TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN HỆ THỐNG THÔNG TIN

BÁO CÁO ĐỒ ÁN
HỆ ĐIỀU HÀNH

Đồ Án 3: ĐA CHƯƠNG VÀ LẬP LỊCH

Lớp: 22CLC06 – Nhóm: THPT

Giáo viên hướng dẫn: Thầy Lê Viết Long

Tp. Hồ Chí Minh, tháng 4/2024

MỤC LỤC

A. Giới thiệu đồ án:

I. Mô tả đồ án:

Bằng cách sử dụng hệ điều hành đơn giản chưa xử lý nhiều (NachOS) kết hợp với các kiến thức đã học về đa tiến trình, đồng bộ hóa và lập lịch, nhóm chúng em tạo ra chương trình có thể chạy đa chương, lập lịch và đồng bộ hóa kèm theo một số chương trình minh họa.

Toàn bộ đồ án được thực hiện trên môi trường phát triển tích hợp VSCode, khởi động hệ điều hành ảo NachOS bằng VMWare và tài liệu được cung cấp từ giảng viên cùng một số tài liệu tham khảo ngoài.

II. Mục đích của đồ án:

Đồ án này được đề ra nhằm mục đích tìm hiểu cách thức hệ điều hành nhập, xuất file, tìm hiểu các vấn đề về đa chương, lập lịch và đồng bộ hóa và vận dụng chúng trong các chương trình minh họa.

Cụ thể, đồ án gồm:

- Thiết lập các system call để chuyển đổi hệ thống NachOS từ đơn chương thành đa chương. Bao gồm quá trình viết các hàm: Join, Exec, Exit, CreateSemaphore, Up, Down, Fork, Yield,…
- Tạo các chương trình minh họa Ping Pong (cho phép chương trình chạy tuần tự xen kẽ nhau ở mỗi bước xuất ra màn hình) và thống kê sử dụng máy quét (cho phép nhiều chương trình thực hiện cùng lúc).

Tóm lại, thông qua đồ án, chúng em hiểu rõ hơn về cách hệ điều hành thực hiện đa chương trình và ứng dụng kiến thức này trong các chương trình minh họa.

B. Chi tiết đồ án:

I. Các system call về đa chương, lập lịch và đồng bộ hóa trong NachOS:

1. SC_Join:

System call này được sử dụng để nhập tiến trình con vào tiến trình cha.

System call này hoạt động bằng cách lấy id của tiến trình con từ thanh ghi R4, kiểm tra nó có nằm trong bảng processTab không, nếu có thì đợi tiến trình con hoàn thành và gọi system call SC_Exit với exitcode là 0 để xóa tiến trình con khỏi processTab. Nếu nhập thành công thì trả về không, ngược lại sẽ trả về -1.

2. SC_Exec:

System call này được sử dụng để khởi chạy tiến tiến trình con từ tiến trình cha.

System call này hoạt động bằng cách lấy chuỗi chứa đường dẫn đến file thực thi từ thanh ghi R4, kiểm tra có tồn tại file đó không, tiến trình cha có tự gọi lại chính nó không, bảng processTab còn vị trí trống không,… Nếu có thì khởi chạy tiến trình con và trả về 0, ngược lại sẽ trả về -1.

3. SC_Exit:

System call này được sử dụng để trả về exitcode của một tiến trình con cho tiến trình cha khi đã hoàn thành.

System call này hoạt động bằng cách lấy exitcode từ thanh ghi R4, kiểm tra tiến trình gọi system call này có tồn tại hay không,… Nếu tiến trình là tiến trình chính (nachos) thì Halt, nếu tiến trình là tiến trình thông thường và tiến trình cha đã gọi system call SC_Join thì gán exitcode vào tiến trình và xóa chương trình, nếu tiến trình cha chưa gọi thì đợi đến khi tiến trình cha gọi. Nếu thực hiện thành công thì trả về Exitcode cho tiến trình con, không thì trả về -1.

4. SC_CreateSemaphore:

System call này được sử dụng để tạo một semaphore giúp cho việc điều phối các tiến trình.

System call này hoạt động bằng cách lấy tên semaphore từ thanh ghi R4, giá trị khởi tạo của semaphore từ thanh ghi R5, sau đó kiểm tra xem semaphore có tồn tại hay chưa (có semaphore nào trong bảng semaphoreTable trùng tên không) và còn vị trí trống nào không, nếu có thì tạo semaphore lưu vào đó và trả về 0, nếu không thì trả về -1.

