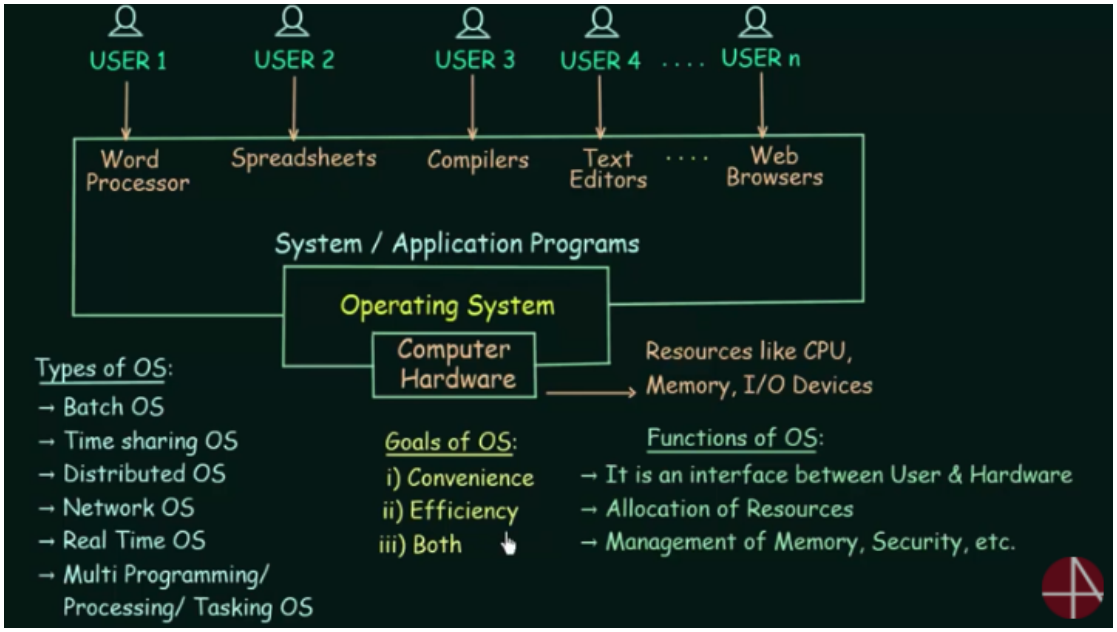
CSC369 Week 1 Notes

Hyungmo Gu

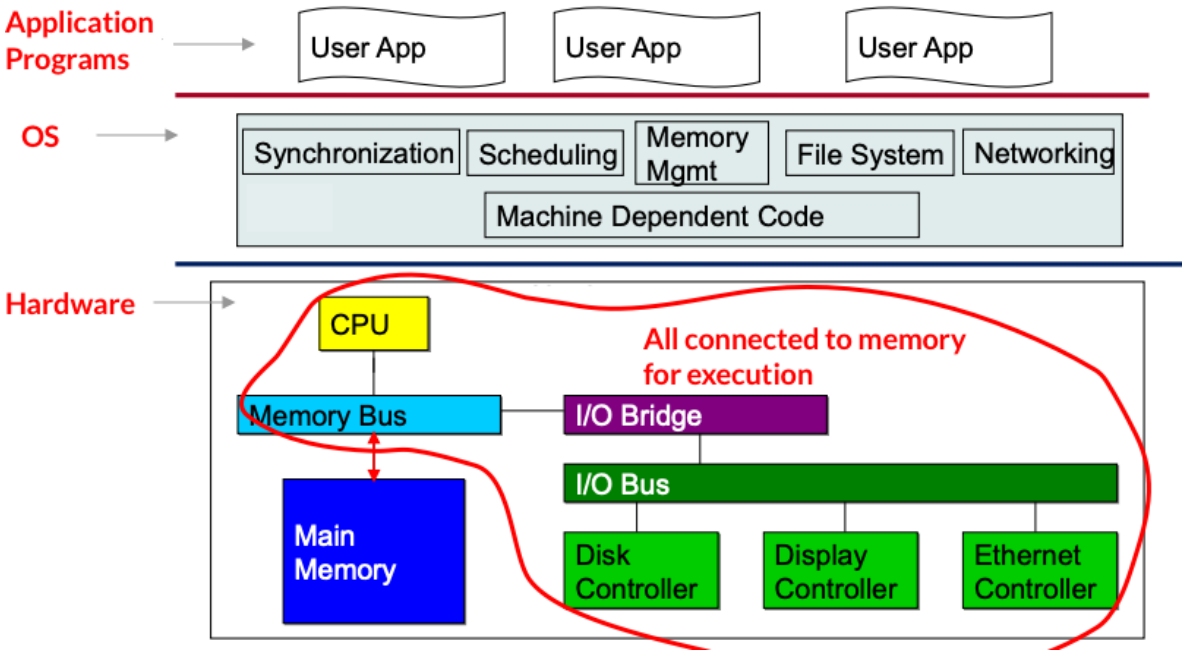
September 11, 2020

1 Intro to OS

- What is Operating System
  - is the program that manages the computer hardware
  - is the software layer between user applications and hardware
  - is used for
