Bài tập lớn số 2

Giả lập bộ nhớ ảo

Ghi chú:

- Sinh viên nộp bài tại trang web: `www.cse.hcmut.edu.vn/portal`
- File nộp bài đặt tên là: *`ass2.tar.bz2`*
- SV có thể chỉnh sửa source code mẫu nếu thấy cần thiết
- Mọi gian lận sẽ nhận điểm KHÔNG nếu bị phát hiện

1 Giới thiệu

1.1 Mục tiêu

Viết một chương trình C trên Linux để giả lập giải thuật thay trang **LRU** trong quản lý bộ nhớ ảo

1.2 Kiến thức cần biết

- Các giải thuật phân trang
- Lập trình C trên Linux
- Makefile

1.3 Mô tả chương trình

Đầu vào (Input) của chương trình:

- Cấu hình hệ thống: gồm thông tin về bộ nhớ chính
- Đặc tả các hoạt động truy cập bộ nhớ trong hệ thống, chẳng hạn: kích thước bộ nhớ cần cấp phát cho một process mới, process đó truy cập bộ nhớ ở địa chỉ luận lý nào, process nào vừa kết thúc

Đầu ra (Output) của chương trình:

- Ghi lại các hoạt động xử lý trên bộ nhớ chính: cấp phát frame, truy cập địa chỉ vật lý, load dữ liệu từ đĩa cứng vô, đưa dữ liệu từ bộ nhớ chính ra đĩa cứng, ...

1.4 Chương trình sườn

1.4.1 Cấu trúc chương trình sườn

Gồm có 3 phần: `common, fileop, memop`:

- Phần common**:** Chứa thông tin đặc tả các cấu trúc được dùng trong việc hiện thực quá trình quản lý bộ nhớ ảo.
- Phần fileop**:** Hiện thực các chức năng liên quan đến tương tác file (đọc file input, đọc file cấu hình, xuất ra file output).
- Phần memop: Hiện thực các chức năng liên quan đến bộ nhớ: cấp phát bộ nhớ, truy cập địa chỉ luận lý, quản lý cấu trúc bộ nhớ, ...

