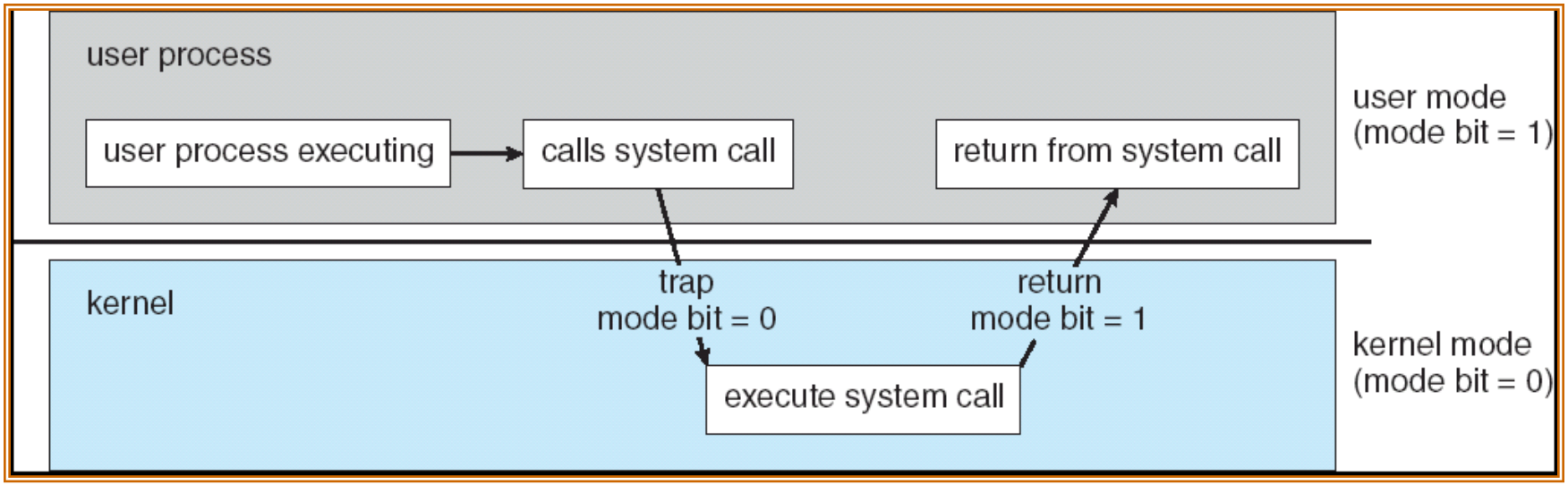
Nachos System Calls

Two mode of Excution

- User mode: execute instructions which only access the user space
- Kernel mode:
  - Execute when NachOS start up
  - Or when an instruction causes a trap: illegal instruction, page fault, **system call**

To protect the system from aberrant users and processors, some instructions are restricted to use only by the OS, but in kernel mode, the OS can do all these things

