Time slot 16:** PID=5 thực thi và kết thúc.
- **Time slot 18:** PID=6 thực thi và kết thúc.
- **Time slot 20:** PID=4 thực thi và kết thúc.
- **Time slot 23:** PID=8 thực thi và kết thúc.
- **Time slot 23:** PID=1 thực thi và kết thúc.
- **Time slot 27:** PID=7 thực thi và kết thúc.

◦ Kết quả cuối cùng

- Tất cả tiến trình đều được hoàn tất trong vòng 28 time slot.
- Các vùng nhớ được ghi và đọc đúng, thể hiện phân trang và ánh xạ bộ nhớ chính xác.
- **Tổng thời gian thực thi**: 28 đơn vị thời gian.

4.3 Lời gọi hệ thống (System Call)

4.3.1 os_syscall

☐ **Input**

Thông tin đầu vào

- Time slide: 2
- Number of CPU: 1
- Number of processes: 1
- RAM size: 2048 bytes
- SWAP size 1: 16777216 bytes
- SWAP size 2: 0 bytes
- SWAP size 3: 0 bytes
- SWAP size 4: 0 bytes

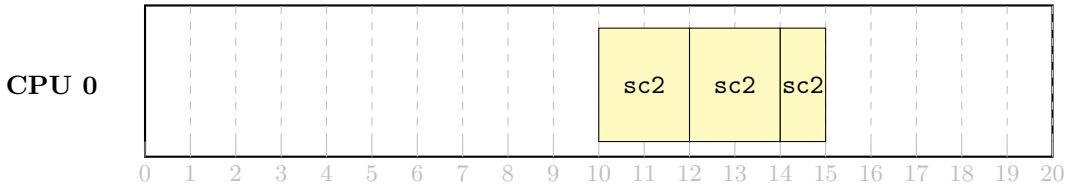
Process	Arrival Time	Priority
sc2 (p1)	9	15

Process	sc2 (p1)
Process priority and Numbers of codes	20 5
Code	alloc 100 1 write 80 1 0 write 48 1 1 write -1 1 2 syscall 101 1

☐ **Output**

```
Time slot   0
ld_routine
Time slot   1
Time slot   2
Time slot   3
Time slot   4
Time slot   5
Time slot   6
Time slot   7
Time slot   8
Time slot   9
        Loaded a process at input/proc/sc2, PID: 1 PRIO: 15
Time slot  10
        CPU 0: Dispatched process  1
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=1 - Region= 1 - Address=00000000 - Size=100 byte
print_pgtbl: 0 - 256
00000000: 80000000
Page Number: 0 -> Frame Number: 0
================================================================
Time slot  11
===== PHYSICAL MEMORY AFTER WRITING =====
write region=1 offset=0 value=80
print_pgtbl: 0 - 256
00000000: 80000000
Page Number: 0 -> Frame Number: 0
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000000: 80
===== PHYSICAL MEMORY END-DUMP =====
================================================================
Time slot  12
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
===== PHYSICAL MEMORY AFTER WRITING =====
write region=1 offset=1 value=48
print_pgtbl: 0 - 256
00000000: 80000000
Page Number: 0 -> Frame Number: 0
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000000: 80
BYTE 00000001: 48
===== PHYSICAL MEMORY END-DUMP =====
================================================================
Time slot  13
===== PHYSICAL MEMORY AFTER WRITING =====
write region=1 offset=2 value=-1
print_pgtbl: 0 - 256
00000000: 80000000
Page Number: 0 -> Frame Number: 0
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000000: 80
BYTE 00000001: 48
BYTE 00000002: -1
===== PHYSICAL MEMORY END-DUMP =====
================================================================
Time slot  14
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
===== PHYSICAL MEMORY AFTER READING =====
read region=1 offset=0 value=80
print_pgtbl: 0 - 256
00000000: 80000000
Page Number: 0 -> Frame Number: 0
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000000: 80
BYTE 00000001: 48
BYTE 00000002: -1
===== PHYSICAL MEMORY END-DUMP =====
================================================================
===== PHYSICAL MEMORY AFTER READING =====
read region=1 offset=1 value=48
print_pgtbl: 0 - 256
00000000: 80000000
Page Number: 0 -> Frame Number: 0
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000000: 80
BYTE 00000001: 48
BYTE 00000002: -1
===== PHYSICAL MEMORY END-DUMP =====
================================================================
===== PHYSICAL MEMORY AFTER READING =====
read region=1 offset=2 value=-1
print_pgtbl: 0 - 256
00000000: 80000000
Page Number: 0 -> Frame Number: 0
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000000: 80
BYTE 00000001: 48
BYTE 00000002: -1
===== PHYSICAL MEMORY END-DUMP =====
================================================================
The procname retrieved from memregionid 1 is "P0"
Time slot  15
        CPU 0: Processed  1 has finished
        CPU 0 stopped
```

☐ **Biểu đồ Gantt**



□ **Giải thích**

- Giai đoạn khởi tạo
  - **Time slot 9:** Tiến trình `sc2` được nạp vào hệ thống với `PID=1`, độ ưu tiên `PRIO=15`.
  - **Time slot 10:** `CPU 0` bắt đầu thực thi `sc2`.
- Thực thi và lập lịch
  - **Time slot 12:** `sc2` bị preempt, được đưa lại vào hàng đợi và lập lịch chạy lần nữa.
  - **Time slot 14:** `sc2` tiếp tục được cấp phát CPU.
- Cấp phát bộ nhớ
  - **Time slot 10:** Cấp phát cho `PID=1` vùng 1 (địa chỉ `0x00000000`, kích thước 100 byte).
- Truy xuất đọc ghi cho vùng nhớ
  - **Time slot 11:** Ghi (write) vùng 1, `offset=0`, giá trị `80`. Xuất ra `BYTE 00000000: 80`.
  - **Time slot 12:** Ghi vùng 1, `offset=1`, giá trị `48`. Xuất ra `BYTE 00000001: 48`.
  - **Time slot 13:** Ghi vùng 1, `offset=2`, giá trị `-1`. Xuất ra `BYTE 00000002: -1`.
- Hoàn tất tiến trình
  - **Time slot 15:** `CPU 0` dừng sau khi hoàn thành thực thi tiến trình `sc2`.
- Sự kiện đặc biệt
  - **Time slot 14:** Tiến hành system call `killall`.

5 Kết luận

Qua bài tập lớn này, nhóm đã mô phỏng được các thành phần cốt lõi của một hệ điều hành tối giản:

- Bộ định lịch `MLQ` (`Multilevel Queue`) – quản lý nhóm tiến trình theo mức ưu tiên, bảo đảm phân chia thời gian CPU công bằng và đáp ứng yêu cầu thời gian thực.
- Quản lý bộ nhớ – hiện thực cơ chế cấp phát/thu hồi vùng nhớ ảo, ánh xạ trang (`paging`) sang khung trang vật lý, xử lý lỗi trang và hoán đổi trang cơ bản.
- Hệ thống lời gọi hệ thống – bổ sung các `syscall` còn thiếu, đặc biệt lệnh `killall` để truy xuất bảng `PCB`, xác định tiến trình theo tên và kết thúc chúng an toàn.