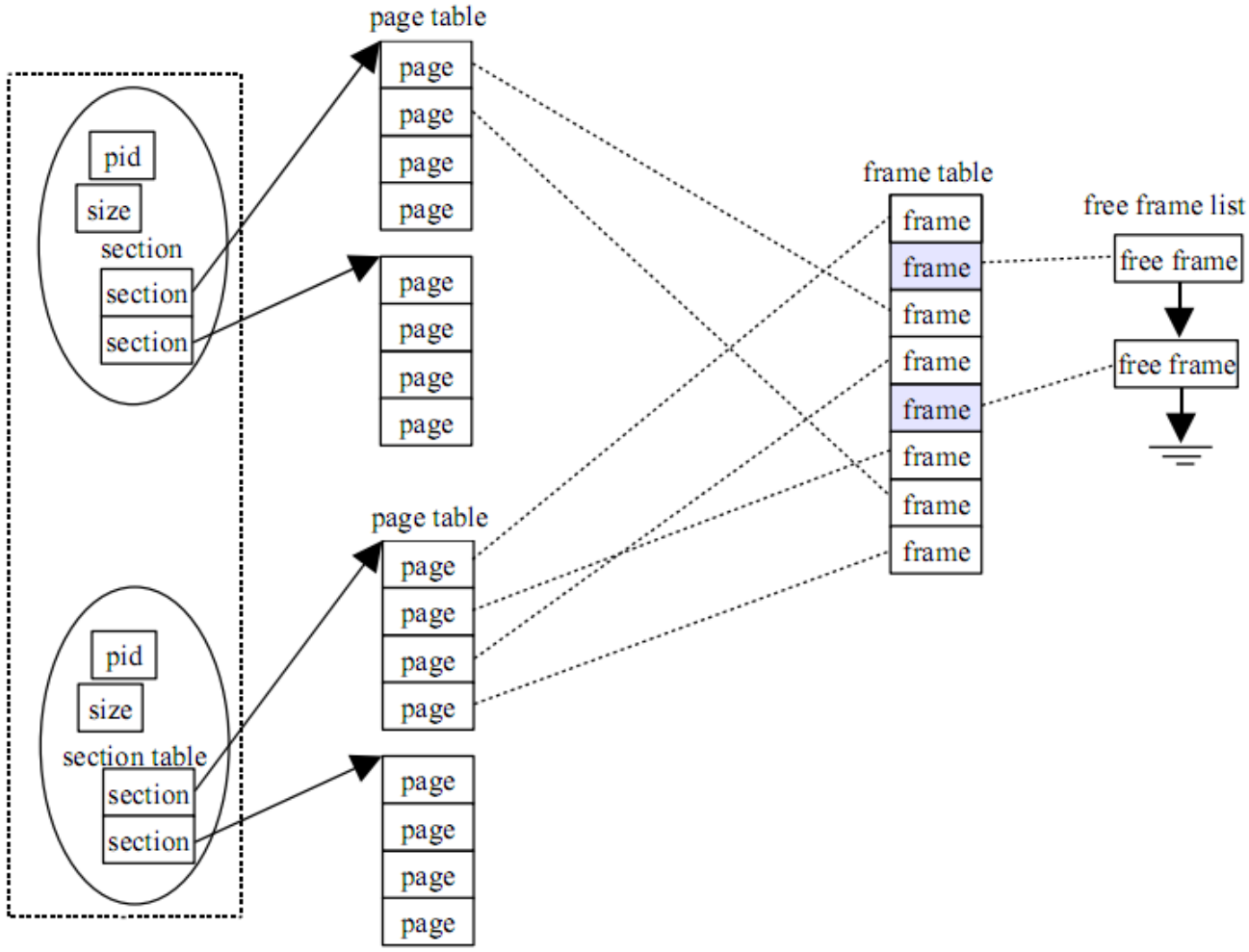
1.4.2 Ý tưởng hiện thực chương trình

Các process đi vào hệ thống được quản lý bởi một danh sách liên kết. Khi có một process mới đi vào hệ thống thì process đó được đưa vào danh sách theo thứ tự pid của chúng.

Mỗi process chứa các thông tin liên quan đến process (process id), và các thông tin phục vụ cho việc cấp phát trang nhớ (kích thước bộ nhớ process yêu cầu, và bảng phân trang hiện tại của process đó).

Trong đề bài yêu cầu, bảng phân trang có 2 mức, do đó ta biểu diễn bảng phân trang thành 2 mảng liên kết với nhau như hình vẽ. Mỗi entry trong bảng section (mức 1) trỏ đến một mảng các trang (mức 2). Mỗi trang trong mảng mức 2 chứa các thông tin về tình trạng của trang đó (đang sử dụng frame nào, trạng thái có valid hay không).

Để quản lý tình trạng đang sử dụng của các ô nhớ, ta sử dụng một bảng các frame, mỗi frame biểu diễn cho một ô nhớ tương ứng trong bộ nhớ chính. Khi một trang nào đó của bảng phân trang trỏ đến một ô nhớ trong bộ nhớ chính, thực sự ta cho nó trỏ đến frame tương ứng trong bảng frame. Để tiện việc truy hồi trang nào đang trỏ đến, frame có lưu giữ thông tin về pid của process và số thứ tự luận lý của trang đó trong bảng phân trang.



Để tiện trong việc tìm kiếm một frame trống cấp phát cho process, ta lưu các frame trống này dưới dạng danh sách liên kết. Khi cần tìm một frame trống, ta chỉ việc lấy frame đầu tiên trong danh sách này và cấp phát cho process đó.

Kể từ khi một process đi vào hệ thống đến khi nó kết thúc có thể được chia ra làm 3 giai đoạn: giai đoạn process vừa vào hệ thống và yêu cầu cấp phát vùng nhớ đê chạy, giai đoạn chạy chương trình, truy cập vào một địa chỉ ô nhớ nào đó, và giai đoạn kết thúc quá trình.

- Giai đoạn process vừa vào hệ thống: Process yêu cầu hệ thống cấp phát vùng nhớ để chạy, lúc này hệ thống sẽ cố gắng cấp phát tất cả các vùng nhớ mà process yêu cầu. Nếu trong hệ thống còn frame trống thì hệ thống ưu tiên cấp phát frame trống trước bằng cách lấy frame đầu tiên trong danh sách *free frame list* và map frame này với ô nhớ tương ứng với process, loại bỏ frame này ra khỏi danh sách *free frame list*. Nếu không còn frame trống để cấp phát, hệ thống sẽ lấy các frame đang được sử dụng bởi các process khác để cấp phát cho process này. Giải thuật thay trang được hiện thực ở đây, thể hiện việc chọn ra frame nào trong danh sách các frame của hệ thống. Sau khi chọn frame để thay thế, hệ thống swap out frame đó ra ngoài đĩa cứng, load nội dung của trang nhớ tương ứng vào frame đó, sau đó map frame đó với trang nhớ tương ứng bằng cách set trang đó thành *valid* và set trang của process lúc trước trỏ đến trang này thành *invalid*.

