VNU - University of Science

Faculty of Information Technology

BÁO CÁO ĐỒ ÁN

Đồ án 2 : NachOS - Đa chương

Instructors: Phạm Tuấn Sơn

Lê Viết Long

Group Members: Võ Phạm Thanh Phương - 21127677

Nguyễn Quốc Huy - 21127511

Lê Hoàng Sang - 21127158

Hệ điều hành

Second Semester - 2022 - 2023

Table of content

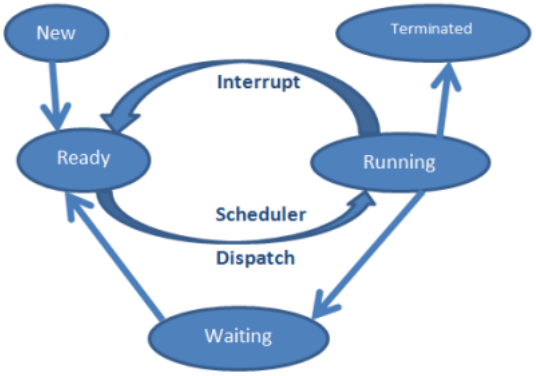
1 Thông tin thành viên

MSSV	Họ tên	Đóng góp
21127677	Võ Phạm Thanh Phương	100%
21127511	Nguyễn Quốc Huy	100%
21127158	Lê Hoàng Sang	100%

2 Thông tin chi tiết

2.1 Mô hình đa chương

- Hiện thời, chúng ta đang làm việc với hệ điều hành NachOS vốn chỉ hỗ trợ đơn chương và chưa được hỗ trợ system call để thực thi chương trình đa chương này. Do đó, ta cần phải thực hiện quản lý bộ nhớ, đồng thời đồng bộ hóa các tiến trình và threads của chương trình.



Hình 1: Đa chương

2.2 Mô hình giải quyết

- Trong lớp **Thread**, ta thực hiện cấp phát vùng nhớ và nạp các tiểu trình để quản lý các trạng thái hoạt động của tiến trình. Ta có:
  - **Addrspace *space:** để lưu vùng nhớ của tiến trình lên bộ nhớ ảo.
  - Hàm **void Fork(VoidFunctionPtr func, int arg)** để cấp phát bộ nhớ cho tiến trình cần sử dụng.

- Trong file **system.cc** và **system.h**
  - Thêm biến toàn cục, để quản lí, đánh dấu các bit đã được sử dụng hay chưa, bit đã được sử dụng gán 1, còn lại gán 0.

```
extern BitMap *bitmapPhysPage;
```

- Trong **machine.h**, ta thay đổi số trang từ 32 lên 255 với mục đích cung cấp thêm không gian trong bộ nhớ để việc thực hiện đa tiến trình không xảy ra trường hợp thiếu trang.
- Trong **addrspace.h** và **addrspace.cc**
  - Thêm biến **int spaceID** để lưu ID của các tiến trình phục vụ cho việc trả về thanh ghi.
  - Hàm **unsigned int InitializeSpaceID()** để khởi tạo một ID ngẫu nhiên tránh việc trùng lặp ID của các tiến trình (Ngoài ra, có thể kiểm soát các spaceID của các tiến trình bằng các hàm nâng cao hơn).
  - Thay đổi cách gán bảng trang vật lý từ **i** sang **bitmapPhysPage->Find()** với mục đích tìm các ô nhớ còn trống trong bộ nhớ vật lý.
  - Thêm các biến unsigned int để hỗ trợ việc copy segments:
    - **unsigned int numPages**: Số trang ảo trong bảng trang.
    - **unsigned int numCodePage**: Số trang cần thiết để lưu trữ mã máy (Code Page).
    - **unsigned int lastCodePageSize**: Kích thước của trang cuối cùng lưu trữ mã máy (Kích thước này có thể không bằng với **Page Size** nên việc khởi tạo một biến phân biệt là hoàn toàn cần thiết).
    - **unsigned int numDataPage**: Số trang lưu trữ dữ liệu khởi tạo (Data Page).
    - **unsigned int firstDataPageSize**: Kích thước của trang đầu tiên lưu trữ dữ liệu khởi tạo (Nếu **lastCodePageSize** không bằng kích thước với **Page Size** thì khả năng cao trang đầu tiên trong vùng Code Page **firstDataPageSize** cũng sẽ có kích thước khác với **Page Size**).
    - **unsigned int lastDataPageSize**: Kích thước của trang cuối cùng lưu trữ dữ liệu khởi tạo (Kích thước này có thể không bằng với **Page Size** nên việc khởi tạo một biến phân biệt là hoàn toàn cần thiết).