5. SC_Up:

System call này được sử dụng để giải phóng tiến trình đang chờ, tăng giá trị semaphore bằng cách gọi hàm S (Signal) trong lớp semaphore.

System call này hoạt động bằng cách lấy tên semaphore từ thanh ghi R4, sau đó kiểm tra xem tiến trình này có trong bảng semaphoreTable hay chưa rồi gọi phương thức Signal và trả về 0, ngược lại trả về -1.

6. SC_Down:

System call này được sử dụng để chờ bằng cách gọi hàm P (Wait) trong lớp semaphore nếu giá trị semaphore bị giảm xuống giá trị âm.

System call này hoạt động bằng cách lấy tên semaphore từ thanh ghi R4, sau đó kiểm tra xem tiến trình này có trong bảng semaphoreTable hay chưa rồi gọi phương thức Wait và trả về 0, ngược lại trả về -1.

7. SC_Fork:

System call này được sử dụng để khởi tạo các thông tin cần thiết cho một tiểu trình và đưa nó vào CPU để thực hiện.

System call này lấy giá trị con trỏ hàm của tiểu trình ở thanh ghi R4, sau đó gọi hàm Fork() của lớp Thread để đưa vào CPU.

System call này ban đầu được nhóm định nghĩa bởi vì có ý định dùng cho scanner nếu cần thiết.

8. SC_Yield:

System call này dùng để thread đang chạy trả CPU cho một thread khác trong ready list, bằng cách gọi hàm Yield() của lớp thread.

System call này ban đầu được nhóm định nghĩa bởi vì có ý định dùng cho chương trình máy quét nếu cần thiết.

9. SC_WriteInt2File:

System call này dùng để ghi một số nguyên dương vào một file được chỉ định.

Đầu tiên WriteInt2File lấy các giá trị là số nguyên dương cần ghi ở thanh ghi R4 và tham số id của file cần ghi ở thanh ghi R5.

Sau đó system call này sẽ chuyển đổi số nguyên dương vừa lấy được thành một mảng char* rồi dùng phương thức Write của lớp OpenFile để ghi mảng kí tự vào file đích.

System call này ban đầu được nhóm định nghĩa bởi vì có ý định dùng cho chương trình máy quét nếu cần thiết.

II. Cài đặt các chương trình minh họa:

1. Ping Pong:

Chương trình PingPong sẽ gọi system call SC_CreateSemaphore để khởi tạo 2 semaphore lần lượt là semaphore “First” (với giá trị là 1 để kiểm soát việc A chạy trước và chỉ được chạy trước B 1 lần) và semaphore “Second” (với giá trị là 0 để kiểm soát việc B chạy sau A).

Sau khi đã khởi tạo 2 semaphore trên, chương trình sẽ gọi system call SC_Exec để chạy 2 chương trình Ping và Pong.

Khi 2 chương trình chạy, 2 semaphore sẽ có các cặp giá trị (First, Second) tương ứng là (1, 0) và (0, 1) sau mỗi lần system call SC_Up được gọi.

Trước lệnh in ký tự A thì semaphore “First” sẽ giảm, còn trước lệnh in ký tự B thì semaphore “Second” sẽ giảm, kết hợp với giá trị được khởi tạo của 2 semaphores (và cặp giá trị semaphores sau SC_Up nói trên) sẽ đảm bảo được A và B sẽ được in xen kẽ với nhau và A được in đầu tiên (hay B được in cuối cùng).

2. Thống kê sử dụng máy quét:

Hiện tại thì nhóm chưa hoàn thành xong chương trình này.

