TRƯỜNG ĐẠI HỌC BÁCH KHOA TP. HỒ CHÍ MINH

BỘ MÔN TOÁN ỨNG DỤNG

-----📖-----

Bài tập lớn

HỆ ĐIỀU HÀNH

Lớp L07

Giáo viên hướng dẫn: Nguyễn Thanh Quân

Sinh viên thực hiện:

Phan Phạm Thi **MSSV: 2114857**

Phạm Hồng My Sa **MSSV: 2112173**

TP. HỒ CHÍ MINH, tháng 5/2023

MỤC LỤC

SƠ LƯỢC VỀ BÀI BÁO CÁO

Bài báo cáo gồm 3 phần chính là:

1. Định thời
2. Quản lý bộ nhớ
3. Kết luận

Trong đó, ở phần 1. Định thời sẽ gồm 2 phần nhỏ dành cho 2 part. Nhóm dùng file input sched_0 để chạy thử cho cả hai part.

Phần 2. Quản lý bộ nhớ chỉ xét part 2 với file input là os_1_mlq_paging.

Phần 3. Kết luận là về mà một hệ điều hành đơn giản hoàn chỉnh mà nhóm đã hiện thực.

PHẦN 1: ĐỊNH THỜI

I. Trả lời câu hỏi.

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

Answer: *Hàng đợi ưu tiên (Priority Queue):* là một phần mở rộng của Queue. Đây là một cấu trúc dữ liệu đặc biệt được sử dụng để lưu trữ các phần tử. Trong đó, mỗi phần tử có một độ ưu tiên nhất định và thứ tự truy xuất phần tử sẽ phụ thuộc vào độ ưu tiên của nó, tức là một phần tử có độ ưu tiên cao hơn sẽ được xử lí trước một phần tử có độ ưu tiên thấp hơn; nếu hai phần tử có cùng độ ưu tiên, chúng sẽ được xử lí lần lượt theo thứ tự của chúng trong hàng đợi, tức là áp dụng quy tắc first-in-first-out như bình thường: phần tử vào hàng đợi trước sẽ là phần tử đầu tiên bị đưa ra trước. Thông thường, có hai loại hàng đợi ưu tiên: hàng đợi ưu tiên có thứ tự là giá trị tăng dần (Ascending Order Priority Queue) và hàng đợi ưu tiên có thứ tự là giá trị giảm dần (Descending Order Priority Queue).

Ưu điểm:

- Tốc độ xử lý nhanh hơn: Vì hàng đợi ưu tiên xử lý các nhiệm vụ có độ ưu tiên cao hơn. Vì vậy, chúng ta sẽ dựa vào độ quan trọng của mỗi nhiệm vụ mà gán priority phù hợp.
- Phân bố tài nguyên hợp lí: Vì hàng đợi ưu tiên xử lý các nhiệm vụ có độ ưu tiên cao trước nên khi đó, nó sẽ phân bố tài nguyên nhiều hơn cho các nhiệm vụ đó. Điều này tăng hiệu suất của hệ thống một cách đáng kể.
- Sử dụng bộ nhớ hiệu quả: Hàng đợi ưu tiên sử dụng bộ nhớ hiệu quả vì nó chỉ lưu trữ các nhiệm vụ có độ ưu tiên cao nhất. Điều này có nghĩa là kích thước của cấu trúc dữ liệu được giảm thiểu, điều này rất quan trọng trong các môi trường có hạn tài nguyên.
- Dễ triển khai: Đây là một trong những lựa chọn phổ biến nhất của các thuật toán định thời vì nó dễ triển khai. Bên cạnh đó, vì ưu điểm lớn nên đây là cấu trúc dữ liệu quan trọng được ứng dụng đa dạng để giải quyết rất nhiều vấn đề khác nhau trong cuộc sống.
- Thứ tự, tùy chỉnh theo nhu cầu: Các nhiệm vụ khác nhau sẽ được gán mức độ ưu tiên khác nhau, từ đó cho phép có thể điều khiển một cách chi tiết và cụ thể hơn về hệ thống.

II. Cơ sở lý thuyết

1. Multilevel queue scheduling

Đây là một loại thuật toán lập định thời CPU chia hàng đợi sẵn sàng (ready queue) thành nhiều cấp độ hoặc bậc, mỗi cấp có mức độ ưu tiên khác nhau. Sau đó, các quy trình được gán cho cấp độ phù hợp dựa trên các đặc điểm của chúng, chẳng hạn như mức độ ưu tiên của quy trình, loại chương trình, yêu cầu về bộ nhớ và mức độ sử dụng CPU.

Đặt vấn đề: Có hai loại quy trình chính trong hệ thống máy tính: quy trình tương tác (interactive processes) và quy trình nền (background processes). Các quy trình tương tác cần được thực hiện nhanh chóng vì chúng đang được sử dụng bởi user, trong khi các quy trình nền có thể đợi vì chúng không quan trọng. Trong multilevel queue scheduling, có các hàng đợi khác nhau cho các foreground (interactive) và background (batch) processes.

Nhìn chung, multilevel queue scheduling giúp hệ thống máy tính quản lý các tác vụ của nó hiệu quả hơn bằng cách ưu tiên chúng dựa trên tầm quan trọng của chúng và sử dụng các kỹ thuật định thời khác nhau cho các loại tác vụ khác nhau.

2. Đặc điểm của định thời Multi-Level Queue (MLQ)

- *Nhiều hàng đợi:* Trong MLQ, các quy trình được chia thành nhiều hàng đợi dựa trên mức độ ưu tiên của chúng, với mỗi hàng đợi có một mức độ ưu tiên khác nhau. Các quy trình có mức độ ưu tiên cao hơn được đặt trong hàng đợi có mức độ ưu tiên cao hơn, trong khi các quy trình có mức độ ưu tiên thấp hơn được đặt trong hàng đợi có mức độ ưu tiên thấp hơn.
- *Các ưu tiên được gán (Priorities assigned):* Các priorities được gán cho các quy trình dựa trên loại, đặc điểm và tầm quan trọng của chúng. Ví dụ: các quy trình tương tác như đầu vào/đầu ra của người dùng có thể có mức độ ưu tiên cao hơn các quy trình hàng loạt như sao lưu tệp.
- *Quyền ưu tiên (Preemption):* có nghĩa là quy trình có mức độ ưu tiên cao hơn có thể ưu tiên 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 ưu tiên cao được thực hiện kịp thời.
- *Thuật toán định thời (Scheduling algorithm):* Các thuật toán định thời khác nhau có thể được sử dụng cho mỗi hàng đợi, tùy thuộc vào yêu cầu của các tiến trình trong hàng đợi đó. Ví dụ, định

thời Round Robin có thể được sử dụng cho các quy trình tương tác, trong khi định thời FCFS có thể được sử dụng cho các quy trình hàng loạt.