    - Allication of resources
    - Management of memory, security, etc.

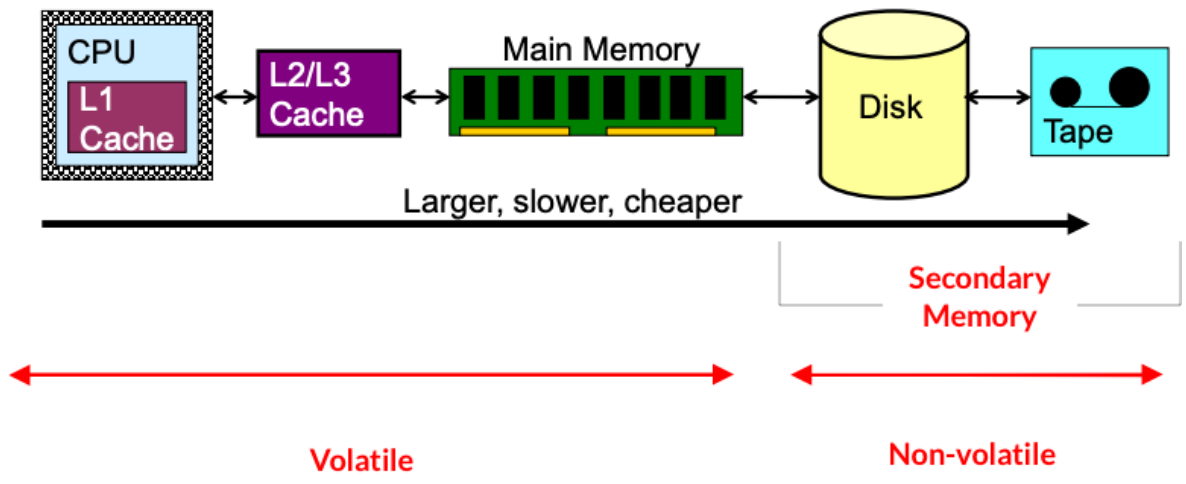


- Overview of Computer System



  - All hardware devices are connected through common **bus** and are loaded to memory for execution.
  - **Synchronization:** to ensure orderly acces to the shared memory

