**CS 35L Notes:**

**Lab 1:**

**Commands:**

man: get manual or man pages

man ls //show the man page for ls command

/ //keyword search within man page

q //quit the man page

pwd: print working directory

cd: change working directory

~: home directory

.: current directory

/: root directory, or directory seperator (root is highest level of hierarchy)

.. parent directory

Environment Variables:

$PATH - ("echo $PATH") List of directories to search for commands

$HOME - Home directory

printenv: prints all env variables

echo: write arguments to stdout (-e options allows the use of tab chars)

mv: mv a file (no undos), can also change the name of the file

cp: copy a file

rm: remove a file

mkdir: make a directory

rmdir: remove a directory

ls: list contents of a directory

-l: show long listing including permission info

-a: list all files including those that are hidden

-s: show size of each file, in blocks

-h: human readable form

cat: concatenate and print files (works with only one arg as well)

tr: translate characters from in set1 to characters in set2

sed: stream editor that requires reg expressions

grep: file pattern searcher

find: recursively search path, matching comparison criteria

locale: display locale settings

sort: sort each line based off of current locale

head: display first lines of a file

tail: display last lines of a file

du: display disk usage stats

ps: process status

kill: terminate or signal process

diff: compare files line by line

cmp: compare two files byte by byte (gives where they differ as default)

wc: display word count stats (lines, words, bytes)

**File matching:**

?: matches any single character

-example: ls file? (file1, filea, fileB, etc.)

\*: matches one or more characters in a filename

[]: matches any one of the characters between the brackets. Use - to separate a range of consecutive characters (e.g. a-k)

**File Permissions:**

For files there are 3 classifications:

User: owner of the file

Group: group of the owner (csugrad) or (csgrad)

Other: others with accounts on the system

With ls -l command the following is shown:

rwx-rwx-rwx - 111 110 110

r: read permission (can file be viewed)

w: write permission (can file be edited)

x: executable permission (can file be executed)

Furthest left corresponds to user permissions

Middle corresponds to group permissions

Furthest right corresponds to group permissions:

chmod: change file permissions

Method 1: use binary translation:

rwx-rw-rw- (111 110 110), binary translate to (766)

chmod 766 file.txt

Method 2:

-use u,g,o abbreviations

-use + to add and - to subtract permissions (u+x) (g-x)

chmod u+x,g-r file.txt or chmod u=rwx, g=r file.txt

**Command History:**

Up-Arrow: previous command

Tab: auto-complete command (double tap for all options)

!!: replace with previous command

-ls

-man !! (in this case it would be "man ls")

**STDIN/OUT:**

command > file.txt write stdout to file.txt

command >> file.txt append stdout to file.txt

< file.txt use contents of a file as stdin

program 2> file.txt stderr writes to file.txt

Can pipeline:

cat < file | sort > file2

much easier to run multiple commands w/o having to continually save output

**Changing File Attributes:**

ln: create a link

-Hard links: points to physical data

-once two files are linked, no difference between the two

-Soft Links (symbolic links) -s: points to a file

-if the file the link points to is deleted, there’s no content

touch: update access & modification time to current time

Find Command:

options:

-type: type of file (e.g. directory, symlink)

-perm: permission of a file (-perm -u=x) - any user executable

-name: name of a file

-prune: don't descend into a directory (not recursive)

-ls: list current file

**Lab 2:**

Compiled Languages:

-C/C++

-Programs are translated from their original source code into object code that is executed by hardware

-efficient

-work at low level, dealing with bytes, ints, floating points

Scripting Languages:

-interpreted

-interpreter reads program, translates it into internal form

and execute programs

Shell script is a file with shell commands

When shell script is executed a new child "shell" process is spawned

First line of script specifices which child "shell" to use

#! /bin/csh -f

#! /bin/awk -f

#! /bin/sh

#! /bin/bash

Ubuntu uses default "dash" shell which is POSIX compliant

-/bin/sh links to /bin/dash (bash and dash practically same)

**Declaring Variables:**

Var="hello" //no spaces between words

echo $Var //prints variable value

Built-in Vars:

# Number of arguments given to current process

@ Command-line arguments to current process. Inside double quotes, expands to individual arguments

\* Command-line arguments to current process. Inside double quotes, expands to single argument

