四川大學
国家示范性软件学院
SCU Software college..

Operating System

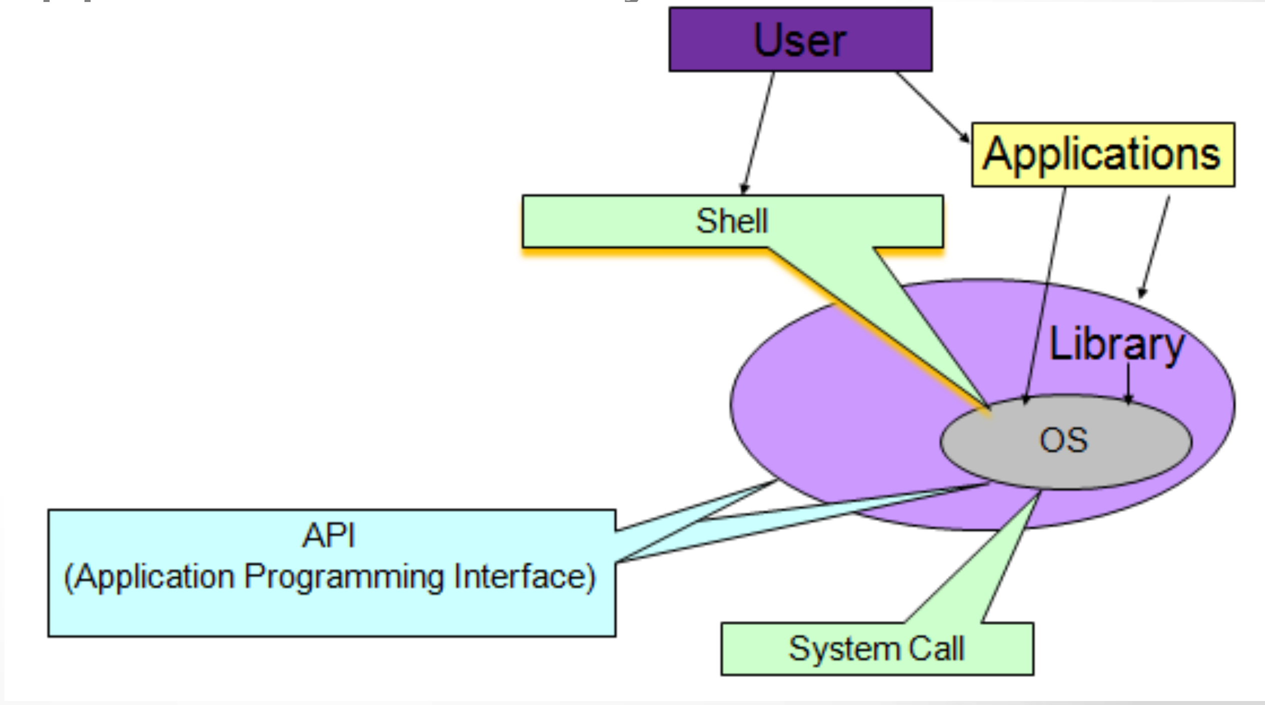
Nachos Project I: System Call
Introduction & Requirement

目录

目录

Review系统调用$_{1/2}$

- What is it
  - Funcitons
    - provided/implemented by OS
    - used by Application or Library
- How many
  - Unix/Linux
  - Windows
  - …
-



