**HỆ ĐIỀU HÀNH**



* **GV: TRẦN TRUNG DŨNG**
* **GV: LÊ QUỐC HÒA**
* **GV: CHUNG THÙY LINH**

**MỤC LỤC**

[I. Thành viên nhóm: 3](#_Toc496388922)

[II. Báo cáo đồ án 4](#_Toc496388923)

**[2.1 Mô hình hoạt động của chương trình và hàm trên hệ điều hành Nachos 4](#_Toc496388924)**

**[2.2 Qui trình cài đặt 9](#_Toc496388925)**

**[a. Qui trình viết một syscall 9](#_Toc496388926)**

**[b. Viết và biên dịch một chương trình 10](#_Toc496388927)**

**[2.3 Chi tiết thiết kế và cài đặt 12](#_Toc496388928)**

[III. Demo chương trình 15](#_Toc496388929)

**DANH SÁCH THÀNH VIÊN THAM GIA ĐỒ ÁN**

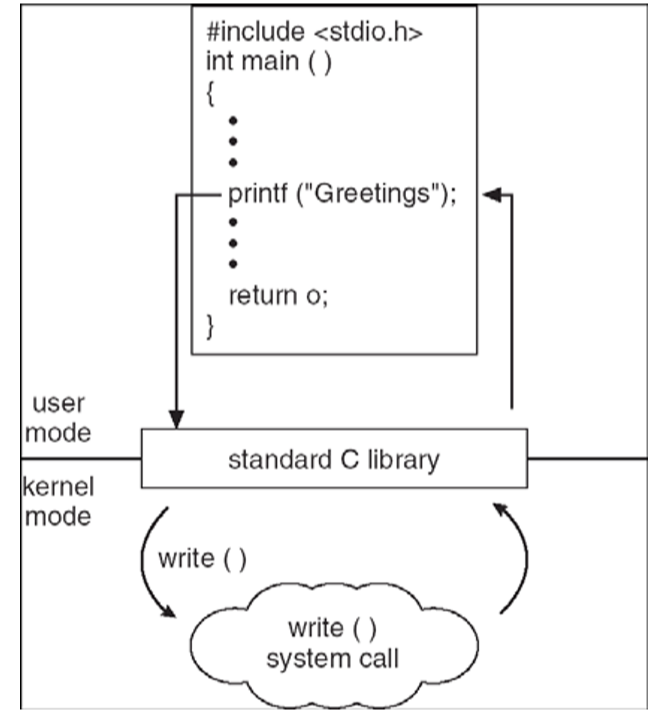
# **Thành viên nhóm:**

|  |  |
| --- | --- |
| * **Họ tên:** * **MSSV:** * **Email:** * **SĐT:** | * **Cao Minh Hiếu** * **1512157** * 1512157@students.hcmus.edu.vn * **0165.259.2239** |
| * **Họ tên:** * **MSSV:** * **Email:** * **SĐT:** | * **Nguyễn Duy Tâm** * **1512479** * [tam.nguyen@goldeneyetech.com.vn](mailto:tam.nguyen@goldeneyetech.com.vn) * **01666.942.492** |

# **Báo cáo đồ án**

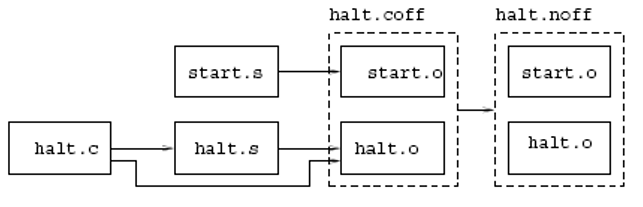
## **2.1 Mô hình hoạt động của chương trình và hàm trên hệ điều hành Nachos**

