>>> LAB TUTORIAL <<<

INTRODUCTION TO OPERATING SYSTEM / Course ID 502047

Lab Part 10 – Further topics

Mục tiêu	Lý thuyết liên quan	Tài nguyên
s		Sử dụng image Ubuntu 14 / 16
Contiguous Memory Allocation		
Dynamic Allocation		

Yêu cầu nộp bài: các tập tin mã nguồn .c và tập tin khả thực thi .out của các "ví dụ" và bài tập cuối hướng dẫn.

Preferences

[1] Abraham Silberschatz, Peter B. Galvin, Greg Gagne, [2018], Operating System Concepts, 10th edition, John Wiley & Sons, New Jersey.

Programming Problems of Chapter 6 and 7.

1. Disk scheduling

1. Viết chương trình thực hiện các thuật toán lập lịch đĩa sau đây:

a. FCFS

b. SCAN

c. C-SCAN

Chương trình phục vụ một đĩa có 5000 hình trụ được đánh số từ 0 đến 4999 (con số này có thể thay đổi và được xác định bằng giá trị ghi ở dòng đầu tiên trong tập tin chứa thông tin truy xuất được truyền vào khi thực hiện lời gọi. Chương trình sẽ tiếp tục đọc các yêu cầu ở từng dòng trong tập tin và phục vụ chúng theo từng thuật toán cũng được truyền vào khi gọi thực thi chương trình. Giả định rằng đầu đọc ban đầu nằm ở cylinder 0 và đọc theo chiều đi ra (với C-SCAN). Chương trình cần báo cáo tổng số bước chuyển động đầu theo yêu cầu của mỗi thuật toán.

Nội dung tập tin disk.txt

4999 35 690 1569 4	Cylinder lớn nhất (cylinder nhỏ nhất = 0). Truy cập đĩa đầu tiên Truy cập đĩa thứ hai …..

>./diskscheduler.out FCFS disk.txt

2. Viết chương trình sinh ra một số lượng truy cập đĩa ngẫu nhiên từ 2 tham số truyền vào khi gọi là số cylinder của đĩa và số lượng truy cập cần sinh ngẫu nhiên. Ví dụ lời gọi sau đây sẽ sinh ra tập tin disk.txt dòng đầu tiên mang giá trị 4999 và 1000 dòng tiếp theo mang các giá trị ngẫu nhiên từ 0 đến 4999.

>./diskrandom.out 5000 1000 disk.txt

3. Viết chương trình đa luồng mà mỗi tiểu trình con sẽ gọi từng giải thuật định thời đĩa. Tiến trình cha sau đó sẽ nhận các thống kê tổng số bước chuyển động của đầu đọc và cho biết giải thuật nào là tối ưu.

2. Contiguous Memory Allocation

Trong chương 9.2, các thuật toán khác nhau để cấp phát bộ nhớ liền kề đã được trình bày. Chương trình này sẽ liên quan đến việc quản lý một vùng bộ nhớ liền kề có kích thước MAX, trong đó các địa chỉ có thể nằm trong khoảng từ 0 đến MAX - 1.

Chương trình có 4 tiến trình con thực hiện 4 nhiệm vụ sau đây trên một không gian nhớ chia sẻ.

1. Yêu cầu cấp phát khối bộ nhớ liền kề.

2. Giải phóng một khối bộ nhớ liền kề.

3.Thu gọn các lỗ bộ nhớ không sử dụng thành một khối duy nhất.