Review系统调用$_{2/2}$

- What is it

  linux内核中设置了一组用于实现各种系统功能的子程序，称为系统调用。用户可以通过系统调用命令在自己的应用程序中调用它们。

- 系统调用和普通函数调用之间的区别

  系统调用由操作系统核心提供，运行于核心态；而普通的函数调用由函数库或用户自己提供，运行于用户态。

目录

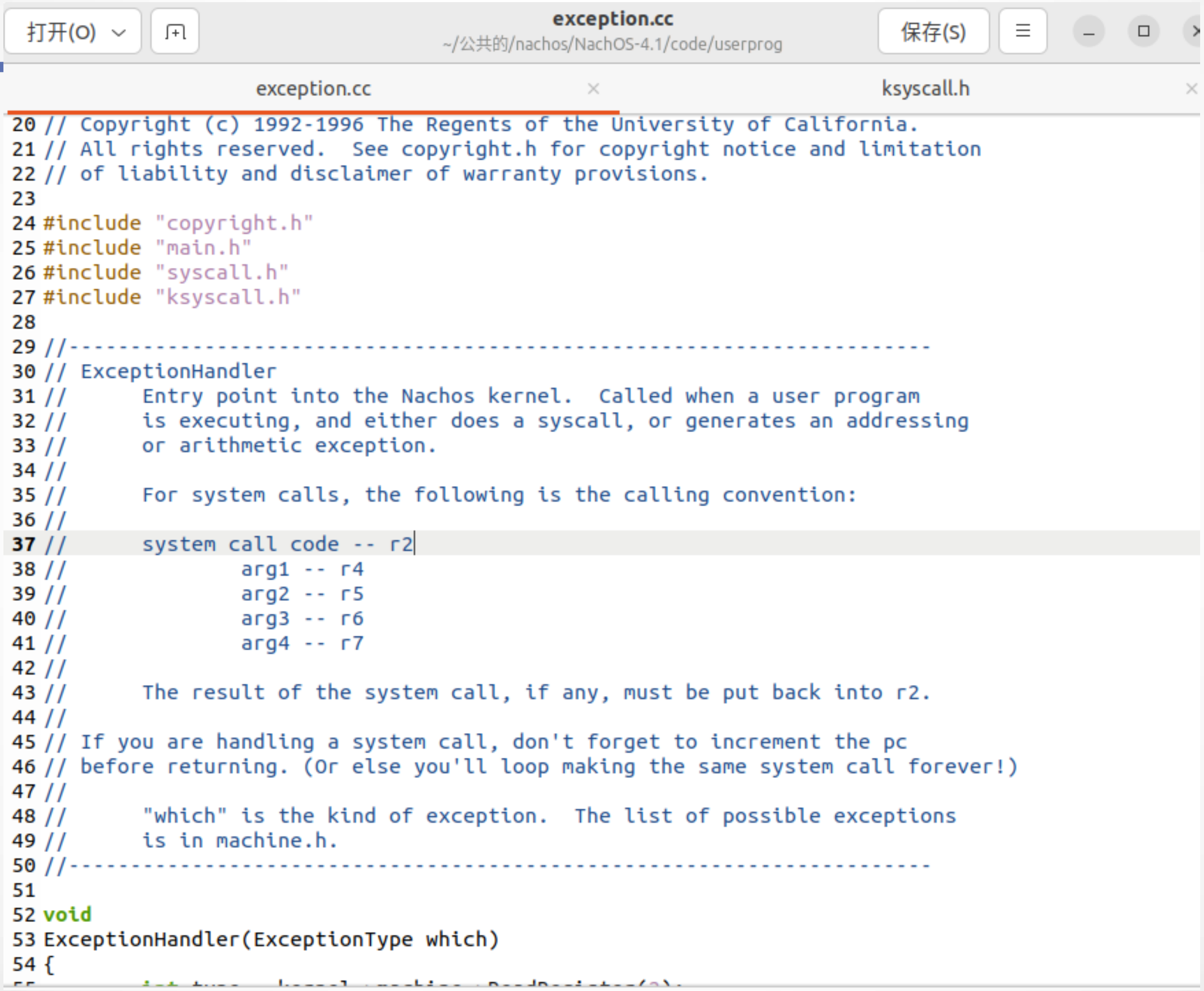
Nachos的系统调用$_{1/4}$

- 定义在ksyscall.h头文件中
  - 定义了两个系统调用：
    - SysHalt( )：实现Halt操作
    - SysAdd( )：实现两个数的相加操作

Nachos的系统调用$_{2/4}$

- 实现:
  - 系统调用的入口处理函数是ExceptionHandler函数
  - 类型为SyscallException
  - start.s同ExceptionHandler()配合使用，完成整个调用过程

四川大學
国家示范性软件学院
SCU Software college.

