**HỌ VÀ TÊN:** NGUYỄN TRÍ LUẬN NGỮ TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

**MÃ SỐ SV:** 1112201 KHOA CÔNG NGHỆ THÔNG TIN

**BÁO CÁO**

**TÌM HIỂU HỆ ĐIỀU HÀNH NACHOS**

Môn: Hệ Điều Hành

**GVC:** Trần Trung Dũng

**GVHD:** Lê Giang Thanh

**Yêu cầu:** Đọc kỹ về class machine, intterrupt, console, viết báo cáo giải thích các hàm và vai trò của từng hàm trong các lớp đó, và quá trình gọi thực thi một interrupt.

**Nội dung:**

**CONSOLE:**

Giả lập thiết bị nhập xuất

* Hàm tạo lập **Console:** khởi tạo một thiết bị phần cứng ảo
* Hàm **PutChar(char ch):** viết ký tự ‘ch’ ra thiết bị xuất console giả lập
* Hàm **GetChar():** đọc trong vùng nhớ đệm ra ký tự ‘ch’ nếu không có trả về giá trị EOF
* Hàm **CheckCharAvail():** kiểm tra các ký tự khi nhập vào có hợp lệ không
* Hàm **WriteDone():** hàm được gọi khi trình quản lý ngắt thông báo hoàn tất việc xuất ký tự

**MACHINE:**

Mô tả sự thực thi chương trình người dùng trên nền Nachos

+ Trong lớp **Instruction:**

* + Hàm **Decode():** giải mã lệnh dưới dạng nhị phân.

+ Trong lớp **Machine:**

* + Hàm **Machine(bool Debug):** khởi tạo một thiết bị phần cứng mô phỏng cho các chương trình người dùng.
  + Hàm **Run():** hàm thực thi một chương trinh người dùng.
  + Hàm **ReadRegister(int num):** đọc nội dung trong thanh ghi thứ ‘num’
  + Hàm **WriteRegister(int num, int value):** ghi nội dung trong biến ‘value’ vào thanh ghi thứ ‘num’.
  + Hàm **OneInstruction(Instruction \*intr):** thực thi một lệnh của chương trình
  + Hàm **DelayedLoad(int nextReg, int nextVal):**
  + Hàm **ReadMem(int addr, int size, int\* value):** đọc giá trị ‘value’ trong địa chỉ ‘addr’
  + Hàm **WriteMem(int addr, int size, int value):** ghi giá trị lưu trong ‘value’ vào địa chỉ ‘addr’
  + Hàm **Translate(int virtAddr, int\* physAddr, int size,bool writing):** chuyển đổi địa chỉ từ địa chỉ vật lý sang địa chỉ ảo, nếu không chuyển được thì trả về một exception.
  + Hàm **RaiseException(ExceptionType which, int badVAddr):** chuyển quyền điều khiển cho hệ thống từ chế độ người dùng, do có lời gọi system call từ chương trình người dùng hoặc có những exception xuất hiện.
  + Hàm **Debugger():** gọi debugger của chương trình người dùng
  + Hàm **DumpState():** xuât ra màn hình trạng thái của CPU và bộ nhớ.

**INTERRUPT**

*Ngắt (Interrupt) là việc xảy ra một điều kiện (sự kiện) làm cho chương trình đang thực thi (chương trình chính) bị tạm dừng để quay sang thực thi một chương trình khác (chương trình xử lý ngắt) rồi sau đó quay trở về thực thi tiếp công việc đang bị tạm dừng. Các ngắt cho phép hệ thống đáp ứng theo cách không đồng bộ và xứ lý sự kiện trong khi một chương trình khác đang thực thi. Một hệ thống được điều khiển bới ngắt cho ta ảo tưởng nhiều công việc đang được thực thi.*

*Lớp Interrupt mô phỏng việc ngắt bởi cách duy trì một hàng đợi các sự kiện cùng với một đồng hồ mô phỏng. Khi đồng hồ diễn ra, hàng đợi sự kiện sẽ kiểm tra các sự kiện được sắp xếp diễn ra hiện tại.*

*Đồng hồ được duy trì thực hiện hoàn toàn trong phần mềm và chạy chỉ khi thỏa các điều kiện sau:*

*+ Mỗi khi interrupts được kích hoạt lại (tức là khi bị ngắt và mở trở lại), đồng hồ sẽ tăng lên 10 ticks. Nachos lập trình thường xuyên tắt và phục hồi những interrupt với mục tích loại trừ lẫn nhau bởi các cách thực hiện việc gọi dứt khoát để tắt và phục hồi.*

