EMACS/LINUX

- Scratch Buffer - provided for evaluating Emacs Lisp expressions interactively. GNU Emacs default bindings: C-x b scratch RET
- Compiling code: M-x compile
- Lisp Mode: M-x emacs lisp mode
- Evaluate Expression upto point: C-x C-e
- emacs C-x d (dired command) shows total available memory. ls -l doesn't
- Man command to print words having keyword: man -k “keyword”. To search within man page: Use /
- Commands: **head** - display first lines of file. **Tail** - display last part of file. **Du** - disk usage stats. **Ps** - process status. **Kill -** terminate or signal a process. **Diff** - compare files line by line. **Cmp -** compare files byte by byte. **Wc** - word, line, character, and byte count. **whereis** -- locate the binary, source, and manual page files for a command
- > file: write stdout to a file >> file: append stdout to a file
- < file: use contents of a file as stdin
- ln: create a link. **Hard links**: points to physical data(inode). **Soft links** aka symbolic links (-s): points to a file, like a shortcut.
- touch: Update access & modification time to current time. Also used to create a file.
- **Chmod** - Modify permissions. read (r), write (w), executable (x). User, group, others.
- type: type of a file (e.g, directory, symbolic link). perm: permission of a file. name:name of a file. prune: don’t descend into a directory
- ls: list current file(s)
- Command Line Interface(CLI) vs Graphical User Interface(GUI):
  - **CLI** : Steep learning curve, Pure control (e.g., scripting), Cumbersome multitasking, Speed: Hack away at keys, Convenient remote access.
  - **GUI** : Intuitive, Limited Control, Easy multitasking, Limited by pointing, Bulky remote access.
- Home directory: **~**, Print working directory: **pwd**, Current directory: **.** , Parent directory **..**
- !!: replace with previous command
- ![str]: refer to previous command with str
- ^[str]: replace with command referred to as str
- ?: matches any single character in a filename
- *: 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.
- ls: list contents of a directory -d: list only directories
  - -a: list all files including hidden ones
  - -l: show long listing including permission info
  - -s: show size of each file, in blocks
  - -h: human readable form (shows size in Byte\KB\MB...)

Shell Scripting and Regular Expressions

- Sample regex, tr, comm, sort command:
  - **cat assign2.html | tr -cs 'A-Za-z' '[\n*]' | sort -u | comm - words**
  - This command filters everything out other than the words having characters A-Z and a-z and displays each word in new line.
  - Displays: In column 1 - Words unique to sorted assign2.html. In column 2 - Words unique to 'words' In column 3 - Words that appear in both assign2.html and words
- Buildwords Script:

```
#!/bin/sh
#select all lines with <td>..</td>
grep "<td>.*</td>"|
#remove all lines having html tags
sed 's/<[^>]*>//g' |
#replace okina with apostrophe
sed "s/\`/'/g" |
#remove blank new lines
sed '/^\s*$/d' |
#convert upper case to lower case
tr '[:upper:]' '[:lower:]' |
#remove every even line
sed -n 'g;n;p'|
#remove leading whitespace
sed 's/^\s*//g' |
#replace commas and spaces  with new line
sed 's/[, ]/\n/g' |
#remove words with non-hawaiian chars
sed  "/[^p^k^'^m^n^w^l^h^a^e^i^o^u]/d" |
#sort words, display just once
sort -u
```