- *Cơ chế phản hồi:* Cơ chế phản hồi có thể được triển khai để điều chỉnh mức độ ưu tiên của quy trình dựa trên hành vi của nó theo thời gian. Ví dụ: nếu một quy trình tương tác đã chờ đợi trong hàng đợi có mức độ ưu tiên thấp hơn trong một thời gian dài, thì mức độ ưu tiên của nó có thể được tăng lên để đảm bảo nó được thực thi kịp thời.
- *Phân bổ hiệu quả thời gian của CPU:* MLQ scheduling 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.
- *Công bằng:* Lập lịch MLQ cung cấp phân bổ thời gian CPU hợp lý cho các loại quy trình khác nhau, dựa trên mức độ ưu tiên và yêu cầu của chúng.
- *Có thể tùy chỉnh (Customizable):* Lập lịch trình MLQ có thể được tùy chỉnh để đáp ứng các yêu cầu cụ thể của các loại quy trình khác nhau.

III. Hiện thực

1. Quy tắc vận hành của MLQ trong bài

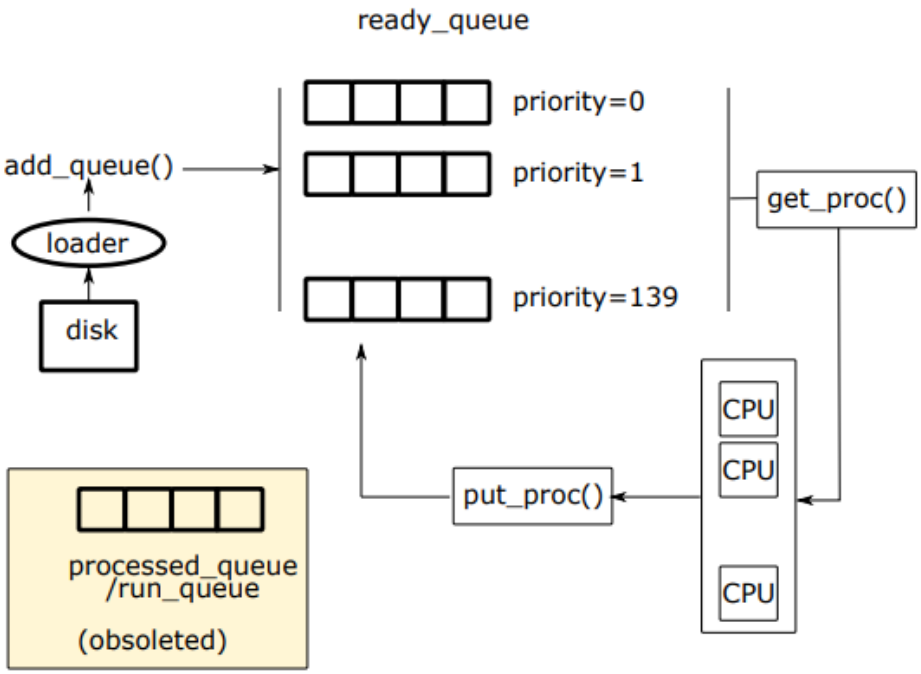


Figure 2: The operation of scheduler in the assignment

Giải thích ý nghĩa mô hình: Với mỗi chương trình, loader sẽ load vào OS một quá trình (process) mới. PCB của quá trình được thêm vào ready queue có cùng một mức ưu tiên với giá trị prio của process này. Sau đó, nó đợi CPU. Mỗi tiến trình được cho phép chạy trong một đoạn thời gian cụ thể (time slot). Sau đó, CPU đưa process trở lại ready queue có cùng độ ưu tiên. CPU sau đó chọn một tiến trình từ ready queue và tiếp tục chạy. Tiến trình này có thể là tiến trình cũ hoặc là một tiến trình mới, phụ thuộc vào độ ưu tiên được hiện thực cho nó.

Mô tả về chính sách MLQ: bước duyệt qua của danh sách hàng đợi sẵn sàng là một số có công thức cố định dựa trên mức độ ưu tiên, tức là slot = (MAX PRIO - prio), mỗi hàng đợi chỉ có một số lượng slot cố định để sử dụng CPU, và khi nó được sử dụng hết, hệ thống phải chuyển tài nguyên cho tiến trình khác trong hàng đợi kế tiếp và để lại công việc còn lại cho slot trong tương lai, ngay cả khi nó cần một vòng lặp hoàn thành của hàng đợi sẵn sàng.

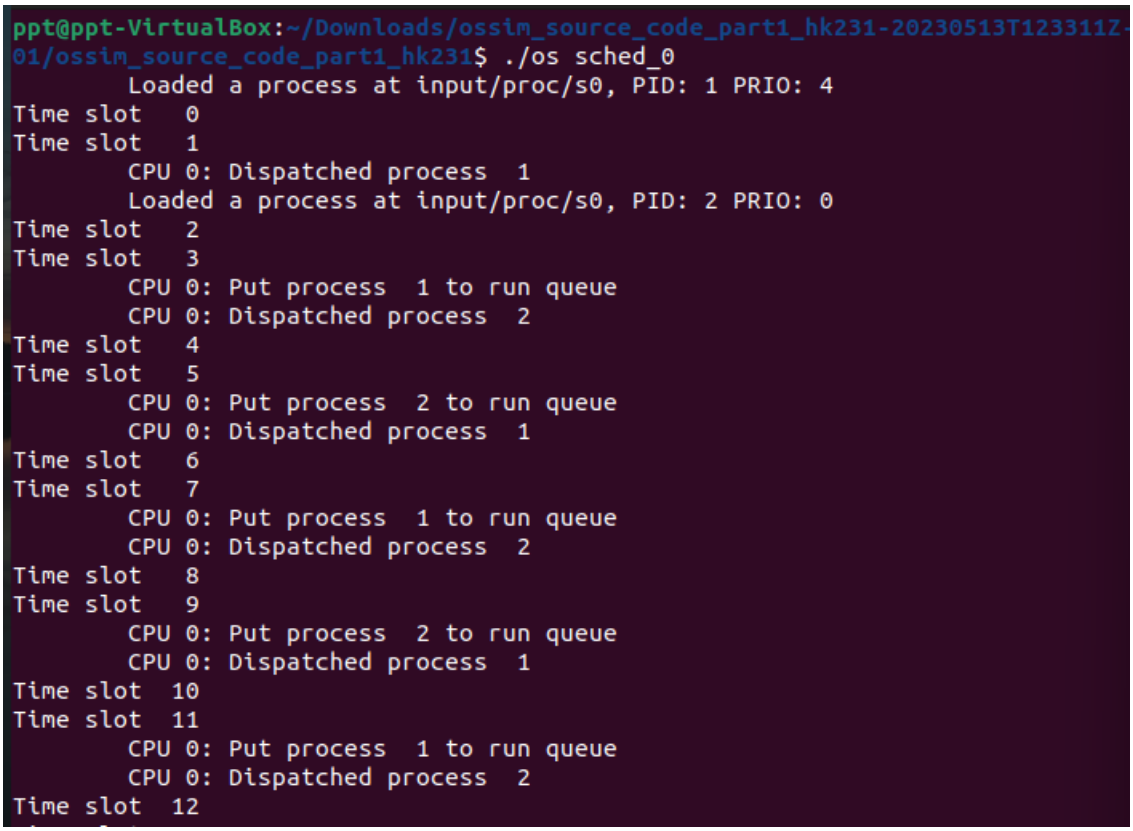
Dựa theo kiến thức được học, nhóm đã hiện thực và chạy thử một input mẫu dưới đây để quan sát:

2. *Part 1*

a. *File input sched_0*

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

b. *Kết quả chạy thử trên phần mềm Ubuntu – hệ điều hành Linux*

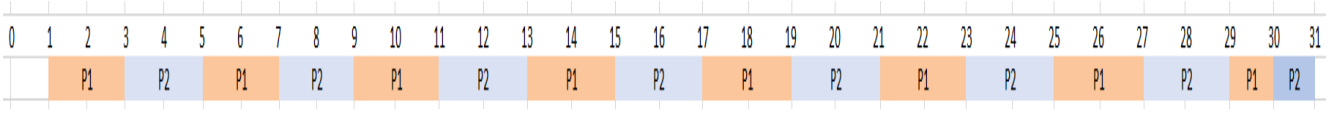


```
Time slot   12
Time slot   13
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  1
Time slot   14
Time slot   15
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   16
Time slot   17
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  1
Time slot   18
Time slot   19
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   20
Time slot   21
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  1
Time slot   22
Time slot   23
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   24
Time slot   25
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  1
```

```
Time slot   26
Time slot   27
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   28
Time slot   29
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  1
Time slot   30
        CPU 0: Processed  1 has finished
        CPU 0: Dispatched process  2
Time slot   31
        CPU 0: Processed  2 has finished
        CPU 0 stopped
ppt@ppt-VirtualBox:~/Downloads/ossim_source_code_part1_hk231-20230513T123311Z-0
```

c. *Giải thích và vẽ biểu đồ Gantt*

Từ file input sched_0 ta nhận các giá trị time_slot = 2, num_plus = 1, num_processes = 2. Các processes được load lên và đưa vào ready_queue (tối đa 10 process) hiện thực bằng Min heap để chọn được process có priority giá trị thấp nhất – mức độ ưu tiên cao nhất. Biểu đồ Gantt:



Khi 1 process thực hiện hết đoạn thời gian time_slot (ở đây là 2) nó sẽ được đưa vào run_queue để chờ được nạp vào ready_queue.

3. *Part 2 MLQ*

a. *Kết quả chạy thử trên phần mềm Ubuntu – hệ điều hành Linux*

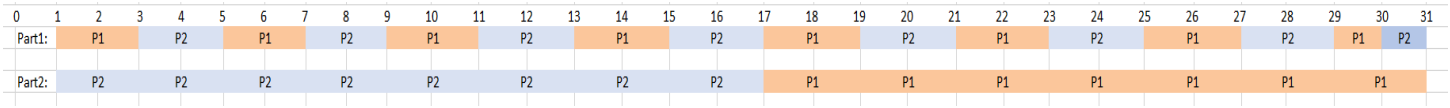
```
ppt@ppt-VirtualBox:~/Downloads/ossim_source_code_part2_hk231_paging-20230513T123307Z-001/ossim_source_code_part2_hk231_paging$ ./os sched_0
Time slot   0
ld_routine
        Loaded a process at input/proc/s0, PID: 1 PRIO: 4
        CPU 0: Dispatched process  1
Time slot   1
        Loaded a process at input/proc/s0, PID: 2 PRIO: 0
Time slot   2
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   3
Time slot   4
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   5
Time slot   6
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   7
Time slot   8
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   9
Time slot  10
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  11
Time slot  12
```

```
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  13
Time slot  14
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  15
Time slot  16
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  17
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  1
Time slot  18
Time slot  19
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  20
Time slot  21
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  22
Time slot  23
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  24
Time slot  25
```

```
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  26
Time slot  27
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  28
Time slot  29
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  30
        CPU 0: Processed  1 has finished
        CPU 0 stopped
```

b. Vẽ biểu đồ Gantt và giải thích

Tương tự ở part 1, ta nhận các giá trị từ file input sched_0 ta nhận các giá trị time_slot = 2, num_plus = 1, num_processes = 2. Các processes được load lên và đưa vào một mảng/ danh sách các hàng đợi (Multilevel queue) mà trong đó mỗi hàng đợi được gắn với một mức ưu tiên. Khi load processes lên, dựa vào chỉ số prio của chúng mà thêm vào vị trí queue tương ứng với quy tắc: process với prio sẽ được thêm vào queue có slot là (MAX_PRIO – prio) (MAX_PRIO = 140), mỗi hàng đợi được hiện thực bằng Min heap để chọn được process có priority giá trị thấp nhất – mức độ ưu tiên cao nhất. Biểu đồ Gantt của cả part 1 và part 2:



PHẦN 2: QUẢN LÝ BỘ NHỚ

I. Trả lời câu hỏi

1. ***Question 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?

Answer:

a. Đặt vấn đề

Nhìn lại kỹ thuật phân trang (paging): Kỹ thuật phân trang cho phép không gian địa chỉ vật lý (physical address space) của một process có thể không liên lục nhau. Bộ nhớ thực được chia thành các khối cố định và có kích thước bằng nhau.

