COMP1521 25T1

Week 5 Lecture 2

Floating Point, Operating Systems and File Systems

Adapted from Hammond Pearce,
Andrew Taylor and John Shepherd's slides

Assignment 1 is due Friday 6pm

Week 4: test: due thursday 9pm (MIPS basics, control, arrays)

Flex week next week!

No lectures, tutorials or labs. Nothing due!

Lab 5 will be due in week 7. There is no lab 6.

Test 5 and Test 6 will be due in week 7

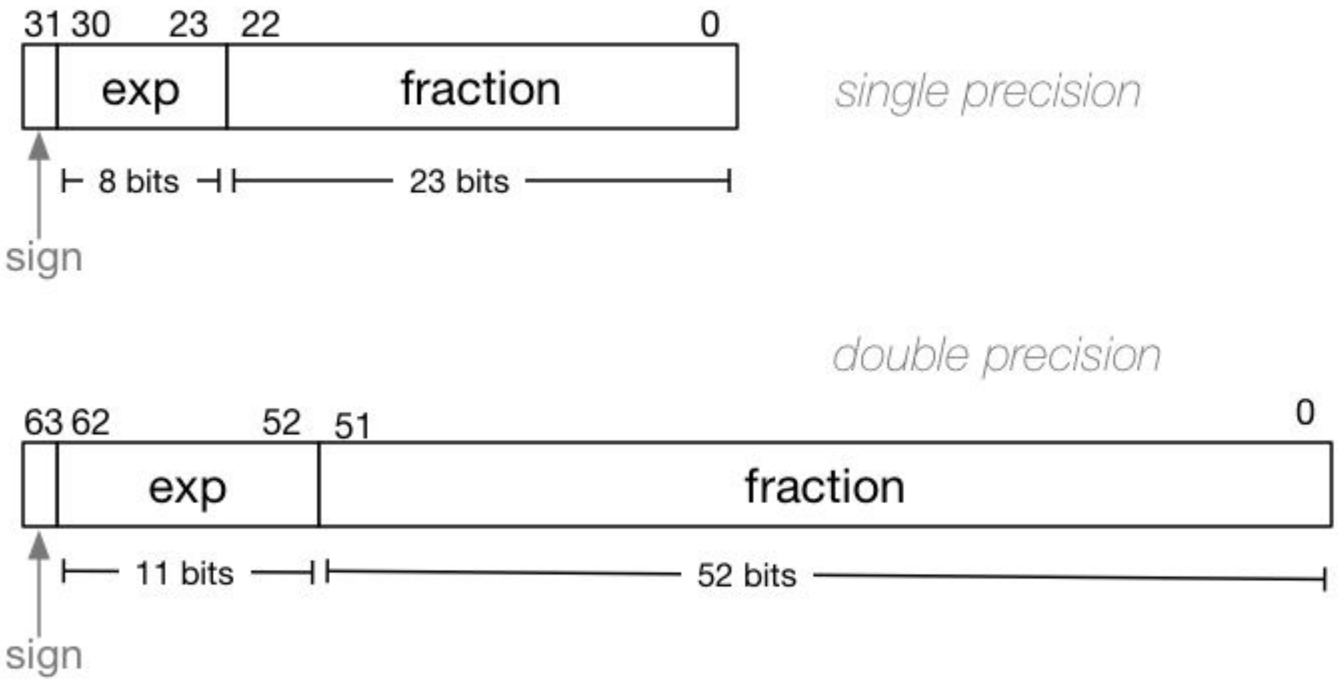
There will still be help sessions on.

There will be bitwise operators revision sessions on - stay tuned to course forum announcements for details

Today's Lecture

- Floating Point Representation
- Operating Systems
- File Systems
  - System Calls

IEEE 754 Standard



Note:
float in C is represented in this single precision format.
double in C is represented in this double precision format

Note: the fraction part is often called the mantissa

IEEE 754 Example

150.75 = 10010110.11

// normalise fraction, compute exponent

= 1.001011011 × 2^7

// determine sign bit,

// map fraction to 24 bits, (don't store the leading 1)

// map exponent to 8 bits after adding on the bias of 127

= 01000011000101101100000000000000

where red is sign bit, green is exponent, blue is fraction

Note: B=127, e=2^7, so exponent = 127+7 = 134 = **10000110**

Check using explain_float_representation.c or Floating Point Calculator

Floating Point Recap Exercise

Keep in mind 10000000 = 2^7 = 128

Convert -42.5 to IEEE 754 float?