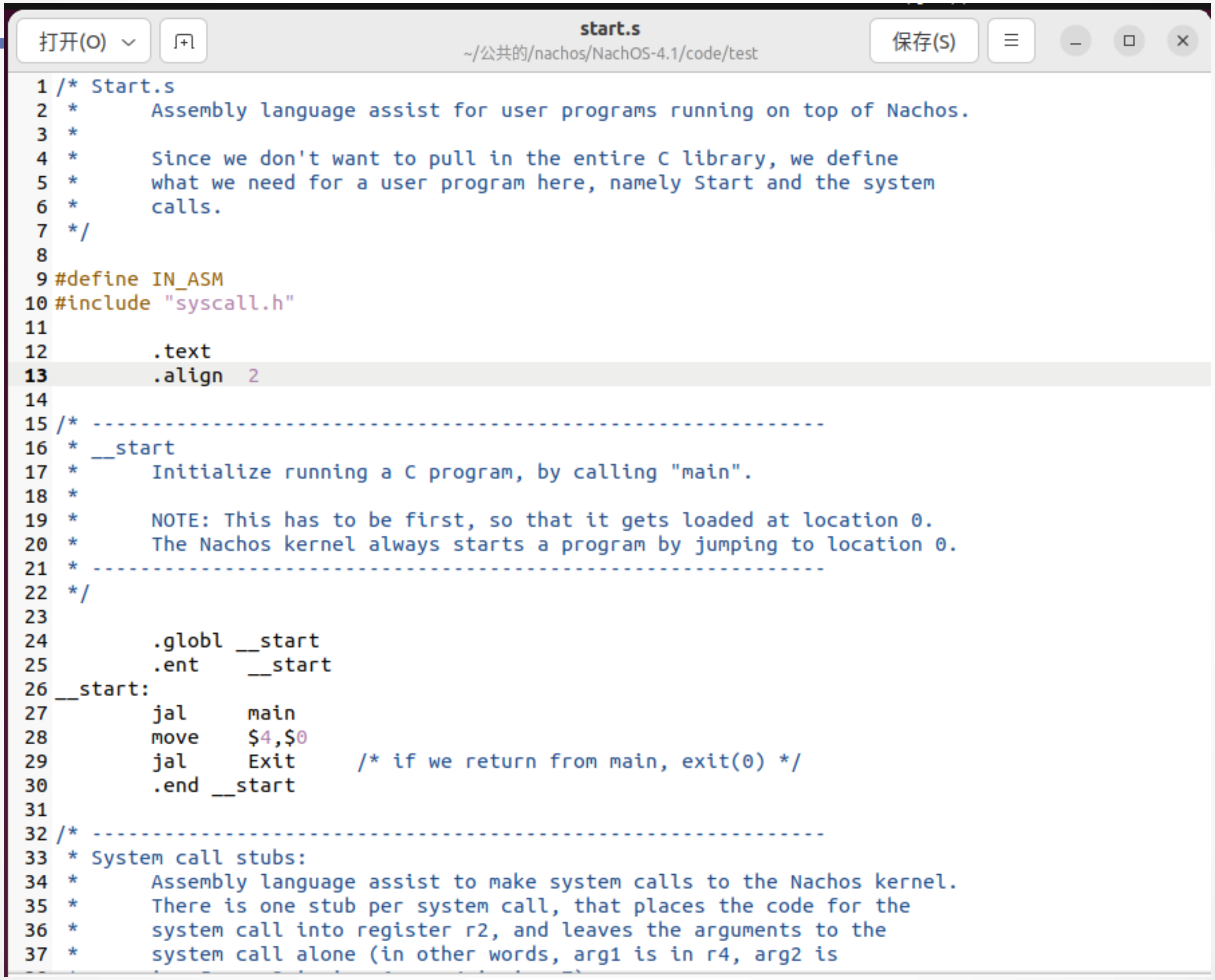


```
// Copyright (c) 1992-1996 The Regents of the University of California.
// All rights reserved.  See copyright.h for copyright notice and limitation
// of liability and disclaimer of warranty provisions.

#include "copyright.h"
#include "main.h"
#include "syscall.h"
#include "ksyscall.h"

//----------------------------------------------------------------------
// ExceptionHandler
//      Entry point into the Nachos kernel.  Called when a user program
//      is executing, and either does a syscall, or generates an addressing
//      or arithmetic exception.
//
//      For system calls, the following is the calling convention:
//
//      system call code -- r2
//              arg1 -- r4
//              arg2 -- r5
//              arg3 -- r6
//              arg4 -- r7
//
//      The result of the system call, if any, must be put back into r2.
//
// If you are handling a system call, don't forget to increment the pc
// before returning. (Or else you'll loop making the same system call forever!)
//
//      "which" is the kind of exception.  The list of possible exceptions
//      is in machine.h.
//----------------------------------------------------------------------

void
ExceptionHandler(ExceptionType which)
{
```