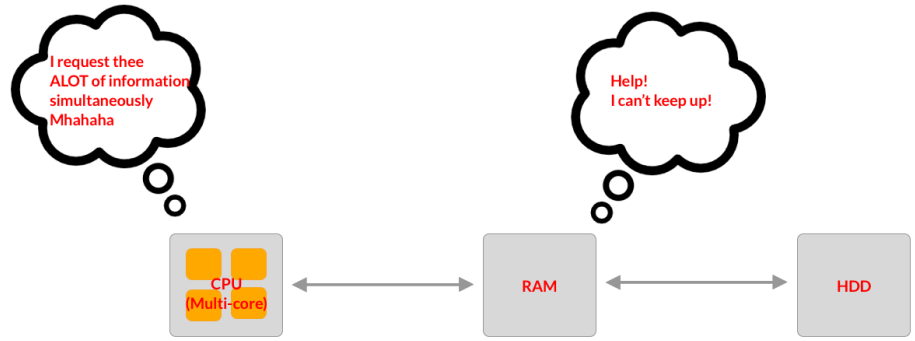
- Storge Hierarchy / Storage Structure

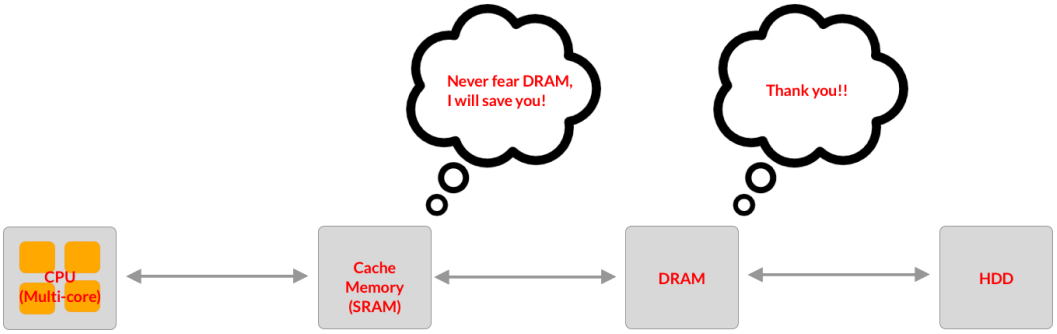


  - **Volatile** → Loses contents when power is removed
  - **Non-volatile** → Retains contents even when power is removed

- Caching / Cache Memory

  - Is also called **Static Random Access Memory (SRAM)**
  - Is more costly
  - Hides performance differences when large access-time gap exsists between two levels
    * Quad-quare requesting RAM for information





  - More can be found here

- Concurrency
  - Is execution of several instruction sequences at same time
    * i.e, CPU and device controllers
  - **Interrupt:** are signals sent to the CPU by external devices, (usually I/O devices)
    * It's like telling 'Hey CPU, please stop this process, and do y instead, since this is more important'
    * i.e. Network Packet has arrived, Disk I/O comeplete occured
  - **System Call:** are interrupt signals sent by software
    * Is a programmatic way of a program requesting for service to kernel of operating system
    * i.e. Accessing a hard-disk drive
  - IMPORTANT: An operating system is an event-driven program.

2 Process Threads

- Part 1: The Process Concept
  - **Process:** is a program in execution
  - **Threads:** is the unit of execution within a process.

$$\text{Thread} = \frac{\text{Job}}{\text{Unit of Work}} \tag{1}$$

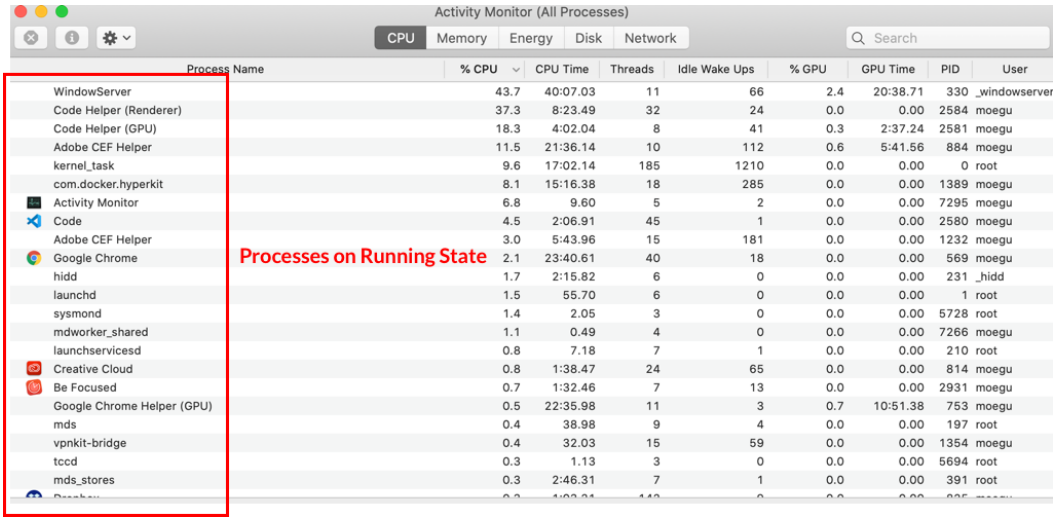
    - A process can have anywhere from one thread to many threads

- Process Data Structure (PCB)
  - Is called Process Control Block
  - Is OS data structur representing each process
  - Generally includes
    1. Process State
       - (Ready, running, blocked)
    2. Program Counter
       - Is an address that indicates the line of code that has to be executed next
       - i.e. the next line of code i need to execute is line 2 :)

```
print("Hello World");
print("Hi World!") //<- Line 2
```

