NachOS – Not Another Completely Heuristic Operating System . là một phần mềm giả lập hệ điều hành với kiến trúc MIPS ( Million Instruction Per Second ) . Được viết bằng C++ cho MIPS , NachOS chạy như một user-process trên hệ điều hành máy chủ.

Các thành phần của NachOS :

* NachOS Machine : NachOS mô phỏng đại khái gần giống với kiến trúc MIPS, có thanh ghi , bộ nhớ và CPU . NachOS Machine được thực thi bởi Machine Object khi NachOS được khởi động, nó chứa các methods như Run, ReadRegister, WriteRegister ,… .
* NachOS Threads : Trong NachOS, một lớp Threads được định nghĩa và có thể có những trạng thái như là ready, running, blocked hoặc just created. Thread object cũng có chứa các methods như PutThreadToSleep, YieldCPU, ThreadFork, ThreadStackAllocate, … .
* NachOS UserPrograms : NachOS chạy chương trình người dung trong chính không gian bảo mật của chúng . NachOS có thể chạy bất kỳ tệp nhị phân MIPS nào . NachOS yêu cầu thực thi tệp phải ở định dạng “Noff” trong khi các tệp đa phần được lưu dưới dạng “a.out” hay còn gọi là định dạng “Coff”.

Hiện nay , các hệ điều hành của máy tính “hiện đại” thường tách riêng bộ nhớ ảo thành 2 khu vực : không gian nhân (kernel space) và không gian người dùng (user space) . Sự tách biệt này nhằm mục đích bảo vệ bộ nhớ và phần cứng khỏi những phần mềm hoặc lỗi độc hại .

|  |  |
| --- | --- |
| Kernel | User |
| Chạy nhân hệ điều hành đặc quyền , phần mở rộng của nhân và đa số các drivers. | Vùng bộ nhớ chỉ để chạy ứng dụng phần mềm hoặc để thực thi một số drivers. |
| Mã thực thi có quyền truy cập vào bất kỳ không gian địa chỉ nào của memory và phần cứng. | Mã thực thi bị giới hạn truy cập và không thể truy cập trực tiếp đến kernel space được. (Truy cập gián tiếp qua một phần nhỏ của kernel gọi là the system call). |
| Bất kỳ sự “ không ổn định “ nào bên trong mã thực thi kernel đều có thể dẫn đến lỗi hệ thống hoàn toàn. | Nếu có xảy ra trục trặc hoặc vấn đề thì đều có thể khôi phục được vì truy cập bị hạn chế và giới hạn trong không gian hệ thống mà user đang hoạt động. |

Kernel space code có đặc tính để run trong kernel mode, gọi là mã thực thi Ring 0.

Điển hình trong kiến trúc x86 có 4 Ring :

* Ring 0 : Kernel Mode. ( Tương tác trực tiếp với phần cứng vật lý như CPU hoặc bộ nhớ )
* Ring 1 : Có thể sử dụng bởi máy ảo hoặc drivers.
* Ring 2 : Sử dụng bởi drivers.
* Ring 3 : Là những ứng dụng.

Đặc quyền sắp xếp từ thấp đến cao : Ring 3 > Ring 2 > Ring 1 > Ring 0 .

