**BÁO CÁO HỆ ĐIỀU HÀNH**

1. **Scheduler:**

Kết quả: các kết quả có thể khác biệt so với output mẫu sẵn có do loader và scheduler được chạy đồng thời với nhau.

- sched\_0

Text

Description automatically generated

Text

Description automatically generated

Ở ví dụ này, chương trình thực thi 2 process:

+ s0: số câu lệnh: 15, độ ưu tiên: 12, thời gian bắt đầu: 0 🡪 process 1

+ s1: số câu lệnh: 7, độ ưu tiên: 20, thời gian bắt đầu: 4 🡪 process 2

Các process được thực thi trên duy nhất 1 CPU (CPU 0), sử dụng *“priority feedback queue”* với timeslice = 2 cho việc định thời.

Ta thấy, s0 được load tại timeslot 0, s1 được load tại timeslot 4. Các process sẽ lần lượt được phân quyền sử dụng CPU và thực thi với timeslice = 2, có nghĩa là mỗi process mỗi lần chiếm giữ CPU sẽ chỉ thực hiện được tốt đa 2 câu lệnh (1 câu lệnh thực hiện trong 1 timeslot).

Ta có thể tóm tắt thứ tự thực thi của các process như sau:

s0 – s0 – s1 – s1 – s0 – s1 – s0 – s1 (**finished**) – s0 – s0 – s0 – s0 (**finished**)

Do số câu lệnh của s0 và s1 lần lượt là 15 và 7 nên sau khi s0 chiếm dụng CPU 8 lần thì kết thúc tại timeslot 23, s1 chiếm dụng CPU 4 lần và kết thúc tại timeslot 16.

- sched\_1

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Tương tự như vậy, ở ví dụ này, chương trình sẽ thực thi với 4 process:

+ s0: số câu lệnh: 15, độ ưu tiên: 12, thời gian bắt đầu: 0 🡪 process 1

+ s1: số câu lệnh: 7, độ ưu tiên: 20, thời gian bắt đầu: 4 🡪 process 2

+ s2: số câu lệnh: 12, độ ưu tiên: 20, thời gian bắt đầu: 6 🡪 process 3

+ s3: số câu lệnh: 11, độ ưu tiên: 7, thời gian bắt đầu: 7 🡪 process 4

Các process được thực thi trên 1 CPU (CPU 0), sử dụng *“priority feedback queue”* với timeslice = 2 cho việc định thời.

Với các thông số ở trên, tương tự như cách giải thích cho sched\_0, ta thấy được:

+ s0: loaded: timeslot 0, finished: timeslot 46, số lần chiếm CPU: 8

+ s1: loaded: timeslot 4, finished: timeslot 28, số lần chiếm CPU: 4

+ s2: loaded: timeslot 6, finished: timeslot 42, số lần chiếm CPU: 6

+ s3: loaded: timeslot 7, finished: timeslot 45, số lần chiếm CPU: 6

**Q:** Những lợi ích của việc sử dụng *“priority feedback queue”* so với các giải thuât đã học như FIFO, Round Robin:

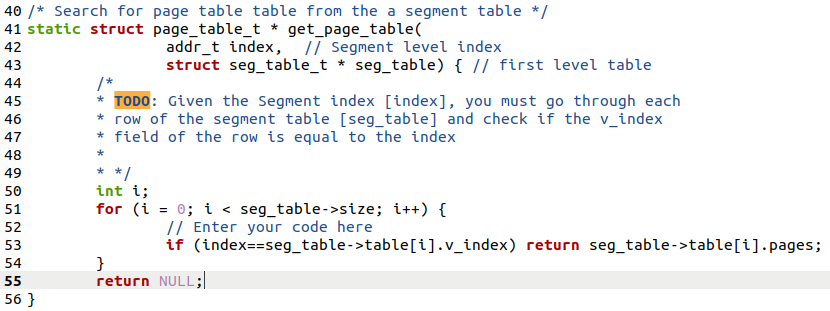
Các process đều được thực thi lần lượt trên CPU với một chu kì timeslice nhất định sau đó chuyển qua quyền qua cho process khác, do đó tối ưu hơn về thời gian phản hồi như đối với FIFO, SJF,... các process có thể thực thi xen kẽ nhau thay vì một process phải đợi cho tất cả các process đến trước nó hoàn thành mới có thể phản hồi.