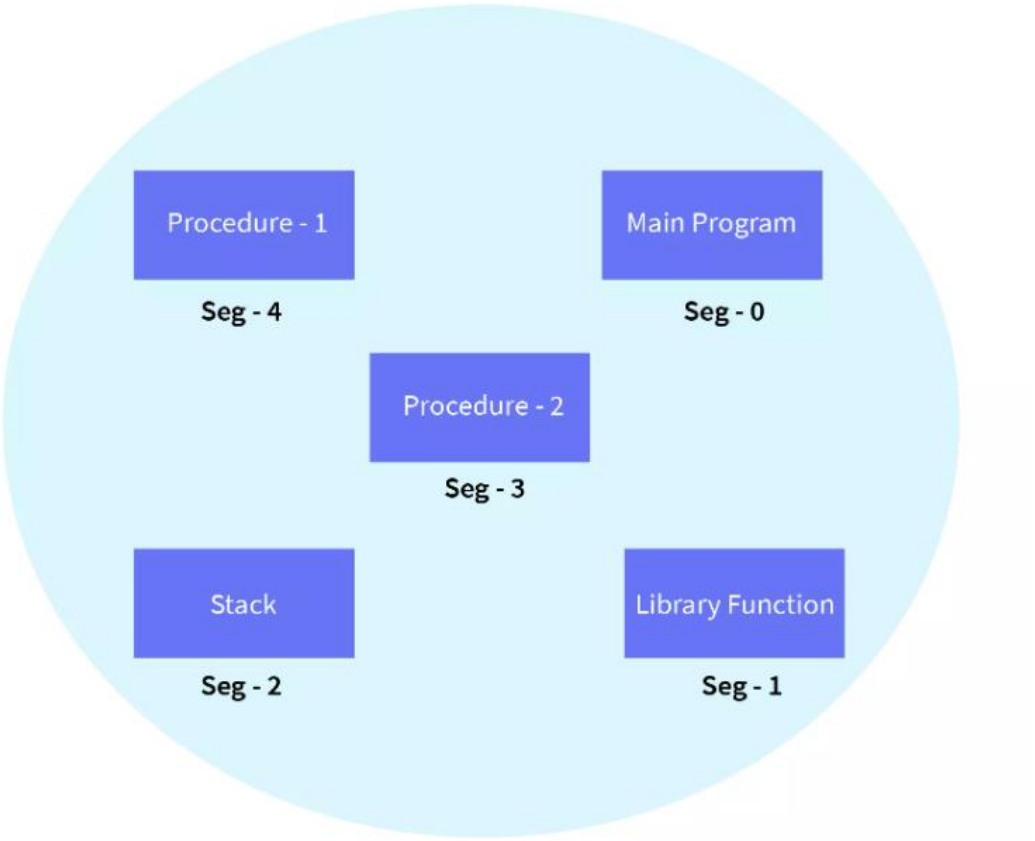
Trong thực tế, dưới góc nhìn của một user, một chương trình cấu thành từ nhiều đoạn (segment). Mỗi đoạn là một đơn vị luận lý của chương trình như: main program, procedure, function; local variables, global variables, common block, stack, symbol table, arrays,… Trong hệ điều hành, chương trình và dữ liệu được lưu trữ trong bộ nhớ, bộ nhớ lại được chia thành nhiều phân đoạn khác nhau để phục vụ cho mục đích khác nhau. Ví dụ, một hệ điều hành có thể chia bộ nhớ thành các phân đoạn sau: Code segment: Lưu trữ mã máy của chương trình; Data segment: Lưu trữ dữ liệu được sử dụng bởi chương trình; Heap segment: Lưu trữ dữ liệu động được tạo ra trong khi chương trình đang chạy; Stack segment: Lưu trữ các biến cục bộ và các giá trị của hàm được gọi trong chương trình.

Thông thường, một chương trình sẽ được biên dịch. Trình biên dịch sẽ tự động xây dựng các segment. Trình loader sẽ gán mỗi segment một số định danh riêng.

b. Kỹ thuật phân đoạn (Segmentation)

Phân đoạn phân chia các quá trình thành các phân nhóm nhỏ hơn được gọi là modules. Các segments được chia không cần phải đặt liên tục nhau trong bộ nhớ. Vì không có sự phân bổ bộ nhớ liên tục (contiguous memory allocation), việc phân mảnh nội sẽ không diễn ra. Độ dài của các phân đoạn của chương trình và bộ nhớ được quyết định bởi mục đích của việc phân đoạn trong chương

trình người dùng. Chúng ta có thể nói rằng không gian địa chỉ logic hoặc bộ nhớ chính là một tập hợp các phân đoạn. Phân đoạn phân chia chương trình người dùng và bộ nhớ thứ cấp thành các khối không đồng đều nhau.Một bảng phân đoạn được sử dụng để lưu trữ thông tin của tất cả các segment của quy trình hiện đang thực hiện. Việc hoán đổi các segment của quá trình dẫn đến việc chia không gian bộ nhớ (free memory) thành các mảnh nhỏ, từ đó dẫn đến phân mảnh ngoại.



Minh họa Segments

c. Ưu điểm

Phân đoạn ra đời vì các vấn đề trong kỹ thuật phân trang. Trong trường hợp kỹ thuật phân trang, một hàm hoặc đoạn mã được chia thành các trang mà không xem xét rằng các phần liên quan đến nhau cũng có thể bị phân chia. Do đó, đối với quá trình thực thi, CPU phải tải nhiều hơn một trang vào các khung để có code hoàn chỉnh của phần đó (complete related code) để chạy. Phân trang đã mất nhiều trang hơn cho một quá trình được tải vào bộ nhớ chính. Do đó, phân đoạn sẽ để code được chia thành các modules để mà phần code có liên quan đến nhau sẽ đươc kết hợp trong một khối duy

nhất (single block). Các kỹ thuật quản lý bộ nhớ khác cũng có một nhược điểm quan trọng - quan điểm thực tế về bộ nhớ vật lý được tách ra khỏi quan điểm của người dùng về bộ nhớ vật lý. Phân đoạn trong HĐH giúp khắc phục vấn đề bằng cách chia chương trình của người dùng thành các phân đoạn theo nhu cầu cụ thể. Tóm lại, các ưu điểm chính của kỹ thuật này:

- Không có phân mảnh nội.
- Bảng phân đoạn được sử dụng để lưu giữ thông tin của các phân đoạn. Bảng phân đoạn chiếm bộ nhớ nhỏ so với bảng trang trong kỹ thuật phân trang.
- Phân đoạn giúp cho việc sử dụng CPU tốt hơn vì một module trọn vẹn sẽ được tải cùng một lúc.
- Phân đoạn gần hơn với chế độ user's view trong bộ nhớ chính. Phân đoạn cho phép người dùng phân vùng các chương trình người dùng thành các modules. Các modules này chính là từng đoạn code riêng biệt nhau trong process hiện tại.
- Kích thước phân đoạn được chỉ định bởi người dùng nhưng trong phân trang, phần cứng quyết định kích thước trang.
- Phân đoạn trong HĐH có thể được sử dụng để phân tách các quy trình và dữ liệu bảo mật.

d. So sánh cơ bản để thấy sự khác biệt giữa hai loại kỹ thuật phân trang và phân đoạn

Cơ sở để so sánh	Phân trang (Paging)	Phân đoạn (Segmentation)
Primary	Trong phân trang, chương trình được chia thành các trang kích thước cố định hoặc được gắn sẵn. Hệ điều hành chịu trách nhiệm cho việc phân trang.	Trong phân đoạn, chương trình được chia thành các phần kích thước khác nhau. Trình biên dịch đoạn (segment complier) chịu trách nhiệm cho việc phân đoạn.
Partition (phân mảnh)	Phân trang có thể dẫn đến phân mảnh nội.	Phân khúc có thể dẫn đến sự phân mảnh ngoại.
Addresss	Địa chỉ do người dùng chỉ định được chia cho CPU thành page number và page offset.	Địa chỉ do người dùng chỉ định được chia thành segment number và segment offset.
Size	Phần cứng quyết định kích thước trang.	Kích thước phân khúc được chỉ định bởi người dùng.
Table	Phân trang liên quan đến một bảng trang có chứa địa chỉ cơ sở của mỗi trang.	Phân đoạn liên quan đến bảng phân đoạn có chứa segment number và offset (độ dài segment).