Built-in:

-(hyphen) Options given to shell on invocation

? Exit status of previous command

$ Process ID of shell process

0 (zero) The name of the shell program

! Process ID of last background command. Use this to save process ID numbers for later use with the wait command

IFS Internal field separator; i.e. the list of character that act as word separator. (Default: space, tab, and newline default)

LANG

LC\_ALL Name of current locale, overrides or LC\_ vars

LC\_COLLATE Name of current locale for character collation (soring)

LC\_TYPE Name of current locale for character class det during pattern matching

LINENO Line number in script or function of the line that just ran

PWD Current working directory

Positional parameters represent a shell script's command-line args

Enclose number in braces if > 9

ex.

echo first arg is $1

echo tenth arg is ${10}

If statements use the test command or []

Include spaces between phrase and '[' ']''

; after condition

Tons of options for test command

Comparisons:

|  |  |
| --- | --- |
| -eq | equal to |
| -ne | not equal to |
| -lt | less than |
| -le | Less than or = to |
| -gt | Greater than |
| -ge | Greater than or = to |

**IF**

if [ 5 -gt 1 ]; then

echo "5 greater than 1"

else

echo "not possible"

fi //DON'T FORGET fi

**DON’T FORGET TO USE $ WHEN TALKING ABOUT VARIABLE VALUE AND TO USE ‘’ ANY TIME YOU WANT THE LITERAL TRANSLATION.**

**Return values - check last command with $?**

0 Command exited successfully

>0 Failure during redirection or word expansion

1-125 Command exited unsuccesfully (each # has meaning)

126 Command found, file not executed

127 Command not found

>128 Command died due to receiving a signal

Quotes:

Backticks ` `

-expand as shell commands

-temp=`ls` ; echo $temp (result of command is saved to var)

Single quotes ' '

-do not expand at all, literal meaning

-temp='$hello$hello' ; echo $temp ($hello$hello is temp)

Double quotes " "

-almost like single quotes but expand backticks,

$, \ characters

ex is if hello="goodbye" ; temp="$hello" (temp is goodbye)

**While Loop**

let command is used to do arithmetic

count=6

while [ $count -gt 0]; do

echo Value of count is $count

let count=count-1

done

**For Loop**

temp=`ls`

for f in $temp; do

echo $f

done

f here refers to each word in the ls output

also

for (( x=0; x < 4; x=x+1))

echo The count of x is $x

done

Output using echo and printf

Echo can't output escape characters: echo "Hello \n world" 🡪 Hello \n world

Printf can output escape characters: printf "Hello \n world" 🡪

Hello

world

Execution Tracing

-print out each command with "+"

set -x: to turn it on

set +x: to turn it off

Searching for Text:

grep: Uses basic regular expressions (BRE)

egrep: Extended grep that uses extended regular expressions (ERE)

grep -E

egrep

sed -r

Fgrep: Fast grep that matches fixed strings instead of reg exp.

grep -F

fgrep

Reg expressions:

\ Both Turn off special meaning of character (need 2 to escape .)

--need one to escape | ( and )

. Both Match any single character except NUL. (1)

\* Both Match any number of the char that precedes it.

^ Both Match the following reg exp at the beginning of the line/string

$ Both Match the preceding reg exp at the end of the line/string (the $ is last character)

[...] Both Matches any one of the enclosed characters. Use - for a consecutive range

[^...] Both Matches any character not enclosed by brackets

+ ERE Match 1 or more instances of the preceding regular expression

? ERE Match 0 or 1 instances of the preceding reg exp.

| ERE Match the regular expression specified before or after

() ERE Apply a match the enclosed group of reg exps

Posix Bracket Expressions:

[:alnum:] alphanumeric chars

[:alpha:] alphabetic chars

[:blank:] Space and tabs chars

[:cntrl:] Control chars

[:graph:] Nonspace characters

[:lower:] Lowercase chars

[:print:] Printable chars

[:space:] Whitespace characters

[:upper:] Uppercase characters

[:xdigit:] Hexadecimal digits

sed:

replace parts of text:

sed 's/:.\*//' /etc/passwd Remove everything after :

sed '/patternstart/,patternstop/d' to delete pattern start to patternstop

sed 's/^/ /' insert 5 white spaces at beginning of each line

tr -d to delete

r -s to squeeze multiple into one

tr "[A-Z]" "[a-z]" //make all lowercase

tr -s '[:blank:]' //delete blank spaces

tr -s '[\n]' '[ ]' //delete all newlines

**Lab 3:**

decompress tar file with:

tar -xzvf filename.tar.gz

-x: extract

-z: gzip

-v: verbose

-f: file

Make file allows only files that change to recompile

Basic example:

# Makefile - A Basic Example

-the tab (second line with g++ or CXX or gcc) is REQUIRED

all : shop #usually first

shop : item.o shoppingList.o shop.o

g++ -g -Wall -o shop item.o shoppingList.o shop.o

item.o : item.cpp item.h

g++ -g -Wall -c item.cpp

shoppingList.o : shoppingList.cpp item.h shoppingList.h

g++ -g -Wall -c shoppingList.cpp

shop.o : shop.cpp item.h shoppingList.h

g++ -g -Wall -c shop.cpp

clean :

rm -f item.o shoppingList.o shop.o shop

Example 2:

makefiles:

CXX=g++

foo.o : foo.h foo.cpp

$(CXX) -c foo.cpp -o foo.o

main.o : foo.h main.cpp

$(CXX) -c main.cpp -o main.o (-c option says not to run the linker, producing .o files)

clean :

rm -rf \*.o mainexec

Practice Final:

CXX=gcc

srot : srot.c

$(CXX) srot.c -o srot

clean :

rm -f srot

**Building Process:**

configure:

-check machine details and generates Makefile

make:

(requires Makefile --- duh)

-Compiles all the program code as instructed in the Makefile

-stores executables in working directory

make install:

-make utility searches for a label named install within the Makefile and executes only that section

-executables are copied into the final directories (system dirs)

Patches are diff files

**diff output:**

---path/to/original\_file

+++path/to/modified\_file

@@ -l,s +l,s @@

@@: beginning of a hunk

-l: beginning line number

-s: number of lines the change hunk applies to for each file

-straight up lines not changes

A line with:

- was deleted from original file

+ was added to the original

stayed the same

Python:

Not just scripting but Object-Oriented too

-classes

-member function

Python compiles

-compiled to bytecode

-bytecode interpretted by Python interpreter

-implicit compilation process

Slower than C

Optparse Library:

-Argument

string from command line passed into script

elements of sys.argv[1] --(first arg)

-Option

-Supplies extra info to customize execution

-Option Argument

-arg following an option that is closely associated with it

List:

Dynamic - expands as new items are added

Heterogeneous - can hold different types of items

List\_name[number]

t = [123, 3.0, 'hello!']

print t[0] --> 123

print t[2] --> hello!

Merging Lists: (if list1 = 1,2,3,4 and list2 = 5,6 ,7,8)

mlist = list1 + list2

print mlist --> (1, 2, 3, 4, 5, 6, 7, 8)

Dictionary:

Essentially a hashtable

key-pair storage

dist{} -creates an empty dictionary

-keys are unique, values are not

-keys must be immutable (strings, numbers, tuples)

dist['hello'] = "world"

print dist['hello'] --> prints world

dist['power'] = 9001

if (dist['power'] > 9000)

print "It's done due as seen by, ", dist['power']

del dist['hello']

del dist

sets:

unordered collections of unique elements

no indexing, slicing, or sequence-like behavior

For-loops in Python:

for word in list\_name:

print word

for i in range(len(list\_name)):

print i

Python has no delimiter

-no braces or keywords for code

-tabs change codes meaning

import random, sys //similar to C include statements

**Exception Handling in Python:**

Even if a program is syntactically correct, there can still be errors that occur during execution; these are called exception. Exception handling is a programming language feature that allows execution errors to be handled gracefully.

In python this is done with try and except blocks. These are blocks of code that is attempted to be executed. If no errors occur then the execution of the program continues, skipping over the except block. If an exception happens, execution of the try block immediately ceases and the except code begins. These handlers can be categorized (if it’s divide by 0 error write this, if it’s argument error write that…) or the can be left as general errors stating that an error occurred.

try:

numlines = int(options.numlines)

except:

parser.error("invalid NUMLINES: {0}". format(options.numlines))

try:

int div = 5/ui //user input (bad if 0)

except ZeroDivisionError:

print “Error: Division by 0!” //corresponding except block to catch errors

Control Flow:

If it goes well then the catch block is skipped

If an exception happens, then you catch the exception and the try block is immediately skipped

**Lab 4**

Software/Version Control

-track changes to code and other files related to the software

-what new files were added?

