CSCI 3753: Operating Systems Fall 2024

Dylan Sain

Department of Computer Science

University of Colorado Boulder

Programming Assignment 1

Q & A

Week 3: Loadable Kernel Modules & Input/Output in C

How to Extend/Interact with Kernel?

- Method 1: System calls

Recompile the kernel !!! 

- Method 2: Loadable Kernel Module (LKM)
  - An object file that contains a chunk of code to add to the base kernel of an operating system while it is RUNNING.
    - Typically, it is used to add support for *device drivers, filesystem drivers, etc.*

LKM Pros & Cons

- Pros:
  - DON’T have to REBUILD the kernel
  - Be MUCH faster to maintain and debug
  - Save memory
    - Loaded only when we are actually using them
    - Unloaded in order to free memory and other resources when it is no longer required
- Cons:
  - FRAGMENTATION penalty
  - Security

LKM Command on Terminal

- ***lsmod***: List currently loaded LKMs.
- ***insmod***: Insert an LKM into the kernel.
- ***rmmod***: Remove an LKM from the kernel.
- ***modprobe***: Insert/remove an LKM or set of LKMs intelligently.
  - e.g., if you must load A before loading B, modprobe will automatically load A when you tell it to load B.
- ***kerneld***: Kernel daemon program
  - allows kernel modules to be loaded automatically rather than manually with insmod/modprobe
- ***depmod***: Determine interdependencies between LKMs

Exercise 1: lsmod

Test out lsmod! Do you see your function from PA1?

LKM command – ***insmod***

- ***insmod*** makes an init_module system call to load the LKM into kernel memory.

- **init_module** system call invokes the LKM's initialization routine (module_init) right after it loads the LKM.

LKM command – ***rmmod***

- ***rmmod*** makes a delete_module system call to unload the LKM into kernel memory.

- **delete_module** system call invokes the LKM's cleanup routine (module_exit) right before it unloads the LKM.

Input-Output in C

Different Ways to Access I/O for File in C

- I/O streams library (C standard library functions)
  - Associate with FILE pointer

BUFFER

A user-space array maintained by the C I/O streams library.

- I/O system calls
  - Associate with a file descriptor

I/O stream Library vs I/O System Calls

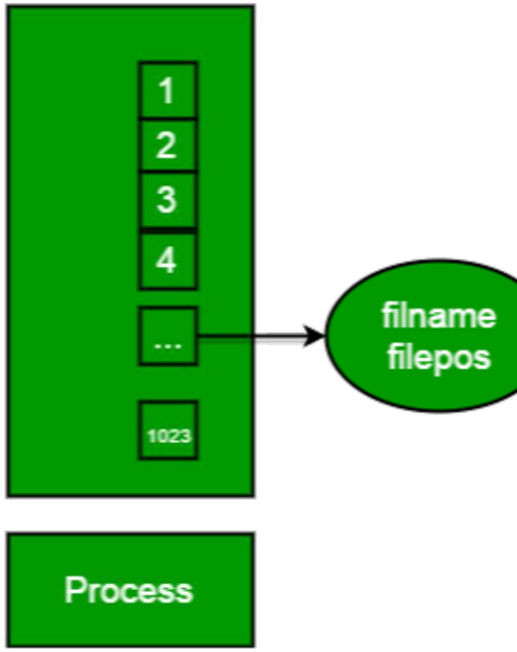
I/O Stream Library

- More high level
- Creates a pointer to the file stream
- Has a file structure and buffered I/O
- Built on top of I/O Systems Calls

I/O System Calls

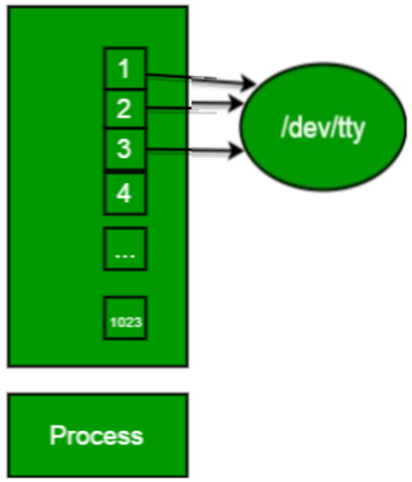
- Low level interactions
- Provides a file descriptor
- Useful for working with other lower level system calls
- FDs have many different functions that go beyond the stream library

I/O for File Terminology



- File descriptor (fd)
  - Integer that uniquely identifies an open file of the process
- File descriptor table
  - 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.
- File table entries
  - A structure in-memory surrogate for an open file, which is created when process request to opens file and these entries maintains file position.

