COMP1521 26T1

Week 7 Lecture 1

File Systems

Announcements

Test 5 (MIPS Strings) **and Test 6** (C bitwise)are due **Thursday 9pm**

Assignment 1

- automarking available (may not be available yet if you had an extension)
- tutor marking ASAP

Assignment 2 coming out on Wednesday!

- Lots of syscalls…

Announcements

Friday is a public holiday:

If you are usually in a Friday tut_lab

- Please attend another one this week.
- Either a make-up tut_lab or any regular tut_lab

Monday next week is a public holiday:

- No live lecture on Monday Week 8. Pre-recorded lecture instead
- **Lab 7**: is Due Tuesday Week 8 midday

Today's Lecture

- **Recap**
  - System Calls, File Descriptors
- Basic File Operations
  - **Libc** Wrappers
    - open, close, read, write
  - **stdio**
    - fopen, fclose, fgetc, fgets, fputs, fread, fwrite
- Comparing Libc and stdio

Recap

Recap Questions

- What is the difference between privileged and non-privileged mode?
- How can user code access hardware?

Operating Systems: Privileged Mode

- Needs hardware to provide a **privileged** mode
    - Code can access all hardware, memory and CPU instructions
    - **OS kernel** runs in this mode
        - The OS kernel is the core of the operating system that manages the hardware and system resources.
- Needs hardware to provide a **non-privileged** mode which
    - code can not access hardware directly
    - code can only access the memory it was allocated
    - **user code** runs in this mode

Operating Systems: System Calls

- System calls allow user level code to request hardware operations
- System calls transfer execution to OS kernel code in **privileged** mode
  - includes arguments specifying details of request being made
  - OS checks operation is valid & permitted
  - OS carries out operation
  - transfers execution back to user code in **non-privileged** mode

3 ways to do System Calls in Linux

syscall function:

- not portable
- no type checking at compile time

Libc named syscall wrapper functions (section 2 of man):

- More portable (likely to work on MacOS and other Linux systems)
- Compile time type checking
- Less cryptic

stdio.h higher level library functions (section 3 of man):

- Portable
- Calls named syscall wrapper function for you
- Does other helpful stuff behind the scenes

System Calls to Manipulate Files

Important file related system calls

Id	Name	Function
0	read	read some bytes from a **file descriptor**
1	write	write some bytes to a **file descriptor**
2	open	open a file system object, returning a **file descriptor**
3	close	close a **file descriptor**
4	stat	get file system metadata for a pathname
8	lseek	move **file descriptor** to a specified offset within a file

Recap Example:

hello.c

File Descriptors

Every process starts with the 3 standard streams, 0, 1, 2.

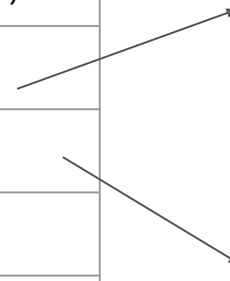
When a file is opened, the next available free file descriptor is used.

When a file is closed the file descriptor is released.

When a file is read to or written from, the file offset is updated

Per Process File Descriptor Table

System Wide File Table



File Basics with Libc Wrapper Functions

Code Demo

- open_read.c
  - Modify to read in filename from command line
- open_write.c
  - O_CREAT
  - O_TRUNC
  - O_APPEND

- open_issue.c

errno

- C library has an interesting way of returning error information
  - functions typically return **-1** to indicate error
  - and set **errno** to integer value indicating reason for error
  - you can think of **errno** as a global integer variable

- These integer values are **#define**-d in **errno.h**
  - see man errno for more information
  - **perror()** looks at **errno** and prints message with reason
  - **strerror()** converts **errno** to string describing reason for error

- To see all error codes type **errno -l** on command line

Libc syscall Wrappers: Common Types

ssize_t:

- Used for a count of bytes or an error indication.
- A signed integer type that can store range [-1, SSIZE_MAX]

size_t:

- used for a count of bytes
- An unsigned integer type that can store range [0, SIZE_MAX]

off_t:

- used files sizes and offsets. A signed integer type.

