

HỆ ĐIỀU HÀNH

ĐỒ ÁN 1

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

MỤC LỤC

I. Thành viên nhóm:	3
II. Báo cáo đồ án:	4
2.1 Mô hình hoạt động của chương trình và hàm trên hệ điều hành Nachos.....	4
2.2 Quy trình cài đặt.....	9
a. Quy trình viết một syscall.....	9
b. Viết và biên dịch một chương trình.....	11
2.3 Chi tiết thiết kế và cài đặt.....	12
III. Demo chương trình:	15

DANH SÁCH THÀNH VIÊN THAM GIA ĐỒ ÁN

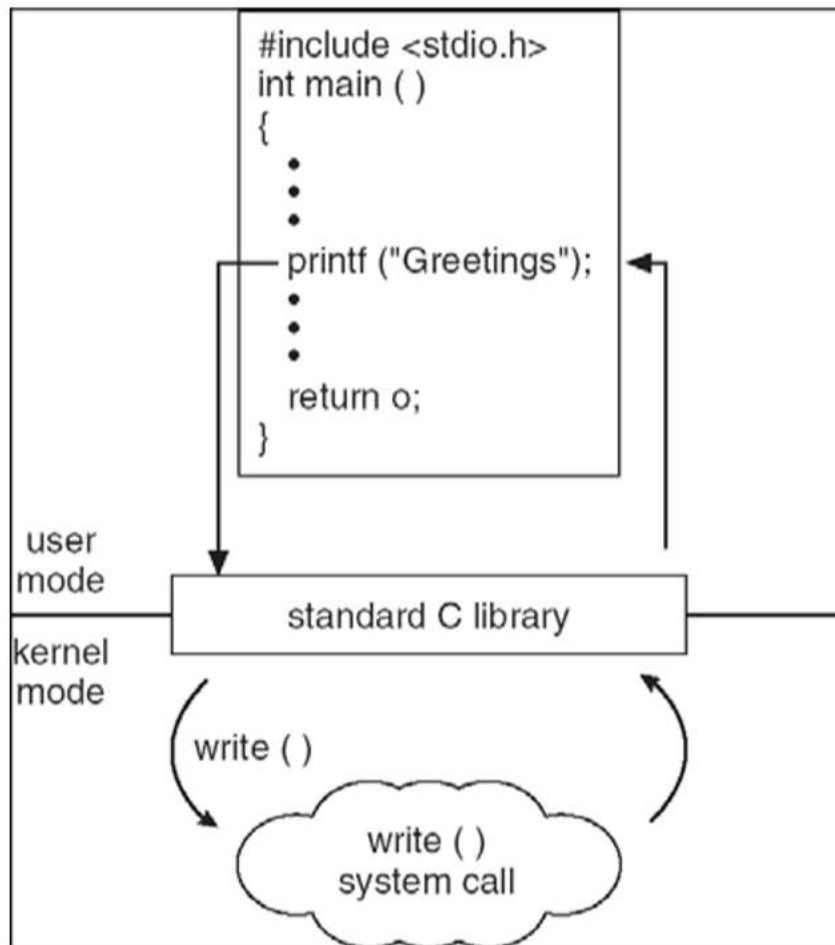
I. Thành viên nhóm:

<ul style="list-style-type: none">• Họ tên:• MSSV:• Email:• SĐT:	<ul style="list-style-type: none">➤ Cao Minh Hiếu➤ 1512157➤ 1512157@students.hcmus.edu.vn➤ 0165.259.2239
<ul style="list-style-type: none">• Họ tên:• MSSV:• Email:• SĐT:	<ul style="list-style-type: none">➤ Nguyễn Duy Tâm➤ 1512479➤ tam.nguyen@goldeneyetech.com.vn➤ 01666.942.492

