Software Construction Laboratory

Week 5 Part 2

Lab 3

Mushi Zhou

Winter 2017

UCLA

Prof. Paul Eggert

System Call Programming and debugging

Outline

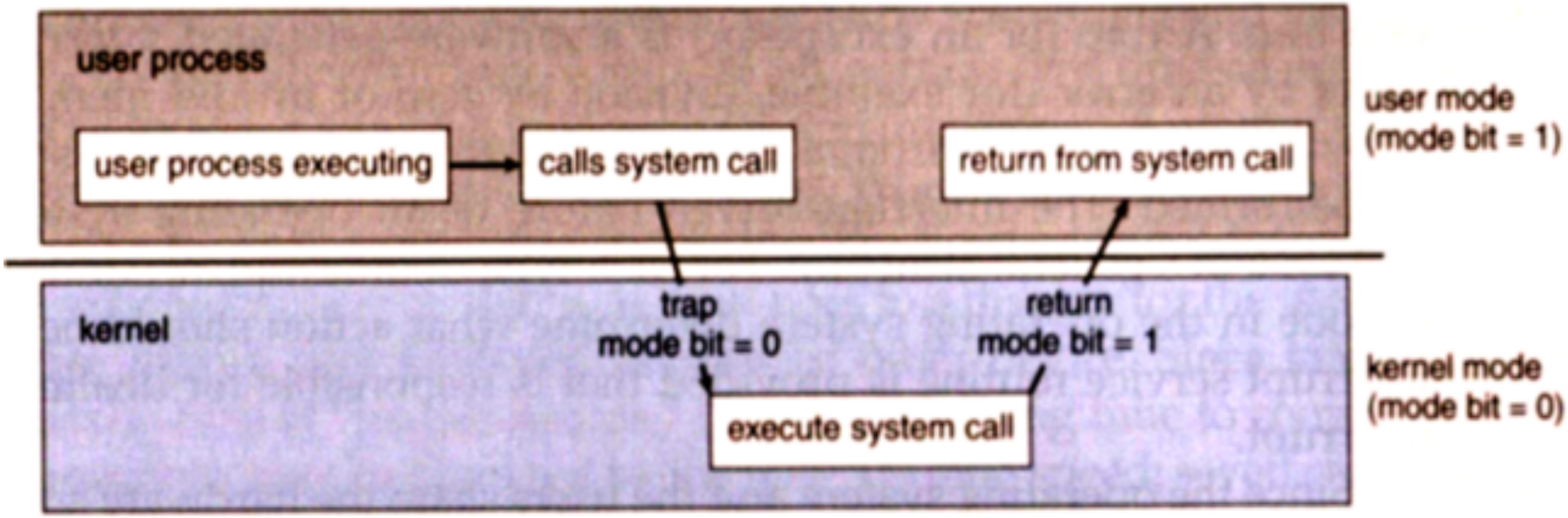
- Buffers, Buffered I/O
- Why do we want to use buffer
- Buffer overruns, and techniques for avoiding them
- System Calls vs Library calls
- How to use system calls in C
- C and system programming

What Are System Calls

- A system call, sometimes referred to as a kernel call, is a request in a Unix-like operating system made via a software interrupt by an active process for a service performed by the kernel

- Kernel?
- Software Interrupt??
- Active process???

Illustration



What Are Library Calls?

- Same as Procedure call & Function Call
- They are basically API for one or more system calls
- They achieve the same goal by calling system calls
- They are provided by libraries, languages
- API: Application Program Interface

A layer between user and actual system, usually warps things around nicely and reduces complications for users

Why Do We Mostly Use Procedure Calls?

- The actually system calls may very system to system
- Some system calls may not be easy to directly interact with
- Procedure calls allow same user interface from many different systems as long as the system support the API of the procedure call

  i.e. c library functions are available as long as c can be installed on a system
- Procedure calls provide nice, clean group of system calls for user with additional features, including but not limited to safety features
- We are using system calls in this assignment just to give you an idea and help you to understand the systems you are using

Some Examples of System Calls

System Calls for I/O:

There are 5 basic system calls that Unix provides for file I/O

1. int open(char *path, int flags [, int mode]);
2. int close(int fd);
3. int read(int fd, char *buf, int size);
4. int write(int fd, char *buf, int size);
5. off_t lseek(int fd, off_t offset, int whence);

Some Details About System Calls

- They look very similar to procedure calls
- But they are different, there is not a library function that defines what this call would do, they are direct instructions resides in the kernel
- For example, fopen() is a library call, that it eventually will call open() to achieve its goal
- When you invoke a system call, your program/process entered kernel mode
- Programs perform privileged tasks in kernel mode through system calls
- When the kernel has satisfied the request made by a process, it restores that process to user mode

Two Distinct Execution Modes of Operation

- User mode: *non-privileged* mode in which each process starts out
- In this mode, processes are not allowed to access those portions of memory that have been allocated to the kernel or to other programs

- Kernel mode has *root* (i.e., administrative) privileges, including *root access permissions*
- This allows the process to perform restricted actions such as accessing hardware devices or the memory management unit.

Some Details About System Calls

System calls can be classified into six groups:

- Process management, Inter-process communication, Memory management, File system, Initialization and Other
- The kernel maintains a list of all registered system calls in the system call table, with a unique number for each system call that are not recycled
- Processes do not refer to system calls by name, but rather by their system call number
- Processes perform system call by passing parameters to CPU registers (You do not need to know details about this in this class, probably will learn this in an OS class)

Some Help for Assignment 5

For Lab 5,

- Compare the performance of getchar/putchar vs read/write
- Read/write are system calls, i.e. they are unbuffered if you make these calls directly
- They take the specified number of bytes each time directly from the stream
- Getchar/putchar are buffered with OS buffers even you don’t really see the buffers
- They take all everything from stream at once into a buffer and them read one byte at a time from the temporary buffer

Some Help for Assignment 5

For HW 5:

- We are focusing on I/O system calls
- You are revising your Sorting program by utilizing system calls
- The 5 system calls for I/O all have detailed man page for them
- do '**man -s 2 open**', '**man -s 2 close**', etc.
- They are among the easiest system calls, very similar to library calls
- Remember to
- #include <unistd.h> // Allow C to recognize your system calls
- #include <sys/syscall.h> // Get entry point for system calls
- #include <errno.h> // The errno variable contains error number when
  // your system call fails

Strace

- Strace, as we have mentioned before, is a debugging tool, that can monitor interactions between processes and Linux kernel, i.e. system calls
- Just do strace *programname* (executable, not your source file)
- By default, strace output all system calls made
- Options:
- -e to show only selected system calls, exp: strace –e open ls
- - t print time stamp of system calls
- - r print relative time of execution
- - c show a statistic report summary

Some Help for Assignment 5

- Modify your HW5 to use system calls instead
- Adding the non-case sensitivity to comparisons
- Note: the topper() takes in unsigned characters, but the default char is signed, so you'll have to shift the value before changing to upper case
- Compare execution time of your two HWs in regards to number of comparisons
- What one do you expect to run faster? (as input sizes goes really large)
- Write a shell script to achieve the same goal, do you expect this to perform faster? (Depends on the way your write it)

If You Want to Learn More…

- System Call Definition:
- http://www.linfo.org/system_call.html

- I/O System Call Examples:
- http://web.eecs.utk.edu/~huangj/cs360/360/notes/Syscall-Intro/lecture.html