Resident Monitor

- Early (oldest) OS resides in memory and monitor execution of the programs. If it fails, error is reported.
- OS provides hardware interfacing that can be reused by all the programs.

Batch Systems

- The batch/group of similar programs is loaded in the computer, from which OS loads one program in the memory and execute it. The programs are executed one after another.
- In this case, if any process is performing IO, CPU will wait for that process and hence not utilized efficiently.

Multi-Programming

- In multi-programming systems, multiple program can be loaded in the memory.
- The number of program that can be loaded in the memory at the same time, is called as "degree of multi-programming".
- In these systems, if one of the process is performing IO, CPU can continue execution of another program. This will increase CPU utilization.
- Each process will spend some time for CPU computation (CPU burst) and some time for IO (IO burst).
    - If CPU burst > IO burst, then process is called as "CPU bound".
    - If IO burst > CPU burst, then process is called as "IO bound".
- To efficiently utilize CPU, a good mix of CPU bound and IO bound processes should be loaded into memory. This task is performed by an unit of OS called as "Job scheduler" OR "Long term scheduler".
- If multiple programs are loaded into the RAM by job scheduler, then one of process need to be executed (dispatched) on the CPU. This selection is done by another unit of OS called as "CPU scheduler" OR "Short term scheduler".

Multi-tasking OR time-sharing

- CPU time is shared amoung multiple processes in the main memory is called as "multi-tasking".
- In such system, a small amount of CPU time is given to each process repeatedly, so that response time for any process < 1 sec.
- With this mechanism, multiple tasks (ready for execution) can execute concurrently.
- There are two types of multi-tasking:
    - Process based multitasking: Multiple independent processes are executing concurrently. Processes running on multiple processors called as "multi-processing".
    - Thread based multi-tasking OR multi-threading: Multiple parts/functions in a process are executing concurrently.

Thread concept

- Threads are used to execute multiple tasks concurrently in the same program/process.
- Thread is a light-weight process.
    - For each thread new control block and stack is created. Other sections (text, data, heap, ...) are shared with the parent process.
    - Inter-thread communication is much faster than inter-process communication.
    - Context switch between two threads in the same process is faster.

- Thread stack is used to create function activation records of the functions called/executed by the thread.

Process vs Thread

- In modern OS, process is a container holding resources required for execution, while thread is unit of execution/scheduling.
- Process holds resources like memory, open files, IPC (e.g. signal table, shared memory, pipe, etc.).
- PCB contains resources information like pid, exit status, open files, signals/ipc, memory info, etc.
- CPU time is allocated to the threads. Thread is unit of execution.
- TCB contains execution information like tid, scheduling info (priority, sched algo, time left, ...), Execution context, Kernel stack, etc.
- terminal> ps -e -o pid,nlwp,cmd
- terminal> ps -e -m -o pid,tid,nlwp

main thread

- For each process one thread is created by default called as main thread.
- The main thread executes entry-point function of the process.
- The main thread use the process stack.
- When main thread is terminated, the process is terminated.
- When a process is terminated, all threads in the process are terminated.

Multi-user

- Multiple users can execute multiple tasks concurrently on the same systems. e.g. IBM 360, UNIX, Windows Servers, etc.
- Each user can access system via different terminal.
- There are many UNIX commands to track users and terminals.
    - tty, who, who am i, whoami, w

Process

- Process is program in execution.
- Process has multiple sections i.e. text, data, rodata, heap, stack. ... into user space and its metadata is stored into kernel space in form of PCB struct.
- PCB contains
    - id, exit status,
    - scheduling info (state, priority, time left, scheduling policy, ...),
    - files info (current directory, root directory, open file descriptor table, ...),
    - memory information (base & limit, segment table, or page table),
    - ipc information (signals, ...),
    - execution context, kernel stack, ...

OS Data Structures:

- Job queue / Process table: PCBs of all processes in the system are maintained here.
- Ready queue: PCBs of all processes ready for the CPU execution and kept here.

- Waiting queue: Each IO device is associated with its waiting queue and processes waiting for that IO device will be kept in that queue

Process Life Cycle

Process States

- New
  - New process PCB is created and added into job queue. PCB is initialized and process get ready for execution.
- Ready
  - The ready process is added into the ready queue. Scheduler pick a process for scheduling from ready queue and dispatch it on CPU.
- Running
  - The process runs on CPU. If process keeps running on CPU, the timer interrupt is used to forcibly put it into ready state and allocate CPU time to other process.
- Waiting
  - If running process request for IO device, the process waits for completion of the IO. The waiting state is also called as sleeping or blocked state.
- Terminated
  - If running process exits, it is terminated.