- First line is used to state which “child shell” to use: #! /bin/csh –f #! /bin/awk –f #! /bin/sh
- IFS (Internal Field Separator): This variable determines how Bash recognizes fields, or word boundaries, when it interprets character strings.
- $IFS defaults to whitespace (space, tab, and newline), but may be changed.
- Accessing Shell Script arguments: `echo first arg is $1 echo tenth arg is ${10}`

```
#!/bin/bash
# Built in shell variables
ls -l
echo "My first script!"
echo "Script name = $0, First arg = $1, Second arg = $2, ..."
echo "Number of arguments = $#"
#int and string variables
varValue=10
strValue="hello world strValue"
echo $varValue
echo $strValue
#For loops
for (( c=1; c<=5; c++ ))
do
    echo "Value of c is $c"
done
#While loops
c=1
while [ $c -lt 6 ]
do
    echo "Value of c is $c"
    let c=c+1
done
#Conditionals
COUNT=5
if [ $COUNT -gt 1 ]
then
    echo "Hello World!"
elif [ $COUNT -eq 5 ]
then
    echo "Hello World again!"
else
    echo "End of world!"
fi
#Reading from STDIN
while read NAME
do
    echo "Input text is:$NAME"
done
#Meaning of backticks, single quotes and double quotes
c=100
echo "Value of c = $c"
echo 'Value of c = $c'
temp=`ls -l`
temp1=$(ls -l)
echo "Backticks output : $temp"
echo "Dollar( output : $temp1"
#Another variant of iterating using for loops
printf "$temp \n"
for foo in $temp
do
    echo "next word is $foo"
done
#For debugging
set -x
```

Modifying and rewriting software

- Compiling: ./configure, make, make install
- Makefile:

  SHELL = /bin/sh
  MAKE = make
  CC = g++
  LIBS=
  CFLAGS=-DSIGSETJMP -O
  hello: main.o hello.o factorial.o functions.h
  ${CC} ${CFLAGS} -o $@ main.o hello.o factorial.o ${LIBS}
  Clean:
  rm -f *.o
- Compiling: ./configure, make, make install
- patch -p[num] < patchfile

  patch [options] originalfile patchfile

- ```
  1  randmain: randcpuid.o randmain.o
  2      gcc $(CFLAGS) -ldl -Wl,-rpath=$(PWD) -o randmain randcpuid.o randmain.o
  3
  4  randlibhw.so: randlibhw.c
  5      gcc  $(CFLAGS) randlibhw.c -fPIC -shared -o randlibhw.so
  6
  7  randlibsw.so: randlibsw.c
  8      gcc $(CFLAGS) randlibsw.c -fPIC -shared -o randlibsw.so
  9
  10
  ```
  ```

Change Management

- Disadvantages of diff and patch - Two people may edit the same file on the same date, 2 patches need to be sent and merged, Changes to one file might affect other files (.h & .c),
- Source/Version Control - **Local SCS** - No server involved,
- **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 get the changes (SVN, CVS). Pro: Everyone can see changes at the same time, Simple to design, Con: Single point of failure, The full project history is only stored in one central place.
- **Distributed SCS** - Version history is replicated on every user's machine, Users have version control all the time, Changes can be communicated between users, Git is distributed (Git, Mercurial). Pros: Commit changes/revert to an old version while offline, Commands run extremely fast because tool accesses the hard drive and not a remote server, Share changes with a few people before showing changes to everyone. Con: Long time to download, lot of disk space required to store all versions.
- Objects used by Git to implement source control
  - **Blobs**: Sequence of bytes
  - **Trees**: Groups blobs/trees together
  - **Commit**: Refers to a particular "git commit", Contains all information about the commit
  - **Tags**: A named commit object for convenience (e.g. versions of software)
  - Objects uniquely identified with hashes
- **Head** - Refers to commit object, many heads in repo. **HEAD** - refers to current head. **Branch** - refers to a head and its set of ancestor commits, Detached HEAD - commit not pointed to by a branch.
- **Git revert** - For reverting a commit, can itself be reverted. **Git rebase** - rewrites commit history, loses context,
- For creating human readable pointer: **git tag -a v1.0 -m 'Version 1.0'.** This will name the HEAD commit as v1.0
- **Backporting:** Taking a certain software modification (patch) and applying it to an older version of the software than it was initially created for.
- *git format-patch -1 [COMMIT NO] --stdout > quote-patch.txt*
- *patch -p1 < quote-patch.txt*
- **Gitk** - Repository browser, understand structure of repo, visualize commit graphs.

C Programming and Debugging

- malloc(size_t size): allocates a block of memory whose size is at least size.
- free(void *ptr): frees the block of memory pointed to by ptr
- realloc(void *ptr, size_t newSize) : Resizes allocated memory
- FILE *fopen(const char * restrict filename, const char * restrict mode);
- int fclose(FILE *fp);
- Reading/Writing characters
  - getc(FILE *fp);
  - putc(int c, FILE *pf);
- Reading/Writing Lines
  - char *fgets(char *buf, int n, FILE *fp);
  - int fputs(const char *s, FILE *fp);
- Format Specifiers: **%s → Strings, %d → Decimal integers, %f → floating points**
- Eg: fprintf(stdout, "%s has %d points.\n", player, score);

