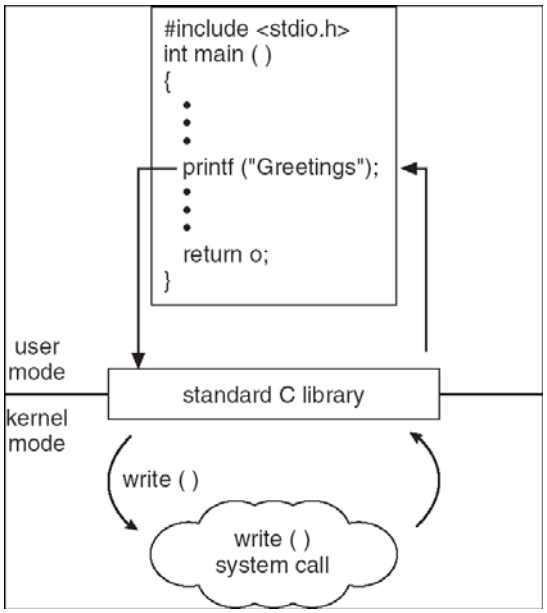
GIAO TIẾP GIỮA HĐH NACHOS VÀ CHƯƠNG TRÌNH

Mục tiêu:

- Trình bày giao tiếp giữa HĐH Nachos và chương trình người dùng
- Hướng dẫn cách xây dựng một system call trên HĐH Nachos

1. Mô hình giao tiếp chung giữa HĐH và chương trình người dùng



2. Giao tiếp giữa HĐH Nachos và chương trình người dùng

Cũng tương tự như mô hình giao tiếp chung giữa HĐH và chương trình người dùng ở trên, tuy nhiên Nachos là một HĐH nhỏ gọn nhằm mục đích phục vụ cho việc tìm hiểu và xây dựng HĐH nên có một số đặc điểm riêng. Trong phần này, sẽ trình bày chi tiết về cách thức giao tiếp giữa HĐH Nachos và chương trình ứng dụng để từ đó có được những ý niệm về cách thức giao tiếp chung giữa HĐH và chương trình người dùng

Một câu hỏi đặt ra là chương trình người dùng sẽ được thực thi như thế nào trên các hệ điều hành ?

Ta sẽ phân tích quá trình thực thi của một chương trình được viết trên hệ điều hành Nachos để trả lời câu hỏi này. Từ đó, có thể thấy được toàn cảnh cách giao tiếp của hệ điều hành Nachos với chương trình người dùng.

Xem xét chương trình **halt.c** (trong thư mục */code/test/*) được viết để chạy trên hệ điều hành Nachos. Chương trình này gọi một hàm duy nhất Halt() để tắt máy ảo Nachos. Chúng ta sẽ phân tích quá trình thực hiện chương trình này:

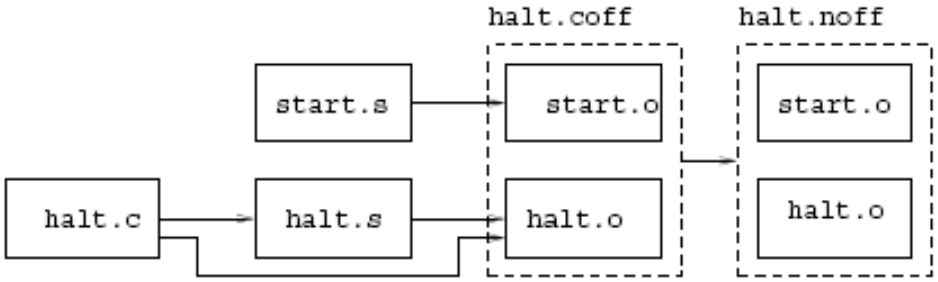
```
#include "syscall.h"
int main()
{
   Halt();
   /* not reached */
}
```

Trước khi chương trình này có thể thực thi được, nó phải được biên dịch. Đối với Nachos, tất cả các chương trình trong thư mục */code/test/* đều được biên dịch khi biên dịch Nachos, tuy nhiên ta hoàn toàn có thể biên dịch các chương trình này mà không cần phải biên dịch lại Nachos như sau:

% …../cross-compiler/decstation-ultrix/bin/gcc -I …../nachos/code/userprog -I…../nachos/code/threads -o ……/test/halt …./test/halt.c

Quá trình biên dịch chương trình này gồm 3 giai đoạn như sau:

- Chương trình **halt.c** được biên dịch bởi cross-compiler gcc (được tạo ra trong bài "Biên dịch và Cài đặt Nachos") thành tập tin **halt.s** là mã hợp ngữ chạy trên kiến trúc MIPS
- Tập tin **halt.s** này sẽ được liên kết với tập tin **start.s** để tạo thành tập tin **halt.coff**, là định dạng thực thi trên hệ điều hành Linux cho kiến trúc máy MIPS. (Lưu ý, do Nachos được xây dựng đơn giản nhằm mục đích giúp tìm hiểu nên người ta sử dụng tập tin start.s thay cho toàn bộ thư viện C trong mô hình chung ở trên.)
- Tập tin **halt.coff** được chuyển thành tập tin **halt.noff**, là định dạng thực thi trên hệ điều hành Nachos cho kiến trúc máy MIPS, sử dụng tiện ích "coff2noff" được cung cấp sẵn trong thư mục */code/bin/*



Như vậy, sau khi biên dịch thành công, ta có tập tin **halt**(.noff) là chương trình thực thi trên Nachos. Để thực hiện chương trình halt này, gõ lệnh:

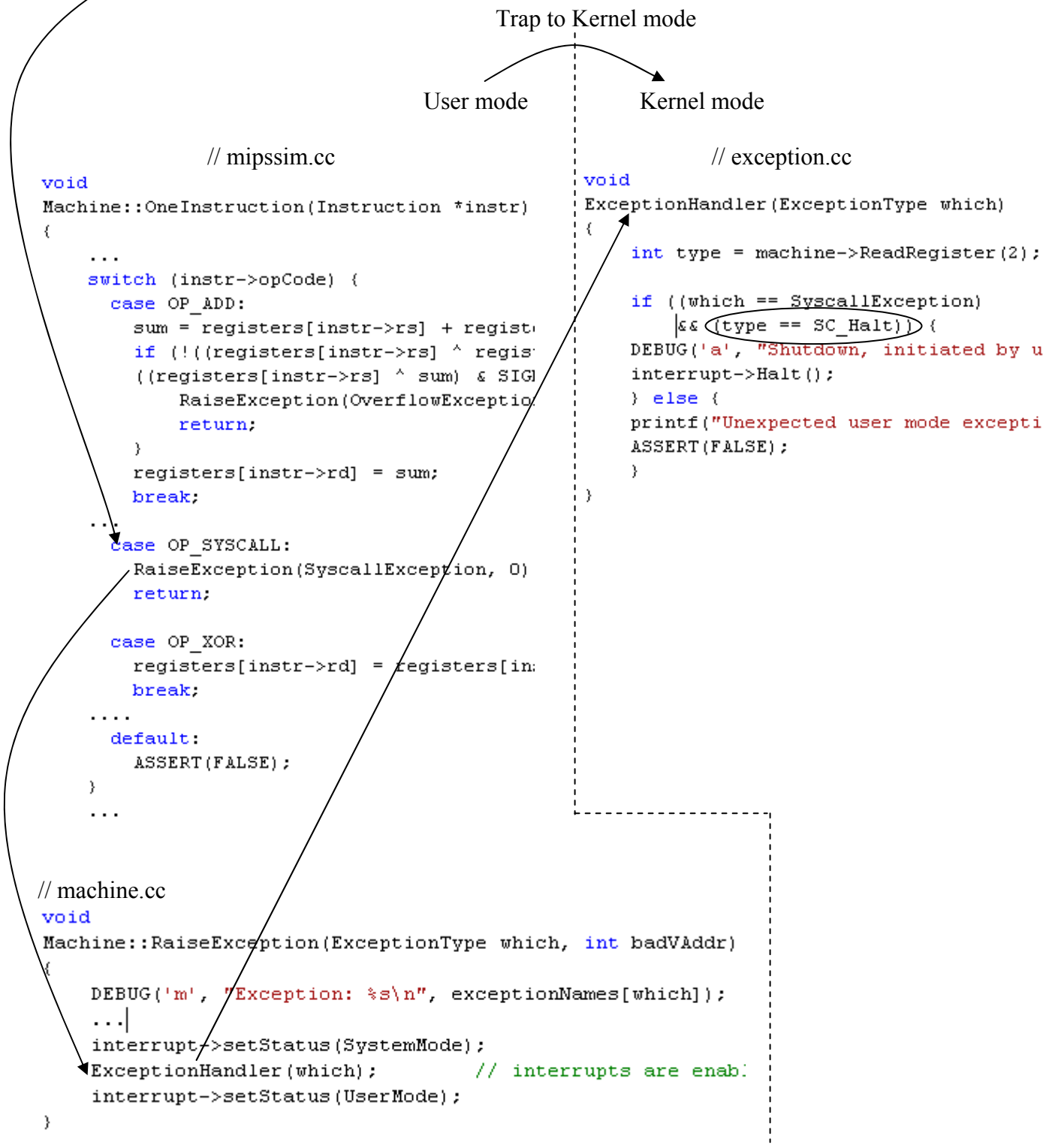
% ./userprog/nachos –rs 1023 –x ./test/halt