Nachos的系统调用$_{3/4}$

- ExceptionHandler()的部分代码

```
void ExceptionHandler(ExceptionType which){
    //取出系统调用代码
    int type = machine->ReadRegister(2);
    DEBUG(dbgSys, "Received Exception " <<
    which << " type: " << type << "\n");
      switch (which) {
      case SyscallException:
        switch(type) {
        case SC_Halt:
          DEBUG(dbgSys, "Shutdown, initiated by
    user program.\n");
          SysHalt();
          ASSERTNOTREACHED();
          break;
        default:
          cerr << "Unexpected system call "
    << type << "\n";
          break;
        }
        break;
      default:
        cerr << "Unexpected user mode
    exception" << (int)which << "\n";
        break;
      }
      ASSERTNOTREACHED();
    }
```



```
/* 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

/* --------------------------------------------------------------
 * __start
 *      Initialize running a C program, by calling "main".
 *
 *      NOTE: This has to be first, so that it gets loaded at location 0.
 *      The Nachos kernel always starts a program by jumping to location 0.
 * --------------------------------------------------------------
 */

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

/* --------------------------------------------------------------
 * System call stubs:
 *      Assembly language assist to make system calls to the Nachos kernel.
 *      There is one stub per system call, that places the code for the
 *      system call into register r2, and leaves the arguments to the
 *      system call alone (in other words, arg1 is in r4, arg2 is
```

Nachos的系统调用$_{4/4}$

- start.s的部分代码

```
.globl              Halt
.ent                Halt
Halt:     addin               $2, $0, SC_Halt
syscall
J                   $31
.end      Halt
```