```
GDB COMMANDS FOR DEBUGGING:
: break <function_name> - set breakpoint at function_name
: s - step into function
: n - for next line of code
: list - for giving lines near
: bt - shows stack
: f - Gives name of current function
: finish - runs everything till exit
: c - continue
: q - quit


gdb --args ./src/ls -lt /tmp/mihirm1 /tmp/now1 /tmp/now2
gdb: break sort_file
gdb: run
breaks at sort_file()
gdb 'n' and 's' repeatedly.
gdp p for getting the values
gdp 'p' on qsort gives (int (*)(V, V)) 0x406e10 <compare_mtime>
<compare_mtime> looks like a function that compares based on time.
compare_mtime returns cmp_mtime.

I ran gdb again and set another breakpoint at cmp_mtime.
gdp: break cmp_mtime
gdb:p
Function timespec_cmp is called:
timespec_cmp (b=..., a=...) at ../lib/timespec.h
int diff = a.tv_sec - b.tv_sec;

Examined timespec_cmp in timespec.h.
The problem is that a.tv_sec - b.tv_sec gives a very large negative
value that causes an integer overflow. It can be easily fixed by assigning
diff 0 or 1 based on which is greater, a.tv_sec or b.tv_sec.

Made new directory coreutils-buggy and untared the buggy version into it.

The local file system on SEASnet is a 64 bit file system.
The tmp folder is NFS(Network File System) and uses a 32bit file system.
UNIX epoch time starts from 1970. Time is stored in a 64 bit system
as a combination of '+' or '-' sign and time elapsed since 1970.
When a time is truncated from 64bit to 32bit, the bit containing the
'-' sign is lost and it appears that the date is in the future.

```

- Pointers to functions: double (*func_ptr) (double, double);
- func_ptr = [&]pow; // func_ptr points to pow()
- // Call the function referenced by func_ptr: double result = (*func_ptr)(1.5, 2.0);
- // The same function call: result = func_ptr(1.5, 2.0);

SSH - Secure Shell

- What type of guarantees do we want?
  - Confidentiality - Message secrecy
  - Data integrity - Message consistency
  - Authentication - Identity confirmation
  - Authorization - Specifying access rights to resources
- Ciphertext - Encrypted message. Secret Key - Part of mathematical function used to encrypt/decrypt.
- **Symmetric Key Encryption** - Same key used for encryption/decryption.
- **Public Key Encryption (Asymetric):**
  - Uses pair of keys: Public - published, known to everyone. Private - Known to owner.
  - Public key used for encryption, Private key for decryption
  - Eg: RSA (Rivest, Shamir & Adelman) - Property used - Difficulty in factoring.
- Telnet: Remote access, not encrypted, packet sniffers can intercept sensitive info.
- **SSH** - run processes remotely, encrypted session, session key during session
- Client and server agree on a **symmetric encryption key** (session key). All messages sent between client and server. Encrypted at the sender with session key. Decrypted at the receiver with session key.
- Passwordless login with keys: Private/Public key used for authentication
  - **ssh-keygen** : Passphrase for protecting pvt. keys, needed to access keys.
  - **ssh-copy-id** username@ugrad.seas.ucla.edu - Copies public key to server.
  - **Ssh- agent:** authentication agent, manages private key. To avoid entering passphrase when key is used.
  - **Ssh add** - registers private key with the agent
- Digital Signature - extra data attached to document, ensure integrity.
- Message digest - shorter version of doc, generated during hashing, slight tampering in doc, will change digest.
- Steps for digital signature gen. : Sender - Gen. message digest, encrypt using private key(called digital sign. now), attach to message and send.
- Receiver - Decrypt digital sign using pub key, generate message digest by hashing,

Compare with the digest of the digital sign, if not same => tampered message.