-what changes made?

-which version had what changes?

-track entire history of software

-Source control softwares

-GIT, Subversion, Perforce

Local SCS

-organize different versions as folders on the local machine

-no server involved

-other users should copy it via disk/network

Centralized SCS

-version history sits on a central server

-users will get a working copy of the files

-changes have to be committed to the server

-all users can ge the changes

Distributed SCS

-version history is replicated at every user's machine

-users have version control all the time

-changes can be communicated between users

-Git is distributed

Distributed Advantages:

The act of cloning an entire repository gives distributed version control tools several advantages over centralized systems:

* Performing actions other than pushing and pulling changesets is *extremely* fast because the tool only needs to access the hard drive, not a remote server.
* Committing new changesets can be done locally without anyone else seeing them. Once you have a group of changesets ready, you can push all of them at once.
* Everything but pushing and pulling can be done without an internet connection. So you can work on a plane, and you won’t be forced to commit several bugfixes as one big changeset.
* Since each programmer has a full copy of the project repository, they can share changes with one or two other people at a time if they want to get some feedback before showing the changes to everyone.

Centralized Advantages:

To be quite honest, there are almost no disadvantages to using a distributed version control system over a centralized one. Distributed systems do *not* prevent you from having a single “central” repository, they just provide more options on top of that.

There are only two major inherent disadvantages to using a distributed system:

* If your project contains many large, binary files that cannot be easily compressed, the space needed to store all versions of these files can accumulate quickly.
* If your project has a very long history (50,000 changesets or more), downloading the entire history can take an impractical amount of time and disk space.

Additionally with centralized version control it is quite easy to determine what the latest revision of the software is.

Repository:

-Files and folder related to the software code

-Full history of the software

Working Copy

-Copy of software's files in the repository

Check-out

-to create a working copy of the repository

Check-in/Commit

-write the changes made in the working copy to the repository

-commits are recorded by the SCS

Git Source Control:

Objects used by GIT:

-Blobs sequence of bytes

-Trees groups blobs/tress together

-Commit refers to a particular commit (all info about committ)

-Tags just a named commit bject for convenience

Objects uniquely identified with hashes

**Terms:**

Head

-refers to a commit object

-there can be many heads in a repository

HEAD

-refers to the currently active head

Detached HEAD

-if a commit is not pointed to by a branch

-this is okay if you want to just take a look and not commit any new changes

-If the new commits have to be preserved then a new branch has to be created

Branch

-refers to a head and its entire set of ancestor commits

Master

-default branch

Repository creation:

git init new repo

git clone copy of an existing repo

Branching

git checkout <tag/commit> -b <new\_branch\_name>

(creates a new branch)

git branch (show available branches)

Commits

git add (Stage modified files) git add foo.h foo.c (can use . to stage all changes)

git commit (check-in the changes to the repo) git commit foo.h foo.c –m “Message about commit”

Getting info

git status (shows modified files, new files, etc)

git diff (compares working copy with staged files)

git log (shows history of commits)

git tag (list of tags used)

git show (show a certain object in the repo)

Getting help

git help

Git Rebase

-rewrites commit history

-loses context

-never use this on public branches!

How to?

git checkout feature

git rebase master //then can merge (git checkout master then git merge testBranch)

Merging

-merging hotfix branch into master

-git checkout master

-git merge hotfix

-Git tries to merge automatically

-simple if its a forward merge

-otherwise, manually resolve conflicts

Refer to multiple parents

-git show hash

-git show hash^2 (shows second parent)

HEAD^^ == HEAD~2

Reverting

-git checkout HEAD main.cpp

-get the EHEAD revision for the working copy

-git checkout --main.cpp

-revert changes in the working directory

-git revert

reverting commits

Cleaning up untracked files

-git clean

Tagging

-Human readable pointers to specific commits

-git tag -a v1.0 -m "Version 1.0"

Lab 5:

Built-in C Types:

-Ints, Floating-points, char strings

-No bool

Compiling 'C' only

-gcc -std=c99 binsortu.c

No classes, but structs

No method and access modifiers in C structs

Ptrs:

int \*iptr;

iptr = &ival;

double x, y, \*ptr;

ptr = &x;

\*ptr = 7.8; //x var gets value of 7.8

\*ptr \*= 2.5; //x multiplied by 2.5

y = \*ptr +0.5 // x + 0.5

Can have multiple pointers to each other.