Bảng so sánh

Cơ chế kết hợp phân trang và phân đoạn được sử dụng để tận dụng tối đa các ưu điểm và hạn chế các khuyết điểm của hai loại kỹ thuật. Có nhiều cách kết hợp, một trong những cách đơn giản là segmentation with paging mà nhóm được nghiên cứu trong bài tập lớn này. Mỗi process sẽ có một bảng phân đoạn và nhiều bảng phân trang, mỗi đoạn sẽ có một bảng phân trang.

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

Answer: Xem xét hệ thống Page Table một mức, các hệ thống hiện đại đều hỗ trợ không gian địa chỉ ảo rất lớn (2^{32} đến 2^{64} bYTE), ở đây giả sử là 2^{32}, kích thước trang nhớ là 4KB (= 2^{12}) thì bảng phân trang sẽ có 2^{20} = 1M mục, nếu mỗi mục cần 4 Byte thì mỗi process sẽ cần 4MB cho bảng phân trang, kích thước rất lớn. Khi kích thước của page table nhỏ hơn kích thước của một khung thì chúng ta không cần phải lo lắng vì chúng ta có thể đặt trực tiếp bảng trang vào khung của bộ nhớ chính, từ đó truy cập trực tiếp vào bảng phân trang. Nhưng nếu kích thước lớn hơn thì sẽ phức tạp hơn nhiều, giải pháp là phân page table thành những đơn vị nhỏ hơn. Nếu bảng phân trang 2 mức vẫn còn quá lớn, tương tự với bảng phân trang 2 mức, ta có thể có bảng phân trang 2,3,4 đến n mức.

Ưu điểm:

- Giảm chiếm bộ nhớ: phân trang đa cấp có thể giúp giảm chiếm bộ nhớ liên quan đến bảng trang. Điều này là do mỗi cấp có chứa ít mục hơn, điều đó có nghĩa là cần ít bộ nhớ để lưu trữ bảng trang.
- Tra cứu bảng phân trang nhanh hơn: Với số lượng mục nhỏ hơn cho mỗi mức, cần ít thời gian hơn để thực hiện tra cứu bảng phân trang. Điều này có thể dẫn đến hiệu suất hệ thống nhanh hơn tổng thể.
- Tính linh hoạt: Phân trang đa cấp cung cấp sự linh hoạt 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.

Tuy nhiên, khi chia địa chỉ thành nhiều hơn 2 cấp, số bit được sử dụng cho mỗi cấp sẽ giảm đi, tương đương với kích thước của frame sẽ giảm đi theo đó. Điều này có thể dẫn đến hiện tượng phân mảnh nội (Internal Fragmentation) trong bộ nhớ, nghĩa là một số phần của frame sẽ không được sử

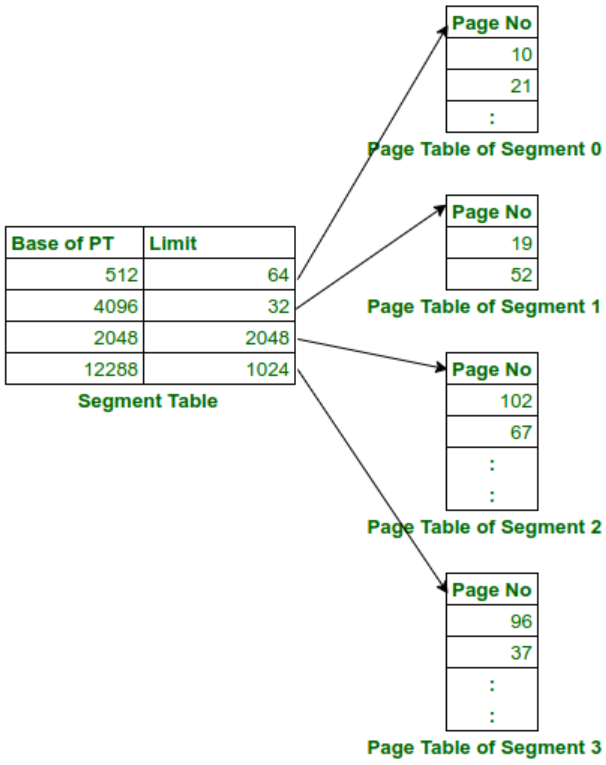
dụng. Ví dụ, nếu chúng ta chia địa chỉ thành 4 cấp, mỗi cấp chỉ sử dụng 4 bit, khi đó mỗi frame chỉ được sử dụng 16Byte thay vì 4KB như ở 2 cấp, dẫn đến lãng phí bộ nhớ.

Nhược điểm:

- Tăng cường độ phức tạp: Phân trang đa cấp làm tăng thêm sự phức tạp cho hệ thống quản lý bộ nhớ, điều này có thể khiến việc thiết kế, thực hiện và fix bug trở nên khó khăn hơn.
- Tăng chi phí: Mặc dù phân trang đa cấp có thể làm giảm chi phí bộ nhớ liên quan đến bảng trang, nhưng nó cũng có thể tăng chi phí liên quan đến tra cứu bảng trang. Điều này là do mỗi cấp độ phải được đi qua để tìm mục nhập bảng trang mong muốn.
- Phân mảnh: Phân trang đa cấp có thể dẫn đến sự phân mảnh của không gian bộ nhớ, có thể làm giảm hiệu suất hệ thống tổng thể. Điều này là do các mục bảng trang có thể không liên tục nhau.

Việc chia địa chỉ thành nhiều cấp đều có những mặt lợi và hạn chế riêng. Cần xem xét nhu cầu cũng như mục tiêu hiện thực để lựa chọn số cấp phân chia địa chỉ vùng nhớ phù hợp nhằm mục đích đạt hiệu quả quản lý bộ nhớ tối đa.

3. ***Question 3:*** What is the advantage and disadvantage of segmentation with paging?



Answer: Như đã trình bày ở Question 1, Segmentation with paging là một trong những cơ chế đơn giản kết hợp hai kỹ thuật phân đoạn và phân trang để tận dụng ưu điểm và hạn chế khuyết điểm lẫn nhau của hai loại kỹ thuật quản lý bộ nhớ.

Ưu điểm:

- Kích thước bảng trang được giảm xuống khi các trang chỉ có mặt cho dữ liệu của các phân đoạn, do đó giảm các yêu cầu bộ nhớ.
- Cung cấp cho lập trình viên những lợi thế của kỹ thuật phân trang.
- Giảm phân mảnh ngoại so với phân đoạn.
- Vì toàn bộ phân đoạn không cần phải swap out, do đó việc swap out vào bộ nhớ ảo trở nên dễ dàng hơn.

Nhược điểm:

- Phân mảnh nội vẫn tồn tại trong các trang.
- Cần có thêm phần cứng.
- Phức tạp và chiếm nhiều tài nguyên hơn để có thể hoạt động, từ đó có thể làm giảm hiệu suất của hệ thống.
- Translation trở nên tuần tự hơn làm tăng thời gian truy cập bộ nhớ.
- Phân mảnh ngoại xảy ra do các kích thước khác nhau của bảng trang và kích thước khác nhau của các bảng phân đoạn trong các hệ thống ngày nay.