Khi thực thi một chương trình trên Nachos thì chương trình **start** luôn được thực thi trước và quá trình thực thi của chương trình **halt** sẽ được thực hiện như hình mô tả trong hình sau:

```
/* Start.s
 *      Assembly language assist for user programs running on top of Nachos.
 *
 *      Since we don't want to pull in the entire C library, we define
 *      what we need for a user program here, namely Start and the system
 *      calls.
 */

#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              # vars= 0, regs= 2/0, args= 16, extra= 0
 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                 # sp not trusted here
 28         lw      $31,20($sp)
 29         lw      $fp,16($sp)
 30         addu    $sp,$sp,24
 31         j       $31
 32         .end    main
```



Trong ví dụ halt thì một system call **SC_Halt** được xây dựng trước và được gọi thực hiện khi chương trình người dùng gọi hàm Halt(). Như vậy, muốn phục vụ người dùng có thể thực thi được các công việc khác nhau như: tạo tập tin, tạo tiến trình,… thì người phát triển hệ điều hành phải xây dựng một bộ system call đủ để phục vụ các yêu cầu này. Một điểm lưu ý, system call cũng là một hàm xử lý nhưng ở *kernel mode* khác với một hàm của chương trình người dùng ở *user mode*

và không thể truyền tham số trực tiếp cho system call. Như vậy nếu một hàm trong chương trình người dùng yêu cầu truyền một tham số thì system call này sẽ nhận tham số này như thế nào? Trong ví dụ tiếp theo, sẽ hướng dẫn cách tạo system call tạo tập tin **SC_Create** nhằm phục vụ yêu cầu của chương trình người dùng muốn tạo một tập tin và cách system call trong hệ điều hành Nachos nhận tham số (trong ví dụ này là tên tập tin)

Bước 1: Trong tập tin */code/userprog/syscall.h* thêm dòng khai báo syscall mới (đã được thực hiện sẵn, nhưng cần sửa lại kiểu trả về của hàm Create từ void thành **int**)

```
#define SC_Create 4
int Create(char *name);
```

Bước 2: Trong tập tin */code/test/start.c* và /code/*test/start.s* thêm dòng (đã được thực hiện sẵn):

```
        .globl  Create
        .ent    Create
Create:
        addiu $2,$0,SC_Create
        syscall
        j       $31
        .end    Create
```

Bước 3: Trong tập tin *code/userprog/exception.cc* sửa điều kiện **if …** thành **switch … case** như sau:

```
// Input: reg4 -filename (string)
// Output: reg2 -1: error and 0: success
// Purpose: process the event SC_Create of System call

// mã system call sẽ được đưa vào thanh ghi r2 (có thể xem lại phần xử lý cho
// system call Halt trong tập tin start.s ở trên)
// tham số thứ 1 sẽ được đưa vào thanh ghi r4
// tham số thứ 2 sẽ được đưa vào thanh ghi r5
// tham số thứ 3 sẽ được đưa vào thanh ghi r6
// tham số thứ 4 sẽ được đưa vào thanh ghi r7
// kết quả thực hiện của system call sẽ được đưa vào thanh ghi r2

switch (which) {
  case NoException:
     return;

  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");
                    // Lấy tham số tên tập tin từ thanh ghi r4
                    virtAddr = machine->ReadRegister(4);
                    DEBUG ('a',"\n Reading filename.");
                    // MaxFileLength là = 32
                    filename = User2System(virtAddr,MaxFileLength+1);
                    if (filename == NULL)
                    {
                        printf("\n Not enough memory in system");
                        DEBUG('a',"\n Not enough memory in system");
                        machine->WriteRegister(2,-1); // trả về lỗi cho chương
                                                      // trình người dùng
                        delete filename;
                        return;
                    }
                    DEBUG('a',"\n Finish reading filename.");
                    //DEBUG('a',"\n File name : '"<<filename<<"'");
                    // Create file with size = 0
                    // Dùng đối tượng fileSystem của lớp OpenFile để tạo file,
                    // việc tạo file này là sử dụng các  thủ tục tạo file của hệ điều
                    // hành Linux, chúng ta không quản ly trực tiếp các block trên
                    // đĩa cứng cấp phát cho file, việc quản ly các block của file
                    // trên ổ đĩa là một đồ án khác
                    if (!fileSystem->Create(filename,0))
                    {
                        printf("\n Error create file '%s'",filename);
                        machine->WriteRegister(2,-1);
                        delete filename;
                        return;
                    }
                    machine->WriteRegister(2,0); // trả về cho chương trình
                                                 // người dùng thành công
                    delete filename;
                    break;
                }
            default:
                printf("\n Unexpected user mode exception (%d %d)", which,
                                                                    type);
                interrupt->Halt();
        }
    }
```