Các process được phân chia và thực thi theo mức độ ưu tiên. Đây là một điểm đặc biệt mà các giải thuật trước đó như FIFO, Round Robin không có. Các process cần thiết hơn sẽ được gán cho mức độ ưu tiên cao hơn, do đó sẽ được thực thi và phản hồi trước cho người sử dụng. Tránh được tình trạng với những chương trình quan trọng, cần phải phản hồi ngay lập tức, tuy nhiên lại phải chờ đợi CPU từ những chương trình ít quan trọng hơn, gây khó chịu cho người sử dụng.

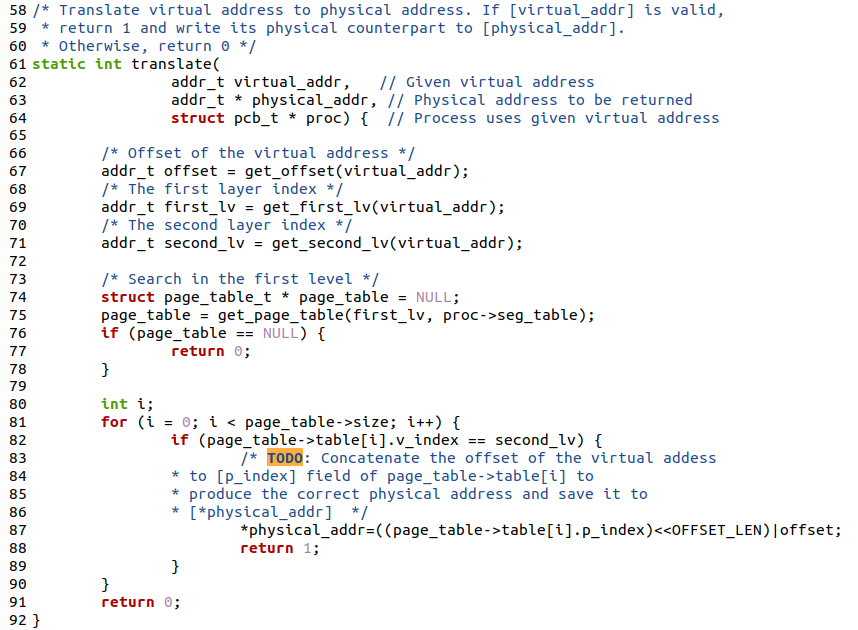
1. **Memory Management:**

Thực hiện các đoạn code trong file mem.c như sau:

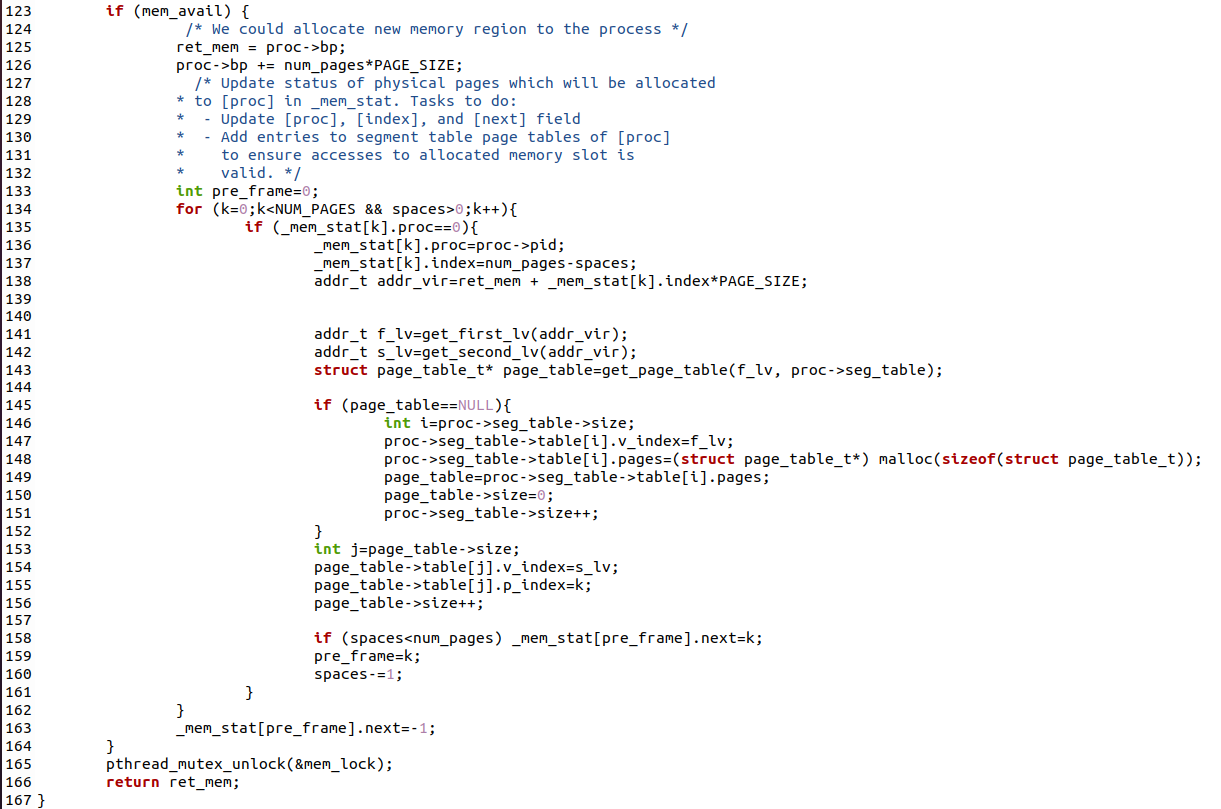
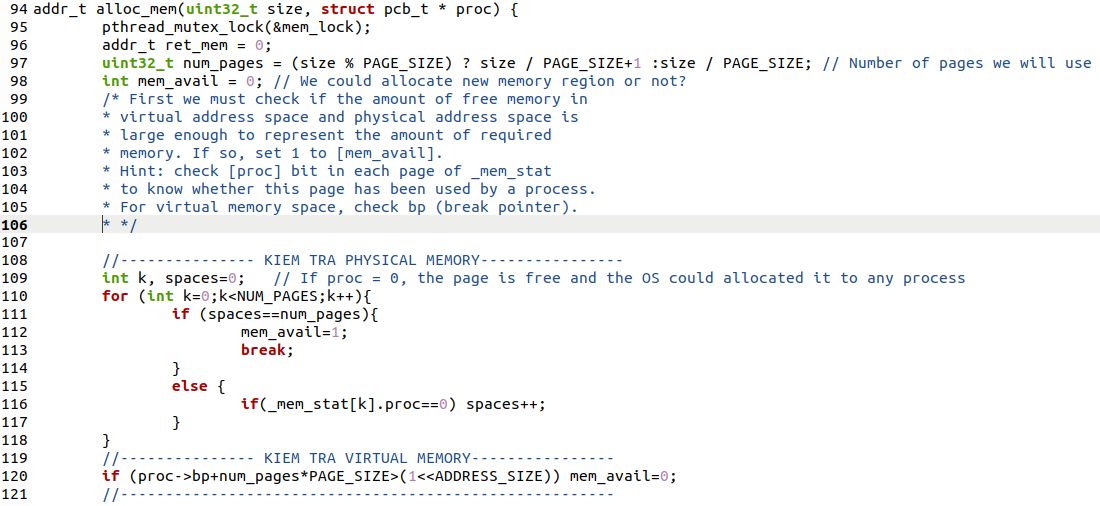
* Hàm tìm bảng phân trang (page\_table) từ seg\_table