- Mô hình thực thi khi gọi hàm của hệ thống trong Nachos



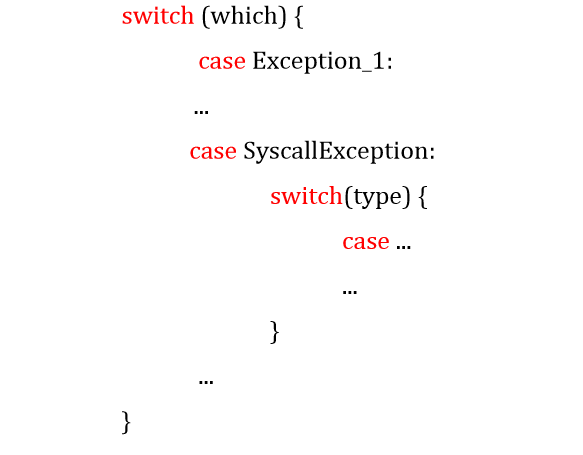
***Hình 1. Gọi hàm trong Nachos***

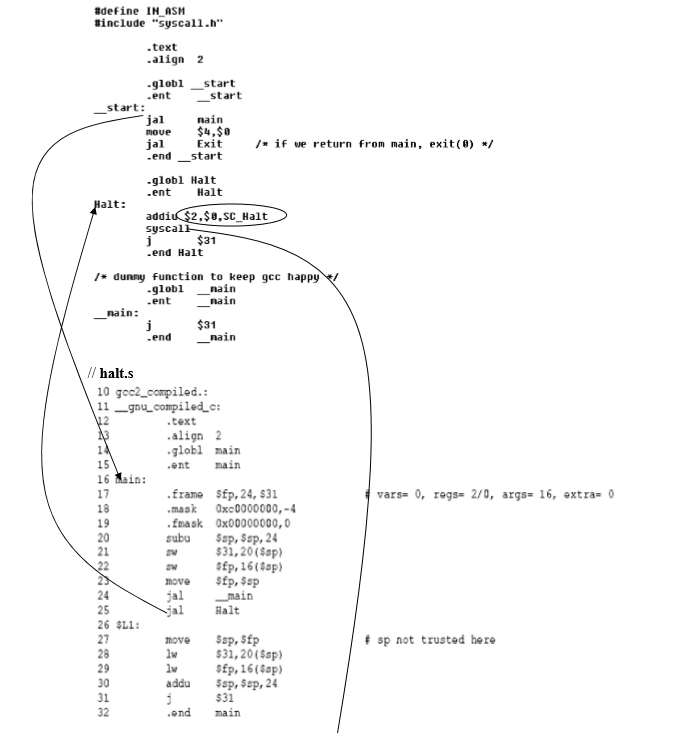
* Một chương trình khi thực thi trong chế độ người dùng (user mode) mà gọi một hàm trong bộ thư viện C chuẩn thì hàm sẽ được thực thi bằng cách gọi syscall tương ứng trong chế độ hệ thống (kernel mode) để thực hiện công việc cần làm sau đó trả kết quả về cho hàm gọi.
* Như vậy, để có thể thực thi những hàm yêu cầu ta cần viết các syscall tương ứng để thực hiện công việc.

