ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN

MÔN HỌC: HỆ ĐIỀU HÀNH

ĐỒ ÁN NACHOS

SYSTEM CALLS VÀ EXCEPTIONS

Thành viên:
HOÀNG PHƯỚC NGUYÊN - 20127258
NGÔ ĐÔNG THỨC - 20127640
LÊ ĐÔNG ĐÔNG - 20127465

Ngày 11 tháng 3 năm 2022

Mục lục

Phần I

LỜI CẢM ƠN

Chúng em xin cảm ơn thầy: Nguyễn Văn Giang - giảng viên lý thuyết và thầy Lê Quốc Hòa - hướng dẫn thực hành đã chỉ dẫn chúng em đồ án khó nhằn này. Thông qua đó, chúng em bước đầu hiểu thêm về hệ điều hành và cách vận hành phía sau của nó thông qua NachOS, từ đó hiểu hơn về môn học này.

Phần II

THÔNG TIN NHÓM

1. **Thông tin các thành viên**

Họ Tên	MSSV	Đánh Giá
Lê Đông Đông	20127465	100%
Ngô Đông Thức	20127640	100%
Hoàng Phước Nguyên	20127258	100%

2. **Đánh giá tiến độ hoàn thành**

Đã hoàn thành tất cả các yêu cầu được giao (100%)

Phần III

NỘI DUNG ĐỒ ÁN

1. **Thiết kế và cài đặt các exception:**

B1 : Các exceptions đã được liệt kê trong file exception.cc ở folder có đường dẫn code/machine/machine.h, chúng em vào file đó để tìm các mô tả về các exceptions dự định sẽ viết.

```
enum ExceptionType { NoException,           // Everything ok!
                     SyscallException,      // A program executed a system call.
                     PageFaultException,    // No valid translation found
                     ReadOnlyException,     // Write attempted to page marked
                                            // "read-only"
                     BusErrorException,     // Translation resulted in an
                                            // invalid physical address
                     AddressErrorException, // Unaligned reference or one that
                                            // was beyond the end of the
                                            // address space
                     OverflowException,     // Integer overflow in add or sub.
                     IllegalInstrException, // Unimplemented or reserved instr.

                     NumExceptionTypes
};
```

B2 : Với mỗi exception sẽ có các thông báo cụ thể cho người dùng biết mình gặp các vấn đề gì. Sau khi thông báo xong thì chúng em sử dụng lệnh Interupt->Halt để tắt hệ điều hành, tránh để nó bị treo khi gặp các lỗi này.

B3 : Riêng về syscall exception sẽ có các trường hợp được xử lý bằng các hàm viết cho user system calls, còn no exception thì trả quyền điều khiển về cho hệ điều hành.

2. **Thiết kế và cài đặt program counter:**

B1 : Program counter sẽ lưu địa chỉ lệnh kế tiếp sẽ được thực hiện, nếu không viết hàm tăng biến đếm chương trình thì hệ điều hành sẽ bị treo.

B2 : Chúng em tìm đoạn mã program counter trong thư mục machine, cụ thể là ở trong file mipssim.cc, đoạn lệnh như sau:

```
// Advance program counters.
registers[PrevPCReg] = registers[PCReg];    // for debugging, in case we
                                            // are jumping into lala-land
registers[PCReg] = registers[NextPCReg];
registers[NextPCReg] = pcAfter;
```

B3 : Copy đoạn mã trên vào chương trình IncreasePC() trong file exception.cc, ta có được một hàm program counter sử dụng cho mỗi system call. Vì mỗi thanh ghi có 4 bytes, nên sau khi đầu tiên ta cộng giá trị thanh ghi là 4 để có thể qua được lệnh kế tiếp.

3. **Thiết kế và cài đặt SC int ReadNum()**

B1 : Để đọc và cài đặt các hàm như readNum (chúng em sửa tên hàm là ReadInt), PrintNum (printInt),... Chúng em sử dụng lớp SynchConsole (lớp cho phép ta thao tác với màn hình), lớp này đã cài đặt như hướng dẫn trong file các thầy đã cho.