```
Commands for setup:
Opened command line on Linux and ran:
    sudo apt-get update
    sudo apt-get install openssh-server
    sudo apt-get install openssh-client

Created new user for lab partner:
    sudo useradd -d /home/hung -m hung
    sudo passwd hung
    cd /home/hung
Created new directory:
    sudo mkdir .ssh
For changing ownership:
    sudo chown -R hung .ssh
    sudo chmod 700 .ssh
    (Flag 700: owner can read, write and execute)

Ran ifconfig for getting IP address:
My IP: 10.97.85.50
Gave this to partner.

Lab partner created a user for me.

Client steps:
Partner's IP: 10.97.85.26
ping server

Logging in:
    ssh-keygen - generated a public/private key pair
    ssh-copy-id -i mathur@10.97.85.26 - For copying key to partner's server
    ssh -add - adds identity to the authentication agent

ssh mathur@10.97.85.26

Running applications:
    ssh -X mathur@10.97.85.26 - connect with X-Forwarding
    Ran commands xterm, gedit, firefox
```

--gen key	generating new keys
--armor	ASCII format
--export	exporting public key
--import	import public key
--detach-sign	creates a file with just the signature
--verify	verify signature with a public key
--encrypt	encrypt document
--decrypt	decrypt document
--list-keys	list all keys in the keyring
--send-keys	register key with a public server/-keyserver option
--search-keys	search for someone's key

System Call Programming

- **Processor Modes**: Mode bit used to distinguish between execution on behalf of OS and on behalf of user.
- **Supervisor mode**: processor executes every instruction in it's hardware repertoire.
- **User mode:** can only use a subset of instructions.
- Mode bit may define areas of memory to be used when the processor is in supervisor mode vs user mode
- The **kernel** executes privileged operations on behalf of untrusted user processes
- **Kernel**: Code of OS executing in supervisor state. Trusted software:manages hardware resources (CPU, memory, and I/O). Implements protection mechanisms that could not be changed through actions of untrusted software in user space.
- **System call interface** is a safe way to expos privileged functionality and services of the processor
- Instructions can be executed in supervisor mode are supervisor privileges, or protection instruction
  - **I/O instructions** are protected. If an application needs to do I/O, it needs to get the OS to do it on it's behalf
  - Instructions that can change the protection state of the system are privileges (e.g. process' authorization status, pointers to resources, etc).
- Instructions can be executed in supervisor mode are supervisor privileges, or protection instruction.
- System Call is executed by the OS, performs a simple operation.
- **System call** causes a switch from user mode to kernel mode. It involves:
  - The system call causes a 'trap' that interrupts the execution of the user process (user mode)
  - The kernel takes control of the processor(kernel mode\privilege switch)
  - The kernel executes the system call on behalf of the user process
  - The user process gets back control of the processor (user mode\privilege switch)

System calls

- `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

More examples: System calls

```
struct stat {
dev_t     st_dev;         /* ID of device containing file */
ino_t     st_ino;         /* inode number */
mode_t    st_mode;        /* protection */
nlink_t   st_nlink;       /* number of hard links */
uid_t     st_uid;         /* user ID of owner */
gid_t     st_gid;         /* group ID of owner */
dev_t     st_rdev;        /* device ID (if special file) */
off_t     st_size;        /* total size, in bytes */
blksize_t st_blksize;     /* blocksize for filesystem I/O */
blkcnt_t  st_blocks;      /* number of 512B blocks allocated */