Convert to decimal from IEEE 754 float
00111110100000000000000000000000

IEEE 754 Standard: Special Cases

Value	Exponent	Fraction	Example
0 (+ve or -ve)	all 0’s	all 0’s	
inf (∞ and -∞)	all 1’s	all 0’s	1.0/0
nan	all 1’s	Not all 0’s	0.0/0

IEEE 754 infinity.c

Representation of +- infinity : propagates sensibly through calculations

```
double x = 1.0/0.0;

printf("%lf\n", x); //prints inf

printf("%lf\n", -x); //prints -inf

printf("%lf\n", x - 1); // prints inf

printf("%lf\n", 2 * atan(x)); // prints 3.141593

printf("%d\n", 42 < x); // prints 1 (true)

printf("%d\n", x == INFINITY); // prints 1 (true)
```

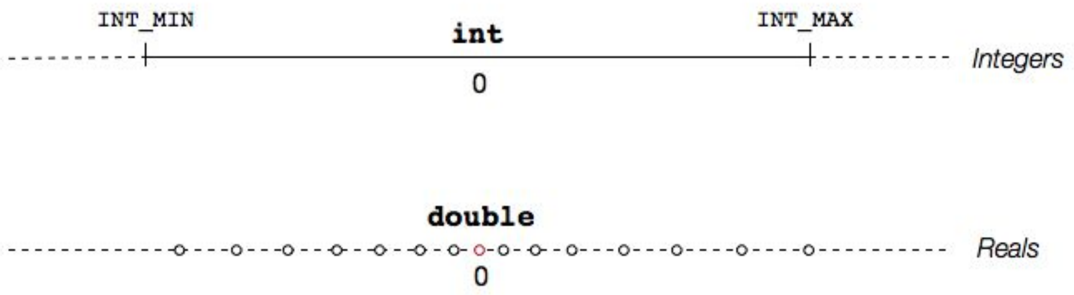
IEEE 754 nan.c

Representation for invalid results NaN (not a number)

- ensures errors propagates sensibly through calculations

```
double x = 0.0/0.0;

printf("%lf\n", x); //prints nan

printf("%lf\n", x - 1); // prints nan

printf("%d\n", x == x); // prints 0 (false)

printf("%d\n", isnan(x)); // prints 1 (true)
```

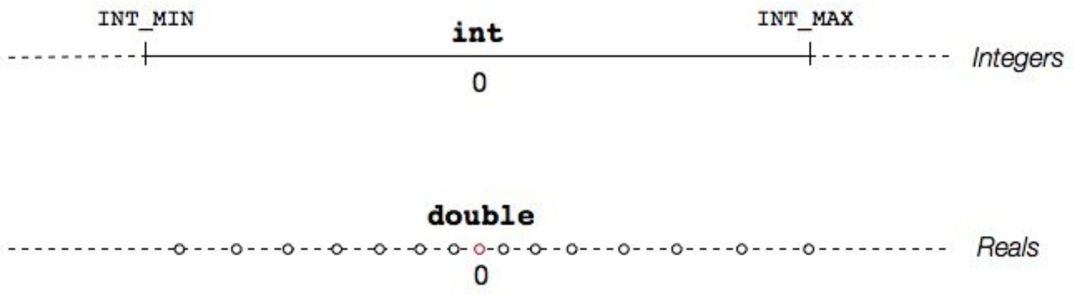
Distribution of Floating Point Numbers



integers ... subset (range) of the mathematical integers

- can represent all integer values in that subset
- each integer is 1 away from the next one and previous one
- all integers are represented accurately

Distribution of Floating Point Numbers



floating point ... subset of the mathematical real numbers

- floating point numbers not evenly distributed
  - numbers closer to 0 have higher precision which is good
  - representations get further apart as values get bigger
  - this works well for most calculations but can cause weird bugs

Distribution of Floating Point Numbers

A 64-bit **double** uses 52 bits for the fraction (mantissa).

- Between 2^n and 2^{n+1} there are 2^{52} doubles evenly spaced
  - e.g. in the interval 2^{-42} and 2^{-43} there are 2^{52} doubles
  - and in the interval between 1 and 2 there are 2^{52} doubles
  - and in the interval between 2^{42} and 2^{43} there are 2^{52} doubles

- near 0.001 - doubles are about 0.00000000000000000002 apart
- near 1000 - doubles are about 0.0000000000002 apart
- near 1000000000000000 - doubles are about 0.25 apart
- **above 2^{53} - doubles are more than 1 apart**

Code Demos

double_disaster.c
double_catastrophe.c
explain_float_representation.c

Operating Systems and File Systems

Reminder: Linux Manual

The linux manual (**`man`**) is divided into sections.

Important sections for this course include:

1. Executable programs eg. ls, cp
2. System calls
   - we will be looking at many of these today and in the coming weeks
3. Library calls eg. strcpy, scanf

And other sections that you can find out about by using the command **`man man`**

Advice: **`man`** will be available in the exam. Get used to using it!