  Nếu một process yêu cầu hệ thống cấp phát vượt quá số tài nguyên mà hệ thống đang có, chương trình sẽ dùng giải thuật thay trang để thay các trang của chính process đó.

- Giai đoạn process truy cập một địa chỉ ô nhớ: Dựa vào địa chỉ này (địa chỉ luận lý), process biết được nó sẽ truy cập vào trang nào trong bảng phân trang, do đó nó có thể lấy được thông tin của trang này.
  - Nếu trạng thái của trang này là *valid* (trang hợp lệ), thông tin frame chứa trong trang này sẽ trỏ đến số thứ tự của frame trong bộ nhớ chính, từ đó process có thể truy cập vào địa chỉ thực trên bộ nhớ chính (địa chỉ vật lý).
  - Nếu trạng thái của trang này là *invalid* (trang không hợp lệ), điều này có nghĩa là trang này không trỏ đến frame nào trong bộ nhớ chính. Lúc này hệ thống sẽ cấp phát cho process này một frame và map frame này với trang tương ứng của process đó. Quá trình cấp phát một frame mới cũng tương tự như ở trên (ưu tiên cấp phát frame trống trước).

  Nếu process truy cập vào một địa chỉ không hợp lệ (vượt quá vùng địa chỉ mà nó xin cấp phát), hệ thống sẽ gây ra lỗi SEGFAULT và giải phóng tất cả các tài nguyên liên quan đến process đó.

- Giai đoạn kết thúc process: Hệ thống giải phóng tất cả các tài nguyên liên quan đến process này gồm bảng phân trang, frame mà process này đang trỏ đến, vùng nhớ chứa process này.

1.5 Một số kỹ thuật lập trình

1.5.1 Xử lý thông số nhập vào từ chương trình

Một chương trình tốt thường cho phép người dùng thiết lập một vài thông số khi chạy chương trình, chẳng hạn khi thực hiện lệnh:

```
$ ls -R
```

người dùng đã truyền vào thông số `-R` để liệt kê các file và thư mục không chỉ trong thư mục hiện hành mà còn cả những thư mục con của thư mục hiện hành nếu có.

Sau đây là đoạn chương trình mẫu, sử dụng hàm `getopt()` để xử lý thông số -R ở trên:

```
int opt;
extern char *optarg;

while ((opt = getopt(argc, argv, "R")) != EOF) {
    switch (opt) {
        case 'R':
            // Option -R occurs
            // Proccess that option here
            break;
        default:
            // Other options
            break;
    }
}
```

1.5.2 Xử lý file cấu hình

File cấu hình thường ở dạng text và có cấu trúc (đơn giản). Ta thường sử dụng các hàm sau để xử lý những dạng file này:

- fopen: mở một file
- fclose: đóng file đã mở
- fscanf: đọc/lọc thông tin một dòng trong file
- fprintf: ghi thông tin lên file

2 Yêu cầu

Hiện thực giải thuật thay trang ***LRU***.

Chương trình sau khi biên dịch có tên là *memsim*, hỗ trợ các thông số sau:

- `-h`: Hiển thị thông tin hướng dẫn sử dụng chương trình
- `-s` *sysfile*: Chọn file cấu hình hệ thống. Đây là file chứa các thông tin về hệ thống. Tên file hệ thống mặc đinh là *sys.conf*
- `-i` *infile*: Chọn file input. Đây là file chứa thông tin các tác vụ tác động lên hệ thống. Tên file input mặc định là *input.txt*
- `-o` *outfile*: Chọn file output. Đây là file chứa kết quả xử lý trên bộ nhớ chính. Tên file output mặc định là *output.txt*

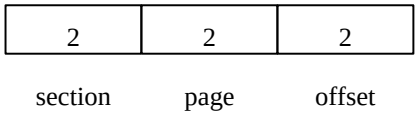
Cú pháp chạy chương trình `memsim`:

`memsim [-h] [-s` *sysfile*`] [-i` *infile*`] [-o` *outfile*`]`

3 Định dạng file input, output

Mô tả hệ thống

Hệ thống sử dụng 6 bit địa chỉ, phân trang theo 2 mức, trong đó 2 bit đầu dành cho phần section (bảng phân trang mức 1), 2 bit tiếp theo dành cho phần page (bảng phân trang mức 2), 2 bit cuối cùng là phần offset. Định dạng địa chỉ của hệ thống được mô tả như hình dưới:



Vì hệ thống sử dụng 6 bit địa chỉ nên không gian bộ nhớ ảo mà một process có thể có được là 64 bytes. Một process không thể xin cấp phát một vùng nhớ lớn hơn kích thước này.