B2 : Sử dụng một biến toàn cục từ lớp synchConsole để tiến hành thao tác. Đối với hàm readNum này thì chúng em sử dụng hàm con read của nó để đọc số từ màn hình vào biến tạm, biến này có size là 255 - 4 bytes theo chuẩn C++. Biến đọc được sẽ là một chuỗi, với mỗi xử lý ở dưới này, chuỗi này sẽ lần lượt được chuyển đổi qua các số hợp lệ.

B3 : Tiến hành xử lý các ngoại lệ người thông tin đã đọc được như là nếu người dùng không nhập gì, kiểm tra số đọc được có phải số âm, kiểm tra số có dấu chấm, hay đó không phải là số, số đọc được quá lớn hay quá bé vượt ra ngoài phạm vi cho phép.

B4 : Sau khi đọc xong thì ghi kết quả vào thanh ghi số 2 (thanh ghi mặc định chứa kết quả), tăng program counter, viết một chương trình thử nghiệm và kết quả đúng.

4. **Thiết kế và cài đặt SC void PrintNum(int num-**

ber)

B1 : Đọc số từ màn hình người nhập bằng cách lấy số đó từ thanh ghi số 4.

B2 : Với các trường hợp ngoại lệ như đó là số âm thì chúng em sẽ chuyển xuống dưới hàm để xử lý được dễ dàng, chuyển nó thành số dương cho dễ tính toán. Sau đó chuyển đổi các số thành chuỗi kí tự để có thể in ra được màn hình.

B3 : Khi đọc vào ta có một chuỗi bị đảo ngược, muốn in ra đúng với kết quả ta mong muốn thì chúng em sử dụng hàm reverse để chuyển ra kết quả cuối cùng.

B4 : Sử dụng biến toàn cục từ lớp synchConsole để ghi ra nmàn hình, cuối cùng tăng thanh ghi và kết thúc system call này.

5. **Thiết kế và cài đặt SC char ReadChar()**

B1 : Sử dụng một biến tạm để lưu chuỗi đọc được trên màn hình (dùng biến toàn cục từ lớp synchConsole để lấy), đồng thời lúc nó đọc sẽ trả ra độ dài chuỗi, lưu độ dài chuỗi này vào một biến để có thể kiểm tra tính hợp lệ của hàm đang viết.

B2 : Sử dụng biến lưu số kí tự ở trên để xét các exception, nếu độ dài lớn hơn 1 thì đây không còn là kí tự mà là chuỗi, không khớp với yêu cầu hàm đặt ra, tương tự nếu như người dùng nhập vào kí tự trống hoặc không nhập gì, tất cả trường hợp này sẽ được ghi vào thanh ghi thứ 2 là 0.

B3 : Nếu không có trường hợp đặc biệt nào thì ghi vào thành ghi số 2 kí tự đầu tiên cũng như là duy nhất của chuỗi trên màn hình. Tăng program counter và xóa các tài nguyên đã cấp phát tạm thời.

6. **Thiết kế và cài đặt SC void PrintChar(char character)**

B1 : Đọc kí tự từ màn hình người nhập bằng cách lấy số đó từ thanh ghi số 4.

B2 : Ép kiểu về char để có thể in ra được màn hình.

B3 : Sử dụng biến toàn cục từ lớp synchConsole để viết ra màn hình, với size là 1 byte ứng với kí tự đó. Cuối cùng tăng program counter.

7. **Thiết kế và cài đặt SC int RandomNum()**

B1 : Sử dụng thư viện time.h để tiến hành dùng hàm random.

B2 : Random sẽ ra một số, ta không thể in ra số này ra màn hình vì nó không phải kiểu char. Thế nên tiến hành biến đổi số thành một chuỗi và in nó ra màn hình bằng biến toàn cục từ lớp synchConsole. Tăng program counter.

8. **Thiết kế và cài đặt SC void ReadString(char[] buffer, int length)**

B1 : Vì thanh ghi chỉ chứa số, truyền vào string thì nó là địa chỉ nên chúng em tạo một biến địa chỉ đọc từ thanh ghi thứ 4.

B2 : Lấy size của chuỗi từ thanh ghi thứ 5, kiểm tra size có hợp lệ hay không rồi tiếp tục thực hiện.