Operating Systems

- This course is a great way to see different areas in computing to
  - See what electives you might be interested in!!
  - See what area you might want to work in!!
- **Question** : What is YOUR favourite operating system?
  - Write in the chat
- **Question 2**: What do operating systems do?
  - Write in the chat

A World without Operating Systems

- Manually Boot Your Computer
  - No OS means no automatic booting into a familiar environment.
- Write your own file system
  - No folders, no directories, your hard drive is just raw data
- Run Programs… If You Can
  - Multi-tasking?? Good luck.
- Security?
  - Your dodgy game can steal your passwords you typed into your online banking… if you could connect to the internet… because…
- Why won't my mouse, printer, usb port, internet connection work??
  - No OS = No drivers. Every program must **talk directly to the hardware**

A World without Operating Systems

- You would need to learn to do this for every specific computer unless it happened to have the same exact configuration of hardware
  - You would not be too keen to use a different device
  - Or get an upgrade would you??

- You would need to write different code for all different configurations of hardware!

A world with Operating Systems

We want to **generalise** computers and provide functionality so:

- Users can easily use different machines with different configurations of hardware
- We can write code that can target lots of computers regardless of their hardware
  - **Abstraction:** We can write higher level code where we don't have to understand the exact hardware specs, or voltages etc.
  - **Portable code:** We can write code that runs on other machines!

Operating Systems

- Operating system (OS) sits between the user and the hardware
- The OS effectively provides a virtual machine to each user.
  - much easier for user to write code and use machine
  - difficult (bug-prone) code implemented by operating system
  - coordinates access to resources e.g. file systems, multiple processes
  - The virtual machine interface can stay the same across different hardware making it easier for user to write portable code

Operating Systems: Privileged Mode

- Needs hardware to provide a **privileged** mode
  - OS kernel runs in this mode
  - code can access all hardware, memory and CPU instructions
- 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

System Calls

- Different operating system have different system calls
  - Linux system calls are very different Windows system calls
  - Linux provides 400+ system calls
  - type man syscalls to find out more information
- Examples of operations that might be provided by system call
  - read or write bytes to a file
  - create a process (run a program) or terminate a process
  - send information over the network

Mipsy System Calls

- **mipsy** provides a virtual machine which can execute MIPS programs
- **mipsy** also provides a tiny operating system
- **mipsy** system calls
- **syscall** instruction
  - small number of very specific system calls
  - designed for students writing small programs with no library functions
  - MIPS programs running on real hardware and real OS also use **syscall**

Experimenting with Linux System Calls

- Linux system calls also have a number
  - e.g system call **1** is **write** bytes to a file
- Linux provides 400+ system calls

```
$ cat /usr/include/x86_64-linux-gnu/asm/unistd_64.h
...
#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
#define __NR_close 3
...
#define __NR_set_mempolicy_home_node 450
```

System Calls in Linux

syscall command

- not usually used in practice
- syscalls vary between operating system code is less portable
- hard to understand

Libc syscall wrapper:

- more meaningful names
- does syscall for you and helps with error checking
- more portable than syscall but not portable
    - some work on POSIX compliant systems (like linux and MacOS)

System Calls in Linux

Higher level library functions like **stdio.h:**

- useful most of the time
- calls syscall wrapper for you
- portable
- does other cool stuff to make thing easier
- you have been using these to indirectly do your system calls the whole time!

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

Unix Files

- On Unix-like systems a **file** is sequence/stream of zero or more bytes
  - file metadata doesn't record that it is e.g. ASCII, MP4, JPG, …
  - file extensions are just hints

Demo: Different File formats on Linux

Files and File Systems

- Files typically live on a hard drive or solid state drive
  - To interact with their data - they need to be read into RAM
  - We need to use system calls to do this!
    - A system call to open the file
    - System calls to read or write bytes from/to the file
    - A system call to close the file when we finish
- File Systems provide a mapping from the file name to where the files are stored on the drive.

File Descriptors

- **file descriptors** are small integers
  - Uniquely identify a stream/file that is open within a process
  - Are indexes into a per process OS file descriptor table

- OS stores info for each file descriptor such as:
  - File offset: current position in the file
  - File status: read-only, write-only etc
  - Information to locate the actual bytes related to the file/stream

