COP4610: Operating Systems

Project 1

Zhi Wang

Florida State University

Spring 2015

Run xv6

- ssh linprog.cs.fsu.edu
- wget `http://www.cs.fsu.edu/~zwang/files/cop4610/Spring2015/xv6_v8.tar.gz`
- tar -xf xv6_v8.tar.gz
- cd xv6
- to compile only: make
  ➠ to compile and run: make qemu-nox
- to quit qemu: ctrl-a x

User Space

- We use close as the example, read is similar.
  - ➠ assuming a program calls close (5), 5 is an invalid file descriptor

User Space

- We use close as the example, read is similar.

  ➠ assuming a program calls close (5), 5 is an invalid file descriptor
- close is declared in user.h: int close(int);
- close is defined in usys.S: SYSCALL(close), which expands to

  .globl close; →declare close as a global symbol

  close: →definition of close

  movl $SYS_close, %eax; →put system call number in register eax

  int $T_SYSCALL; →trigger a software interrupt, enter the kernel

  ret →return to the caller of close

User Space

- We use close as the example, read is similar.
  ➠ assuming a program calls close (5), 5 is an invalid file descriptor
- close is declared in user.h: int close(int);
- close is defined in usys.S: SYSCALL(close), which expands to
  .globl close; →declare close as a global symbol
  close: →definition of close
    movl $SYS_close, %eax; →put system call number in register eax
    int $T_SYSCALL; →trigger a software interrupt, enter the kernel
    ret →return to the caller of close
- T_SYSCALL defined in traps.h, SYS_close defined in syscall.h

Entering the Kernel

- int $T_SYSCALL triggers a software interrupt (T_SYSCALL=64)

  ➠ CPU saves the current state, and calls the interrupt handler

  ➠ Interrupt handler for T_SYSCALL is vector64 (vectors.S)

  ➠ vector64 jumps to alltraps function, which creates the trapframe, and calls trap (struct trapframe *tf)

Entering the Kernel

- int $T_SYSCALL triggers a software interrupt (T_SYSCALL=64)
  - ➠ CPU saves the current state, and calls the interrupt handler
  - ➠ Interrupt handler for T_SYSCALL is vector64 (vectors.S)
  - ➠ vector64 jumps to alltraps function, which creates the trapframe, and calls trap (struct trapframe *tf)
- struct trapframe saves the user-space registers. tf→eax contains the system call number (SYS_close)

```
struct trapframe {
  uint edi;
  uint esi;
  uint ebp;
  uint oesp;
  uint ebx;
  uint edx;
  ...
```

Syscall Dispatch

- trap (tf) calls syscall (void) because (tf→trapno == T_SYSCALL)
  ➠ trapframe tf saved to the current process control block
- syscall reads the syscall number in eax, and calls sys_close
  ➠ syscalls[SYS_close]
  ➠ return value is saved in tf→eax, the kernel restores tf before returning to user space

sys_close and Return

- sys_close reads the parameter from user stack with argfd
  - ➠ $fd = 5$, an invalid file descriptor
  - ➠ sys_close returns -1
- it returns to syscall(void), which saves return value to eax and returns to trap
- trap returns to alltraps, which restores user registers and returns to user space with iret

Define a New Syscall

- User space:
  - ➠ declare getprocs in user.h, define getprocs in usys.S
  - ➠ create a program called ps and add it to the makefile (refer cat.c)
- Kernel space:
  - ➠ add a new system call number: SYS_ps in syscall.h
  - ➠ create the syscall handler (sys_ps) in sysproc.c for SYS_ps
  - ➠ add sys_ps to the syscall dispatcher syscalls

Hint:

follow close/sys_close as an example to complete this part.

sys_ps

- proc.c has a global variable ptable with all the process control blocks
- sys_ps copies each PCB to user-provided memory
  - ➠ note: not all PCB states are copied
  - ➠ check the size of the user-provided memory