Can have pointers to functions

double (\*funcPtr)(double, double);

double result;

funcPtr = pow; //pow() is a function here

result = (\*funcPtr)(1.5, 2.0);

result = funcPtr(1.5, 2.0); //same call as above

typedef struct Point { double x, y;} Point\_t;

typedef struct

{

Point\_t top\_left;

Point\_t bottom\_right;

} Rectangle\_t;

Memory Allocation:

malloc(size\_t size): allocates a block of mem at least the size of specified size

free(void \*ptr): free the block of mem pointed to by ptr

realloc(void \*ptr, size\_t newSize): Resizes allocated mem

Opening & Closing Files:

FILE \*fopen(const char \* restrict filename, const char \* restrict mode):

int fclose(FILE \*fp);

Reading/Writing characters

getc(FILE \*fp); //read

putc(int c, FILE \*pf); //write

Reading/Writing Lines

char \*fgets(char \* buf, int n, FILE \*fp);

int fputs(const char \*s, FILE \*fp)

Formatted Output:

int fprintf(FILE \* restrict fp, const char \* restrict format, ..);

int fscanf(FILE \* retrict fp, const char \* restrict format, ...);

getchar() takes char from stdin

getc(..) allows char input from a different stream

fscanf (scanf but allow different input streams)

Similarly,

putchar() writes to stdout

putc allows char output to a different stream

fprintf allows output to different stream

#include <stdio.h> //printf, scanf, null

#include <stdlib.h> /malloc, free, rand

Format String:

int score = 120;

char player[] = "Mary";

printf( "%s has %d points.\n", player, score );

fprintf(stdout, "%s has %d points. \n", player, score)

fscanf(stdin, "%d", &score);

fprintf(fPtr, "%s has %d points.\n", player, score);

fscanf(fPtr, "%d", &score);

%s - strings

%d - decimal integers

%f - floating point

Ternary Op ?

result = a > b ? x : y;

Equivalent to:

if (a >b){

result =x;

} else {

result = y;

}

void qsort (void\* base, size\_t num, size\_t size, int (\*compare)(const void\*, const void\*));

-base: Records to be sorted

-num: Number of records

-size: Size of each record

-compare function pointer

Compare func – int (\*comp)(const void\* r1, const void\* r2)

// Takes 2 records as arguments and returns an int

Return value < 0 then 'r1' goes before 'r2'

Return value = 0 then 'r1' and 'r2' are equivalent

Return value > 0 then 'r2' goes before 'r1'

**Debugging in C:**

gcc -o ExecutableFile -g main.c

-o is name of exeFile

-g option indicates to include symbol and source-line info for debugging

**Source Code Display:**

list file\_name: line\_number (list sfrobs.c: 34)

list line\_number (uses current source file)

list from, to (displays range of src code, to from can be lines or function names)

list (show more lines from last command)

Breakpoints:

break file\_name: line\_number (shortcut use b)

break function (set a beginning)

break (break at current line)

delete breakpoint #/range (straight delete shortcut d))

delete breakpoint (deletes all)

disable #/range (temporarily disable)

enable #/range (restores disabled breaks)

ignore # iterations (pass over for x amount of time)

break \_position\_ if expression

continue (run until exits or breakpoint)

step (lines) (execute current line of program)

next (lines) (like step but doesn't stop for function calls)

quit, q

finish (finish current function)