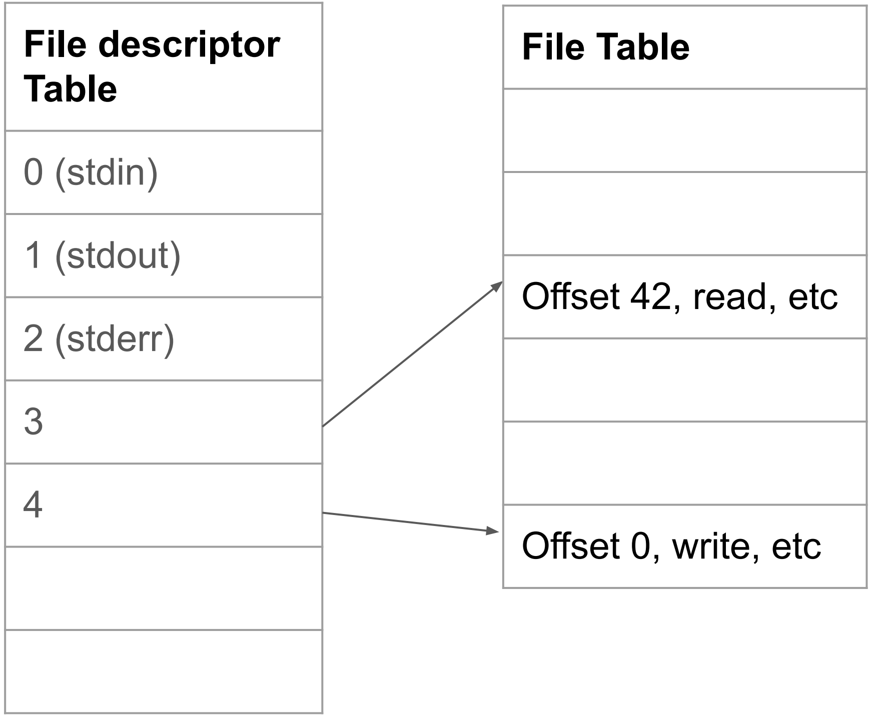
File Descriptors

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

When a file is opened a new file descriptor is added to the table.

When a file is closed the file descriptor is removed

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



What on earth is stderr?

- There are 3 standard streams in linux
  - stdin (0), stdout (1), stderr (2)
- They are treated like they are files in linux
  - They are a sequence of bytes like a file is
- By default
  - stdin : connected to keyboard
  - stdout: connected to terminal
  - stderr: connected to terminal

What on earth is stderr?

- The user can use redirection to send stdout and stderr to different places to separate the output from the error messages
    - ./prog > output #redirects stdout to a file
    - ./prog 2> error_msgs #redirects stderr to a file
- Demo: stderr_example.c

System call to print a message to stdout

`syscall` : make a system call without writing assembler code

- not usually used by programmers
- use to experiment and learn

```
char bytes[13] = "Hello, Zac!\n";

// argument 1 to syscall is the system call number, 1 is write
// remaining arguments are specific to each system call

// write system call takes 3 arguments:
//   1) file descriptor, 1 == stdout
//   2) memory address of first byte to write
//   3) number of bytes to write

syscall(1, 1, bytes, 12); // prints Hello, Zac! on stdout
```

Source code for hello_syscalls.c

Unix C Library Wrappers for System Calls

- Unix-like systems have C library wrapper functions corresponding to most system calls
  - e.g. **open**, **read**, **write**, **close**
  - not portable
    - some are POSIX compliant and will run on some non-Unix systems
  - typically return **-1** on error and set the error code **errno**
  - Better to use library functions (eg stdio.h functions) when possible.

Libc wrapper to print message to stdout

```
char bytes[13] = "Hello, Zac!\n";

// write takes 3 arguments:
//    1) file descriptor, 1 == stdout
//    2) memory address of first byte to write
//    3) number of bytes to write
write(1, bytes, 12); // prints Hello, Zac! on stdout
```

Source code for hello_libc.c

stdio.h - C Standard Library I/O Functions

- **stdio.h** provides a portable higher-level API to manipulate files.
    - part of standard C library
    - available in every C implementation that can do I/O
    - functions are portable, convenient & efficient
    - on Unix-like systems they will call open()/read()/write() ... with buffering
- Use stdio.h functions for file operations unless you have a good reason not to
    - e.g .program with special I/O requirements like a database implementation

stdio library to print message to stdout

```
char bytes[] = "Hello, Zac!\n";
printf("%s",bytes);
```

`printf` will do the write system call for us!

See more ways to print using `stdio.h` with `hello_stdio.c`

Source code for hello_stdio.c

Live Coding

syscall vs libc vs stdio.h

hello.c printing to stdout

read_char.c reading byte from stdin

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`	open for reading
`O_WRONLY`	open for writing
`O_APPEND`	append on each write
`O_RDWR`	open object for reading and writing
`O_CREAT`	create file if doesn't exist
`O_TRUNC`	truncate to size 0

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

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 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()`**

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, `0` returned
- if error, **-1** is returned and **errno** set
- file descriptor **current position** in file is updated

Libc library wrapper for read 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

Code Demo

open_read.c

open_write.c

open_issue.c

Coming up after flex week

Working with stdio.h library and files

And much more about file systems!

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/hP0wEPPFPX

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
	Mind HUB	student.unsw.edu.au/**mind-hub** Online Self-Help Resources
	In Australia Call Afterhours UNSW Mental Health Support Line	1300 787 026 5pm-9am
	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**