- Thực hiện copy data segments vào bộ nhớ:
  - Đầu tiên, ta tính số trang cần thiết để lưu trữ mã máy và dữ liệu khởi tạo, tính kích thước của trang cuối cùng lưu trữ mã máy và kích thước tạm thời của trang đầu tiên lưu trữ dữ liệu khởi tạo.
  - Sau đó, kiểm tra điều kiện nếu kích thước tạm thời của dữ liệu khởi tạo không đủ để tạo một trang, thì ta không cần phải lưu trữ dữ liệu khởi tạo, ngược lại thì ta tính số trang cần thiết và kích thước của trang cuối cùng, đồng thời tính lại kích thước của trang đầu tiên lưu trữ dữ liệu khởi tạo.
  - Tiếp theo, ta đọc từng trang cần thiết để lưu trữ mã máy và dữ liệu khởi tạo từ tệp thực thi và lưu vào bộ nhớ vật lý tại địa chỉ ảo tương ứng. Nếu đó là trang cuối cùng thì sử dụng kích thước của trang cuối cùng đã được tính toán từ trước đó, ngược lại thì sử dụng kích thước của trang mặc định.
  - Và kiểm tra điều kiện nếu kích thước của trang cuối cùng của CodePage nhỏ hơn kích thước trang mặc định, chỉ cần đọc dữ liệu khởi tạo và lưu vào trang đầu tiên lưu trữ dữ liệu khởi tạo.
  - Sau đó ta cũng thực hiện đọc từng trang dữ liệu khởi tạo từ tệp thực thi và lưu vào bộ nhớ vật lý tại địa chỉ ảo tương ứng. Nếu là trang cuối cùng, sử dụng kích thước của trang cuối cùng đã tính toán trước đó, ngược lại thì sử dụng kích thước của trang mặc định.
  - Cuối cùng là khởi tạo ID cho biến **spaceID** bằng hàm **InitializeSpaceID()**.

- Trong **SCHandle.h** thêm hàm **StartProcess** với tham số truyền vào là **int**:
  - Mô tả void StartProcess(int _which):
    - Input: Số nguyên, là địa chỉ của tên file thực thi
    - Output: Void, và bắt đầu thực thi tiến trình đó
  - Tên hàm StartProcess giống với hàm trong file progtest.cc nhưng tham số là dạng số nguyên, chứa địa chỉ file thực thi. Hàm sẽ tiến hành mở file.
  - Sau đó ta thực hiện kiểm tra coi tên file có phải **NULL** hay không. Nếu là **NULL** thì return và ghi vào thanh ghi **r2** giá trị -1. Tạo file thực thi và cấp phát vùng nhớ cho chương trình này.
  - Cuối cùng, hàm sẽ gọi phương thức Run của lớp machine để chạy tiến trình.