Kết hợp giữa phân đoạn và phân trang giúp khắc phục những nhược điểm của từng kỹ thuật đơn lẻ. Tuy nhiên, kết hợp này cũng có nhược điểm của cả hai phương pháp. Tùy trường hợp mà chúng ta sử dụng cơ chế kết hợp thích hợp, một cơ chế kết hợp khác là Paged Segmentation cũng là một trong những cơ chế kết hợp thường được sử dụng.

II. Cơ sở lí thuyết

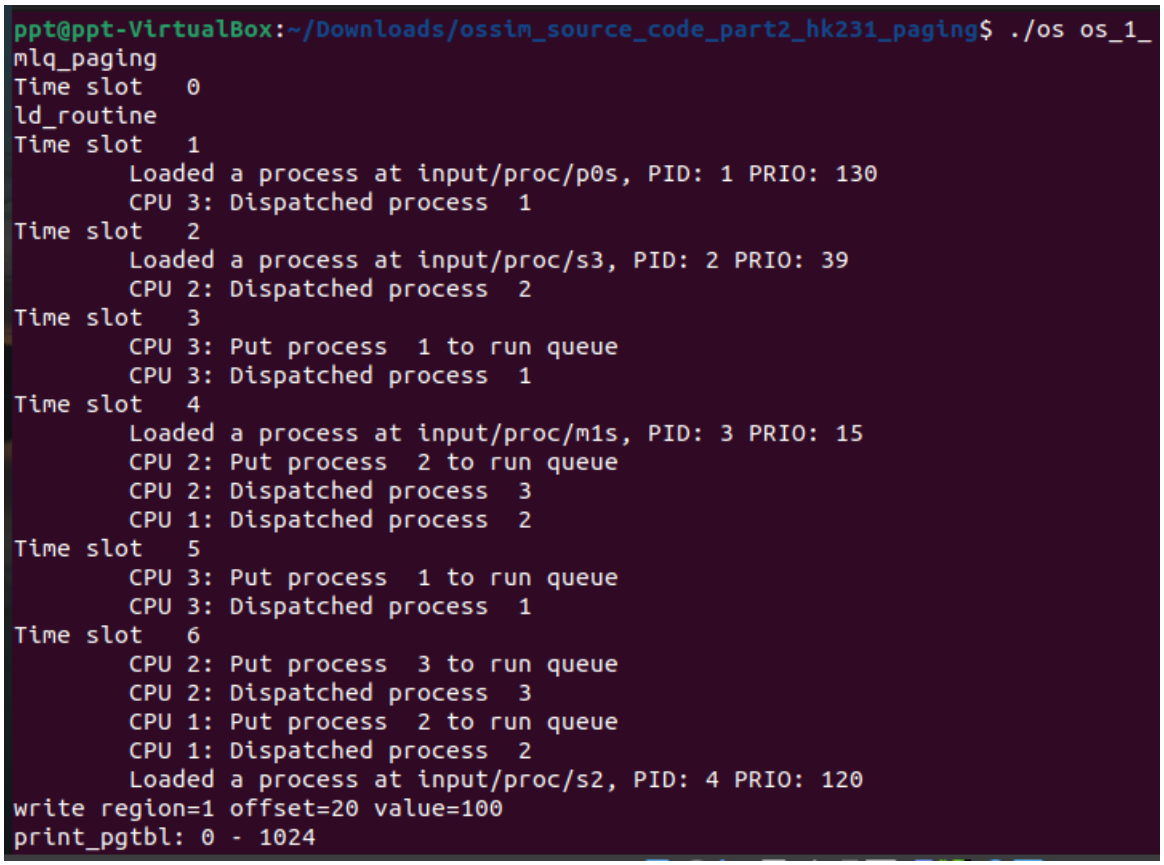
(Đề bài đã cung cấp đầy đủ về cơ sở lí thuyết)

III. Hiện thực

1. File input os_1_mlq_paging

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

2. Kết quả chạy thử trên Ubuntu



```
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Start MEMPHY_dump
End MEMPHY_dump
Time slot   7
        CPU 0: Dispatched process  4
        Loaded a process at input/proc/m0s, PID: 5 PRIO: 120
        CPU 3: Put process  1 to run queue
        CPU 3: Dispatched process  5
Time slot   8
        CPU 2: Put process  3 to run queue
        CPU 2: Dispatched process  3
        CPU 1: Put process  2 to run queue
        CPU 1: Dispatched process  2
Time slot   9
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
        Loaded a process at input/proc/p1s, PID: 6 PRIO: 15
        CPU 3: Put process  5 to run queue
        CPU 3: Dispatched process  6
Time slot  10
        CPU 2: Put process  3 to run queue
        CPU 2: Dispatched process  3
        CPU 1: Put process  2 to run queue
        CPU 1: Dispatched process  2
Time slot  11
        CPU 0: Put process  4 to run queue
```

```
        CPU 0: Dispatched process  5
        Loaded a process at input/proc/s0, PID: 7 PRIO: 38
        CPU 3: Put process  6 to run queue
        CPU 3: Dispatched process  6
Time slot  12
        CPU 2: Processed  3 has finished
        CPU 2: Dispatched process  7
        CPU 1: Put process  2 to run queue
        CPU 1: Dispatched process  2
Time slot  13
        CPU 1: Processed  2 has finished
        CPU 1: Dispatched process  4
        CPU 0: Put process  5 to run queue
        CPU 0: Dispatched process  5
write region=1 offset=20 value=102
print_pgtbl: 0 - 512
00000000: 80000007
00000004: 80000006
Start MEMPHY_dump
At physical address 220 has data: 100
        CPU 3: Put process  6 to run queue
        CPU 3: Dispatched process  6
End MEMPHY_dump
Time slot  14
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
write region=2 offset=1000 value=1
print_pgtbl: 0 - 512
00000000: 80000007
```

```
00000004: 80000006
Start MEMPHY_dump
At physical address 64 has data: 102
At physical address 220 has data: 100
End MEMPHY_dump
Time slot   15
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  4
        CPU 3: Put process  6 to run queue
        CPU 3: Dispatched process  6
        CPU 0: Put process  5 to run queue
        CPU 0: Dispatched process  5
write region=0 offset=0 value=0
print_pgtbl: 0 - 512
00000000: c0000000
00000004: 80000006
Start MEMPHY_dump
At physical address 64 has data: 102
At physical address 176 has data: 1
At physical address 220 has data: 100
End MEMPHY_dump
Time slot   16
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
        CPU 0: Processed  5 has finished
        CPU 0: Dispatched process  1
read region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
```