4. Báo cáo các vùng bộ nhớ trống và vừng đã được cấp phát.

5. Kết thúc chương trình.

Dung lượng bộ nhớ giả lập được truyền vào khi gọi chương trình. Ví dụ: lệnh sau khởi tạo chương trình với bộ nhớ 1 MB (1.048.576 byte):

./allocator 1048576

Sau đó, chương trình sẽ hiển thị dấu nhắc cho người dùng chọn tác vụ, và sau khi hoàn thành tác vụ thì lại quay về dấu nhắc cho đến khi nào người dùng chọn tác vụ "Kết thúc chương trình".

allocator>

Người dùng có thể gọi các lệnh sau: RQ (yêu cầu), RL (giải phóng), C (thu gọn), STAT (thống ke) và X (thoát). Ví dụ: một yêu cầu 40.000 byte sẽ được gọi như sau:

allocator>RQ P0 40000 W

Tham số đầu tiên trong lệnh RQ là tiến trình mới yêu cầu bộ nhớ, theo sau là lượng bộ nhớ được yêu cầu và cuối cùng là chiến lược cấp phát. (Trong ví dụ trên, W là chiến lược tồi tệ nhất [Worst Fit].)

Tương tự, một yêu cầu giải phóng sẽ được gọi:

allocator>RL P0

Lệnh này sẽ giải phóng bộ nhớ đã được cấp phát cho tiến trình P0.

Khi bộ nhớ bị phân mảnh ngoại, người dùng có thể gom các lỗ trống lại thành một lỗ trống lớn, lệnh sau đây sẽ được gọi:

allocator>C

Cuối cùng, lệnh STAT để báo cáo trạng thái của bộ nhớ được gọi dưới dạng:

allocator> STAT

Chương trình sẽ báo cáo các vùng bộ nhớ được cấp phát và các vùng không được sử dụng.

Ví dụ, một sự sắp xếp có thể cấp phát bộ nhớ sẽ như sau:

Addresses [0:315000] Process P1

Addresses [315001: 512500]

Process P3

Addresses [512501:625575]

Unused Addresses [625575:725100]

Process P6 Addresses [725001] . . .

Cấp phát bộ nhớ

Chương trình sẽ cấp phát bộ nhớ bằng một trong ba cách tiếp cận được giới thiệu trong chương 9.2.2, tùy thuộc vào tham số được truyền cho lệnh RQ. Các lựa chọn có thể là

• F—first fit

• B—best fit

• W—worst fit

Chương trình cần theo dõi các lỗ trống khác nhau thể hiện bộ nhớ khả dụng. Khi một yêu cầu cấp phát bộ nhớ đến, nó sẽ cấp phát bộ nhớ từ một trong các lỗ có sẵn tùy vào chiến lược được chỉ định (First fit, Best fit, Worst fit). Nếu không đủ bộ nhớ để cấp phát cho một yêu cầu, nó sẽ xuất ra một thông báo lỗi và từ chối yêu cầu.

Chương trình cũng sẽ cần theo dõi vùng bộ nhớ nào đã được cấp phát cho quá trình nào. Điều này là cần thiết để hỗ trợ lệnh STAT. Những thay đổi diễn ra trong bộ nhớ giả lập bao gồm: lệnh RQ nếu thành công sẽ cung cấp tên tiến trình, kích thước của nó và kích thước lỗ trống còn lại sau khi cấp phát, lệnh RL sẽ thu hồi vùng nhớ đã cấp cho tiến trình, và nếu vùng nhớ thu hồi liền kề một lỗ trống hiện hữu, hãy chắc chắn chúng sẽ được kết hợp thành một lỗ duy nhất.

Dọn phân mảnh

Nếu người dùng nhập lệnh C, chương trình sẽ nén tập hợp các lỗ thành một lỗ lớn hơn. Ví dụ: nếu bộ nhớ có bốn lỗ riêng biệt có kích thước 550 KB, 375 KB, 1900 KB và 4500 KB, chương trình sẽ kết hợp bốn lỗ này thành một lỗ lớn có kích thước 7325 KB.

Có một số chiến lược để thực hiện dọn phân mảnh, có thể tham khảo ở chương 9.2.3. Các tiến trình đã được cấp phát cũng cần cập nhật lại địa chỉ bộ nhớ.