*+ Khi nào giả lập MIPS thực hiện một trong các chỉ thị, đồng hồ sẽ tăng lên 1 tick.*

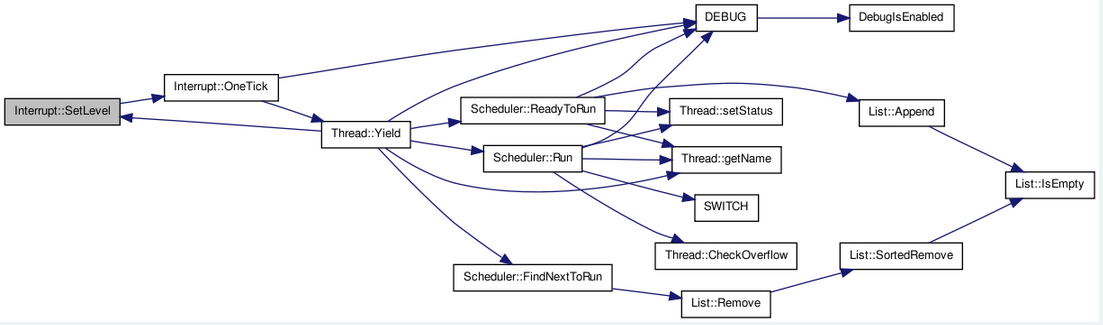
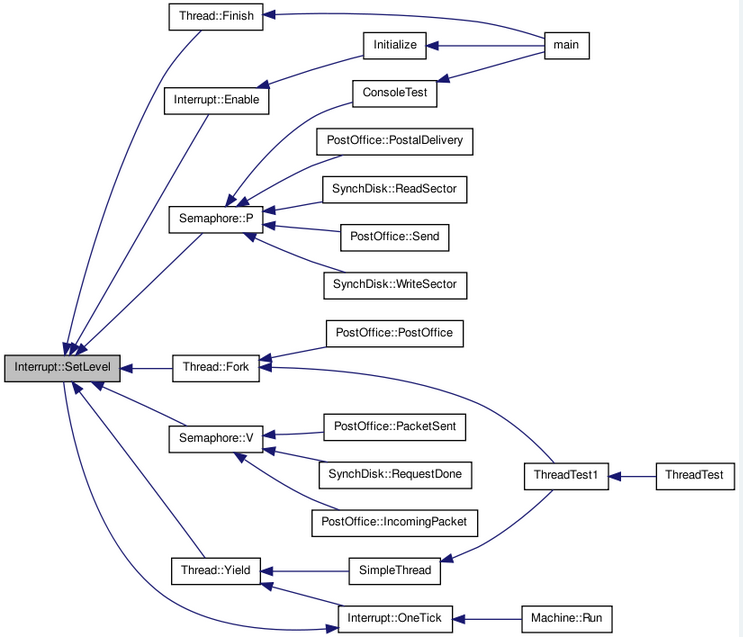
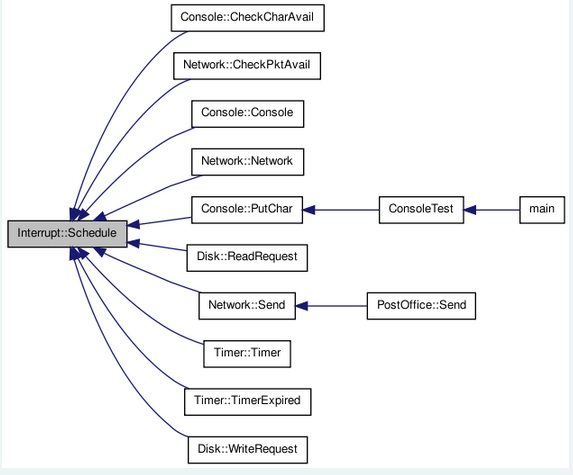
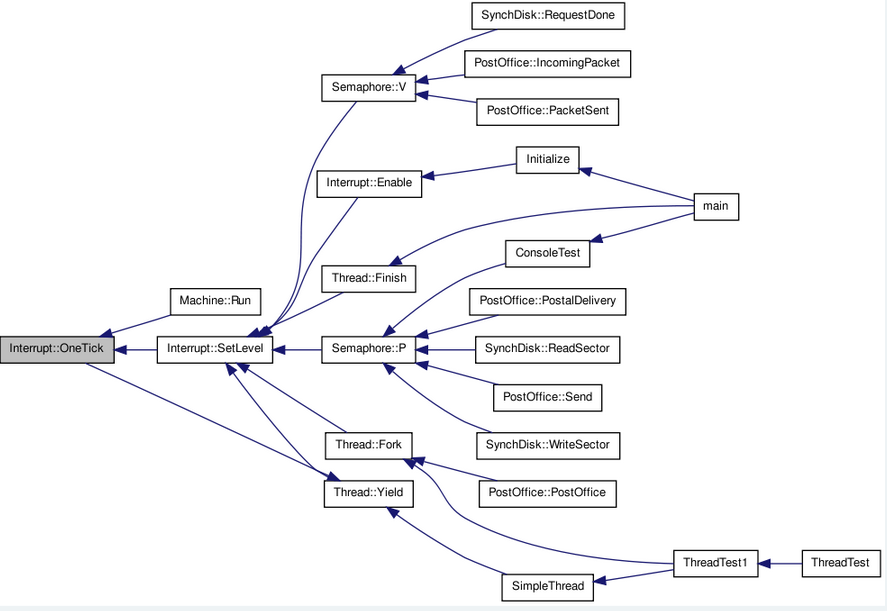
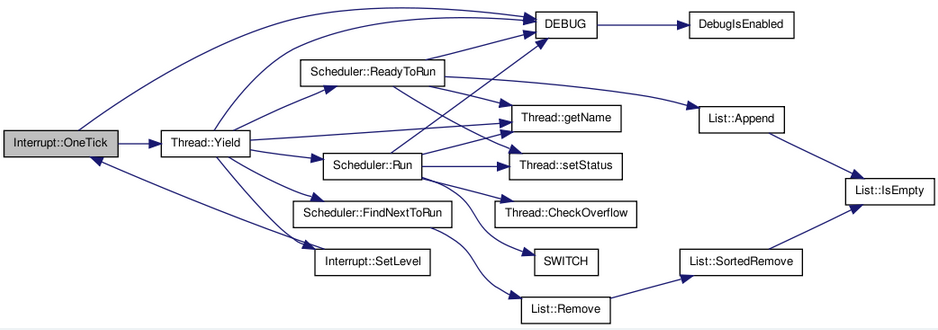
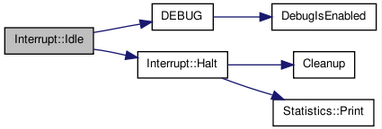
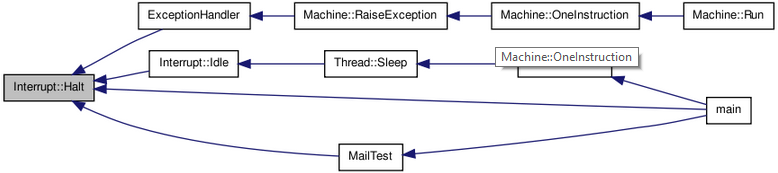
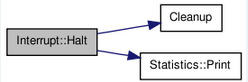
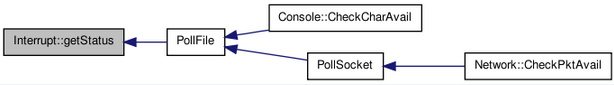