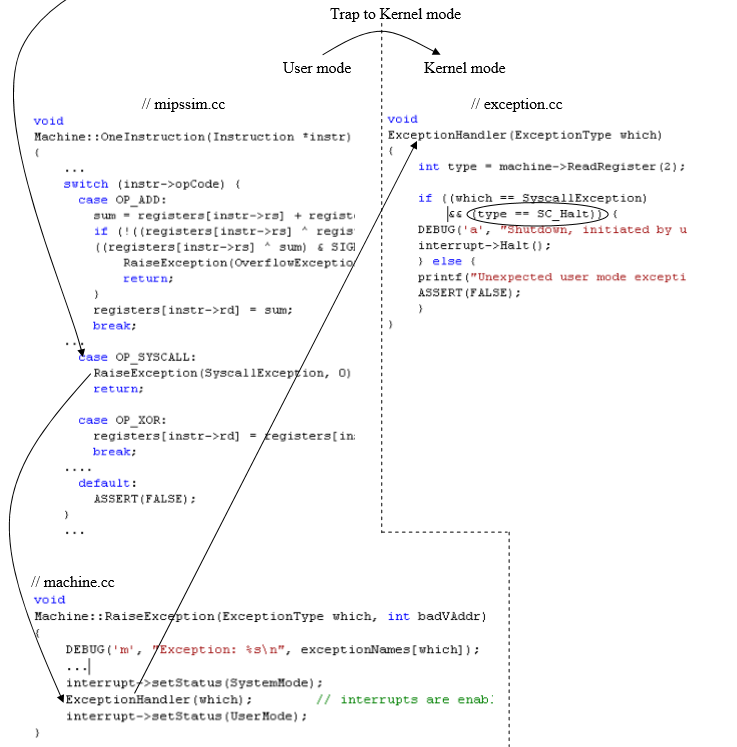


***Hình 2. Thành phần file thực thi trên Nachos***

* Phân tích mô hình biên dịch một file thực thi trong Nachos:
* Chương trình viết trong file halt.c biên dịch bằng cross-compiler thành halt.s
* halt.s và start.s liên kết nhau thành halt.coff là file thực thi trên Linux (với kiến trúc MIPS).
* halt.coff được nachos biến đổi thành halt.noff là định dạng file thực thi trên Nachos (kiến trúc MIPS).
* Hình tiếp theo mô tả sơ đồ gọi hàm trong Nachos khi một file halt.noff được thực thi:
* Phần start trong halt.noff được thực thi trước, gọi nhảy vào hàm main trong halt.s qua lệnh nhảy jal main.
* Cuối thủ tục main lại có lệnh nhảy vào hàm halt: jal halt.
* Trong halt: thanh ghi $v0 ($2) được gán giá trị = SC\_Halt với SC\_Halt là hằng đã định nghĩ trong syscall.h: $v0 = 0 + SC\_Halt. Tiếp đó lời gọi hạm hệ thống syscall được thực hiện để nhảy vào hàm Machine::OneInstruction(…) – hàm này vốn thuộc chế độ user mode.
* Qua bộ lọc switch case lại nhảy tiếp vào hàm RaiseException(…) – thuộc usermode.
* Trong RaiseException(…) lại gọi ExceptionHandler(…) lúc này mới bước vào kernel mode (hàm ExceptionHandler thuộc kernel mode) và thực thi hàm syscall tương ứng - ở đây là interrupt-> halt().
* Ở đây ta thấy ExceptionHandler() chỉ xử lý một trường hợp duy nhất là: which = SyscallException và type = SC\_Halt. Nếu ta chuyển cấu trúc này thành switch thì có thể xử lý được nhiều loại which của exception và type của syscall hơn.







***Hình 3. Sơ đồ gọi hàm thực thi trong nachos***

* Từ mô hình trên, ta có thể thay đổi cấu trúc để xử lý được nhiều exception và syscall hơn.

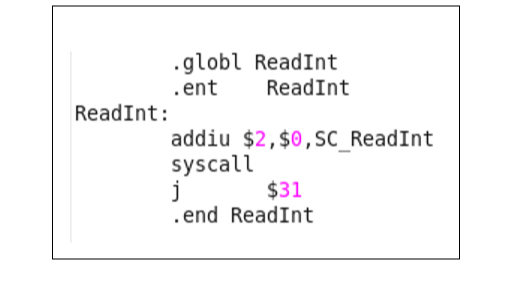
## **2.2 Qui trình cài đặt**

### **a. Qui trình viết một syscall**

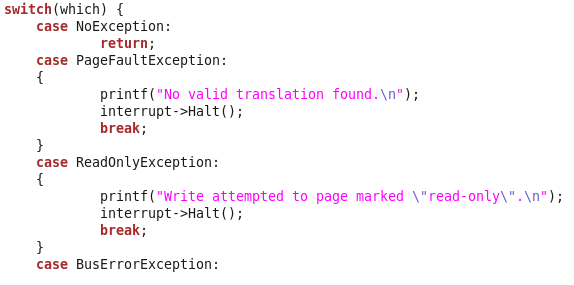
* Khai báo hằng xác định loại syscall và hàm để gọi syscall này trong ./userprog/syscall.h

|  |  |
| --- | --- |
| #define SC\_PrintInt 44  #define SC\_PrintChar 45  #define SC\_ReadInt 46  #define SC\_ReadChar 47  #define SC\_ReadString 48  #define SC\_PrintString 49 | **void** PrintInt (**int** number);  **void** PrintChar (**char** character);  **int** ReadInt ();  **char** ReadChar ();  **void** ReadString(**char** buffer[], **in**t len);  **void** PrintString(**char** buffer[] ); |
| Khai báo loại syscall | Khai báo hàm |

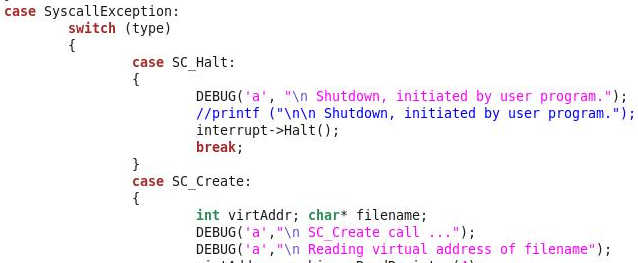
* Khai báo đoạn mã MIPS cho hàm gọi syscall trong file ./test/start.c và start.s



* Khai báo các loại Exception và type syscall của exception Syscall Exception trong file ./userprog/exception.cc



**Khai báo các exception cần xử lý**

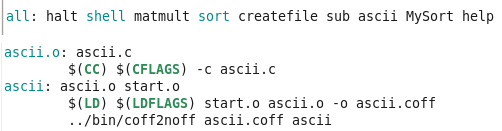


**Khai báo các loại system call cần xử lí trong SyscallException**

* Sau đó cài đặt hàm để xử lý syscall trong file ./userprog/exception.cc
* Biên dịch lại nachos là có thể dùng syscall vừa viết.