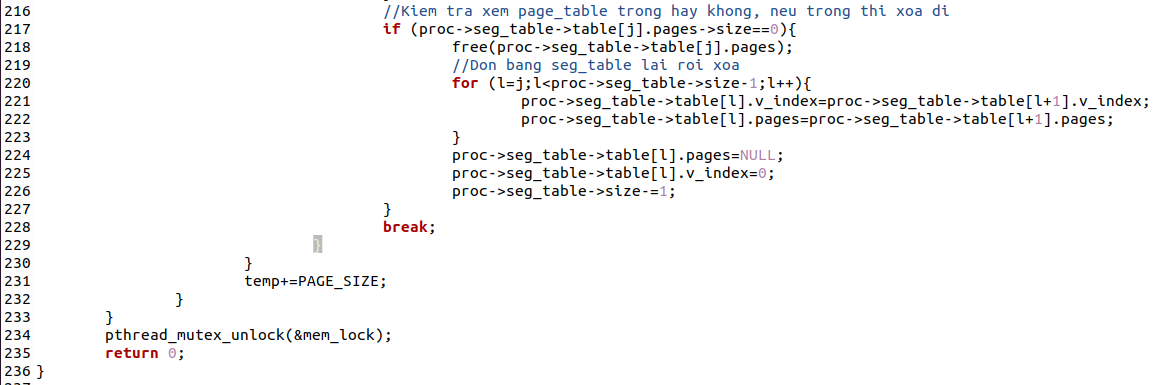
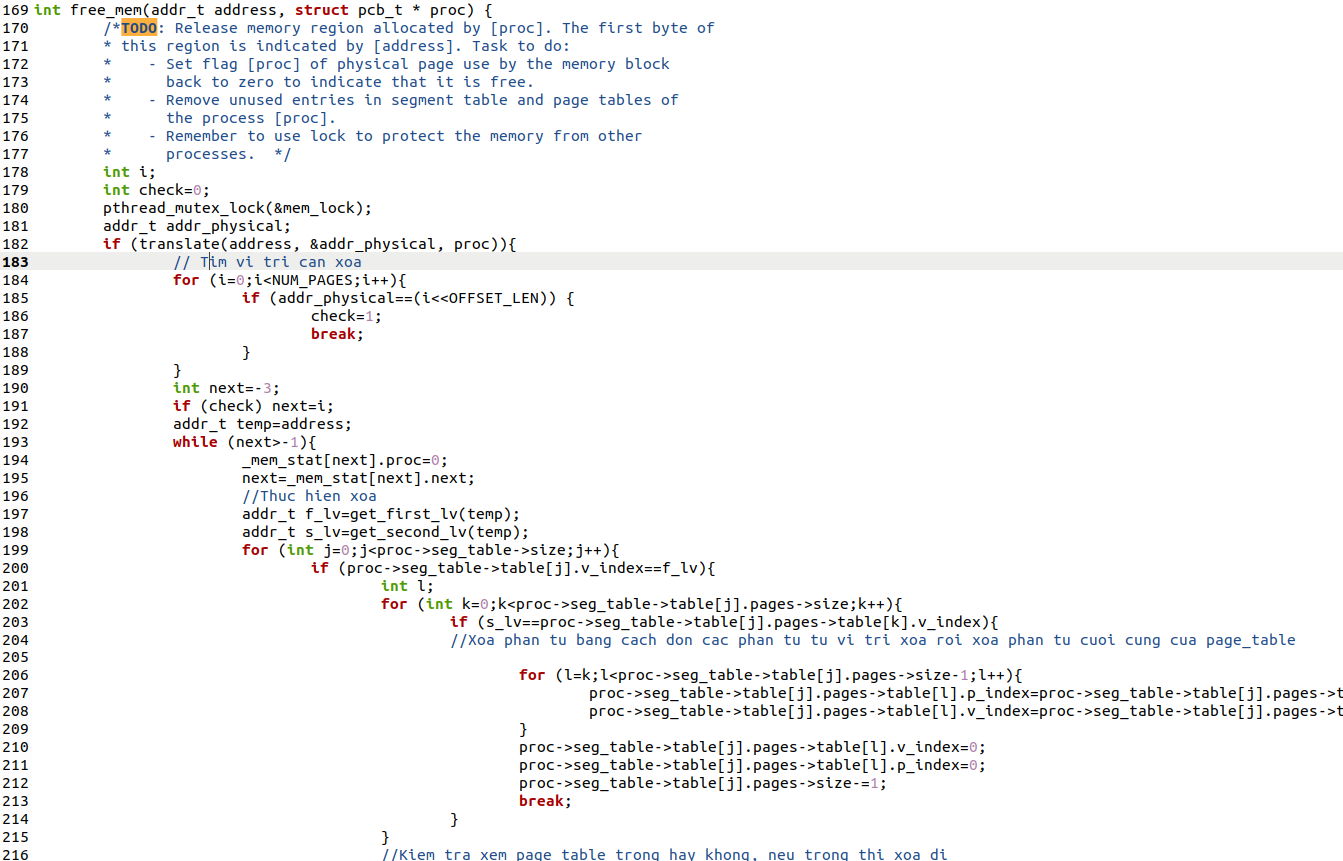
* Hàm ánh xạ (translate) từ địa chỉ ảo (virtual address) sang địa chỉ vật lý (physical address)