B3 : Cấp phát vùng nhớ cho biến tạm chứa chuỗi, đọc chuỗi trên màn hình và lưu vào biến này, đồng thời nó sẽ trả về size mà nó đã đọc được. Sử dụng biến size để kiểm tra tính hợp lệ của chuỗi.

B4 : Dùng hàm System2User để copy giá trị từ system space (buffer) qua user space (địa chỉ lấy ra từ thanh ghi thứ 4). Tăng program counter.

9. **Thiết kế và cài đặt SC PrintString(char[] buffer)**

B1 : Lấy địa chỉ chuỗi từ thanh ghi thứ 4, sử dụng hàm User2System để sao chép vùng nhớ từ user space (địa chỉ chuỗi ở thanh ghi thứ 4) sang system space (buf).

B2 : Tính độ dài của chuỗi, đặt kí tự thoát ở cuối chuỗi và sử dụng biến toàn cục của lớp synchconsole viết ra màn hình chuỗi này. Tăng program counter.

10. **Viết chương trình help**

B1 : Chương trình đơn giản sử dụng các system calls đã viết để sử dụng. Với chương trình này, sau khi import thư viện syscall.h chứa các định danh hàm, chúng em sử dụng system call PrintString.

B2 : Viết các thông tin cơ bản như là thông tin về nhóm, cách dùng chương trình in ra bảng mã ascii và chương trình sắp xếp nổi bọt.

B3 : Vào file makefile của folder code/test, thêm tên chương trình help vào nhãn all, viết đoạn lệnh như trên hình để thực hiện việc makefile để chạy được chương trình.

```
help.o: help.c
	$(CC) $(CFLAGS) -c help.c
help: help.o start.o
	$(LD) $(LDFLAGS) start.o help.o -o help.coff
	../bin/coff2noff help.coff help
```

B4 : Sử dụng lệnh Halt() ở cuối chương trình để tắt hệ điều hành, bước này để đảm bảo khi chạy xong chương trình thì hệ điều hành sẽ tắt tránh bị treo.

11. **Viết chương trình ascii**

B1 : Tương tự như chương trình help, cài đặt các thuật toán cần thiết để in ra bảng mã ascii (sử dụng các system call như là PrintChar và PrintInt để xuất các kí tự ascii cũng như là các số).

B2 : Vào file makefile của folder code/test, thêm tên chương trình ascii vào nhãn all, viết đoạn lệnh như trên hình

để thực hiện việc makefile thực thi đoạn mã:

```
ascii.o: ascii.c
	$(CC) $(CFLAGS) -c ascii.c
ascii: ascii.o start.o
	$(LD) $(LDFLAGS) start.o ascii.o -o ascii.coff
	../bin/coff2noff ascii.coff ascii
```

B3 : Cuối cùng, sử dụng lệnh halt để tắt hệ điều hành.

12. **Viết chương trình sort**

B1 : Tương tự như chương trình help. Đầu tiên sử dụng các lệnh PrintString để thông báo cho người dùng nhập vào n, với n sẽ nhận gía trị từ syscall ReadInt, sử dụng vòng lặp nhập số để có thể xử lý các ngoại lệ mà người dùng nhập vào (như là nhập quá 100 và nhập nhỏ hơn 0).

B2 : Sử dụng vòng lặp để người dùng có thể nhập vào các phần tử trong mảng, sử dụng các syscall PrintString để thông báo, PrintInt để in ra chỉ số phần tử trong thông báo và các phần tử trong mảng sẽ nhận giá trị từ hàm ReadInt.

B3 : Cài đặt các thuật toán để sắp xếp nổi bọt mảng, sử dụng 0 và 1 để chọn chiều sắp xếp, ở đây cũng xử lý các ngoại lệ nếu người dùng nhập khác 2 số 0 và 1 đó.

B4 : In ra các phần tử đã được sắp xếp bằng các syscall cần thiết. Cuối cùng dùng lệnh Halt để tắt hệ điều hành.

B5 : Tiến hành thêm tên chương trình vào nhãn all trong file makefile của folder code/test, sau đó thêm các dòng lệnh cần thiết để chương trình có thể chạy được.