II. Báo cáo đồ án

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

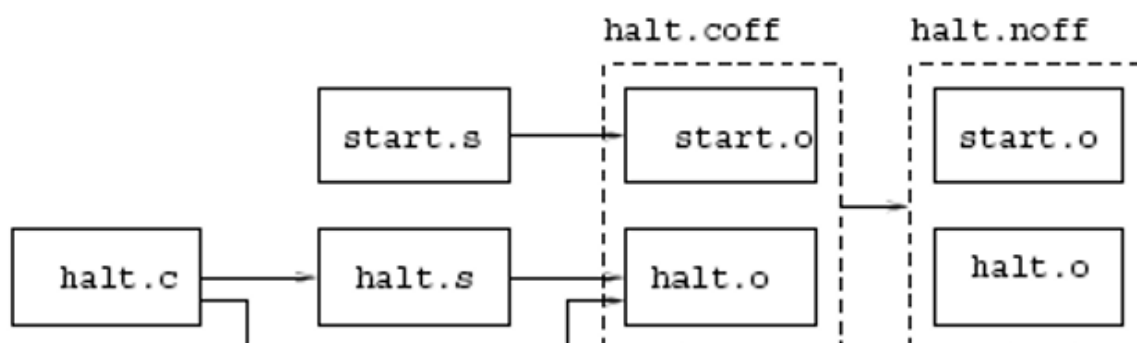
- Mô hình thực thi khi gọi hàm của hệ thống trong Nachos



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 nghĩa trong syscall.h: $\$v0 = 0 + \text{SC_Halt}$. Tiếp đó lời gọi hàm 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.

```

switch (which) {
    case Exception_1:
        ...
    case SyscallException:
        switch(type) {
            case ...
                ...
        }
        ...
}

```

```

#define IN_ASM
#include "syscall.h"

        .text
        .align 2

        .globl __start
        .ent  __start

__start:
        jal    main
        move   $4,$0
        jal    Exit    /* if we return from main, exit(0) */
        .end  __start

        .globl Halt
        .ent   Halt

Halt:
        addiu  $2,$0,SC_Halt
        syscall
        j      $31
        .end  Halt

/* dummy function to keep gcc happy */
        .globl __main
        .ent   __main

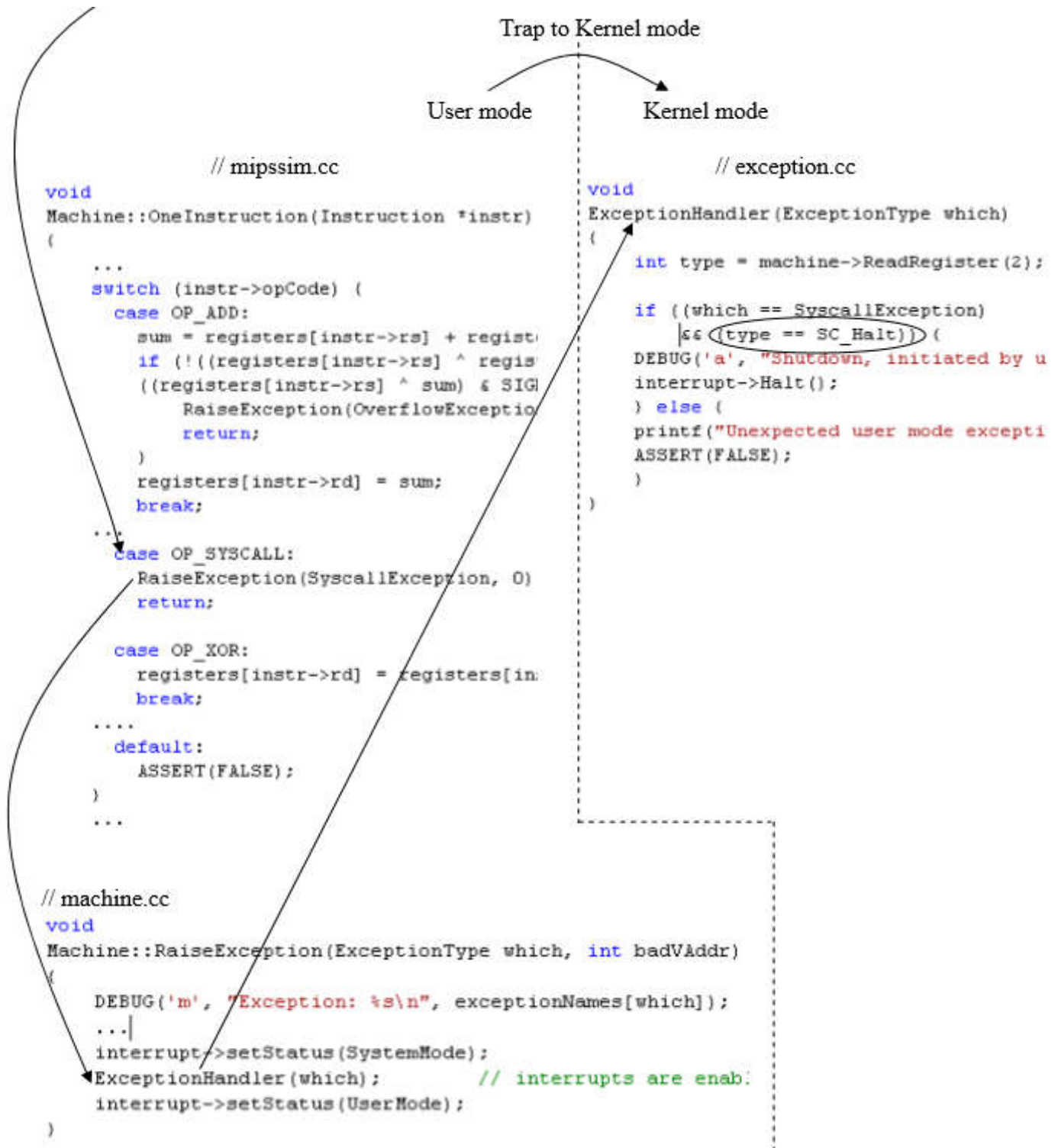
__main:
        j      $31
        .end  __main

// halt.s
10 gcc2_compiled.:
11 __gnu_compiled_c:
12         .text
13         .align 2
14         .globl main
15         .ent  main
16 main:
17         .frame $fp,24,$31
18         .mask 0xc0000000,-4
19         .fmask 0x00000000,0
20         subu  $sp,$sp,24
21         sw    $31,20($sp)
22         sw    $fp,16($sp)
23         move  $fp,$sp
24         jal   __main
25         jal   Halt
26 $L1:
27         move  $sp,$fp
28         lw    $31,20($sp)
29         lw    $fp,16($sp)
30         addu  $sp,$sp,24
31         j     $31
32         .end  main