Standard File Descriptors



- When any process starts, that process file descriptors table's fd 0, 1, 2 open automatically.
- By default, each of these 3 fd references file table entry for a file named **/dev/tty**
  - Read from stdin => read from fd 0: Whenever we write any character from keyboard, it read from stdin through fd 0 and save to file named /dev/tty.
  - Write to stdout => write to fd 1: Whenever we see any output to the screen, it's from the file named /dev/tty and written to stdout in screen through fd 1.
  - Write to stderr => write to fd 2: We see any error to the screen, it is also from that file write to stderr in screen through fd 2.

Different Ways to Access I/O for File in C

	I/O Stream Library	I/O System Calls
Create a new file	`fopen()`	`creat()`
Open a file	`fopen()`	`open()`
Close a file	`fclose()`	`close()`
Read from a file	`fscanf(), getc(), getw(), fread()`	`read()`
Write to a file	`fprintf(), putc(), putw(), fwrite()`	`write()`
Sets the position of a file pointer to a specified location	`fseek()`	`lseek()`

fopen() vs open()

```
FILE* fopen(const char *filename, const char *mode)
```

- #include <stdio.h>
- Parameters:
  - filename : path to file
  - mode : ”w”, “r”, “a”, “w+”, “r+”, “a+”

```
int open(const char* Path, int flags[, mode_t mode])
```

- #include <fcntl.h>
- Parameters:
  - path : path to file
  - flags : O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, etc.
  - Mode : indicates permissions of new file

Reading & Writing in I/O Stream

I/O Stream Function	Description

`fprintf()`
`fwrite()`
`fputs()`
	Write a **block** of data to a file

`fscanf()`
`fread()`
`fgets()`
	Read a **block** of data from a file

`fgetc()`	Read a **single character** from a file
`fputc()`	Write a **single character** to a file
`ftell()`	Returns the current position of a file pointer
`rewind()`	Sets the file pointer at the beginning of a file

Reading & Writing in I/O Standard Stream

I/O Stream Function	Description
`getc()`	Reads a **single character** from stdin
`putc()`	Writes a **single character** to stdout
`getw()`	Reads an **integer** from stdin
`putw()`	Writing an **integer** to stdout

`scanf()`
`getline()`
~~`gets()`~~
	Reads a **block** of data from stdin

`printf()`	Writing a **block** of data to stdout

Appendix

I/O System calls

- creat(): create a new empty file
- open(): open the file for reading, writing or both
- close(): close the file which pointed by fd
- read(): read n bytes of input from the file indicated by the file descriptor fd into the memory area indicated by buf
- write(): write n bytes from buf to the file or socket associated with fd
- lseek(): change the location of the read/write pointer of the fd

I/O System calls

- creat(): create a new empty file
  - Syntax

```
int creat(char *filename, mode_t mode)
```

  - Parameters:
    - filename : name of the file which you want to create
    - mode : indicates permissions of new file
  - How it works in OS
    - Create new empty file on disk
    - Create file table entry
    - Set first unused file descriptor to point to file table entry
    - Return file descriptor used or -1 upon failure

I/O System calls

- open(): open the file for reading, writing or both
  - Syntax

```
int open(const char* Path, int flags[,
                mode_t mode])
```

  - Parameters:
    - path : path to file
    - flags : O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, etc.
    - mode: indicates permissions of new file
  - How it works in OS
    - Find existing file on disk
    - Create file table entry
    - Set first unused file descriptor to point to file table entry
    - Return file descriptor used or -1 upon failure

I/O System calls

- close(): close the file which pointed by fd
  - Syntax

```
int close(int fd)
```

  - Parameters:
    - fd: file descriptor
  - How it works in OS
    - Destroy file table entry referenced by element fd of file descriptor table as long as no other process is pointing to it !!!
    - Set element fd of file descriptor table to NULL
    - Return 0 on success or -1 on error

I/O System calls

- read(): read bytes of input from fd into memory area
  - Syntax

```
size_t read(int fd, void* buf, size_t n)
```

  - Parameters:
    - fd: file descripter
    - buf: buffer to read data from
    - cnt: length of buffer
  - Returns:
    - 0 on reaching end of file
    - -1 on error
    - -1 on signal interrupt
    - Number of bytes read on success

I/O System calls

- write(): write bytes of input from memory area to fd
  - Syntax

```
size_t write(int fd, void* buf, size_t n)
```

  - Parameters:
    - fd: file descripter
    - buf: buffer to read data from
    - cnt: length of buffer
  - Returns:
    - 0 on reaching end of file
    - -1 on error
    - -1 on signal interrupt
    - Number of bytes written on success

I/O System calls

- lseek(): change the location of the read/write pointer of the fd
  - Syntax

```
off_t lseek(int fildes, off_t offset, int
                        whence)
```

  - Parameters:
    - fildes: fd of the pointer that is going to be moved
    - offset: offset of the pointer (measured in bytes)
    - whence : The method in which the offset is to be interpreted
  - Returns:
    - the offset of the pointer (in bytes) from the beginning of the file.
    - -1 on an error moving the pointer