Syscalls

Indranil Sen Gupta (odd section)
and Mainack Mondal (even section)
CS39002

Spring 2019-20

The story so far

- A brief historical overview of OS
  - Batch processing systems
  - Multiprogramming
  - Multitasking
  - Some practice problems
- Today's OS (multitasking, like Unix)
  - Dual mode of operation
  - Uses of timer

Today's class

- System calls
- Some practice problems
- Start of processes

System calls

What are system calls?

- The mechanism used by an application program to request service from the operating system
  - So how does it work?

Earlier days: interrupt

- Originally, system calls issued using “int” instruction
  - Kernel creates an array of Interrupt descriptors in memory, called Interrupt Descriptor Table, or IDT

Earlier days: interrupt

- Originally, system calls issued using “int” instruction
  - Kernel creates an array of Interrupt descriptors in memory, called Interrupt Descriptor Table, or IDT
  - The handler routine was just an interrupt handler

Earlier days: interrupt

- Originally, system calls issued using “int” instruction
  - Kernel creates an array of Interrupt descriptors in memory, called Interrupt Descriptor Table, or IDT
  - The handler routine was just an interrupt handler
  - Like interrupts, system calls are arranged in a table

Earlier days: interrupt

- Originally, system calls issued using “int” instruction
  - Kernel creates an array of Interrupt descriptors in memory, called Interrupt Descriptor Table, or IDT
  - The handler routine was just an interrupt handler
  - Like interrupts, system calls are arranged in a table

- Whenever a syscall (interrupt driven) came

Earlier days: interrupt

- Originally, system calls issued using “int” instruction
  - Kernel creates an array of Interrupt descriptors in memory, called Interrupt Descriptor Table, or IDT
  - The handler routine was just an interrupt handler
  - Like interrupts, system calls are arranged in a table
- Whenever a syscall (interrupt driven) came
  - Kernel selected syscall placing index in eax register

Earlier days: interrupt

- Originally, system calls issued using “int” instruction
  - Kernel creates an array of Interrupt descriptors in memory, called Interrupt Descriptor Table, or IDT
  - The handler routine was just an interrupt handler
  - Like interrupts, system calls are arranged in a table
- Whenever a syscall (interrupt driven) came
  - Kernel selected syscall placing index in eax register
  - Arguments go in the other registers

Earlier days: interrupt

- Originally, system calls issued using “int” instruction
  - Kernel creates an array of Interrupt descriptors in memory, called Interrupt Descriptor Table, or IDT
  - The handler routine was just an interrupt handler
  - Like interrupts, system calls are arranged in a table

- Whenever a syscall (interrupt driven) came
  - Kernel selected syscall placing index in eax register
  - Arguments go in the other registers
  - Return value goes in eax

However it is slow

- Today, Processors are totally pipelined
  - Pipeline stalls are very expensive
  - Cache misses can cause pipeline stalls
- Now recall that IDT is in memory
  - May not be in cache
  - Makes it expensive

Idea: new instruction

- What if we cache the IDT entry for a system call in a special CPU register?
  - No more cache misses for the IDT!

- What is the cost?

Idea: new instruction

- What if we cache the IDT entry for a system call in a special CPU register?
  - No more cache misses for the IDT!

- What is the cost?
  - system calls should be frequent enough to be worth the transistor budget

How to leverage the new instruction?

- There is a machine instruction (new architectures)
  - Essentially for asking your processor to perform task
  - Hardware specific
  - Can be called “syscall”, “trap”, “svc”, “swi”
  - For x86-64 architecture its called “syscall”

Example: How are system calls used in kernel

- syscall machine instruction takes operands
  - syscall 10 // integer number for x86
    // some of these are fixed by intel

Example: How are system calls used in kernel

- syscall machine instruction takes operands
  - syscall 10 // integer number for x86
    // some of these are fixed by intel

- When we are writing programs in HLL (higher level language, think C)
  - We think of syscalls in a somewhat higher level context

Example: How are system calls used in kernel

- syscall machine instruction takes operands
  - syscall 10 // integer number for x86
    // some of these are fixed by intel

- When we are writing programs in HLL (higher level language, think C)
  - We think of syscalls in a somewhat higher level context

Characteristics of system calls

- Provide an interface to the services made available by an operating system

Characteristics of system calls

- Provide an interface to the services made available by an operating system
  - Typically executes hundreds of thousands time every second

Characteristics of system calls

- Provide an interface to the services made available by an operating system
  - Typically executes hundreds of thousands time every second
  - User application programs do not see this level of detail

Characteristics of system calls

- Provide an interface to the services made available by an operating system
  - Typically executes hundreds of thousands time every second
  - User application programs do not see this level of detail
  - Use Application programming interface (API)

Characteristics of system calls

- Provide an interface to the services made available by an operating system
  - Typically executes hundreds of thousands time every second
  - User application programs do not see this level of detail
  - Use Application programming interface (API)
  - fork, pipe, execvp etc.