time_t st_atime;  /* time of last access */
time_t st_mtime;  /* time of last modification */
time_t st_ctime;  /* time of last status change */
};
```

- **pid_t getpid(void)**: returns the process id of the calling process
- **int dup(int fd)**:Duplicates a file descriptor fd. Returns a second file descriptor that points to the same file table entry as fd does.
- **int fstat(int filedes, struct stat *buf)**: Returns information about the file with the descriptor filedes to buf
- 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. These functions perform privileged operations by making system calls. Equivalent lib. Functions make less sys calls=> less overhead.

A buffer, also called buffer memory, is a portion of a computer's memory that is set aside as a temporary holding place for data that is being sent to or received from an external device, such as a hard disk drive (HDD), keyboard or printer.

- **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
- Buffered output improves I/O performance and can reduce system calls.
- Unbuffered output when you want to ensure that the output has been written before continuing.
- stderr under a C runtime library is unbuffered by default. Errors are infrequent, but want to know about them immediately.
- stdout is buffered because it's assumed there will be far more data going through it. logging: log messages of a process.
- Library calls
  - executed in the user program
  - may perform several task
  - may call system calls

System calls vs library call conventions

- Library functions often return pointers
  - `FILE *fp = fopen("cs35l","r")`
  - `NULL for return for failure`
- System calls usually return an integer
  - `int res=system_call_function(a_few_args)`
  - Where the return value
    - res >= 0 → all is well
    - res < 0 → failure
    - See the global variable `errorno` for more info

Reminder of how System calls work

1. program get to the system call in the user's code
   int res = sys_call(a_few_params)
2. puts the parameters on the stack
3. performs a system 'trap' -- hardware switch
   now in system mode
4. operating system code may copy large data structures into system memory
5. starts operation…
6. operation complete!
7. if necessary copies result data structures back to user program's memory
   return to user mode
8. user program puts return code into res(the return value from the system call)
9. program recommences

System calls

- `int res = ssize_t read(int fildes, void *buf, size_t nbyte)`
  - fildes: file descriptor
  - buf: buffer to write to, is not NULL terminated
  - nbyte: number of bytes to read
  - res: is 0 at end of file, negative for error
- `int res = ssize_t write(int fildes,const void *buf,size_t nbyte)`
  - fildes: file descriptor
  - buf: buffer to write to, need not be NULL terminated
  - nbyte: number of bytes to write
    - res might be less than nbyte if OS buffers full
    - should check and repeat until all gone
  - res: is 0 for end of file, negative for error

`time` and `strace`

- **time** [*options*] *command* [*arguments...*]
- Output:
  - real 0m4.866s: elapsed time as read from a wall clock
  - user 0m0.001s: the CPU time used by your process
  - sys 0m0.021s: the CPU time used by the system on behalf of your process
- **strace**: intercepts and prints out system calls to stderr or to an output file
  - $ strace –o strace_output ./tr2b 'AB' 'XY' < input.txt
  - $ strace –o strace_output2 ./tr2u 'AB' 'XY' < input.txt

- Eg. strace usage: strace -o strace_output1 ./tr2b ABC XYZ < book.txt
- For printing calls and time: strace -c ./tr2b ABC XYZ < book.txt
- Time: time cat book.txt | ./tr2b abcdef ghijkl > book_tr2b.txt

Multithreading/Parallel Processing

- **Multitasking**: Run multiple processes simultaneously to increase performance
- Processes do not share internal structures (stacks,globals,etc)
- Communicate via **IPC (inter-process communication)** methods. Pipes, sockets, signals, message queues
- **Single core**: Illusion of parallelism by switching processes quickly (time-sharing). Why is illusion good?
- **Multi-core**: True parallelism. Multiple processes execute concurrently on different CPU cores.
- 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 the following with other threads belonging to the same process:Code, Global Data, OS resources (files,I/O).
- **Time Sharing:** Illusion of multithreaded parallelism. Thread switching has less overhead than process switching.
- Multithreading properties:
  - Efficient way to parallelize tasks

- Thread switches are less expensive compared to process switches (context switching)
- Inter-thread communication is easy, via shared global data
- Need synchronization among threads accessing same data

Pthread API

```
#include <pthread.h>
```

- `int pthread_create(pthread_t *thread,`
  `const pthread_attr_t *attr,void*`
  `(*thread_function) (void*), void *arg);`
  - Returns 0 on success, otherwise returns non-zero number
- `void pthread_exit(void *retval);`
- `int pthread_join(pthread_t thread, void **retval);`
  - Returns 0 on success, otherwise returns non zero error number