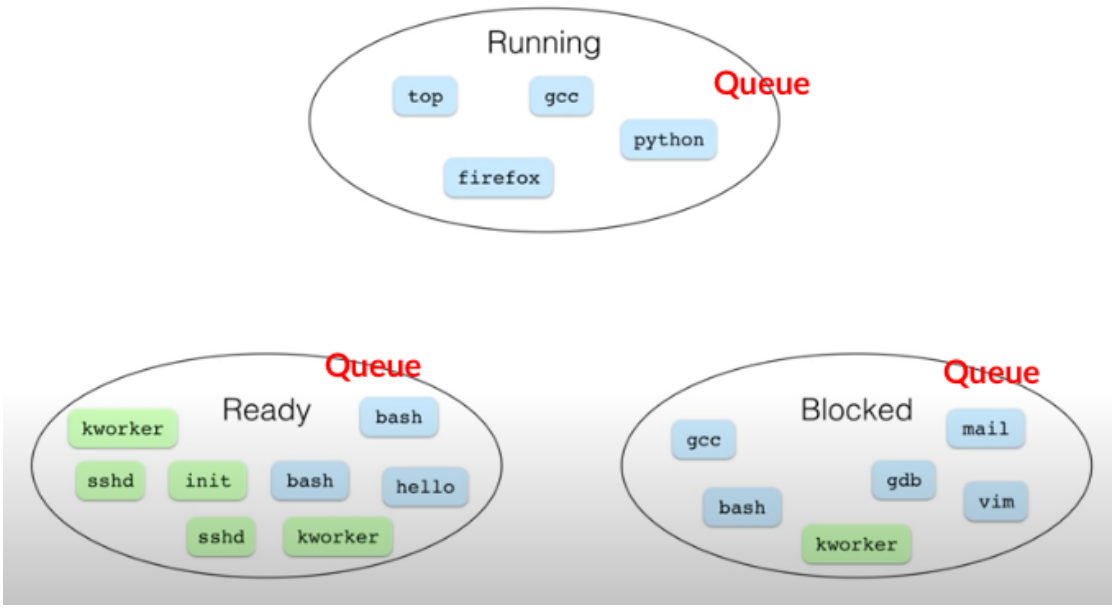
    3. CPU Register ***Need to come back*
    4. CPU Scheduling Information
       - Priority of process
       - Higher the priority → executed first
    5. Memory Management ***Need to come back*
    6. I/O Status Information
       - Is list of input output devices assigned to this process
       - Is used during execution
       - i.e. Sound, Mouse, Keyboard

- State Queues
  - Is a part of **process scheduling**
    - keeps CPU busy at all times to deliver minimum response time for all programs