3. Designing a Virtual Memory Manager

This project consists of writing a program that translates logical to physical addresses for a virtual address space of size 216 = 65,536 bytes. Your program will read from a file containing logical addresses and, using a TLB and a page table, will translate each logical address to its corresponding physical address and output the value of the byte stored at the translated physical address. Your learning goal is to use simulation to understand the steps involved in translating logical to physical addresses. This will include resolving page faults using demand paging, managing a TLB, and implementing a page-replacement algorithm. Specific Your program will read a file containing several 32-bit integer numbers that represent logical addresses. However, you need only be concerned with 16- bit addresses, so you must mask the rightmost 16 bits of each logical address. These 16 bits are divided into (1) an 8-bit page number and (2) an 8-bit page offset. Hence, the addresses are structured as shown as:

offset 31 1516 78 0 page number

Other specifics include the following:

• 28 entries in the page table

• Page size of 28 bytes

• 16 entries in the TLB

• Frame size of 28 bytes

• 256 frames

• Physical memory of 65,536 bytes (256 frames × 256-byte frame size)

Ngoài ra, chương trình chỉ cần quan tâm đến việc đọc địa chỉ logic và dịch chúng sang địa chỉ vật lý tương ứng. Chương trình không cần phải hỗ trợ ghi vào không gian địa chỉ logic.

Dịch địa chỉ

Chương trình của bạn sẽ dịch logic sang các địa chỉ vật lý bằng cách sử dụng bảng TLB và bảng phân trang đã được giới thiệu trong chương 9.3. Đầu tiên, số trang được trích xuất từ địa chỉ logic và TLB được truy vấn. Trong trường hợp tìm thấy trong TLB, số khung được lấy từ TLB. Trong trường hợp trượt TLB, bảng phân trang phải được truy vấn; khi đó, số khung trang được lấy từ bảng phân trang hoặc sẽ xảy ra lỗi trang. Sơ đồ mô tả quá trình dịch địa chỉ được cho như sau:

Xử lý lỗi trang

Chương trình sẽ triển khai phân trang theo yêu cầu như được mô tả trong Chương 10.2. Vùng nhớ sao lưu [backing store] được hiện thực bằng tập tin BACKING_STORE.bin, một tập tin nhị phân có kích thước 65.536 byte. Khi xảy ra lỗi trang, chương trình sẽ đọc trong trang 256 byte từ tập tin BACKING_STORE.bin và lưu nó trong một khung trang có sẵn trong bộ nhớ vật lý. Ví dụ: nếu một địa chỉ logic có số trang 15 gây ra lỗi trang, chương trình sẽ đọc ở trang 15 từ BACKING_STORE.bin (hãy nhớ rằng các trang bắt đầu từ 0 và có kích thước 256 byte) và lưu trữ nó trong một khung trang trong bộ nhớ vật lý. Khi khung này được lưu trữ (và bảng phân trang và TLB được cập nhật), các lần truy cập tiếp theo vào trang 15 sẽ được phân giải bằng TLB hoặc bảng phân trang.

Tập tin BACKING_STORE.bin là một tập tin truy cập ngẫu nhiên để có thể ngẫu nhiên tìm đến các vị trí nhất định của tập tin để đọc. Chúng tôi khuyên bạn nên sử dụng các hàm thư viện C tiêu chuẩn để thực hiện I / O, bao gồm fopen(), fread(), fseek() và fclose(). Kích thước của bộ nhớ vật lý giống như kích thước của không gian địa chỉ ảo. 65,536 byte byte, do đó bạn không cần phải lo lắng về việc thay thế trang trong một lỗi trang. Sau đó, chúng tôi mô tả một sửa đổi cho dự án này bằng cách sử dụng một lượng bộ nhớ vật lý nhỏ hơn; tại thời điểm đó, một chiến lược thay thế trang sẽ được yêu cầu.

You will need to treat BACKING STORE.bin as a random-access file so that you can randomly seek to certain positions of the file for reading. We suggest using the standard C library functions for performing I/O, including fopen(), fread(), fseek(), and fclose(). The size of physical memory is the same as the size of the virtual address space—65,536 bytes—so you do not need to be concerned about page replacements during a page fault. Later, we describe a modification to this project using a smaller amount of physical memory; at that point, a page-replacement strategy will be required.