其中addin $2, $0, SC_Halt语句的作用是在r2寄存器中存放系统调用的类别码SC_Halt，即Halt系统调用。

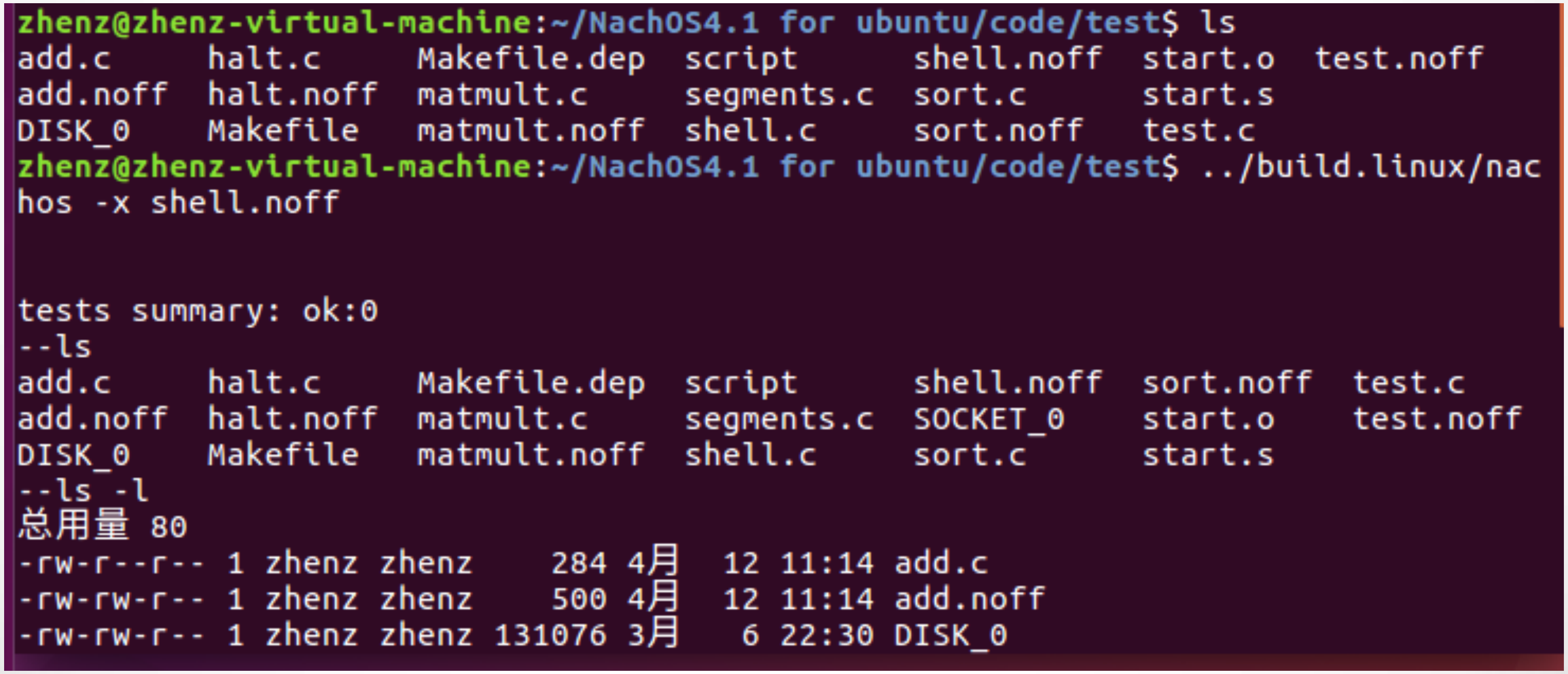
目录

实验要求$_{1/2}$

- 实现Nachos的基本系统调用
  - Write( )
  - Read( )
  - Exec( )
  - Join( )
- 编译和运行code/test目录下的shell