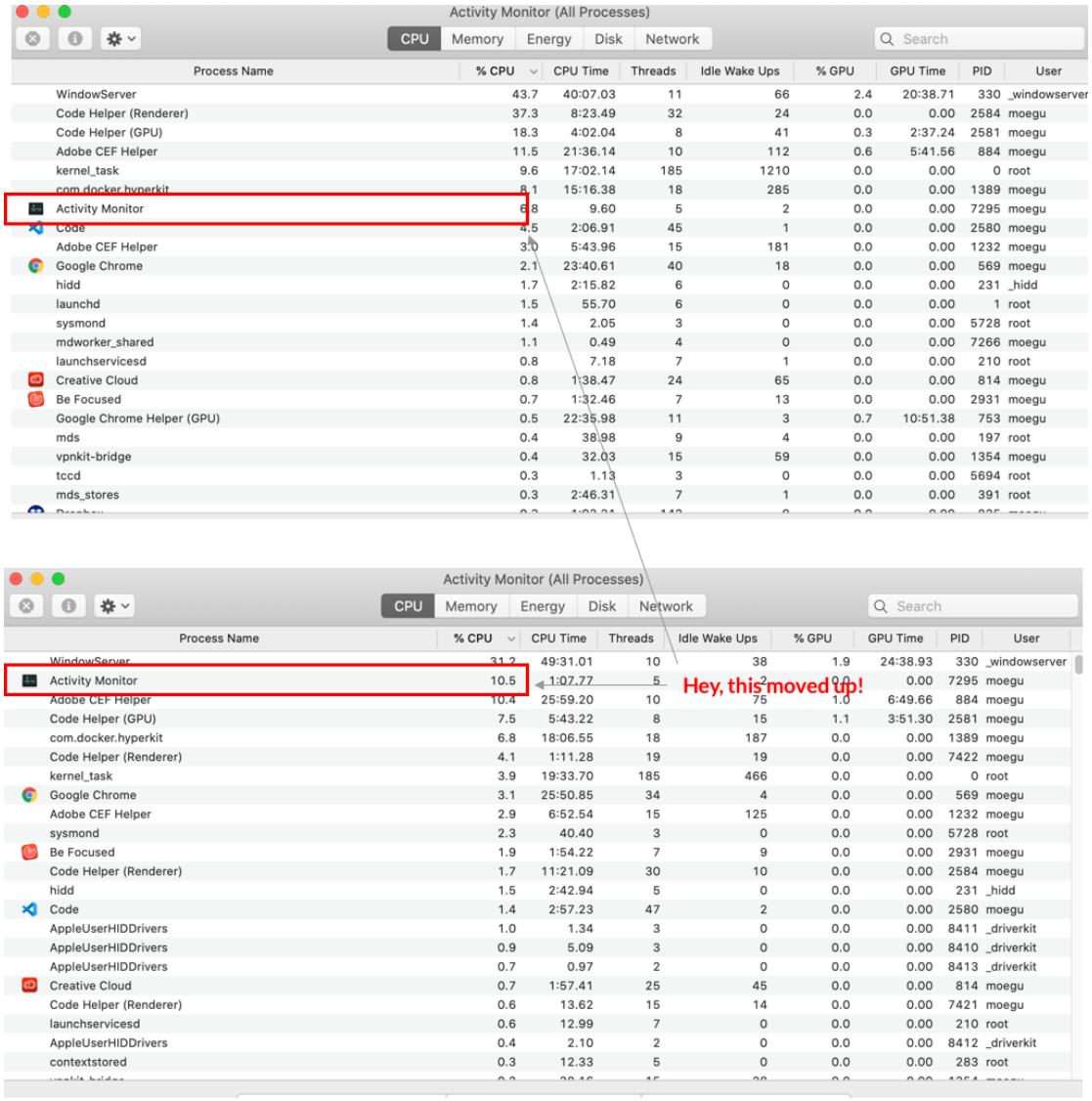
* Here, processes in queue are switched so frequently that user can interact with each program simultaneously while running
  - i.e, listening to music, typing and downloading a picture of a cute puppy all at the same time

– Has one state queue for each process state
  * Job Queue, Ready Queue, Waiting Queue, Blocking Queue