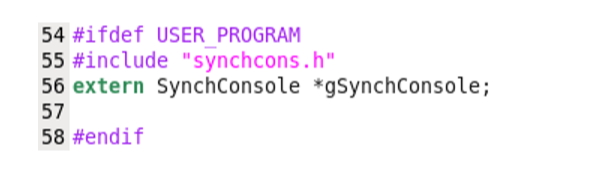
### **b. Viết và biên dịch một chương trình**

* Tạo một file <ten chuong trinh>.c trong ./test/ và viết một chương trình trong đó.
* Thêm khai báo biên dịch trong file ./test/Makefile

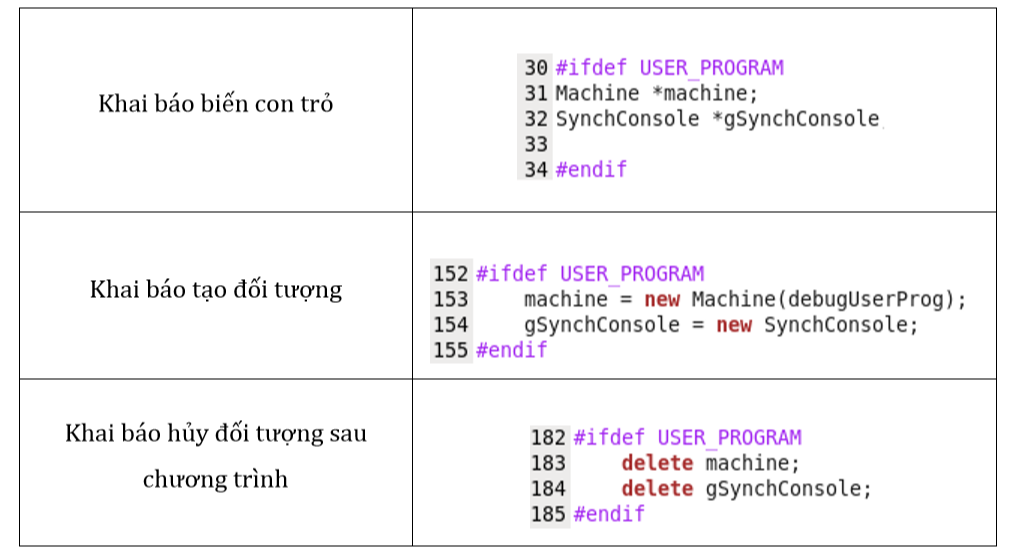


**Khai báo sau all và khai báo đoạn mã biên dịch**

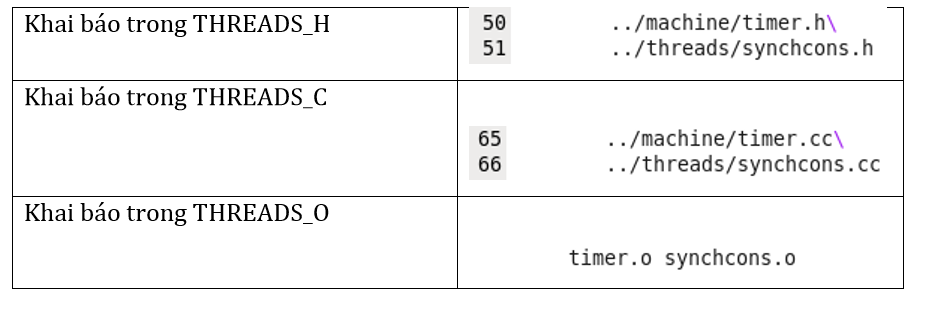
* Biên dịch lại nachos và thực thi qua lệnh: ./userprog/nachos –rs 1023 –x ./test/<ten chuong trinh> c. Thêm lớp vào nachos
* Ở trình bày cách thêm lớp SynchConsole vì đồ án sử dụng lớp này để đọc và xuất từ console.
* Chép file khai báo và cài đặt **synchcons.h** và **synchcons.cc** vào thư mục ./threads/ vào khai báo đối tượng biến toàn cục cho lớp này để có thể gọi sử dụng.
* Trong file system.h: khai báo biến con trỏ toàn cục cho lớp SynchConsole