Bt (show call trace)

info frame (info about current stack)

info locals (lists local vars)

info args (list the arg values of funcall)

watch expression (stop program when value change)

sfrobs.c example:

gcc -o sfx -g sfrob.c

gdb sfx

list sfrob.c:34 //displays surrounding lines of code

list 34 //effectively same as above (depends on code being executed)

list 1, 100 (1-100 line displayed)

list frobcmp, cmpwrap (shows all of code between frobcmp to cmpwrap)

break sfrob.c:34

b cmpwrap

break or b

d breakpoint 2

delete breakpoint 3-5

d (deleting all)

ignore 8 2 (ignore breakpoint 8 2 times)

break 28 if i == limit -1

**Lab 6:**

Plaintext - Actual message

Ciphertext - Encrypted message

Encryption - Plaintext -> ciphertext

Decryption - Ciphertext -> plaintext

Secret key - mathematical function used to encrypt/decrypt

Symmetric-key Encryption

-same key is used for both encryption/decryption

-key distribution is a problem

-has to be deliverd in safe way

-chance of being compromised

Public-key Encryption

(Asymmetric)

Uses a pair of keys for encryption

-public key - published and known to everyone

-private key - secret key known only to the owner

Encryption

- use public key to encrypt

- anyone can encrypt message, but they cannot decrypt ciphertext

Secure Shell (SSH)

Telnet

-remote access

-not encrypted

-packet sniffers can intercept sensitive info (user/psswd)

**SSH**

-run processes remotely

-encrypted session

-session key (secret key) used for encryption during session

Passwordless login with keys

-use private/public keys for authentication

-ssh-keygen

-passphrase (longer version of passwd)

-passphrase for protecting private key

-passphrase neede whenever the keys are accessed

-ssh-copy-id uname@seas.ucla.edu

-copies the publi key to the server

-Login w/out password

Passphrase-less authenticatoin

-ssh-agent -Authentication agent

-manages private key identities for SSH

-to avoid entering the passphrase whenever the key is used

ssh-add

-registers the private key with the agent

-passphrase asked only once

-ssh will ask the ssh-agent whenver the keys are needed

Session encryption

-symmetric encrytpion

-exchange secret key

Host/Client Validation

-public-key encryption

challenge-response

-host sends a challenge that has to be answered by the client

-similarly client sends a challenge that has to be answered by the host

2 way verification

**Account Admin:**

sudo apt-get update //downloads the package lists from repos and updates them to get infromation on the newest versions of packages

" " install openssh-server

install openssh-client //install openssh

sudo useradd -d /home/username -m username

sudo passwd username

cd /home/username/

sudo mkdir .ssh

sudo chown -R username .ssh //recursively change file owner and all subdirectories/files

sudo chmod 700 .shh //gives permissions to only that user

ifconfig //gives IP address of server

ps aux | grep ssh //shows process named sshd the daemon/server

Client

password

ssh username@ip\_address //the one from ifconfig

password-less

ssh-keygen

ssh-copy-id -i username@pip\_addr

ssh username@ip\_addr

passphrase-less

ssh-add

ssh username@ip\_addr

Xforwarding

ssh -X username@ip\_addr

xterm

firefox -- program shows up on client screen, running on server

scp:

based on ssh

used for securely transferring files (encrypted)

**Digital Signatures**

protect integrity of the documents

-receiver recieved the document that the sender intended

digital signature is extra data attached to the doc that can be used to check for tampering

message digest

-shorter version of the document

-generated using hashing algorithms

-even a slight change in the original doc will change message digest (high probablity)

gpg

--gen-key (generating new keys)

--armor (ASCII format)

--export (export public key)

--import (import public key)

--detach-sign (creates file with just signature)

--verify (verify signature with a public key)

--encrypt (encrypt doc)

--decrypt (decrypt doc)

--list-keys all keys in the keyring)

--send-key (register key with a public server)

--search-key (search for someone's key)

**Lab 7:**

System Calls:

Processor Modes:

-Mode bit used to distinguish between execution on behalf of OS & execution on behalf of user

-Supervisor mode: processor executes every instruction on its hardware repertoires

-User mode: can only use a subset of instructions

Supervisor privileges (instructions that can executed in supervisor mode)

I/O instruction are protected. If an app needs to do I/O, it needs to get the OS to do it on its behalf.