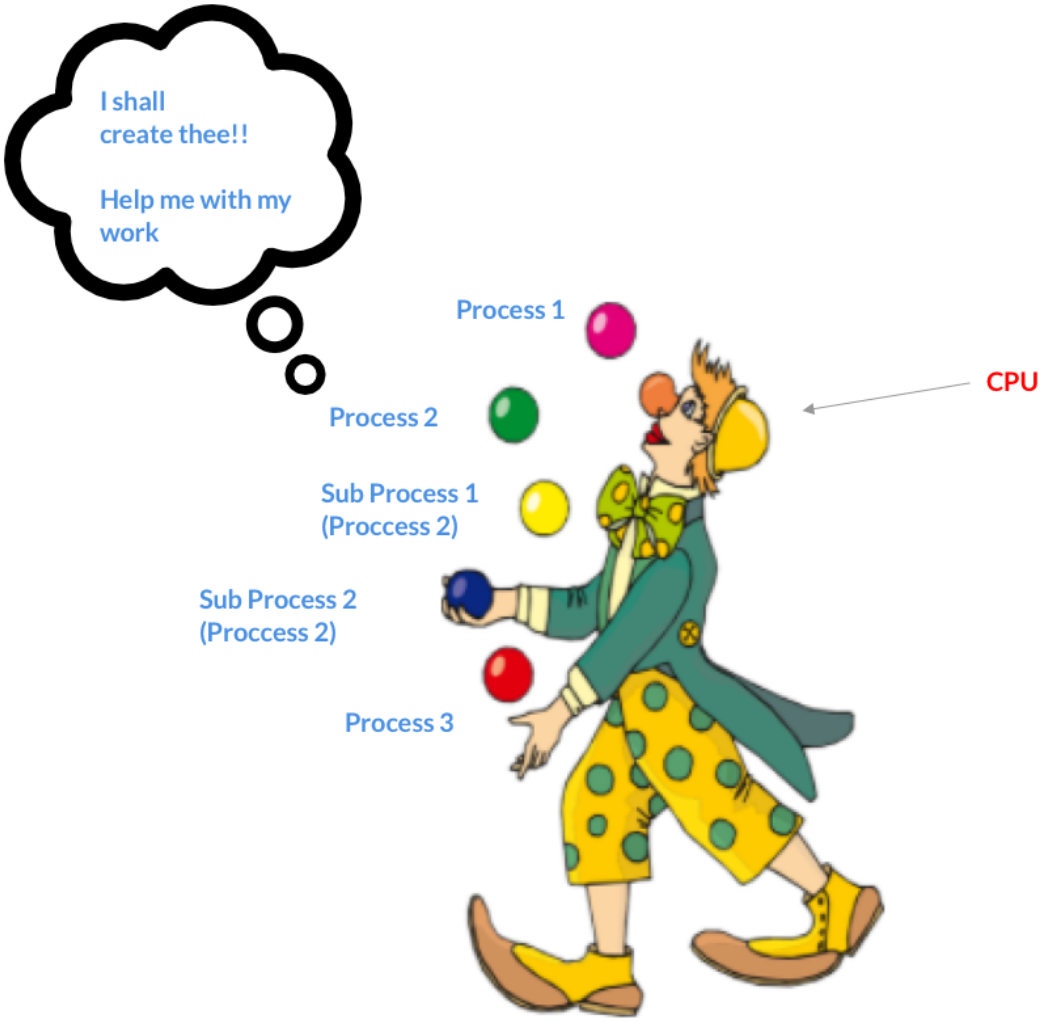


- PCBs And State Queues
  – Process created → OS allocates PCB → Initializes it → Places it on Ready Queue
  – Process terminated → PCB deallocated
- Context Switch

- Switches the CPU to another process on interrupt, saving the state of the old process and loading the saved state of the new process until done
  - i.e. Loading a music in Google Music Player
- Previous process resumes executing when done



- Operations on Processes
  - Process can have multiple new processes during the course of execution, or alone
  - Process execute concurrently and must be created and deleted dynamically
    - Wow, CPU is a master juggler



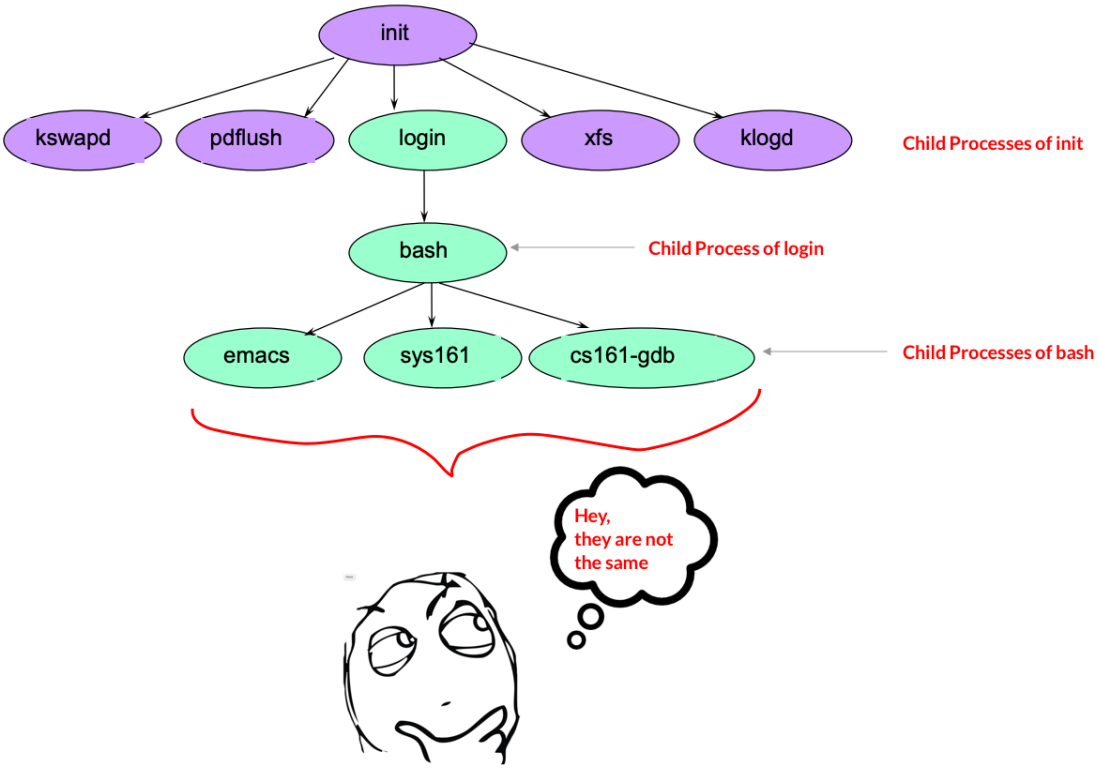
  - Has two types of operations
    - Process Creation
    - Process Termination