Recap System Call

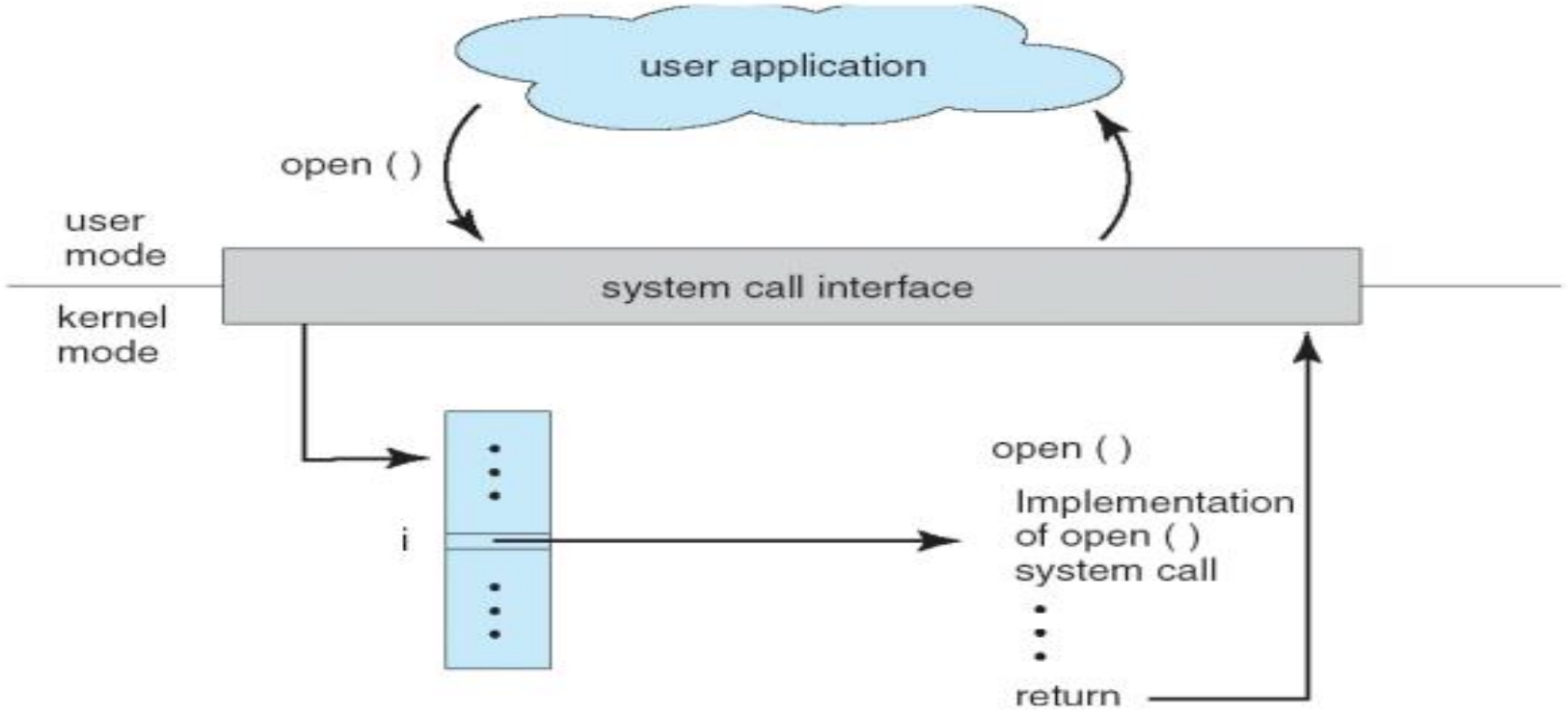


- What are some system calls?
  - I/O: open, close, read, write, lseek
  - Files: delete, mkdir, rmdir, truncate, chown, chgrp, ..
  - Process: exec, join(wait), exit…

System Calls

- Programming interface to the services provided by the OS. OS procedure that executes privileged instructions
- Typically written in a high-level language (C or C++)
- A number associated with each system call. System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values to user mode when finished

System Call – OS relationship



- Example when user application call **open()** function in user mode, which call **open()** system call in kernel mode

System Call Argument Passing

- System Call argument passing:
  - In registers (not very much can be passed)
  - Write into user memory, kernel copies into kernel memory
    - *User addresses must be translated!*
    - Kernel has different view of memory than user
- Nachos Example
  - Register R2: The code for the system call
  - Registers R4,R5,R6: store arg1 to arg3
  - The return value is stored in R2

User vs Kernel Space Address Translation

User2System function

- Purpose: Copy buffer from User memory space to System memory space
- Input:
  - User space address (int)
  - Limit of buffer (int)
- Output: buffer (char*)

User2System function

```
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);
    for (i = 0 ; i < limit ;i++)
    {
        machine->ReadMem(virtAddr+i,1,&oneChar);
        kernelBuf[i] = (char)oneChar;
        if (oneChar == 0)
        break;
    }
    return kernelBuf;
}
```

System2User function

- Purpose: Copy buffer from System memory space to User memory space
- Input:
  - User space address (int)
  - Limit of buffer (int)
  - Buffer (char[])
- Output: number of bytes copied (int)

System2User function

```
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;
}
```

System Call Implement

Key files

- progtest.cc: test routines to run user code.
- addrspace.h: addrspace.cc -- create an address space and load the program from disk.
- syscall.h: the system call interface.
- exception.cc: the handler for system calls and other user-level exceptions such as page faults
- start.c, start.s: which includes all system call stubs

Step 1

- Modify code/userprog/syscall.h:
  - Constants are defined to give somewhat mnemonic names (*SC_Halt*, *SC_Exit*, etc.) to the codes for the system calls recognized by Nachos
    #define SC_Sub 43
  - The remainder of *syscall.h* declares C++ functions that may be called by user programs to produce system call exceptions

    int Sub(int a, int b);

Step 2

- Modify assembly code to make system call 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

Step 2 (cont.)

- Modify 2 following files: following the style of Halt

  code/test/start.c

  code/test/start.s

```
.globl Sub
.ent Sub
Sub:
    addiu $2, $0, SC_Sub
    syscall
    j $31
    .end Sub
```

Step 3

- Modify code/userprog/exception.cc to add all system call entries into ExceptionHandler()
- The only other work that the initial Nachos definition of *ExceptionHandler* accomplishes for you is to fetch the identifying number for the system call from register 2
- You will need them to bring other parameters from user code into the OS kernel code.
- After each system call, increment PC registers so that ExceptionHandler() execution flow returns back to next instruction after user's system call place
- Run `gmake all` command to recompile NachOS

Step 3 (cont.)

```
switch(which) {
    case SystemcallException:
        switch(type) {
            case SC_Sub:
          //Add your code here for processing
            int op1 = machine->ReadRegister(4);
            int  op2 = machine->ReadRegister(5);
            result = op1 - op2;
            machine->WriteRegister(2,result);
            break;
            // Another code...
        }
        // increment value in PC register
        machine->Registers[PrevPCReg] =
                        machine->Registers[PCReg];
        machine->Registers[PCReg] =
                        machine->Registers[NextPCReg];
        machine->Registers[NextPCReg] += 4;
}
```

User program in NachOS

User program Implement

- Write a user program to test your system call
- Following these step:
  - Make a C program in code/test/sub.c
  - Modify code/test/Makefile to compile sub.c to executable file sub in NachOS
  - Execute your user program

/code/test/Makefile

- Add the name of executable file in the end of this line

```
all: … sub
```

- Include these code to compile sub.c to executable file in noff format

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

Compile and Execute User program

- Run `gmake all` command to recompile NachOS
- Typing the following command to execute your user program in NachOS

```
./userprog/nachos -x  [fullpath to the executable file]
```

Example:

```
./userprog/nachos -x ./test/sub
```

❖ *Note: the current directory is code*