```

vars= 0, regs= 2/0, args= 16, extra= 0

sp not trusted here



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	void PrintInt (int number);
#define SC_PrintChar	45	void PrintChar (char character);
#define SC_ReadInt	46	int ReadInt ();
#define SC_ReadChar	47	char ReadChar ();
#define SC_ReadString	48	void ReadString(char buffer[], int len);
#define SC_PrintString	49	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

```

        .globl ReadInt
        .ent   ReadInt
ReadInt:
        addiu $2,$0,SC_ReadInt
        syscall
        j      $31
        .end ReadInt

```

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

```
switch(which) {
    case NoException:
        return;
    case PageFaultException:
    {
        printf("No valid translation found.\n");
        interrupt->Halt();
        break;
    }
    case ReadOnlyException:
    {
        printf("Write attempted to page marked \"read-only\".\n");
        interrupt->Halt();
        break;
    }
    case BusErrorException:
```

Khai báo các exception cần xử lý

```
case SyscallException:
    switch (type)
    {
        case SC_Halt:
        {
            DEBUG('a', "\n Shutdown, initiated by user program.");
            //printf ("\n\n Shutdown, initiated by user program.");
            interrupt->Halt();
            break;
        }
        case SC_Create:
        {
            int virtAddr; char* filename;
            DEBUG('a', "\n SC_Create call ...");
            DEBUG('a', "\n Reading virtual address of filename");
```

Khai báo các loại system call cần xử lý 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 chương trình>.c trong ./test/ và viết một chương trình trong đó.
- Thêm khai báo biên dịch trong file ./test/Makefile

```
all: halt shell matmult sort createfile sub ascii MySort help

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
```

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 chương trình> c. Thêm lớp vào nachos
- Ở trình bày cách thêm lớp SynchConsole vì đề 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

```
54 #ifdef USER_PROGRAM
55 #include "synchcons.h"
56 extern SynchConsole *gSynchConsole;
57
58 #endif
```