Trong phần xử lý cho system call tạo tập tin có sử dụng hàm User2System được viết như sau:

```
// Input: - User space address (int)
//      - Limit of buffer (int)
// Output:- Buffer (char*)
// Purpose: Copy buffer from User memory space to System memory space
char* User2System(int virtAddr,int limit)
{
    int i;// index
    int oneChar;
    char* kernelBuf = NULL;
    kernelBuf = new char[limit +1];//need for terminal string
    if (kernelBuf == NULL)
       return kernelBuf;
    memset(kernelBuf,0,limit+1);
    //printf("\n Filename u2s:");
    for (i = 0 ; i < limit ;i++)
    {
        machine->ReadMem(virtAddr+i,1,&oneChar);
        kernelBuf[i] = (char)oneChar;
        //printf("%c",kernelBuf[i]);
        if (oneChar == 0)
           break;
    }
    return kernelBuf;
}
```

Ngược lại ta có hàm System2User như sau:

```
// Input: - User space address (int)
//      - Limit of buffer (int)
//      - Buffer (char[])
// Output:- Number of bytes copied (int)
// Purpose: Copy buffer from System memory space to User  memory space
int   System2User(int virtAddr,int len,char* buffer)
{
    if (len < 0) return -1;
    if (len == 0)return len;
    int i = 0;
    int oneChar = 0 ;
   do{
         oneChar= (int) buffer[i];
         machine->WriteMem(virtAddr+i,1,oneChar);
         i ++;
  }while(i < len && oneChar != 0);
  return i;
}
```

Bước 4: Viết chương trình ở mức người dùng để kiểm tra – **createfile.c**

```
#include "syscall.h"
#include "copyright.h"
#define maxlen 32

int
main()
{
    int len;
    char filename[maxlen +1];

    /*Create a file*/
    if (Create("text.txt") = = -1)
    {
        // xuất thông báo lỗi tạo tập tin
    }
    else
    {
        // xuất thông báo tạo tập tin thành công
    }
    Halt();
}
```

Bước 5: Thêm đoạn vào **Makefile** trong */code/test/*

Thêm **createfile** vào dòng all:

all: halt shell matmult sort **createfile**

Thêm đoạn sau phía sau **matmult**:

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

Bước 6: Biên dịch lại Nachos

Bước 7: Thực thi chương trình **createfile**.

Nếu chương trình không báo lỗi "**Not enough memory in system**" hoặc "**Error create file …**" thì xem như thành công

Như vậy, ta đã hoàn thành việc tạo system call tạo tập tin để phục vụ cho lời gọi hàm Create trong chương trình người dùng. Tuy nhiên, trong chương trình người dùng, để thể hiện kết quả tạo tập tin rõ ràng hơn, cần thực hiện xuất các thông báo tương ứng như sau:

```
#include "syscall.h"
#include "copyright.h"
#define maxlen 32

int
main()
{
    int len;
    char filename[maxlen +1];

    /*Create a file*/
    if (Create("text.txt") = = -1)
    {
        print("\nCreate file ~");
        print(filename);
        print(" fail.");
    }
    else
    {
        print("\nCreate file ~");
        print(filename);
        print(" success.~");
    }
    Halt();
}
```

Tuy nhiên, hiện tại HĐH Nachos chưa hỗ trợ hàm **print** để xuất 1 chuỗi ra màn hình. Do đó ta cần xây dựng thêm 1 system call phục vụ cho việc xuất 1 chuỗi ký tự trong chương trình người dùng. Quá trình xây dựng system call này cũng tương tự quá trình xây dựng system call tạo tập tin.

……Nhiệm vụ tiếp theo

- Tìm hiểu ý nghĩa các lệnh xử lý của system call tạo tập tin
- Tìm hiểu các thành phần cơ bản về quản lý tập tin, quản lý tiến trình, quản lý bộ nhớ mà HĐH Nachos hỗ trợ sẵn
- Xây dựng system call hỗ trợ việc xuất 1 chuỗi ra màn hình
- Xây dựng các systam call còn lại để quản lý tập tin: mở tập tin (open), đọc tập tin (read), ghi tập tin (write), đóng tập tin (close), định vị (seek).