- Process Creation
  - Is an operation that creates new processes by another process
    - Child process → New Process
    - Parent Process → Creating Process
  - Is achieved through **fork()** system call
  - Occurs in
    1. System Initialization
    2. A running process
    3. A user request
    4. Initialization of a bath job
  - Two possibilities exist on creation:

1. Parent executes with children in parallel
2. Parent waits until some or all of its children are terminated

– More can be found here

- Linux Tree



–

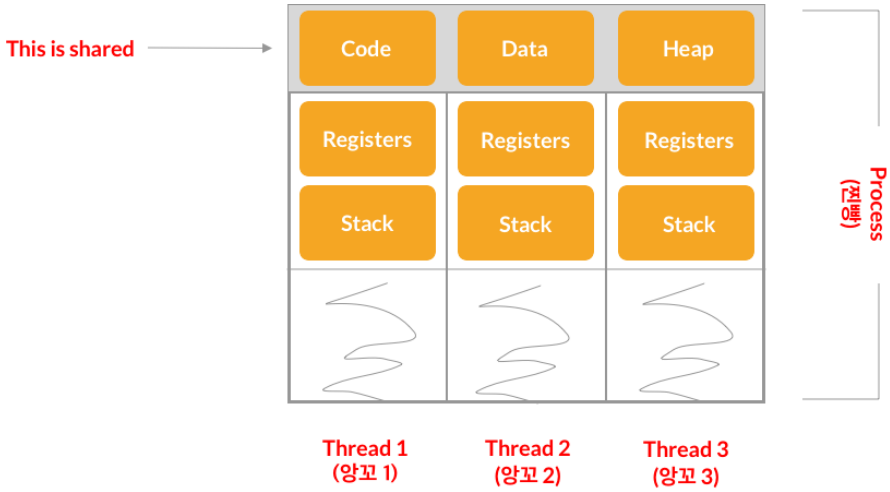
There are two possibilities of in terms of the address space of the new process

1. Child process is the duplicate of the parent process
2. Child process has new program loaded onto it
   * i.e. Unix Shell

```
while (1) {
  char *cmd = read_command();
  int child_pid = fork();
  if (child_pid == 0) {
    exec(cmd);
    } else {
    wait(child_pid);
  }
}
```

By this guy :)

- Threads
  - Thread
    - Is Basic unit of execution / CPU utilization
    - Is bound to a single process
    - Is composed of
      1. Thread ID
      2. Program counter
      3. Register set
      4. Stack
    - multiple threads → Can process more tha one task at a time → Faster
    - Key idea: separate the concept of process from its exeuction state
      - Process: Address space, privileges, resources, etc.
      - Execution State (thread / thread of control): PC, SP, registers
    - It's like 앙꼬만 있는 찐빵, or eggs with only yorks.
  - Process
    - is program under execution
    - Has same code, data (address space), previleges, and resources (files sockets, etc) shared by threads
    - is container in which threads execute
    - is like 찐빵, or egg shell with only egg whites



  - More can be found here

- Cooperating Process
  - **Independent:** if it cannot affect or be affected by the other processes executing in the system
    - Independent process → No data sharing

  - **Cooperating** if is not independent
    - i.e. Threads
    - Cooperating Process → communicates and actions synchronized

- Interprocess Communicator
  - Allows communication between cooperating processes
  - Are done via
    1. Shared memory (i.e. fork())
    2. Message passing
       - send(P, msg) - Send msg to process P
       - receive(Q, msg) - Receive msg from process Q