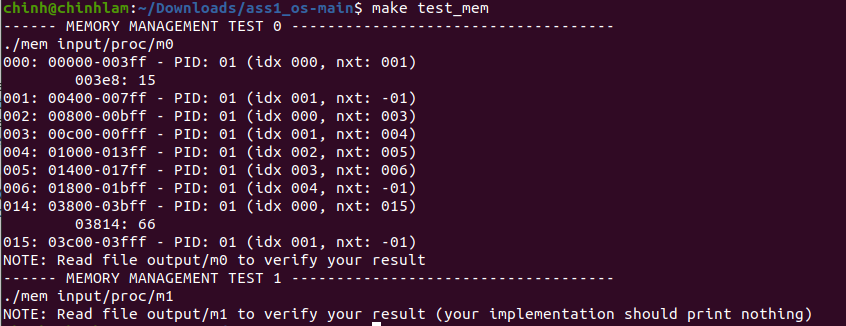
* Hàm alloc memory (cấp phát vùng nhớ)



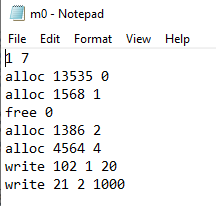
* Hàm giải phóng vùng nhớ (free memory)



***Kết quả khi chạy code:***

******

Với file input m0:



PAGE\_SIZE là 1024

* Độ ưu tiên (priority) của process là 1, số lệnh thực hiện là 7.
* Với dòng lệnh **alloc 13535 0** thì:

Sẽ cấp phát 14 trang \_mem\_stat (từ 000 -> 013) và lưu địa chỉ của byte đầu tiên được cấp phát vào thanh ghi số 0.

* Với dòng lệnh **alloc 1568 1** thì:

Sẽ cấp phát 2 trang \_mem\_stat (từ 014->015) vì 13 trang đầu được cấp phát với lệnh trước đó, và lưu địa chỉ của byte đầu tiên được cấp phát vào thanh ghi số 1.

* Với dòng lệnh **free 0** thì:

Giải phóng các vùng nhớ được cấp phát từ lệnh alloc với thanh ghi số 0 (từ 000->013) và do đó lúc này chỉ còn lại trang 014 và 015.

* Với dòng lệnh **allooc 1386 2** thì:

Sẽ cấp phát 2 trang \_mem\_stat (từ 000->001) và lưu địa chỉ của byte đầu tiên được cấp phát vào thanh ghi số 2.

* Với dòng lệnh **alloc 4564 4** thì:

Sẽ cấp phát 5 trang \_mem\_stat (từ 002->006) và lưu địa chỉ của byte đầu tiên được cấp phát vào thanh ghi số 4.

* Với dòng lệnh **write 102 1 20** thì:

Sẽ viết giá trị 102 vào vị trí có địa chỉ bằng với địa chỉ của thanh ghi số 1 cộng với offset là 20.