```
#include<pthread.h>  //Compile the following code as - gcc main.c -lpthread
#include<stdio.h>
void* ThreadFunction(void *arg) {
  long tID = (long)arg;
  printf("Inside thread function with ID = %ld\n", tID); pthread_exit(0);}

int main(int argc, char *argv[]) {
  const int nthreads = 5; pthread_t threadID[nthreads]; long t;
  for(t = 0; t < nthreads; ++t) {
    int rs = pthread_create(&threadID[t], 0, ThreadFunction, (void*)t);
    if(rs) {
      fprintf(stderr, "Error creating thread\n");
      return -1; }}
  printf("Main thread finished creating threads\n");
  for(t = 0; t < nthreads; ++t) {
    void *retVal;
    int rs = pthread_join(threadID[t], &retVal);
    if(rs) {
      fprintf(stderr, "Error joining thread\n");
      return -1;
  }}
  printf("Main thread finished execution!\n");
  return 0; }
```

Pthread API

- **Thread safe function** - safe to be called by multiple threads at the same time. Function is free of ‘race conditions’ when called by multiple threads simultaneously.
- **Race condition** - the output depends on the order of execution
- For thread synchronization - **Mutex: Mutual Exclusion**. Only one thread will get the mutex. Other thread will block in Mutex.lock(). Other thread can start execution only when the thread that holds the mutex calls Mutex.unlock().
- **Ray Tracing.** Powerful rendering technique in Computer Graphics
  - Yields high quality rendering
  - Suited for scenes with complex light interactions
  - Visually realistic
- Trace the path of light in the scene
- Computationally expensive
  - Not suited for real-time rendering (e.g. games)
  - Suited for rendering high quality pictures (e.g. movies)
- Embarrassingly parallel
  - Good candidate for multi-threading
  - Threads need not synchronize with each other, because each thread works on a different pixel.

Since ray-tracing is an Embarrassingly Parallel problem, there is no communication between threads and workload can be divided equally.

Commands From Lab:
$ od -An -t f -N 80000000 < /dev/urandom | tr -s ' ' '\n' > random.txt
-An: specifies that input offset base should not be written
-t f: specifies that we need double-precision floating point numbers.
-N : specifies bytes of input, double is 8 bytes, so we need
80,000,000 bytes for 10,000,000 doubles.

$ time -p sort -g --parallel=8 random.txt > /dev/null

As the number of threads increase, the speed increases and time of execution decreases.

pthread_join Example

```
#include <pthread.h> …
#define NUM_THREADS 5

void *PrintHello(void *thread_ num) {
      printf("\n%d: Hello World!\n", (int) thread_num); }

int main() {
      pthread_t threads[NUM_THREADS];
      int ret, t;
      for(t = 0; t < NUM_THREADS; t++) {
            printf("Creating thread %d\n", t);
            ret = pthread_create(&threads[t], NULL, PrintHello, (void *) t);
            // check return value }