Types of Scheduling

Non-preemptive

- The current process gives up CPU voluntarily (for IO, terminate or yield).
- Then CPU scheduler picks next process for the execution.
- If each process yields CPU so that other process can get CPU for the execution, it is referred as "Co-operative scheduling".

Preemptive

- The current process may give up CPU voluntarily or paused forcibly (for high priority process or upon completion of its time quantum)

Scheduling criteria's

CPU utilization: Ideal - max

- On server systems, CPU utilization should be more than 90%.
- On desktop systems, CPU utilization should around 70%.

Throughput: Ideal - max

- The amount of work done in unit time.

Waiting time: Ideal - min

- Time spent by the process in the ready queue to get scheduled on the CPU.

- If waiting time is more (not getting CPU time for execution) -- Starvation.

Turn-around time: Ideal - CPU burst + IO burst

- Time from arrival of the process till completion of the process.
- CPU burst + IO burst + (CPU) Waiting time + IO Waiting time

Response time: Ideal - min

- Time from arrival of process (in ready queue) till allocated CPU for first time.

Scheduling Algorithms

FCFS

- Process added first in ready queue should be scheduled first.
- Non-preemptive scheduling
- Scheduler is invoked when process is terminated, blocked or gives up CPU is ready for execution.
- Convoy Effect: Larger processes slow down execution of other processes.

SJF

- Process with lowest burst time is scheduled first.
- Non-preemptive scheduling
- Minimum waiting time

SRTF - Shortest Remaining Time First

- Similar to SJF - but Preemptive scheduling
- Minimum waiting time

Priority

- Each process is associated with some priority level. Usually lower the number, higher is the priority.
- Preemptive scheduling or Non Preemptive scheduling
- Starvation
    - Problem may arise in priority scheduling.
    - Process not getting CPU time due to other high priority processes.
    - Process is in ready state (ready queue).
    - May be handled with aging -- dynamically increasing priority of the process.

Round-Robin

- Preemptive scheduling
- Process is assigned a time quantum/slice.
- Once time slice is completed/expired, then process is forcibly preempted and other process is scheduled.
- Min response time.

Fair-share

- CPU time is divided into epoch times.
- Each ready process gets some time share in each epoch time.
- Process is assigned a time share in proportion with its priority.
- In Linux, processes with time-sharing (TS) class have nice value. Range of nice value is -20 (highest priority) to +19 (lowest priority).

Regular Expressions

- Find a pattern in text file(s).
- Regular expressions are patterns used to match character combinations in strings.
- A regular expression pattern is composed of simple characters, or a combination of simple and special characters e.g. /abc/, /ab*c/
- Pattern is given using regex wild-card characters.
    - Basic wild-card characters
        - $ - find at the end of line.
        - ^ - find at the start of line.
        - [ ] - any single char in give range or set of chars
        - [^] - any single char not in give range or set of chars
        - . - any single character
        - zero or more occurrences of previous character
    - Extended wild-card characters
        - ? - zero or one occurrence of previous character
        - one or more occurrences of previous character
        - {n} - n occurrences of previous character
        - {,n} - max n occurrences of previous character
        - {m,} - min m occurrences of previous character
        - {m,n} - min m and max n occurrences of previous character
        - () - grouping (chars)
        - (|) - find one of the group of characters

grep

- Regex commands
    - grep - GNU Regular Expression Parser - Basic wild-card
    - egrep - Extended Grep - Basic + Extended wild-card
    - fgrep - Fixed Grep - No wild-card
- Command syntax
    - grep "pattern" filepath
    - grep [options] "pattern" filepath
        - -c : count number of occurrences
        - -v : invert the find output
        - -i : case insensitive search
        - -w : search whole words only
        - -R : search recursively in a directory
        - -n : show line number.

VI Editor

- sudo apt-get install vim
- VI editor works in two modes
    - command mode
    - insert mode
- press i - to go into insert mode
- press Esc - to go into command node
- VI editor commands:
    - :w - write/save into file
    - :q - quit vi editor
    - :wq - save and quit
    - :y - to copy current line
    - yy - to copy current line
    - nyy - copy n lines from current line
    - :m,ny - copy fomr mth line to nth line
    - :d - to cut current line
    - dd - to cut current line
    - ndd - cut n lines from current line
    - :m,nd - cut fomr mth line to nth line
    - press p - to paste copied line on next line of current line
- To do setting in vi editor
    - create file into home directory as below:

```
vim ~/.vimrc
```

- add below content into it

```
set number
set autoindent
set tabstop=4
set nowrap
```