Instruction that can change the protection state of the system are privileges

The Kernel:

-Core of OS software executing in supervisor state

Trusted software

-manages hardware resources (CPU, Memory and I/O)

-implements protection mechanisms that could not be change through actions of untrusted software

System call interface is a safe way to access privileges of kernel

System Calls:

A system call involves the following:

-the system call causes a 'trap' that interrupts the execution of the user process

-takes control of the processor

-executes the system call on behalf of the user process

-the user process get back control of processor

System calls have to be used wisely

-expensive due to privilege switching

ssize\_t read(int fildes, void \*buf, size\_t nbyte)

-fildes file descriptor

-buf buffer to write to

-nbyte number of bytes to read

ssize\_t write(int fildes, const void \*buf, size\_t nbyte)

-fildes file descriptor

-buf buffer to write to

-nbyte number of bytes to write

int open(const char \*pathname, int flags, mode\_t mode)

int close(int fd);

File descriptors:

0 stdin

1 stdout

2 stderr

stat struct contains info about file (mtime, owner, protection, etc.)

int fstat (int fildes, struct stat \*buf);

-buf struct hold all file information that can accessed

Buffering is done to decrease the number of system calls and thus improve efficiency. This is the process of storing I/O results before system until a threshold is reached before have the system call.

In read/write cases this is a specific nbyte size that has been optimzied to whatever user program is being run.

Faster to write to a file than the terminal (formatting and less buffering for terminal)

**Lab 8:**

Multitasking

-run multiple processes simultaneously to increase performance

-processes do not share internal structres (stack, globals, etc)

-communicate via inter-process communication methods

-pipes, sockets, signals, messages queues, etc.

single-core: Illusion of parallelism by switching process quicky (time-sharing)

Multi-core: True parallelism. Multiple processes execute concurrently on different CPU cores

Threads:

A process can be:

single-threaded

multi-threaded

Threads in a process can run in parallel

A thread is a lightweight process

It is a basic unit of CPU utilization

Each thread has its own

Stack

Registers

Thread ID

Each thread shares:

Code

Global Data

OS resources (files, I/O)

Multithreading properties

-efficient way to parallelize task

-thread switches are less expensive compare to process switching (context switching)

-inter-thread communication is easy, via shared global data

-need synchronization among thread accessing same data

Thread safey/synchronization

-thread safe function

-safe to be called by multipled threads at the same time. Function is free of race conditions

-race conditions - the output depends on the order of execution (reading/writing from same place)

int pthread\_create(pthread\_t \**thread*, const pthread\_attr\_t \**attr*, void \*(\**start\_routine*) (void \*), void \**arg*);

int pthread\_join(pthread\_t thread, void \*\*retval);

Dynamic Linking:

C/C++ Compilation:

preprocessor, compiler, linker, loader

Linux Libraries

-Static Library (.a)

-statically linked

-every program has its own copy

-more space in memory

-tied to a specific version of the lib

-new verision require recompilation

-Shared Library (.so)

-Dynamic Loaded/Linking

-Dynamic Linking

-The OS loads the library when needed

-Dynamic Loading

-The program "actively" loads the library it needs. More control to the program at run time

-Shared by multiple programs

-Lower memory footprint

-New verison o the lib does not require a recompile of a source code using lib

extern and static qualifiers

-Declaration vs. Definition

-definition allocates memory,

-declaration tells the compiler about the existence of a function or a variable (extern)

extern

-Function are extern by default

-extern int j; //just tells it exists (no mem allocated)

static

-Functions - static function is visible only in that file, and hence cannot be invoked from outside the file it is defined in.

ELF - Executable and Linking format

-format of executables, shared libraries, object code on Linux

ldd - Shows shared library dependencies

-libdl.so (the DLAPI), the GNU C library (libc.so), etc.

readelf - parsing ELF files

objdump - info about object files

attribute \_\_(constructor))\_\_

& des are called when a shared library is loaded by the dynamic loader, these two attributes are looked for and run if found (constructor when opened and destructor when closed.

-ldl -Wl,-rpath=$(PWD), needed to set path to working directory for dynamic linking. ldl specifies to use dynamic link library

-fPic

nm - List symbols from object files

void \* dlopen(const char \* filename, int flag);

void \*dlsym(void \**handle*, const char \**symbol*);

int dlclose(void \* handle)

dlp = dlopen("randlibhw.so", RTLD\_NOW);

rand64 = dlsym(dlp, "rand64");

dlclose(dlp);