- Viết system call **SpaceID Exce(char *name)**

- Mô tả SC_Exce:
  - Input: Địa chỉ tên file ở User space
  - Output: -1 nếu bị lỗi hoặc Process SpaceID của chương trình người dùng vừa được tạo nếu thành công
- Đầu tiên, ta lấy tham số tên tập tin từ thanh ghi **r4**, sau đó lưu giá trị vừa lưu ở **r4** từ User space sang System space bằng hàm **User2System()**.
- Sau đó ta thực hiện kiểm tra coi tên file có phải **NULL** hay không. Nếu là **NULL** thì return và ghi vào thanh ghi **r2** giá trị -1.
- Nếu không là **NULL** thì tiếp tục kiểm tra coi độ dài tên file có bằng 0 hay quá dài so với giới hạn cho phép, nếu có lỗi thì trả về -1.
- Tiếp theo, tạo chương trình thực thi, kiểm tra có **NULL** không, nếu không thì tiếp tục cấp phát vùng nhớ cho chương trình thực thi đó.
- Tạo ra **Thread** để chạy chương trình, và gán vùng nhớ vừa tạo ở trên vào **Thread**.
- Kiểm tra xem luồng đó có **NULL** không, nếu không tiếp tục gọi hàm **Fork()** để bắt đầu chạy luồng chương trình.
- Ở đây, hàm **StartProcess(int)** ở trên được truyền vào dưới dạng con trỏ hàm (được typedef trong **./text/utility.h**.
- Cuối cùng, trả về ID của địa chỉ luồng chương trình đang chạy.

2.3 Viết và chạy chương trình ping pong

- Chỉnh lại **./test/makefile**.
- Thêm các define cho các chương trình ping, pong và scheduler.
- Trong thư mục **test**, thêm các file **ping.c**, **pong.c** và **scheduler.c** như theo đề.
- Cd vào thư mục **Do_an_2**, chạy các lệnh sau:

```
make
./userprog/nachos -rs 1023 -x ./test/scheduler
```

- Demo chạy chương trình Scheduler trên Terminal:

scheduler.c

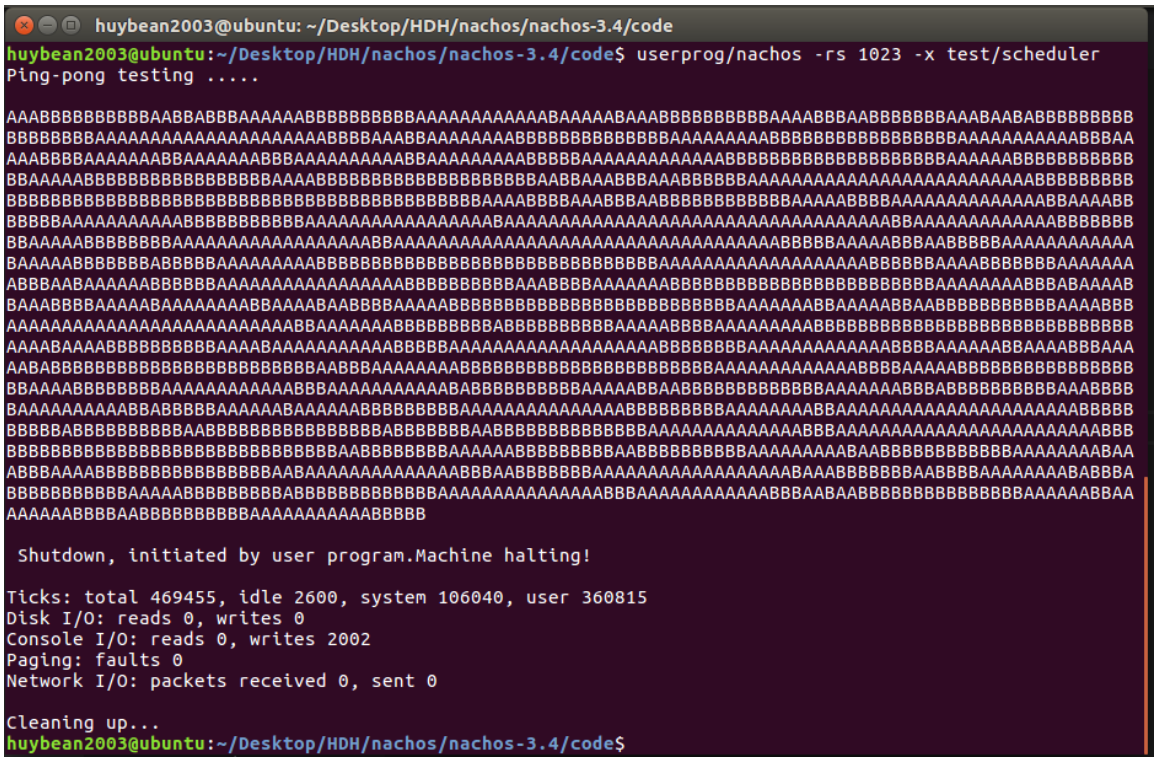
test ▸ scheduler.c ▸ MAXLENGTH

```c
#include "syscall.h"
#include "copyright.h"

#define MAXLENGTH 255

int main()
{
    int pingID, pongID;
    Printf("Ping-pong testing ..... \n\n");
    pingID = Exec("./test/ping");
    pongID = Exec("./test/pong");
    while(1){}
}
```