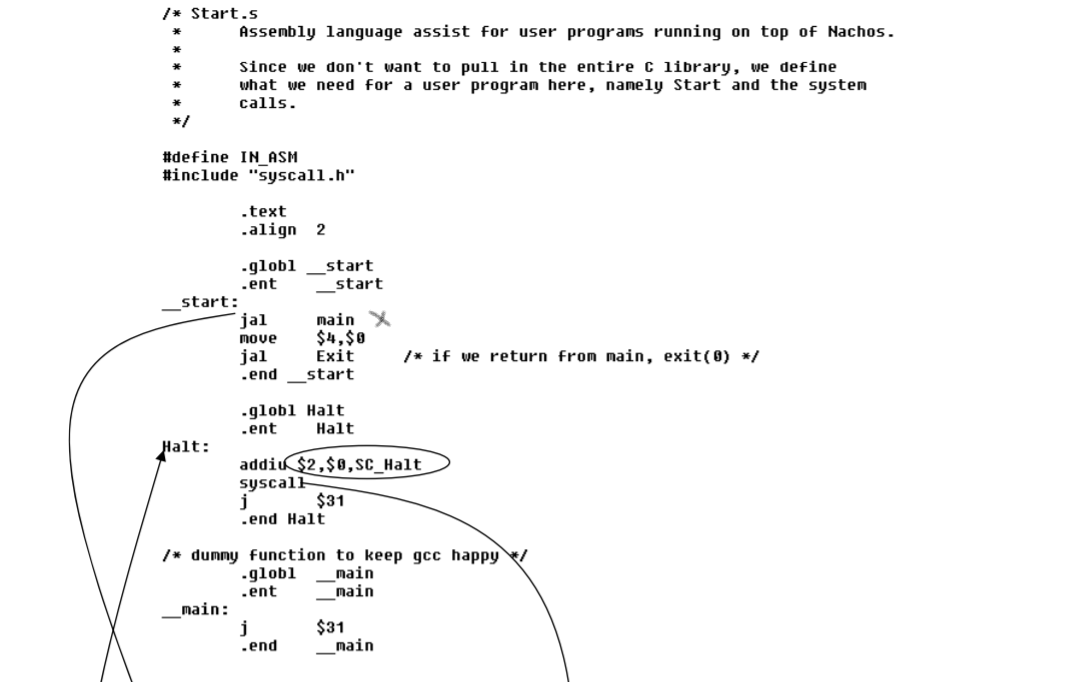
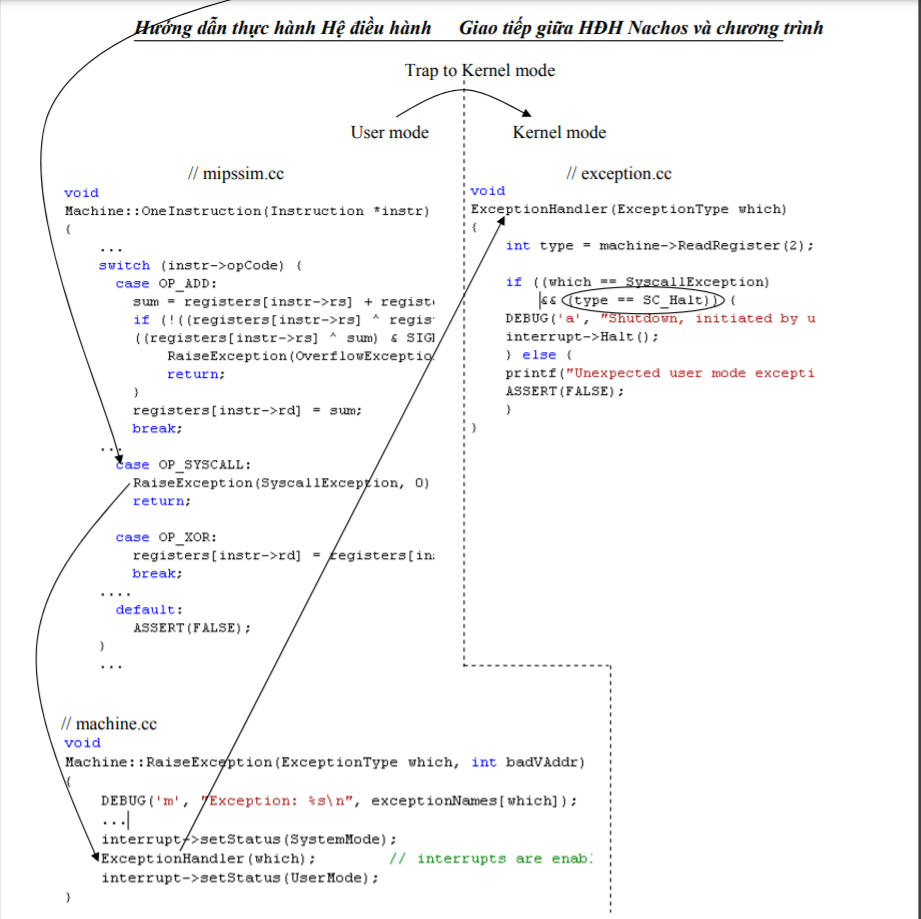
Riêng Linux chỉ sử dụng 2 Ring là : Ring 0 ( cho Kernel ) và Ring 3 ( cho User )