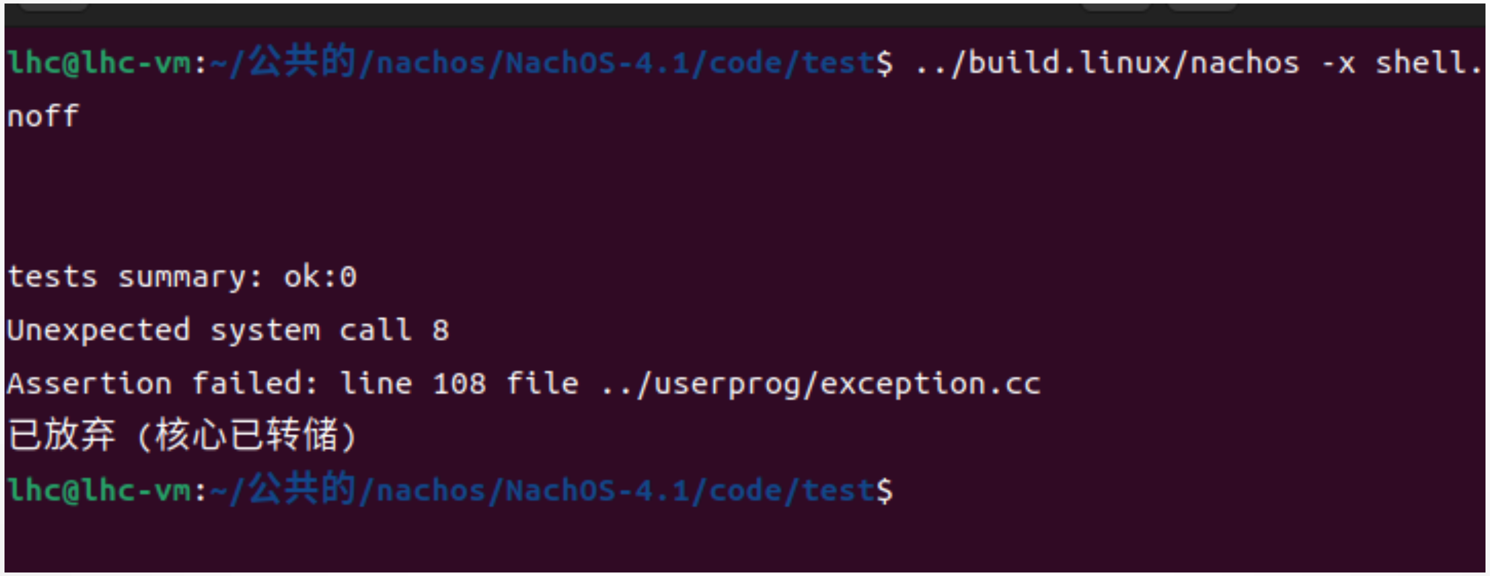
实验要求$_{2/2}$

- 完成了实验后，code/test目录下的shell运行结果



```
zhenz@zhenz-virtual-machine:~/NachOS4.1 for ubuntu/code/test$ ls
add.c     halt.c     Makefile.dep  script      shell.noff  start.o  test.noff
add.noff  halt.noff  matmult.c     segments.c  sort.c      start.s
DISK_0    Makefile   matmult.noff  shell.c     sort.noff   test.c
zhenz@zhenz-virtual-machine:~/NachOS4.1 for ubuntu/code/test$ ../build.linux/nac
hos -x shell.noff


tests summary: ok:0
--ls
add.c     halt.c     Makefile.dep  script      shell.noff  sort.noff  test.c
add.noff  halt.noff  matmult.c     segments.c  SOCKET_0    start.o    test.noff
DISK_0    Makefile   matmult.noff  shell.c     sort.c      start.s
--ls -l
总用量 80
-rw-r--r-- 1 zhenz zhenz    284 4月  12 11:14 add.c
-rw-rw-r-- 1 zhenz zhenz    500 4月  12 11:14 add.noff
-rw-rw-r-- 1 zhenz zhenz 131076 3月   6 22:30 DISK_0
```

- 要先完成对exception.cc和ksyscall.h的修改，否则无法正常编译运行shell.noff



```
lhc@lhc-vm:~/公共的/nachos/NachOS-4.1/code/test$ ../build.linux/nachos -x shell.
noff


tests summary: ok:0
Unexpected system call 8
Assertion failed: line 108 file ../userprog/exception.cc
已放弃 (核心已转储)
lhc@lhc-vm:~/公共的/nachos/NachOS-4.1/code/test$
```