Libc wrapper to open a file

```
int open(char *pathname, int flags);
```

- open file at **`pathname`**, according to **`flags`**
- **`flags`** is a bit-mask defined in **`<fcntl.h>`**

```
int open(char *pathname, int flags, mode_t mode);
```

- Use this version when potentially creating a new file
- **`mode`** is an **octal** number to give the file sensible user access permissions

if successful they return **file descriptor** (small non-negative int)
if unsuccessful they return **`-1`** and set **`errno`** to value indicating reason

Libc wrapper to open a file

Flag	Use

`O_RDONLY`
`O_WRONLY`
`O_RDWR`
	open for reading
	open for writing
	open object for reading and writing

`O_APPEND`	append on each write
`O_CREAT`	create file if doesn't exist
`O_TRUNC`	truncate to size 0

flags can be combined e.g. `(O_WRONLY|O_CREAT|O_TRUNC)`

Libc library wrapper for read system call

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

- read (up to) **count** bytes from **fd** into **buf**
  - **buf** should point to array of at least **count** bytes
  - read cannot check **buf** points to enough space
- if successful, number of bytes actually read is returned
- if no more bytes to read (reached end of file), **0** returned
- if error, **-1** is returned and **errno** set
- file descriptor **current position** in file is updated

Libc library wrapper for write system call

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

- attempt to write **`count`** bytes from **`buf`** into stream identified by **`fd`**
- if successful, number of bytes actually written is returned
- if unsuccessful, **-1** returned and **errno is set**
- file descriptor **current position** in file is updated

Libc wrapper to close a file

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

- release open file descriptor **`fd`**
- if successful, return **`0`**
- if unsuccessful, return **`-1`** and set errno
  - could be unsuccessful if **`fd`** is not an open file descriptor
  - e.g. if **`fd`** has already been closed

number of file descriptors may be limited (maybe to 1024)

- limited number of file open at any time, so use **`close()`**

File Basics with stdio library functions

stdio.h - fopen()

```
FILE *fopen(const char *pathname, const char *mode);
```

- returns NULL if the open failed
- **mode** is string of 1 or more characters including:
    - "r" open file for reading
    - "w" open file for writing
      truncated to 0 length if it exists, created if does not exist
    - "a" open file for writing
      writes append to it if it exists, created if does not exist

See **man 3 fopen** for other modes like "r+", "w+" and "a+"

FILE *

fopen returns a **FILE** pointer

- FILE is an opaque struct - we can not access fields
- FILE stores file descriptor
- FILE may also for efficiency store buffered data
- FILE may store other things too!!!!!

Demo: Modify open_read.c and open_write.c to use stdio.h

stdio.h fclose()

```
int fclose(FILE *stream);
```

- calls close
- number of streams open at any time is limited (to maybe 1024)
- writes unwritten buffered data to the stream

stdio.h reading and writing

```
int fgetc(FILE *stream) ;                    // read a byte
int fputc(int c, FILE *stream);              // write a byte

size_t fread(void *ptr, size_t size, size_t nmemb,
             FILE *stream);

size_t fwrite(const void *ptr, size_t size, size_t nmemb,
              FILE *stream);
```

stdio.h reading and writing text only

```
char *fgets(char *s, int size, FILE *stream);  // read a line
char *fputs(char *s, FILE *stream);            // write a string

// formatted input/output
int fscanf(FILE *stream, const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
```

These functions can not be used for binary data as they may contain zero bytes

- can use to read text (ASCII/Unicode)
- can **not** use to read a *jpg* for example

stdio.h convenience functions

To read/write to stdin/stdout

```
int getchar(void);                    // fgetc(stdin)
int putchar(int c);                   // fputc(c, stdout)
int puts(char *s);                    // fputs(s, stdout)
int scanf(char *format, ...);         // fscanf(stdin, format, ...)
int printf(char *format, ...);        // fprintf(stdout, format, ...)
```

These should never be used: security vulnerability, buffer overflow

```
char *gets(char *s);                  // NEVER USE
scanf("%s", array);                   // Don't use with %s
```

stdio.h - Aside: IO to strings

`stdio.h` provides useful functions which operate on strings

// like scanf, but input comes from char array **`str`**

```
int sscanf(const char *str, const char *format, ...);
```

// like printf, but output goes to char array **`str`**

// handy for creating strings passed to other functions

// size contains size of str

// Do not use similar function sprintf as it is a security vulnerability

```
int snprintf(char *str, size_t size, const char *format, ...);
```

Exercise

Implement linux **cp** command

1. byte at a time stdio.h
2. using fgets and fprintf/fputs - what is the problem with this approach?