Lúc này thanh ghi 1 lưu địa chỉ như kết quả chạy thử là 03800, cộng với offset chuyển từ hex sang decimal thì 20->14 và do đó sẽ là 03814.

Giá trị 102 chuyển từ hex sang decimal là 66.

Do đó kết quả như trên là **03814: 66**

* Với dòng lệnh write 21 2 1000 thì:

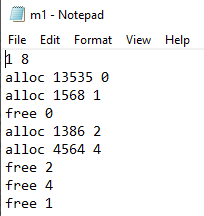
Sẽ viết giá trị 21 vào vị trí có địa chỉ bằng với địa chỉ của thanh ghi số 2 cộng với offset là 1000.

Lúc này thanh ghi 2 lưu địa chỉ như kết quả chạy thử là 00000, cộng với offset chuyển từ hex sang decimal thì 1000-> 3e8 và do đó sẽ là 003e8.

Giá trị 21 chuyển từ hex sang decimal là 15.

Do đó kết quả như trên là **003e8: 15**

Với file input m1:



PAGE\_SIZE là 1024

* Độ ưu tiên (priority) của process là 1, số lệnh thực hiện là 7.
* Với dòng lệnh **alloc 13535 0** thì:

Sẽ cấp phát 14 trang \_mem\_stat (từ 000 -> 013) và lưu địa chỉ của byte đầu tiên được cấp phát vào thanh ghi số 0.

* Với dòng lệnh **alloc 1568 1** thì:

Sẽ cấp phát 2 trang \_mem\_stat (từ 014->015) vì 13 trang đầu được cấp phát với lệnh trước đó, và lưu địa chỉ của byte đầu tiên được cấp phát vào thanh ghi số 1.

* Với dòng lệnh **free 0** thì:

Giải phóng các vùng nhớ được cấp phát từ lệnh alloc với thanh ghi số 0 (từ 000->013) và do đó lúc này chỉ còn lại trang 014 và 015.

* Với dòng lệnh **allooc 1386 2** thì:

Sẽ cấp phát 2 trang \_mem\_stat (từ 000->001) và lưu địa chỉ của byte đầu tiên được cấp phát vào thanh ghi số 2.

* Với dòng lệnh **alloc 4564 4** thì:

Sẽ cấp phát 5 trang \_mem\_stat (từ 002->006) và lưu địa chỉ của byte đầu tiên được cấp phát vào thanh ghi số 4.

* Với dòng lệnh **free 2** thì:

Giải phóng các vùng nhớ được cấp phát từ lệnh alloc với thanh ghi số 2 (từ 000->001 và do đó lúc này chỉ còn lại trang từ 002->006 và 014->015.

* Với dòng lệnh **free 4** thì:

Giải phóng các vùng nhớ được cấp phát từ lệnh alloc với thanh ghi số 4 (từ 002->006) và do đó lúc này chỉ còn lại trang 014 và 015.

* Với dòng lệnh **free 1** thì:

Giải phóng các vùng nhớ được cấp phát từ lệnh alloc với thanh ghi số 1 (từ 014->015) và các vùng nhớ đều đã được giải phóng.

Do đó kết quả in ra là không có gì cả như kết quả đã test thử ở trên.

**Question:**

In which system is segmentation with paging used (give an example of at least one system)? Explain clearly the advantage and disadvantage of segmentation with paging.

Trả lời:

* Hệ điều hảnh sử dụng cơ chế segentation with paging: họ hệ điều hành linux
* Ưu điểm:
  + Không xảy ra phân mảnh ngoại (external fragmentation)
  + Giảm kích thước của page table
  + Toàn bộ segment không cần phải swap out
* Nhược điểm:
  + Có thể xảy ra phân mảnh nội (internal fragmentation)
  + Phức tạp hơn về phần cứng so với segmented paging
  + Nhiều level sẽ dẫn tới delay lớn hơn khi truy xuất bộ nhớ

1. **Put It All Together:**