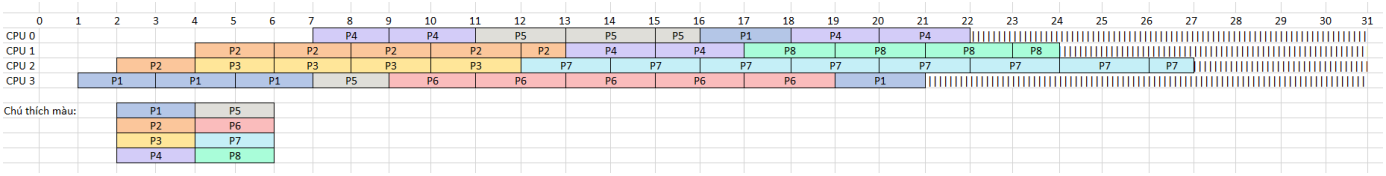
```
00000004: 80000000
00000008: 80000003
00000012: 80000002
Start MEMPHY_dump
At physical address 64 has data: 102
At physical address 220 has data: 100
At physical address 255 has data: 0
        Loaded a process at input/proc/s1, PID: 8 PRIO: 0
End MEMPHY_dump
Time slot   17
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  8
write region=2 offset=20 value=102
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Start MEMPHY_dump
At physical address 64 has data: 102
At physical address 220 has data: 100
At physical address 255 has data: 0
End MEMPHY_dump
        CPU 3: Put process  6 to run queue
        CPU 3: Dispatched process  6
Time slot   18
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
        CPU 0: Put process  1 to run queue
```

```
        CPU 0: Dispatched process  4
Time slot  19
        CPU 1: Put process  8 to run queue
        CPU 1: Dispatched process  8
        CPU 3: Processed  6 has finished
        CPU 3: Dispatched process  1
read region=2 offset=20 value=102
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Start MEMPHY_dump
At physical address 20 has data: 102
At physical address 64 has data: 102
At physical address 220 has data: 100
At physical address 255 has data: 0
End MEMPHY_dump
Time slot  20
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
write region=3 offset=20 value=103
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
```

```
Start MEMPHY_dump
At physical address 20 has data: 102
At physical address 64 has data: 102
At physical address 220 has data: 100
At physical address 255 has data: 0
End MEMPHY_dump
Time slot  21
        CPU 1: Put process  8 to run queue
        CPU 1: Dispatched process  8
        CPU 3: Processed  1 has finished
        CPU 3 stopped
Time slot  22
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
        CPU 0: Processed  4 has finished
        CPU 0 stopped
Time slot  23
        CPU 1: Put process  8 to run queue
        CPU 1: Dispatched process  8
Time slot  24
        CPU 1: Processed  8 has finished
        CPU 1 stopped
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
Time slot  25
Time slot  26
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
Time slot  27
```

```
Time slot  27
        CPU 2: Processed  7 has finished
        CPU 2 stopped
```

3. *Sơ đồ Gannt cho ví dụ trên*



4. *Giải thích về trạng thái của RAM trong các process khi alloc và free.*

Từ file input ta nhận các giá trị time_slot = 2, num_plus = 4, num_processes = 8. Memramsz (kích thước của RAM) = 1048576, giá trị kích thước của 4 SWAP lần lượt là memswpsz 16777216, 0, 0, 0. Ở bài báo cáo này, vì để cho ngắn gọn, nhóm sẽ giải thích chi tiết tiến trình đầu tiên là file input p0s với 10 tác vụ gồm 3 tác vụ alloc và 2 tác vụ free.

Tại time_slot 1, process 1 được load lên với pid = 1, bắt đầu thực hiện tác vụ CALC, sau đó đến ALLOC 300 0 (proc->pc = 2), thực hiện alloc một size bằng 300 ở region index 0 của mảng symrgtbl.

- Tại hàm alloc, ta xét 2 trường hợp là:
    - Tồn tại region free trong vùng area hiện tại
    - Không tồn tại region free trong vùng area hiện tại

Ghi chú: *hàm* ***get_free_vmrg_area*** *lưu lại những vùng region đã được giải phóng. Ví dụ khi giải phóng vùng nhớ "foo" bằng lệnh free(foo) thì vùng nhớ "foo" sẽ không có địa chỉ NULL và giảm program break (sbrk) mà thay vào đó, sẽ thêm vùng nhớ này vào danh sách liên kết* ***vm_freerg_list*** *của vùng area hiện tại. Vùng nhớ này sẽ được tái sử dụng bởi các lần gọi alloc sau đó.*

Như vậy, tại thời điểm hiện tại, không có region free trong vùng area hiện tại. Khi đó, ta sẽ tăng giới hạn của vùng area hiện tại, cụ thể là ***vm_end*** của area hiện tại bằng hàm ***inc_vma_limit*** .

- Tại hàm ***inc_vma_limit*** ta nhận thấy size cần alloc là 300. Mà ta có 1 trang(page) của bộ nhớ ảo có kích thước là 256, như vậy, ta cần cung cấp 2 trang
  => kích thước là 256 * 2 = 512 để có thể alloc kích thước 300.

- Ở đây ta cần xét vùng region bắt đầu từ rg_start = sbrk (program break) đến rg_start + 300 có vượt qua kích thước của vùng area hiện tại hay không. Nếu có, ta cần tăng vm_end đến giá trị có thể chứa đựng được vùng region trên. Nếu không, ta cập nhật lại giá trị sbrk cho vùng area.
    - Cụ thể, vùng region cần so sánh ở đây có rg_start = 0 và rg_end = 300. Vùng area hiện tại có vma_start = 0 và vma_end = 0 (khai báo mặc định ban đầu).
    - Ta tiến hành so sánh bằng hàm ***validate_overlap_vm_area*** và thấy rằng, vùng region cần so sánh có rg_end (300) đã vượt ngưỡng giới hạn của vùng area hiện tại (0) nên ta cần tăng vùng area hiện tại lên 512 (giá trị được đối chiếu với kích thước của trang vùng nhớ) để có thể chứa đựng được kích thước 300.
- Như vậy kết thúc tác vụ ALLOC 300 0, vm_start = 0, vm_end = 512, sbrk = 300, symrgtbl[0] có rg_start = 0, rg_end = 300.

Ghi chú: *sau khi tăng giới hạn vùng area, ta tính toán các page được tạo ra (on-line page) và kết nối chúng với số lượng frame tương ứng trong bộ nhớ vật lý bằng* ***pte_set_fpn*** *ở hàm* ***vmap_page_range.*** *Danh sách entries trong process (pgd) chứa các entries để giữ mối liên kết giữa page trong ram và frame trong bộ nhớ vật lý. Entry thứ pgn(page number) là số thứ tự của frame (fpn).*

Proc->pc = 3, process thực hiện tác vụ thứ 3, ALLOC 300 4, tương tự như trên, ta thấy không có vùng region nào trong danh sách liên kết ***vm_freerg_list*** nên ta tiếp tục tăng giới hạn của vm_end vùng area hiện tại. Xét thấy kích thước cần alloc là 300 nên ta tăng vm_end lên thêm 512. Ở tác vụ thứ 2, vm_end = 512 nên khi tăng thêm 512, ta được vm_end = 1024. Ngoài ra, sbrk lúc này cập nhật bằng sbrk = 300 (sbrk cũ) + 300 (kích thước mới) = 600, symrgtbl[4] có rg_start = 300, rg_end =600.