We also have implementations using syscall and libc we can compare these to later.

Demo: fgetc return type bug

- To make a buggy version:
  - Use char instead of int for fgetc (this creates bugs with getchar too)

- Reminder: getchar and fgetc return int
  - Legal values they can return -1..255. (257 possible values)
  - This can't fit in signed char or unsigned char!

- signed char (or char on our system) can store -1 and detect EOF,
  - but valid byte value 0xFF gets mistaken for EOF

- unsigned char can't store -1 and can't detect EOF

Demo: cp using fgets and fprintf

- Using fgets and fprintf to copy a file
- Seems to work fine when copying text files BUT
    - Breaks for binary files with 0x00 bytes
    - They are interpreted as end of string '\0' character

Reminder: only use fgets, fprintf, fscanf, or fputs for text

Comparing Libc Wrappers vs stdio

IO Performance & Buffering libc vs stdio

Let's compare our implementations of cp!

```
$ clang -O3 cp_x.c -o cp_x
$ dd bs=1M count=10 </dev/urandom >random_file
10485760 bytes (10 MB, 10 MiB) copied, 0.183075 s, 57.3 MB/s
$ time ./cp_x random_file random_file_copy
```

Can we get any insights from strace?

```
$ strace ./cp_x random_file random_file_copy
```

Compare:

Linux cp command, cp_fgetc_one_byte.c, cp_libc_one_byte.c, cp_libc.c

stdio.h buffering for efficiency

- **Goal:** reduce number of system calls (expensive)
- **Reading:**
  - Uses a **read** system call to fill whole buffer
  - Subsequent reads get bytes from the buffer
  - Does not do another **read** system call till it runs out of data in the buffer
- **Writing**:
  - Delays calls to **write** system call by storing data in buffer (array) instead
  - Calls **write** system call only when
    - Buffer is full
    - a newline is encountered for line buffered output (e.g. terminal)
    - `fflush` is called
    - file is `fclosed`
    - Note: files are closed and buffers are flushed on exit

fflush stdio buffers

You can manually flush stdio buffers by using:

```c
int fflush(FILE *stream);
```

For example

- this would force a write system call to stdout and empty the output buffer
  ```c
  fflush(stdout);
  ```
- Can also be used for files that have been opened for writing.
- Should not be used for stdin or files opened for read only.

Demos: fflush_line.c fflush_full.c

What we learnt today

- System calls related to files (found in man section 2):
  - open, close, read, write
- Equivalent stdio portable functions (found in man section 3):
  - fopen, fclose, fgetc, fputc, fgets, fputs, fread, fwrite
- Stdio buffers
  - fflush

Next Lecture

- File Systems:
  - File metadata
    - Permissions
    - system call stat
  - Hard Links and Symbolic Links
  - Working with directories

Feedback Please!

Your feedback is valuable!

If you have any feedback from today's lecture, please follow the link below or use the QR Code.

Please remember to keep your feedback constructive, so I can action it and improve your learning experience.

https://forms.office.com/r/kD5B7X4L47

Reach Out

Content Related Questions:

[Forum]

Admin related Questions email:

cs1521@cse.unsw.edu.au

Student Support | I Need Help With...

I need help with...		Service	Contact		
My Feelings and Mental Health Managing Low Mood, Unusual Feelings & Depression		**Mental Health Connect**	student.unsw.edu.au/**counselling** Telehealth	**In Australia Call Afterhours UNSW Mental Health Support Line**	1300 787 026 5pm-9am
		Mind HUB	student.unsw.edu.au/**mind-hub** Online Self-Help Resources	**Outside Australia Afterhours 24-hour Medibank Hotline**	+61 (2) 8905 0307
Uni and Life Pressures Stress, Financial, Visas, Accommodation & More		**Student Support Indigenous Student Support**	student.unsw.edu.au/**advisors**		
Reporting Sexual Assault/Harassment		**Equity Diversity and Inclusion (EDI)**	edi.unsw.edu.au/**sexual-misconduct**		
Educational Adjustments To Manage my Studies and Disability / Health Condition		**Equitable Learning Service (ELS)**	student.unsw.edu.au/**els**		
Academic and Study Skills		**Academic Language Skills**	student.unsw.edu.au/**skills**		
Special Consideration Because Life Impacts our Studies and Exams		**Special Consideration**	student.unsw.edu.au/**special-consideration**		