* Trong file **system.cc**



- Trong code/Makefile thêm dòng lệnh biên dịch cho lớp SynchConsole



## **2.3 Chi tiết thiết kế và cài đặt**

* Để viết các syscall ReadInt(), PrintInt(), ReadChar(), PrintChar(), ReadString(), PrintString() ta cũng thực hiện các bước khai báo cho một syscall như đã nêu ở mục 2.2.a.
* Sau đây trình bày chi tiết cách cài đặt cho từng syscall
* **ReadInt()**

str = new char[11]; **// tạo chuỗi trung gian lưu số được nhập vào ở dạng chuỗi, độ dài dài nhất của số nguyên int là 11 kí số**

len = gSynchConsole->Read(str,11); **// gọi hàm đọc chuỗi kí số**

if (len < 1) { machine->WriteRegister(2,-1); break; } **// nếu không đọc được kí tự nào, trả về thanh ghi kết quả -1 và break**

number = String2Number(str); **// gọi hàm chuyển chuỗi kí số sang số nguyên thực sự, hàm này cũng kiểm tra nếu xuất hiện kí tự không phải kí số thì trả về 0**

delete str; **// giải phóng bộ nhớ**

* **PrintInt()**

number = machine->ReadRegister (4); **// lấy tham số**

buffer = Number2String(number); **// chuyển số nguyên lấy được sang dạng chuỗi kí số, lưu vào chuỗi đệm**

gSynchConsole->Write(buffer,strlen(buffer)); **// ghi ra màn hình console**

delete buffer; **// giải phóng chuỗi đệm**

* **ReadChar()**

char ch;

gSynchConsole->Read(&ch,1); **// đọc 1 byte kí tự**

machine->WriteRegister(2,ch); **// ghi vào thanh ghi**

* **PrintChar()**

ch = machine->ReadRegister (4); **// lấy kí tự từ thanh ghi**

gSynchConsole->Write(&ch,1); **// ghi kí tự ra màn hình**

* **ReadString()**

virtAddr = machine->ReadRegister(4); **// đọc tham số thứ 1 từ thanh ghi, lấy địa chỉ ảo của chuỗi**

len = machine->ReadRegister(5); **// đọc tham số thứ 2 từ thanh ghi, lấy độ dài chuỗi**

str = new char[len + 1]; **// khai báo chuỗi trung gian, thêm 1 cho null**

numbytes = gSynchConsole->Read(str,len); **// đọc chuỗi trung gian từ console**

str[numbytes] = NULL; **// thêm kí tự kết thúc chuỗi**

System2User(virtAddr,len,str); /**/ sao chép chuỗi từ kernel space sang user space**

machine->WriteRegister(2,numbytes); **// trả về số bytes đọc được**

delete str; **// xóa chuỗi trung gian**

* **PrintString()**

virtAddr = machine->ReadRegister(4); **// đọc địa chỉ ảo từ thanh ghi**

**// đọc và ghi ra màn hình từng kí tự cho đến khi gặp NULL**

do {

buff = User2System(virtAddr, 1) ***// chép 1 ký tự vào kernel space trả địa chỉ ra buff***

ch := \*buff ; **// lấy kí tự có địa chỉ là buff**

gSynchConsole->Write(buff, 1); ***// ghi ký tự vừa đọc ra màn hình***

delete buff; ***// xóa vùng nhớ đã cấp trong kernel space***

virtAddr++; ***// tăng địa chỉ chuỗi vào trỏ tới ký tự tiếp theo trong user space***

} while (ch != '\0')

* **Tăng thanh ghi: Sau mỗi lần thực hiện một syscall phải tăng thanh ghi lên 4 đơn vị để thực hiện tiếp syscall tiếp theo**

machine->registers[PrevPCReg] = machine->registers[PCReg];

machine->registers[PCReg] = machine->registers[NextPCReg];

machine->registers[NextPCReg] += 4;

# **Demo chương trình**

❖ Đối với các hàm đọc và ghi số, kí tự, chuỗi ,nhóm tụi em demo trong cùng một file sub.c.

|  |  |  |
| --- | --- | --- |
| STT | Tên | Ảnh demo |
| 1 | Hàm ReadInt và PrintInt |  |
| 2 | Hàm ReadChar và PrintChar |  |
| 3 | Hàm ReadString và PrintString |  |
| 4 | Program MySort |  |
| 5 | Program ascii |  |
| 6 | Program help |  |