Characteristics of system calls

- Provide an interface to the services made available by an operating system
  - Typically executes hundreds of thousands time every second
  - User application programs do not see this level of detail
  - Use Application programming interface (API)
  - fork, pipe, execvp etc.
  - These APIs are also loosely termed as system calls

More about system calls

- System call numbers in linux for x86 - 64 : https://github.com/torvalds/linux/blob/16f73eb02d7e1765ccab3d2018e0bd98eb93d973/arch/x86/entry/syscalls/syscall_64.tbl
- System call numbers in linux for x86 : https://github.com/torvalds/linux/blob/16f73eb02d7e1765ccab3d2018e0bd98eb93d973/arch/x86/entry/syscalls/syscall_32.tbl

- System call implementations in x86-64 and x86 https://stackoverflow.com/questions/15168822/intel-x86-vs-x64-system-call

Example of a system call API

- There are no fopen, fgets, printf, and fclose system calls in the Linux kernel but open, read, write, and close

- http://man7.org/linux/man-pages/man2/read.2.html

NAME top

read - read from a file descriptor

SYNOPSIS top

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
```

Example of a system call API

- http://man7.org/linux/man-pages/man2/read.2.html

NAME top

read - read from a file descriptor

SYNOPSIS top

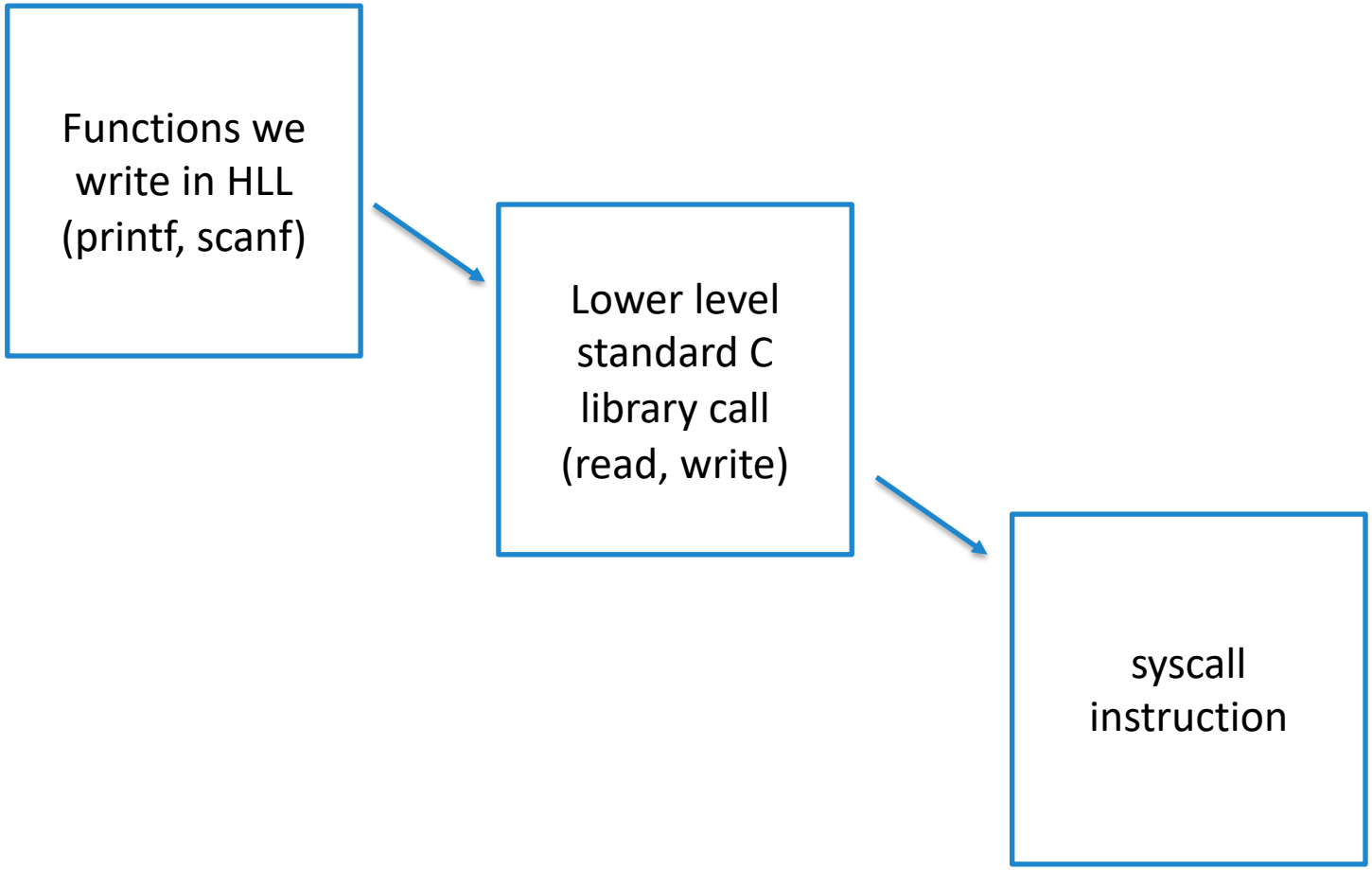
```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
```

- What are these function parameters?
- What is the return values?

How is read invoked?

- Check the board
- Demo: checking what syscalls are invoked in a process
- Demo: checking assembly code in C (using gcc)

Summary: The workflow



Types of system calls

(From silberschatz's slides)

Types of System Calls

- Process control (e.g., fork(), exit(), wait())
  - create process, terminate process (fork, exit)
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - Dump memory if error
  - Debugger for determining bugs, single step execution
  - Locks for managing access to share data between processes

Types of System Calls (Cont.)

- File management (e.g., open(), close(), read(), write())
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes
- Device management (e.g., ioctl(), read(), write())
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

Types of System Calls (Cont.)

- Inter-Process Communications (e.g., pipe(), semget(), semop(), shmget(), shmcat(), shmdt(), shmctl(), signal(), kill())
  - create, delete communication connection
  - send, receive messages if message passing model to host name or process name
  - Shared-memory model create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices

Types of System Calls (Cont.)

- Protection (chmod(), chown(), umask())
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

Today's class

- System calls
- Some practice problems
- Start of processes

Announcement

- The first practice problems sheet is up: http://www.facweb.iitkgp.ac.in/~isg/OS/ASSIGN/Assignment-1.pdf