CS 35L

Software Construction Laboratory

Lecture 5.2

7th February, 2019

Logistics

- Hardware requirement for Week 8
  - Seeed Studio BeagleBone Green Wireless Development Board
- Presentations for Assignment 10
  - **Fill your details min the link below by 8th Feb, 2019**
  - Do not fill a slot without a Presentation Topic
  - https://docs.google.com/spreadsheets/d/1o6r6CKCaB2du3klPflHiquymhBvbn7oP0wkHHMz_q1E/edit?usp=sharing
- Assignment 4 is due on 9th Feb, 2018

Review - Previous Lab

- System Calls
  - Processor modes
  - System Calls
  - System Call Overhead
  - Types of System Calls

System Call Programming

Examples of System Calls

- Open()
- Create()
- Close()
- Read()
- Write()

File Descriptors

- File descriptor is an integer that uniquely identifies an open file of the process
- File descriptor table is the collection of integer array indices that are file descriptors in which elements are pointers to file table entries. One unique file descriptors table is provided in operating system for each process
- Read from stdin => read from fd 0: Whenever we write any character from keyboard, it is read from stdin through fd 0
- Write to stdout => write to fd 1: Whenever we see any output to the video screen it is written to stdout in screen through fd 1.
- Write to stderr => write to fd 2: Whenever we see any error to the video screen, it is written to stderr in screen through fd 2.

Open()

- Used to Open the file for reading, writing or both
- Syntax: int open (const char* Path, int flags);
  - Path: Path to file which is to be opened
    - Use Relative path if you are working in the same working directory as file
    - Otherwise, Absolute path, starting with '/'
  - Flags
    - O_RDONLY: read only,
    - O_WRONLY: write only,
    - O_RDWR: read and write,
    - O_CREAT: create file if it doesn't exist,
    - O_EXCL: prevent creation if it already exists
- Returns a file descriptor

Create()

- Used to create a new empty file
- Syntax: int creat(char *filename, mode_t mode)
  - Filename: name of the file which you want to create
  - Mode: Indicates permission of the new file
- returns first unused file descriptor (generally 3 because 0, 1, 2 fd are reserved) ; returns -1 when error

Difference between open() and create()

- creat function *creates* files, but can not open existing file.
- Create is more of a legacy function now
- creat() is equivalent to open() with flags equal to O_CREAT|O_WRONLY|O_TRUNC

Close()

- Closes the file which pointed by fd
- Syntax: int close(int fd);
  - Fd: File Descriptor
- Returns 0 on success and -1 on error

Read()

- From the file indicated by the file descriptor fd, the read() function reads n bytes of input into the memory area indicated by buf.
- Syntax: size_t read (int fd, void* buf, size_t n);
  - fd: file descripter
  - buf: buffer to read data from
  - n: length of buffer
- Returns:
  - Number of bytes read on success
  - 0 on reaching end of file
  - -1 on error

Write()

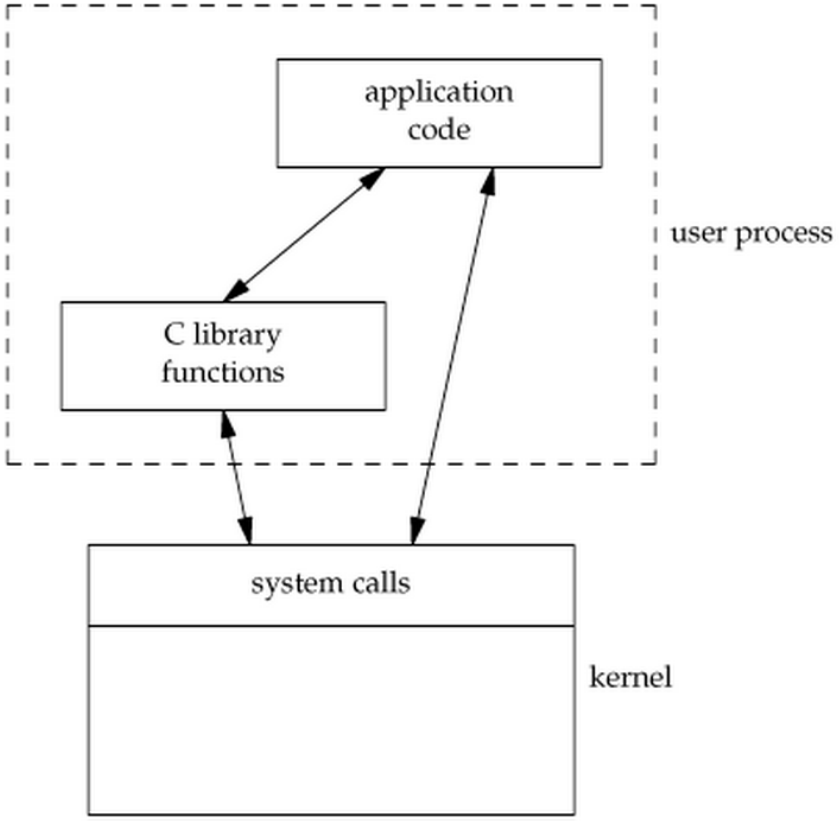
- Writes n bytes from buf to the file associated with fd
- Syntax: size_t write (int fd, void* buf, size_t cnt);
  - fd: file descripter
  - buf: buffer to write data to
  - n: length of buffer
- Returns
  - Number of bytes written on success
  - 0 on reaching end of file
  - -1 on error

Library Functions

- Functions that are a part of standard C library
- To avoid system call overhead use equivalent library functions
  - getchar, putchar vs. read, write (for standard I/O)
  - fopen, fclose vs. open, close (for file I/O), etc.
- How do these functions perform privileged operations?
  - They make system calls

So What's the Point?

- Many library functions invoke system calls indirectly
- So why use library calls?
- Usually equivalent library functions make fewer system calls
- non-frequent switches from user mode to kernel mode => less overhead



Unbuffered vs. Buffered I/O

- Unbuffered
  - Every byte is read/written by the kernel through a system call
- Buffered
  - collect as many bytes as possible (in a buffer) and read more than a single byte (into buffer) at a time and use one system call for a block of bytes Buffered I/O decreases the number of read/write system calls and the corresponding overhead

Test Email!

Questions?