```
* Program and Process
* compile and link
* build process
* Definition vs Declaration of fuctions
* make / Makefile
* git
    * git init, git status, git log, git add, git commit

* Sections
    * .text
    * .data
    * .bss
    * stack
    * heap

* Modular programs
    * Libraries
        * Standard libraries (linux - libc.a and libc.so)
        * User libraries/ 3rd party library
    * Two types
        * Static => *.a
        * Dynamic (Shared Objects) => *.so

* Static Library
    * ar x (extract)
    * ar crv libname.a 1.o 2.o..... n.o => (*.o)
    * lib function definitoin embedded in binary
    * multiple copies of library function loaded in the memory
    * more memory
* Dynamic Library
    * gcc -o libname.so -shared -fPIC
    * lib function deference definitoin embedded in binary
    * single copy of library function loaded in the memory
    * less memory


* PID, PPID, UID
* getpid(), getppid()
* man pages


* fork - create a new process
    * parent and child relationship
    * parent and child have their own address space (text, data, bss, stack and
heap)
    *

* Pseudo parallelism
* True parallelism

* ltrace
* strace
* Program using system call -> write a string inside a file.

* Library
    * User space
    * May be buffered I/O
    * Formatted I/O
* System
    * Kernel space
    * Not buffered I/O
    * No formatted I/O


* open - O_CREAT
* int - file descriptor -
```

```
    * fd - represents an open file in the kernel
    * 0 - standard input
    * 1 - standard output
    * 2 - standard error


Process states:
    * Create
    * Ready (multiple processes)
    * Running (one process -> uniprocessor)
    * Waiting (multiple processes)
    * Destroy

* Scheduling algorithm -> choose the next process that has to execute
* Context switch -> Context Saving (Current process) + Scheduling (Choose the new
process) + Context Restoring (New Process)
* Context -> PC, SP, GPR, Flags -> Hardware Registers -> Uniprocessor (1 copy)

* Blocking -> Process may go into a waiting state
* Non Blocking call -> never block -> immediately return back after doing
functionality

Inter process communication (IPC)
P1 -> P2

* Pipes
    * IPC
    * Unidirectional
    * Related processes (Parent and Child)
    * pipe -> two integers (file descriptors)
    * 0 -> Reading
    * 1 -> Writing
    * Child will inherit file descriptors


* FIFOs
    * IPC
    * Unidirectional
    * Unrelated processes
    * fifos aka named pipes
    * mkfifo (command and API)


* Write a program which does the following:
    * P1: Get two integer inputs
    * P1: Send it to P2
    * P2: Recv two integers
    * P2: Add two integers
    * P2: Send to P1
    * P1: Print the result

* Batch
* Multi programming
* Multi tasking
* Multi processing
* Multi threading
* Multi user

* Design
    * Monolithic Approach
        * eg. Linux
        * Single address space
        * Sharing of information easy -> in same address space
    * Micro Kernel Approach
        * eg. QNX, Minix
        * Multiple address space
```

```
            * System Processes -> Priviledged eg. Networking stack, DD
            * User Processes -> Non-Priv -> MP3 player, Editor
        * Message Queues used for information sharing
* Debate Linus Torvalds and Tannenbaum
* Threads
    * POSIX
    * Library pthread
    * pthread_t
    * pthread_attr_t -> Attributes
        * Joinable - detachable
        * Scheduling policy
        * priority
    * Each thread has its own stack
    * Threads share -> .text, .data, .bss, heap

* Race condition
    * shared data
    * thread accessing shared data -> race condition

* Mutual Exclusion (pthread_mutex_t)
    * Mutex
        * Futex
        * Recursive Mutex
        * Adaptive Mutex
        * Error Checking Mutex (Assignment)

* Semaphore (sem_t)
    * Binary
        * Mutual Exclusion -> initial value -> 1
        * Signalling
    * Counting

/*------------------------------*/
Case 1:
* Create - Webinterface - repo
* Add address of the repo
* git push
* git add
* git commit
* git push

Case 2:
* Existing repo - Webinterface
* git clone GITRepoURL
* Copy your source code to this directory
* git add
* git commit
* git push

Case 3:
* git pull
 ** git add
 ** git commit
* git push
```