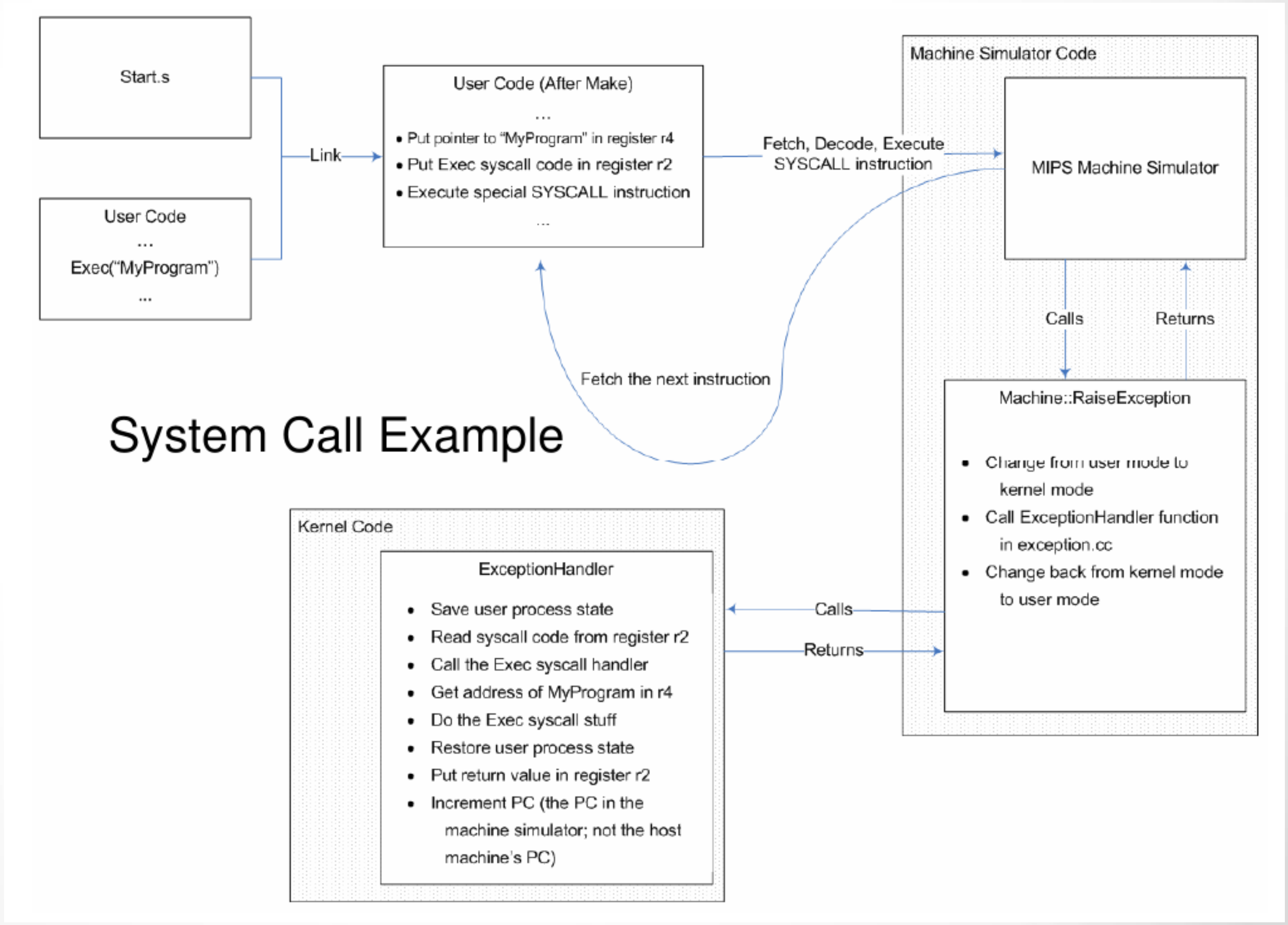
目录

实验提示1/4

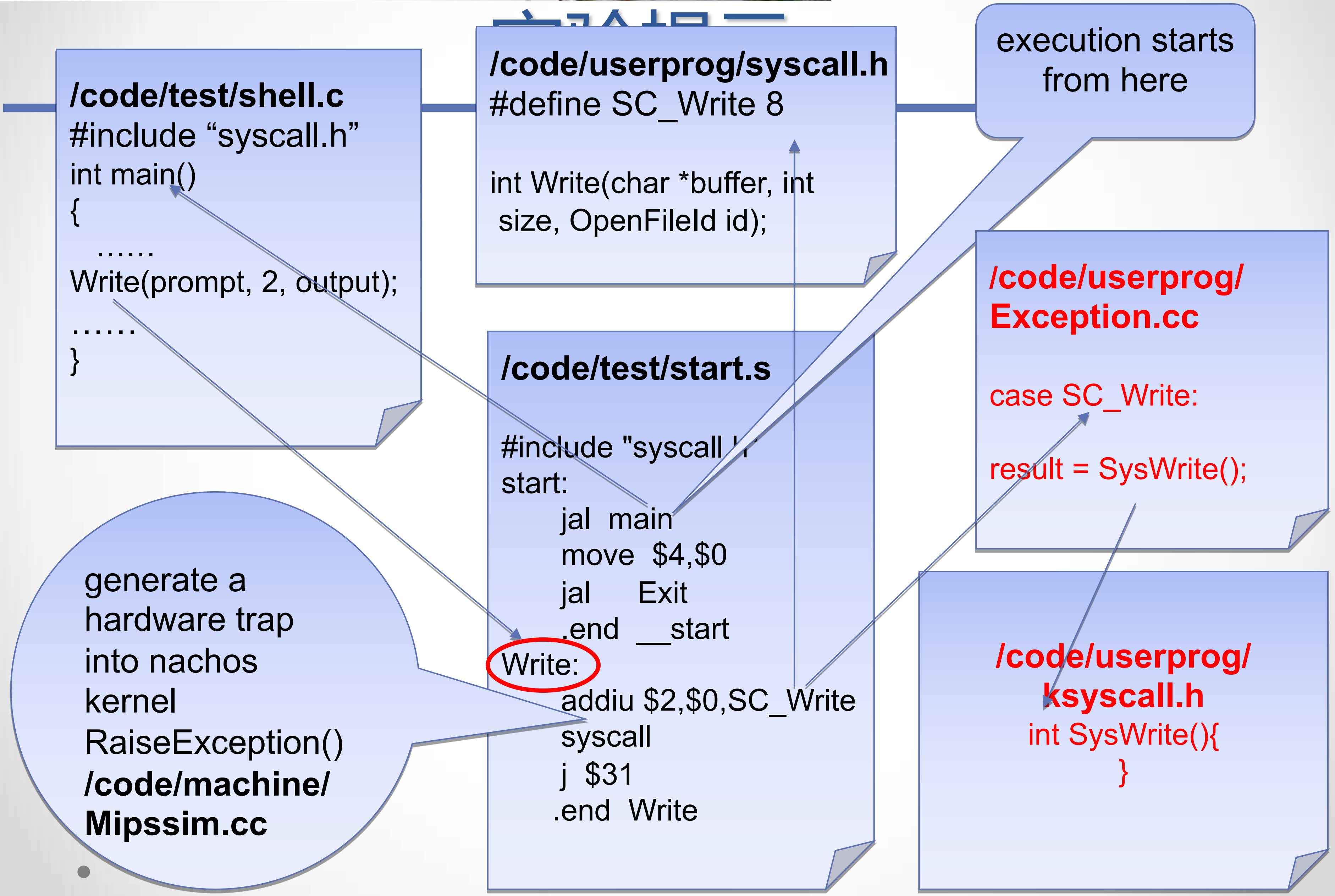
- **过程**：
  - 阅读和修改Nachos源代码(**exception.cc, ksyscall.h**)
  - 编译：
    - 1)code/test目录下，先使用make clean Makefile,再make产生shell.noff
    - 2)code/build.linux 目录下先使用make clean Makefile, make产生nachos（或make clean,再make)
  - 测试运行
    - 在code/build.linux 目录下./nachos –x ../test/shell.noff
    - 或者在code/test目录下../build.linux/nachos –x shell.noff

实验提示$_{3/4}$

- 全局图



Start.s
User Code
...
Exec("MyProgram")
...
Link
User Code (After Make)
...
• Put pointer to "MyProgram" in register r4
• Put Exec syscall code in register r2
• Execute special SYSCALL instruction
...
Fetch, Decode, Execute
SYSCALL instruction
Machine Simulator Code
MIPS Machine Simulator
Calls
Returns
Machine::RaiseException
• Change from user mode to kernel mode
• Call ExceptionHandler function in exception.cc
• Change back from kernel mode to user mode
Fetch the next instruction
System Call Example
Kernel Code
ExceptionHandler
• Save user process state
• Read syscall code from register r2
• Call the Exec syscall handler
• Get address of MyProgram in r4
• Do the Exec syscall stuff
• Restore user process state
• Put return value in register r2
• Increment PC (the PC in the machine simulator; not the host machine's PC)
Calls
Returns



/code/test/shell.c

```
#include “syscall.h”
int main()
{
   ……
Write(prompt, 2, output);
……
}
```

/code/userprog/syscall.h

```
#define SC_Write 8

int Write(char *buffer, int
 size, OpenFileId id);
```

execution starts from here

/code/test/start.s

```
#include "syscall.h"
start:
        jal  main
        move  $4,$0
        jal      Exit
        .end  __start
Write:
        addiu $2,$0,SC_Write
        syscall
        j  $31
       .end  Write
```

generate a hardware trap into nachos kernel

RaiseException()

/code/machine/Mipssim.cc

/code/userprog/Exception.cc

```
case SC_Write:

result = SysWrite();
```

/code/userprog/ksyscall.h

```
int SysWrite(){
}
```

实验提示$_{4/4}$

- MIPS的编译器采用以下参数传递规则

参数1:	r4寄存器
参数2:	r5寄存器
参数3:	r6寄存器
参数4:	r7寄存器

THE END…

THANK YOU~