Proc->pc = 4, process thực hiện tác vụ tiếp theo là FREE 0, giải phóng vùng nhớ ở vị trí index 0 trong mảng symrgtbl.

- Thêm vùng nhớ này vào danh sách liên kết ***vm_freerg_list*** để có thể tái sử dụng.

- Ta thực hiện thay đổi các địa chỉ rg_start, rg_end của vùng nhớ này là -1, biểu thị vùng nhớ chưa được cấp phát.

Proc->pc = 5, process thực hiện tác vụ tiếp theo là ALLOC 100 1.

- Ở thời điểm này, trong danh sách liên kết ***vm_freerg_list*** có chứa một vùng nhớ sẵn có với rg_start = 0 và rg_end = 300.
- Vì để địa chỉ hiện tại của phân vùng không bị nhập nhằng, các vùng region không bị phân mảnh, ta chuyển đổi kích thước của vùng nhớ sẵn có này thành rg_start = 200 và rg_end = 300 để có thể phù hợp với kích thước 100 mà yêu cầu cấp phát.
- Cuối cùng, ta cập nhật giá trị cho vùng nhớ tại index = 1 của mảng, symrgtbl[1] có rg_start = 200, rg_end =300.

Proc->pc = 6, process thực hiện tác vụ write vào vùng nhớ 1 giá trị 100 với offset = 20, WRITE 100 1 20

- Ta truyền vào địa chỉ addr = rg_start (của vùng nhớ cần ghi giá trị) + offset = 200 + 20 = 220 để tìm số thứ tự của page trong page table và từ đó tìm số thứ tự frame trong bộ nhớ vật lý. Ở đây, pgn = 0 và fpn = 0 => địa chỉ vật lý là 220
- Ghi giá trị 100 vào địa chỉ 220 trong storage.

Ghi chú: *symrgtbl[1] có rg_start = 200, rg_end =300.*

Proc->pc = 7, process thực hiện tác vụ read vùng nhớ 1, offset là 20, tham số cuối cùng là thứ tự thanh ghi để lưu giá trị được đọc ra (không được hiện thực trong bài tập lớn này)

- Ta truyền vào địa chỉ = 220 = rg_start (của vùng nhớ cần đọc giá trị) + offset = 200 + 20 để tìm số thứ tự của page trong page table và từ đó tìm số thứ tự frame trong bộ nhớ vật lý. Ở đây, pgn = 0 và fpn = 0 => địa chỉ vật lý là 220
- Đọc dữ liệu tại địa chỉ số 220 ở storage.

Tương tự đối với các tác vụ write, read còn lại.

PHẦN 3: PUT IT ALL TOGETHER

I. Trả lời câu hỏi

Question: What will happen if the synchronization is not handled in your simple OS?

Answer: Nếu đồng bộ hóa không được được xử lý, hay xử lý không được tốt trong hệ điều hành đơn giản, có thể gây ra các vấn đề như:

- Race condition: là một tình huống xảy ra khi hai hay nhiều process, thread cùng đồng thời truy cập vào dữ liệu chia sẻ chung, và có ít nhất 1 thread, process thay đổi giá trị của dữ liệu đó. Từ đó dẫn đến các giá trị ghi đè lẫn nhau, đọc ra giá trị không mong muốn.
- Data corruption (hư hỏng dữ liệu) xảy ra khi hai process cố gắng ghi đồng thời vào cùng một tệp, nội dung của tệp sẽ bị cắt xén hoặc có thể chứa các dữ liệu không mong muốn.
- Deadlocks: là trạng thái xảy ra trong môi trường đa nhiệm (muti-threading) khi hai hoặc nhiều tiến trình đi vào vòng lặp chờ tài nguyên mãi mãi.
- Giảm hiệu suất: Trong trường hợp hệ điều hành không có cơ chế đồng bộ hóa, có thể xảy ra tranh chấp quá mức đối với các tài nguyên được chia sẻ, dẫn đến hiệu suất kém.

Trong bài tập lớn này, nhóm đã sử dụng `static pthread_mutex_t queue_lock` để đồng bộ, ngăn chặn các vấn đề trên, ở trong file code sched.c.

II. Hiện thực

(Để minh họa cho kết quả hiện thực, nhóm đã chạy input mẫu os_1_mlq_paging. Kết quả đã trình bày ở Phần 2)

III. Kết luận

Hệ điều hành nhóm hiện thực:

a. **Chạy đa luồng** với nhiều process, trong đó dùng **Multilevel Queue Scheduling** (Điều phối hàng chờ nhiều mức) để sử dụng nhiều mức hàng chờ với độ ưu tiêu khác nhau bằng một mảng các hàng chờ có vị trí tối đa là 140. Ngoài ra, mỗi hàng chờ dùng giải thuật Priority Queue (Hàng đợi ưu tiên) theo giá trị nhỏ nhất của priority từng tiến trình (tương ứng mức ưu tiên cao nhất) hiện thực bằng cấu trúc dữ liệu Min heap.

b. Về **quản lý bộ nhớ**, bộ nhớ được hiện thực theo cơ chế **segmentation with paging**, bộ nhớ vật lý là một hệ thống gồm 1 RAM và 4 SWAP (bộ nhớ phụ) với RAM là bộ nhớ chính, được CPU truy cập trực tiếp và duy nhất; có cơ chế swapping giữa RAM và SWAP khi cần thiết. Trong phiên bản này, nhóm phát triển một hệ thống single paging tận dụng một thiết bị RAM và một SWAP. Nhóm chỉ tập trung sử dụng phân đoạn đầu tiên và là phân đoạn duy nhất của vm_area (với vmaid = 0).

PHẦN 4: KẾT LUẬN

Trong bài tập lớn này, nhóm đã hoàn thành mô phỏng một hệ điều hành đơn giản bao gồm các hoạt động sau: scheduling (định thời), memory management (quản lý bộ nhớ) cũng như là synchronization (đồng bộ hóa). Từ đó, nhóm đã hiểu thêm về các phần lý thuyết dược học trên lớp và các buổi thí nghiệm. Nhờ có dự án BTL này mà nhóm cũng có cơ hội nâng cao tinh thần làm việc nhóm. Cảm ơn các giáo viên Bộ Môn đã giúp đỡ chúng em hoàn thành dự án.

PHẦN 5: TÀI LIỆU THAM KHẢO

1. https://www.scaler.com/topics/data-structures/priority-queue-in-data-structure/
2. https://www.geeksforgeeks.org/multilevel-queue-mlq-cpu-scheduling/
3. https://www.prepbytes.com/blog/queues/multilevel-queue-mlq-cpu-scheduling
4. https://www.geeksforgeeks.org/difference-between-paging-and-segmentation/
5. https://www.scaler.com/topics/operating-system/segmentation-in-os/
6. https://www.geeksforgeeks.org/multilevel-paging-in-operating-system/
7. https://www.geeksforgeeks.org/paged-segmentation-and-segmented-paging/