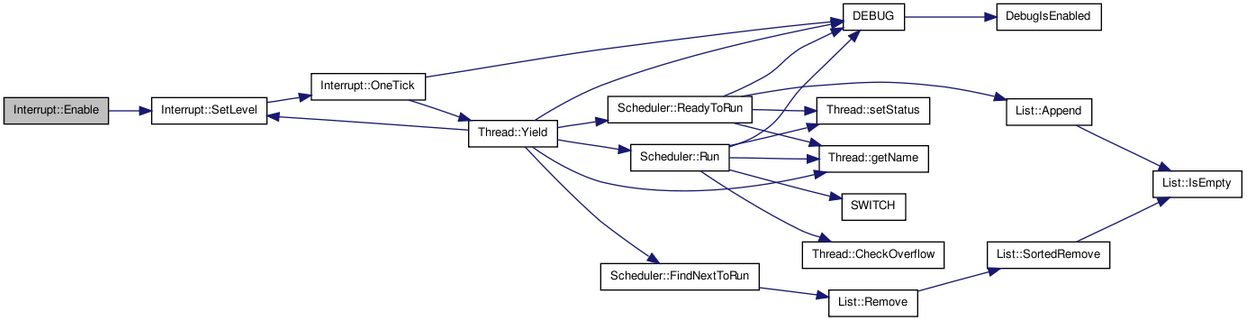
*\*Nachos cho phép thiết lập các thời gian mô phỏng để cải thiện các interrupt kế tiếp mọi lúc khi list hiện tại rỗng. Điều này giúp cải thiện nhỏ lợi ích khi thực hiện, nhưng nó lại tạo sự tương tác tự nhiên giữa nhân và phần cứng, và nó không cần thiết do một hệ điều hành bình thường sử dụng các luồng (thread) vô hiệu quả, và điều này chính xác là Nachos đang tạo ra).*