Hình 2: Chương trình đa chương với 2 tiến trình Ping và Pong



```
huybean2003@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
huybean2003@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ userprog/nachos -rs 1023 -x test/scheduler
Ping-pong testing .....

AAABBBBBBBBBBAABBABBBAAAAAABBBBBBBBBBAAAAAAAAAAAABAAAAABAAABBBBBBBBBBAAAABBBAABBBBBBBBAAABAABABBBBBBBBBB
BBBBBBBBAAAAAAAAAAAAAAAAAAAAAABBBBAABBAAAAAAAAABBBBBBBBBBBBBBBAAAAAAAAABBBBBBBBBBBBBBBBBAAAAAAAAAABBBAA
AAABBBBAAAAAAABBAAAAAAABBBAAAAAAAAAABBAAAAAAAAABBBBBAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBAAAAAABBBBBBBBBBB
BBAAAAABBBBBBBBBBBBBBBBAAAABBBBBBBBBBBBBBBBBBBAABBAAABBBAAABBBBBBBBBAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBAAAABBBBAAABBBAABBBBBBBBBBAAAAABBBBAAAAAAAAAAAAABBAAAABB
BBBBBAAAAAAAAAABBBBBBBBBBAAAAAAAAAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBAAAAABBBBBBBBB
BBAAAAABBBBBBBAAAAAAAAAAAAAAAAAABBAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBAAAAABBAABBBBAAAAAAAAAAA
BAAAAABBBBBBBABBBBAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBAAAAAAAAAAAAAAAAAABBBBBBAAABBBBBBBAAAAAA
ABBBAABAAAAABBBBBBAAAAAAAAAAAAAAAABBBBBBBBBBAAABBBBAAAAAABBBBBBBBBBBBBBBBBBBBBAAAAAAABBBABAAAAB
BAAABBBBAAAAABAAAAAAABBAAAABAABBBBAAAABBBBBBBBBBBBBBBBBBBBBAAAAAABBAAAAABBAABBBBBBBBBBBAAAABBB
AAAAAAAAAAAAAAAAAAAAAAAABBAAAAAABBBBBBBBBABBBBBBBBBBAAAABBBBAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBB
AAABAAAABBBBBBBBBBAAABAAAAAAAAAABBBBBAAAAAAAAAAAAAAAAAABBBBBBBBAAAAAAAAAAAABBBBAAAAABBAAABBBAAA
AABABBBBBBBBBBBBBBBBBBBBBBAABBAAAAAAABBBBBBBBBBBBBBBBBBBBAAAAAAAAAAAABBBBAAAABBBBBBBBBBBBBB
BBAAAABBBBBBBAAAAAAAAAAABBBAAAAAAAAAAABABBBBBBBBBBAAAABAABBBBBBBBBBBAAAAAABBBBBBBBBBBAAABBBB
BAAAAAAAAAABBABBBBBAAAAABAAAAABBBBBBBBBAAAAAAAAAAAAAAABBBBBBBBBAAAAAAAABBAAAAAAAAAAAAAAAABBBBB
BBBBBABBBBBBBBBAABBBBBBBBBBBBBBBBABBBBBBAABBBBBBBBBBBBBAAAAAAAAAAAAABBBAAAAAAAAAAAAAAAAAABBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBAABBBBBBBAAAAABBBBBBBAABBBBBBBBBAAAAAAAAABAABBBBBBBBBBBAAAAAABA
ABBBAAAABBBBBBBBBBBBBBBAABAAAAAAAAAAAAAABBBAABBBBBBBAAAAAAAAAAAAAAAAAABAAABBBBBBBAABBBAAAAAAABABBA
BBBBBBBBBBAAAAABBBBBBBBBABBBBBBBBBBBBBAAAAAAAAAAAAAAABBBAAAAAAAAAAAABBBAABAABBBBBBBBBBBBBAAAAABAA
AAAAAABBBBAABBBBBBBBBBAAAAAAAAAAABBBBB

 Shutdown, initiated by user program.Machine halting!

Ticks: total 469455, idle 2600, system 106040, user 360815
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 2002
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
huybean2003@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$
```