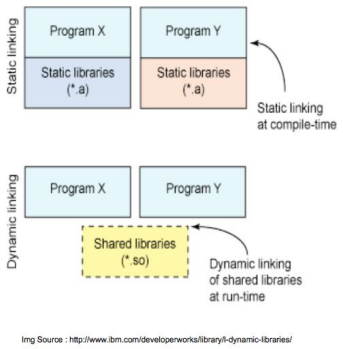
       for(t = 0; t < NUM_THREADS; t++) {
            ret =  pthread_join(threads[t], NULL);
            // check return value }
```

Dynamic Linking

- Linker: Adjusts any memory references to fit the OS's memory model.
- Compilation Process:
- **Preprocessor**
  - Expand Header includes, macros, etc
  - -E option in gcc to show the resulting code
- **Compiler**
  - Generates machine code for certain architecture
- **Linker**
  - Linkall modules together
  - Address resolution
- **Loader**
  - Loads the executable to memory to start execution

Linux Libraries

- **Static Library**
  - Statically linked
  - Every program has its own copy
  - More space in memory
  - Tied to a specific version of the lib. New version of the lib requires recompile of source code.
- **Shared Library (binding at run-time)**
  - Dynamically loaded/linking
    - **Dynamic Linking** – The OS loads the library when needed. A dynamic linker does the linking for the symbol used.
    - **Dynamic Loading** – The program "actively" loads the library it needs (DL API – dlopen(), dlclose()). More control to the program at run-time. Permits extension of programs to have new functionality.
  - Library is shared by multiple programs
  - Lower memory footprint
  - New version of the lib does not require a recompile of source code using the lib



Img Source : http://www.ibm.com/developerworks/library/l-dynamic-libraries/

- **Static Linking:** Carried out only once to produce an executable file. If static libraries are called, the linker will copy all the modules referenced by the program to the executable. Static libraries are typically denoted by the .a file extension
- **Dynamic Linking:** Allows a process to add, remove, replace or relocate object modules during its execution.
- **Ldd** - Print shared object dependencies.
- If shared libraries are called: Only copy a little reference information when the executable file is created. Complete the linking during loading time or running time
- Dynamic libraries are typically denoted by the .so file extension .dll on Windows.
- Linker collects procedures and links them together object modules into one executable program.
- Why isn't everything written as just one big program, saving the necessity of linking?
- Efficiency: if just one function is changed in a 100K line program, why recompile the whole program? Just recompile the one function and relink. Multiple-language programs.
- Unix systems: Code is typically compiled as a **dynamic shared object (DSO).**
- **Advantages of Dynamic Linking:** The executable is typically smaller. When the library is changed, the code that references it does not usually need to be recompiled.The executable accesses the .so at run time; therefore, multiple programs can access the same .so at the same time. Memory footprint amortized across all programs using the same .so
- **Disadvantages of Dynamic Linking:** Performance hit. Need to load shared objects (at least once). Need to resolve addresses (once or every time). What if the necessary dynamic library is missing? What if we have the library, but it is the wrong version?
- Attributes of functions: Used to declare certain things about functions called in your program. Help the compiler optimize calls and check code. Also used to control memory placement, code generation options or call/return conventions within the function being annotated. Introduced by the **attribute** keyword on a declaration, followed by an attribute specification inside double parentheses
- __attribute__ ((__constructor__)) : Is run when dlopen() is called
- __attribute__ ((__destructor__)): Is run when dlclose() is called
- Example:

  __attribute__ ((__constructor__))

  void to_run_before (void) {

```
        printf("pre_func\n");
}
```

Table 1. The DI API

Function	Description
dlopen	Makes an object file accessible to a program
dlsym	Obtains the address of a symbol within a `dlopen`ed object file
dlerror	Returns a string error of the last error that occurred
dlclose	Closes an object file

```c
#include <stdio.h>
#include <dlfcn.h>

int main(int argc, char* argv[]) {
  int i = 10;
  void (*myfunc)(int *); void *dl_handle;
  char *error;
  dl_handle = dlopen("libmymath.so", RTLD_LAZY);//RTLD_NOW
  if(!dl_handle) {
    printf("dlopen() error - %s\n", dlerror()); return 1;
  }
  //Calling mul5(&i);
  myfunc = dlsym(dl_handle, "mul5"); error = dlerror();
  if(error != NULL) {
    printf("dlsym mul5 error - %s\n", error); return 1;
  }
  myfunc(&i);
  //Calling add1(&i);
  myfunc = dlsym(dl_handle, "add1"); error = dlerror();
  if(error != NULL) {
    printf("dlsym add1 error - %s\n", error); return 1;
  }
  myfunc(&i);
  printf("i = %d\n", i);
  dlclose(dl_handle);
  return 0;
}
```

GCC options

- -c : compile and create object files
- -o : name of output file
- -I (upper case i) : Additional folders to search for header files
- -L : Additional folders to search for libraries to link with
- -shared : create shared library
- -l (lower case 'ell') : Name of additional library to link with
- -fpic : Output position independent code. This is required for shared libraries (generating relative addresses instead of absolute addresses)

GCC Flags

- `-fPIC`: Compiler directive to output position independent code, a characteristic required by shared libraries.
- `-lXXX`: Link with "`libXXX.so`"
  - Without `-L` to directly specify the path, `/usr/lib` is used.
- `-L`: At **compile** time, find `.so` from this path.
- `-Wl,rpath=.`: `-Wl` passes options to linker. `-rpath` at **runtime** finds `.so` from this path.
- `-c`: Generate object code from c code.
- `-shared`: Produce a shared object which can then be linked with other objects to form an executable.