File cấu hình hệ thống

```
32
// Dòng trống
```

File này mô tả cấu hình hệ thống. Hiện tại đề bài chỉ cho mô tả kích thước bộ nhớ thực (tính theo byte). Đối với ví dụ trên, file này mô tả hệ thống của chúng ta có 32 bytes bộ nhớ thực.

File input

Mỗi dòng mô tả một yêu cầu của process, yêu cầu này có thể là xin cấp phát vùng nhớ lúc quá trình vừa khởi tạo, truy cập vào một địa chỉ luận lý, kết thúc quá trình. Một yêu cầu được ghi trên 1 dòng, các thông tin cách nhau bởi khoảng trắng. Một yêu cầu có thể thuộc một trong ba dạng sau:

- `START pid size`: process *pid* vừa vào hệ thống và yêu cầu hệ thống cấp phát một vùng nhớ là *size* bytes
- `ACCESS pid addr`: process *pid* truy cập địa chỉ luận lý *addr*
- `FINISH pid num`: process *pid* kết thúc quá trình, trong trường hợp này *num* là một số bất kỳ và không được sử dụng, tuy nhiên ta để số này vào cho việc xử lý file input được dễ dàng (có cùng định dạng với 2 tác vụ ở trên)

```
START 0 2   // Không có ký tự trống sau số 2
ACCESS 0 1
FINISH 0 0
            // Dòng trống
```

Với file input mẫu ở trên, chương trình sẽ hiểu như sau: process 0 đi vào hệ thống, yêu cầu hệ thống cấp phát 2 bytes, sau đó truy cập vào địa chỉ luận lý số 1, sau đó kết thúc chương trình.

File output

Xuất thông tin quá trình tương tác lên bộ nhớ chính và các hoạt động quản lý trang nhớ trên mỗi process. Mỗi dòng kết quả trong file output thuộc một trong những dạng sau:

- `ALLOC pid section page frame`: hệ thống cấp frame trống frame tại trang (*section, page*) cho process *pid*. Chú ý trước khi cấp phát một trang nhớ cho process, hệ thống load dữ liệu tương ứng từ đĩa cứng vào trang *frame*.
- `ACCESS pid addr`: process *pid* truy cập vào địa chỉ vật lý *addr*
- `SEGFAULT pid`: process *pid* truy cập vào địa chỉ không hợp lệ. Sau khi xuất dòng này, process *pid* được giải phóng và hệ thống lấy lại tất cả các tài nguyên liên quan đến process này.
- `INVALIDPAGE pid section page`: khi process *pid* truy cập vào địa chỉ luận lý thuộc trang (*section, page*), và trang này đang có trạng thái là *invalid*. Sau khi xuất dòng này, hệ thống bắt đầu thực hiện quá trình thay trang (tìm trang để thay, swap out nội dung frame ra đĩa cứng, load nội dung vào bộ nhớ, set lại trạng thái trang thành *valid*)
- `SWAPOUT frame`: swap out nội dung của frame ra đĩa cứng để dùng frame này cho các mục đích khác.
- `LOADTO frame pid addr`: load trang dữ liệu từ đĩa cứng tương ứng với địa chỉ luận lý *addr* của quá trình *pid* vào frame *frame*
- `SETVALID pid section page frame`: Cập nhật trang nhớ (*section, page*) của quá trình *pid* sang trạng thái *valid* và map nó với frame *frame*
- `FREE frame`: giải phóng frame *frame*, đưa frame này vào cuối danh sách *free frame list* của hệ thống. Chú ý trước khi giải phóng frame nhớ, hệ thống ghi nội dung của frame đó vào đĩa cứng.

Từ các phân tích ở trên, ta được kết quả của chương trình như sau:

```
LOADTO 0 0 0
ALLOC 0 0 0 0
ACCESS 0 1
SWAPOUT 0
FREE 0
                // Dòng trống
```

4 Tài liệu tham khảo

[1] *GCC Manuals*, http://gcc.gnu.org/onlinedocs/gcc-4.2.3/gcc/

[2] *C programming tutorial*, http://www.cprogramming.com/tutorial.html

[3] *C programming tutorial*, http://www.iu.hio.no/~mark/CTutorial/CTutorial.html

[5] *Tutorial – Make file*, http://www.opussoftware.com/tutorial/TutMakefile.htm

[6] *GNU Make*, http://www.gnu.org/software/make/manual/make.html

[7] *Man page*

[8] A. Silberschatz, P. B. Galvin, G. Gagne, 2006, *Operating System Principles, 7th Edition*, John Wiley & Sons Inc.