- Trong file **system.cc**

Khai báo biến con trỏ	<pre> 30 #ifdef USER_PROGRAM 31 Machine *machine; 32 SynchConsole *gSynchConsole; 33 34 #endif </pre>
Khai báo tạo đối tượng	<pre> 152 #ifdef USER_PROGRAM 153 machine = new Machine(debugUserProg); 154 gSynchConsole = new SynchConsole; 155 #endif </pre>
Khai báo hủy đối tượng sau chương trình	<pre> 182 #ifdef USER_PROGRAM 183 delete machine; 184 delete gSynchConsole; 185 #endif </pre>

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

Khai báo trong THREADS_H	<pre> 50 ../machine/timer.h\ 51 ../threads/synchcons.h </pre>
Khai báo trong THREADS_C	<pre> 65 ../machine/timer.cc\ 66 ../threads/synchcons.cc </pre>
Khai báo trong THREADS_O	<pre> timer.o synchcons.o </pre>

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 trở 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];

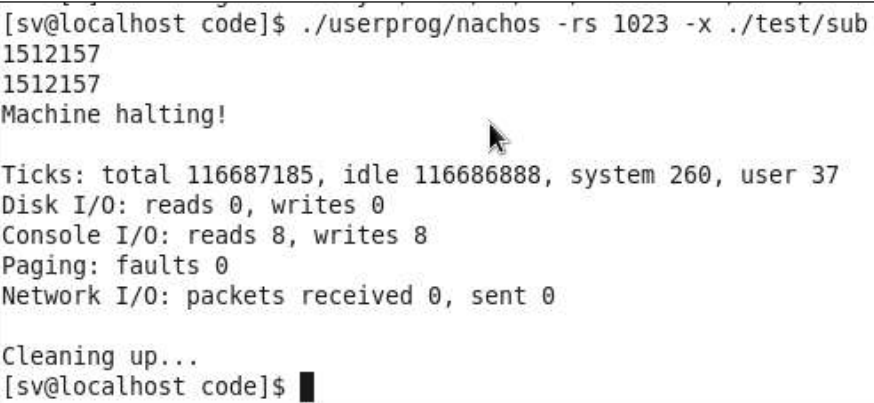
machine->registers[PCReg] = machine->registers[NextPCReg];

machine->registers[NextPCReg] += 4;

```

III. 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	 <pre> [sv@localhost code]\$./userprog/nachos -rs 1023 -x ./test/sub 1512157 1512157 Machine halting! Ticks: total 116687185, idle 116686888, system 260, user 37 Disk I/O: reads 0, writes 0 Console I/O: reads 8, writes 8 Paging: faults 0 Network I/O: packets received 0, sent 0 Cleaning up... [sv@localhost code]\$ █ </pre>

2	Hàm ReadChar và PrintChar	<pre>[sv@localhost code]\$./userprog/nachos -rs 1023 -x ./test/sub x x Machine halting! Ticks: total 144508726, idle 144508558, system 130, user 38 Disk I/O: reads 0, writes 0 Console I/O: reads 2, writes 2 Paging: faults 0 Network I/O: packets received 0, sent 0 Cleaning up... [sv@localhost code]\$</pre>
3	Hàm ReadString và PrintString	<pre>[sv@localhost code]\$./userprog/nachos -rs 1023 -x ./test/sub Em gai mua Em gai mua Machine halting! Ticks: total 160281774, idle 160278633, system 3100, user 41 Disk I/O: reads 0, writes 0 Console I/O: reads 11, writes 257 Paging: faults 0 Network I/O: packets received 0, sent 0 Cleaning up... [sv@localhost code]\$</pre>
4	Program MySort	<pre>[sv@localhost code]\$./userprog/nachos -rs 1023 -x ./test/MySort Nhap N (0 < N <= 100): 5 Nhap a[0] : 9 Nhap a[1] : 2 Nhap a[2] : 5 Nhap a[3] : 1 Nhap a[4] : 7 1 2 5 7 9 Machine halting! Ticks: total 1018392512, idle 1018359064, system 32300, user 1148 Disk I/O: reads 0, writes 0 Console I/O: reads 12, writes 2831 Paging: faults 0 Network I/O: packets received 0, sent 0 Cleaning up... [sv@localhost code]\$</pre>

ĐỒ ÁN #1: NACHOS

[illegible]