Cách hoạt động của quá trình thực thi một chương trình được viết trên hệ điều hành NachOS :

* Một chương trình có định dạng .c ( VD : halt.c , sort.c, start.c , … ) là chương trình được biên dịch bởi cross-compiler GCC thành tập tin có định dạng .s (VD : halt.s , start.s , …) . *Định dạng .c là file input được biên dịch thành định dạng .s là mã hợp ngữ chạy trên kiến trúc MIPS .* Lưu ý : chính vì GCC không hỗ trợ biên dịch các thư viện có sẵn trong C/C++ như stdio.h, conio.h, … , nên các file input định dạng .c phải viết mà không thể sử dụng hàm trong các thư viện có sẵn . Tức là chỉ có thể dùng những toán tử (+, -, \*,/, while, if, … ) mà không thể thao tác với người dùng ( VD : muốn in ra màn hình, nhập dữ liệu từ bàn phím, tạo tập tin , … ) , để làm được điều này thì ta dùng System Call .
* Sau đó tập tin sau khi được biên dịch bởi cross-compiler này sẽ được liên kết với tập tin thư viện của hợp ngữ và tạo thành tập tin với định dạng .coff .
* Kế đến tập tin với định dạng .coff đó sẽ được chuyển thành tập tin định dạng .noff (định dạng thực thi trên hệ điều hành NachOS ) bởi tiện ích “coff2noff” .

System Call là lời gọi từ người dùng khi user không làm được gì đó mà cần đến sự giúp đỡ của hệ điều hành.

Cách mà Halt.noff được thực thi :

* Trong chương trình Halt chỉ có duy nhất một hàm Halt() . Đây là hàm có nhiệm vụ tắt máy ( hay ở đây là máy ảo ) và hàm Halt() được gọi là system call. Như định nghĩa về system call , chương trình Halt không thể tự tắt máy được mà phải cần đến hệ điều hành.
* Sau khi chương trình nạp vào RAM ảo , CPU MIPS bắt đầu nạp lệnh trong halt.noff để thực thi lệnh.
* Lệnh *jal main* ( đánh dấu x ) để nhảy vào hàm main() của halt.c.
* Tiếp đến là lệnh *addiu $2,$0,SC\_Halt* ( được khoanh tròn ) lưu giá trị của SC\_Halt ( là 1 số nguyên ) vào thanh ghi thứ 2.
* Lệnh *syscall* ( bên dưới lệnh *addiu $2,$0,SC\_Halt* ) dùng lệnh này để gọi và xử lý system call. Khi nạp lệnh này , mã hoạt động sẽ nhận thấy lệnh *OP\_SYSCALL* và nó sẽ gọi hàm RaiseException(). Vì là system call nên hàm *syscall* sẽ chuyển qua kernel mode để thực hiện xử lý system call đó, sau đó chuyển về lại cho user mode để chạy tiếp chương trình.
* *void Machine::OneInstruction(Instruction \*instr)* để thực hiện một chỉ dẫn hoặc chương trình từ user mode với các diều kiện switch-case tương ứng. (Như là Add ( thêm vào thanh ghi ), Syscall ( System call ) , XOR , còn lại là trả về giá trị FALSE, ASSERT là syscall exit vì machine->Run ko có return )
* *Void Machine::RaiseException(ExceptionType which, int badVAddr)* 
  + *Interrupt->setStatus(SystemMode);* để ngắt system mode hiện tại rồi chuyển qua system mode khác (ở đây là chuyển từ User mode sang Kernel mode ).
  + *ExceptionHandler(which)* để chuyển sang dùng hàm void ExceptionHandler với ExceptionType là biến which.
  + *Interrupt->setStatus(UserMode);* cũng tương tự là chương trình về lại User mode thôi.
* *Void ExceptionHandler(ExceptionType which)* : hàm này để thực thi system call.
  + *Int type = machine->ReadRegister(2);* để class machine đọc thanh ghi số 2 ( $2 chỗ addiu $2,$0,SC\_Halt ) .
  + Điều kiện *if((which == SyscallException) && (type == SC\_Halt ) )* để xét điều kiện với *which* và *type* sau đó nếu thỏa thì chạy hàm *interrupt->Halt()*
  + Điều kiện *else* : in ra màn hình rồi trả về giá trị FALSE.