*Bất cứ khi nào đồng hồ tăng, hàng đợi của sự kiện sẽ được kiểm tra và các sự kiện interrupt chưa được xử lý bị đem ra xử lý bởi việc gọi các việc xử lý các sự kiện của thiết bị liên quan với sự kiện đó. Lưu ý đó không phải là việc xử lý interrupt (hay gọi là dịch vụ thường xuyên gián đoạn). Việc xử lý interrupt là một phần của chương trình, khi xử lý sự kiện của thiết bị là một phần của mô phỏng phần cứng. Việc xử lý sự kiện từ thiết bị sẽ được gọi xử lý interrupt của phàn mềm cho thiết bị. Vì điều đó, lớp interrupt vô hiệu hóa các interrupt trước khi gọi xử lý sự kiện cùa thiết bị.*

**+** Trong lớp **PendingInterrupt {}:** class này để định nghĩa cho việc ngắt có thể xảy bất ngờ, các cấu trúc nội tại của nó rời khỏi chế độ public để làm đơn giản hơn các thao tác hoạt động.

+ Trong lớp **Interrupt {}:** class này định nghĩa cho cấu trúc dữ liệu của các interrupt của phần cứng tương tự nhau, chúng ghi nhận lại nếu những interrupt được kích hoạt hoặc không, và các interrupt của phần cứng được sắp xếp xảy ra ở tương lai.

* Hàm **IntStatus SetLevel(IntStatus level): d**ùng để bật hoặc tắt tín hiệu ngắt và trả về trạng thái trước đó, do kiểu dữ liệu IntStatus dùng để tắt mở interrupt, và nếu ngắt đang được bật thì tăng bộ đếm của đồng hồ mô phỏng thông qua lời gọi hàm OneTick().
* Hàm **void Enable():** bật tín hiệu ngăt
* Hàm **IntStatus getLevel() {return level;}** : trả về trạng thái ngắt hiện tại
* Hàm **void Idle():** nếu hàng đợi có sẵn là rỗng, đưa thời gian mô phỏng(giả lập) vào trước đến khi thực hiện interrupt kế tiếp. Từ khi chạy một việc phải được chạy để đưa vào thread trên hàng đợi hiện tại, điều duy nhất để làm là thúc đẩy thời gian giả lập đến khi sắp xếp interrupt tiếp theo của phần cứng.
* Hàm **void Halt():** thoát khỏi interrupt và in ra tình trạng hệ thống
* Hàm **void YieldOnReturn():** Quay trở về chương trình đang bị tạm ngưng.
* Hàm **MachineStatus getStatus() { return status; }**: trả về kiểu mode đang thực hiện
* Hàm **void setStatus(MachineStatus st) { status = st; }**: gán kiểu mode muốn vào
* Hàm **void DumpState():** in ra trạng thái của interrupt
* Hàm **void Schedule(VoidFunctionPtr handler,int arg, int when, IntType type):** sắp xếp interrupt xuất hiện tại biến thời gian ‘when’, điều này được gọi bởi việc mô phỏng thiết bị phần cứng.
* Hàm **void OneTick():** tăng bộ đếm đồng hồ mô phỏng và kiểm tra nếu có bất kỳ interrupt đang chờ để được gọi. Hai thứ có gây ra OneTick() được gọi là: interrupt tái kích hoạt do chỉ thị của người dung được thực thi.
* Hàm **bool CheckIfDue(bool advanceClock):**  kiểm tra nếu interrupt được giả sử xảy ra hiện tại, trả về TRUE nếu không có gì trong hàng đợi hiện tại, vì vậy chỉ đơn giản tăng đồng hồ khi interrupt tiếp theo được cấp phát chính xảy ra (nếu có).
* Hàm **void ChangeLevel(IntStatus old, IntStatus now):** thay đổi trạng thái của interrupt hiện tại, nhưng không ảnh hưởng đến đồng hồ mô phỏng.
* ***Quá trình thực thi Interrupt:***

**