C. Hướng dẫn sử dụng chương trình:

I. Cài đặt Cross-compiler:

- Để cài đặt được cross compiler thì yêu cầu cần phải cài đặt trước gcc-3.x trên hệ điều hành Linux.
- Tạo thư mục lưu bài tập hệ điều hành bằng lệnh: % mkdir hdh.
- Copy các tập tin sau vào thư mục hdh (có thể dùng WinSCP)
  - binutils-2.11.2.tar.gz
  - gcc-2.95.3.tar.gz.
- Vào thư mục hdh, giải nén các tập tin trên bằng các lệnh sau:
  - % cd hdh
  - % tar -xzvf binutils-2.11.2.tar.gz
  - % tar –xzvf gcc-2.95.3.tar.gz.
- Tạo thư mục chứa Cross-compiler sẽ được cài đặt, sử dụng lệnh:
  % mkdir cross-compiler
- Biên dịch binutils bằng các lệnh sau:
  - % cd binutils-2.11.2/
  - % ./configure --host=i686-pc-linux-gnu --target=decstation-ultrix --prefix=/root/hdh/cross-compiler
  - % make
  - % make install.
- Cài đặt binutils thành công sẽ tạo ra 6 file sau trong thư mục /root/hdh/cross-compiler/decstation-ultrix/bin/:
  ar as ld nm ranlib strip
- Trước khi tiến hành biên dịch gcc, cần tạo thư mục chứa kết quả biên dịch (thư mục này nên đặt cùng cấp với thư mục cross-compiler tạo ở trên):
  - % cd ..
  - % mkdir gcc-obj.

- Và tạo thư mục giả system-include

  % mkdir cross-compiler/decstation-ultrix/sys-include
- Vào thư mục gcc-obj, thực hiện cấu hình gcc bằng 2 lệnh sau:
  - % cd gcc-obj
  - % ../gcc-2.95.3/configure --host=i686-pc-linux-gnu -- target=decstation-ultrix --prefix=/root/hdh/cross-compiler --with-gnu-as --with-gnu-ld --with-as=/root/hdh/cross-compiler/decstation-ultrix/bin/as --with-ld=/root/hdh/cross-compiler/decstation-ultrix/bin/ld --enable-languages=c –disable-multilib --disable-libgcj
- Biên dịch gcc bằng 2 lệnh sau:
  - % make
  - % make -k install

II. Biên dịch và cài đặt Nachos:

- Copy tập tin sau vào thư mục hdh (có thể dùng WinSCP)

  nachos-3.4.tar.gz
- Giả sử đang làm việc tại thư mục /root. Ta vào thư mục hdh, và tiến hành giải nén các tập tin này bằng 2 lệnh:
  - % cd hdh
  - % tar -xzvf nachos-3.4.tar.gz
- Tiếp theo, ta cần đi tới thư mục code, sử dụng lệnh sau:

  % cd nachos-3.4/code
- Tiếp theo, ta sử dụng lệnh make hoặc make all để biên dịch các file cần thiết trong đồ án: % make hoặc % make all

III. Chạy chương trình trên Nachos:

- Khi đang ở thư mục code, ta di chuyển tới thư mục threads bằng lệnh: cd threads
- Từ đây, ta sử dụng lệnh sau:

  % ./nachos -rs 1234 -x ../test/[…], với […] là tên chương trình cần chạy.

D. Những vấn đề đã làm được và chưa làm được:

- Đã làm được:
  - Định nghĩa các system call theo yêu cầu của đồ án.
  - Thêm các thư viện cần thiết.
  - Chuyển hệ điều hành NachOS từ đơn chương thành đa chương.
  - Hoàn thành chương trình PingPong.
  - Sửa và hiểu được một phần của phần mã nguồn được cho sẵn trong quá trình chuyển từ đơn chương sang đa chương.
- Chưa làm được:
  - Hoàn thành chương trình thống kê sử dụng máy quét.
  - Tìm hiểu lí do khi khai báo con trỏ ListElement trong phương thức SortedRemove của lớp List (trong quá trình xóa mảng pcb) tạo ra lỗi segmentation fault (core dumped).

E. Đánh giá thành viên:

MSSV	Họ và tên	Tỷ lệ đóng góp
22127121	Đào Việt Hoàng	25%
22127320	Bùi Tá Phát	25%
22127432	Đặng Thanh Tú	25%
22127486	Đào Ngọc Thiện	25%

F. Mã nguồn, công cụ hỗ trợ, liên kết ngoài và tài liệu tham khảo:

I. Mã nguồn:

Đường dẫn tới source code GitHub: LancelotMoretti/NachOS-Development: OS developing NachOS project (github.com)

II. Các công cụ hỗ trợ trong quá trình thực hiện đồ án:

- VMware Workstation 17 Player
- Github Copilot

III. Tài liệu tham khảo:

Nhóm có tham khảo các file hướng dẫn do giáo viên cung cấp.

Nhóm có tham khảo 1 đồ án của các anh chị đối với SC_Up và SC_Down