Hình 3: Kết quả chạy chương trình

2.4 Ý tưởng điều tiết tiến trình

- Trong hệ điều hành NachOS, các tiến trình được xử lý theo thuật toán lập lịch tuần tự, nghĩa là các tiến trình sẽ được xử lý tuần tự, mỗi tiến trình được xử lý trong một khoảng thời gian nhất định, sau đó được chuyển sang tiến trình khác. Thuật toán này đảm bảo tính công bằng và logic giữa các tiến trình, mỗi tiến trình được xử lý trong khoảng thời gian tương đối đồng đều.

2.4.1 Độ ưu tiên của các tiến trình

- Để thực hiện ưu tiên chạy của một tiến trình trước đó, ta có thể sử dụng cơ chế "scheduling" của hệ điều hành, ví dụ như sử dụng thuật toán Round Robin hoặc Priority Scheduling. Thuật toán này sẽ đảm bảo rằng tiến trình ưu tiên sẽ được thực thi trước đó trước khi chuyển sang tiến trình khác.

- Cụ thể chúng ta có thể sử dụng class **Scheduler** để lập lịch thực thi các tiến trình. Để ưu tiên thực thi cho một tiến trình cụ thể, ta có thể đặt mức độ ưu tiên cao hơn cho tiến trình đó, ví dụ:

  - Để đặt mức độ ưu tiên cho một tiến trình, có thể sử dụng hàm SetPriority() của lớp Thread.

  - Khi sử dụng **Fork()** để khởi tạo một tiến trình mới, có thể lên lịch tiến trình đó với **Scheduler** bằng cách sử dụng hàm **Yield()** của Thread. Hàm này sẽ báo cho **Scheduler** rằng tiến trình này có thể được thực thi tiếp theo.

- Để quản lý tiến trình và lập lịch, NachOS sử dụng lớp **Scheduler**. Bao gồm một số phương thức quan trọng để quản lý hàng đợi tiến trình và lập lịch, bao gồm:

  - **ReadyToRun(Thread *thread)** - Phương thức này được gọi khi một tiến trình được chuyển từ trạng thái **BLOCKED** sang trạng thái **READY**. Phương thức này đưa tiến trình vào hàng đợi ưu tiên thích hợp dựa trên độ ưu tiên của nó.

  - **Run()** - Phương thức này là trái tim của hệ thống lập lịch **NachOS**. Nó là phương thức được gọi bởi hàm main() và chạy vô thời hạn để lập lịch cho các tiến trình.

  - **FindNextToRun()** - Phương thức này được sử dụng để tìm ra tiến trình kế tiếp sẽ được thực thi. Nó sẽ duyệt qua hàng đợi ưu tiên, tìm kiếm tiến trình có độ ưu tiên cao nhất và chuyển nó sang trạng thái **RUNNING**.

- **PreemptIfNecessary(Thread *nextThread)** - Phương thức này kiểm tra xem tiến trình hiện tại có độ ưu tiên thấp hơn tiến trình kế tiếp sẽ được thực thi không. Nếu đúng, nó sẽ dừng tiến trình hiện tại và đưa nó vào hàng đợi ưu tiên thích hợp, sau đó chuyển sang thực thi tiến trình kế tiếp.