Operating System Concepts

- OS Concepts
- Linux commands
- Shell scripts
- Linux System call Programming

Learning OS

- step 1: End user
    - Linux commands
- step 2: Administrator
    - Install OS (Linux)
    - Configuration - Users, Networking, Storage, ...
    - Shell scripts
- step 3: Programmer
    - Linux System call programming
- step 4: Designer/Internals
    - UNIX & Linux internals

What is OS?

- Interface between end user and computer hardware.
- Interface between Programs and computer hardware.
- Control program that controls execution of all other programs.
- Resource manager/allocator that manage all hardware resources.
- Bootable CD/DVD = Core OS + Applications + Utilities
- Core OS = Kernel -- Performs all basic functions of OS.

OS Functions

- CPU scheduling
- Process Management
- Memory Management
- File & IO Management
- Hardware abstraction
- User interfacing
- Security & Protection
- Networking

Process Management

Program

- Set of instructions given to the computer --> Executable file.
- Program --> Sectioned binary --> "objdump" & "readelf".

- Exe header --> Magic number, Address of entry-point function, Information about all sections. (objdump -h program.out)
- Text --> Machine level code (objdump -S program.out)
- Data --> Global and Static variables (Initialized)
- BSS --> Global and Static variables (Uninitialized)
- RoData --> String constants
- Symbol Table --> Information about the symbols (Name, Size, section, Flags, Address) (objdump -t program.out)
- Program (Executable File) Format
  - Windows -- PE
  - Linux -- ELF
- Program are stored on disk (storage).

Process

- Progam under execution
- Process execute in RAM.
- Process control block contains information about the process (required for the execution of process).
  - Process id
  - Exit status
    - 0 - Indicate successful execution
    - Non-zero - Indicate failure
  - Scheduling information (State, Priority, Sched algorithm, Time, ...)
  - Memory information (Base & Limit, Segment table, or Page table)
  - File information (Open files, Current directory, ...)
  - IPC information (Signals, ...)
  - Execution context (Values of CPU registers)
  - Kernel stack
- PCB is also called as process descriptor (PD), uarea (UNIX), or task_struct (Linux).
- In Linux size of task_struct is approx 4KB

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.

- Linux: TASK_RUNNING (R), TASK_INTERRUPTIBLE (S), TASK_UNINTERRUPTIBLE (D), TASK_STOPPED(T), TASK_ZOMBIE (Z), TASK_DEAD (X)

Classification of OS

- OS can be categorized based on the target system (computers).
    - Mainframe systems
    - Desktop systems
    - Multi-processor (Parallel) systems
    - Distributed systems
    - Hand-held systems
    - Real-time systems

Mainframe systems

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

Desktop systems

- Personal computers -- desktop and laptops
- User convinence and Responsiveness
- Examples: Windows, Mac, Linux, few UNIX, ...

Multiprocessor systems

- The systems in which multiple processors are connected in a close circuit is called as "multiprocessor computer".
- The programs/OS take advantage of multiple processors in the computer are called as "Multiprocssing" programs/OS.
    - Windows Vista: First Windows OS designed for multi-processing.
    - Linux 2.5+: Linux started supporting multi-processing.
- Modern PC architectures are multi-core arch i.e. multiple CPUs on single chip.
- Since multiple tasks can be exeuted on these processors simultaneously, such systems are also called as "parallel systems".
- Parallel systems have more throughput (Number of tasks done in unit time).
- There are two types of multiprocessor systems:
    - Asymmetric Multi-processing
    - Symmetric Multi-processing

Asymmetric Multi-processing

- OS treats one of the processor as master processor and schedule task for it. The task is in turn divided into smaller tasks and get them done from other processors.

Symmetric Multi-processing

- OS considers all processors at same level and schedule tasks on each processor individually.
- All modern desktop systems are SMP.

Distributed systems

- Multiple computers connected together in a close network is called as "distributed system".
- Its advantages are high availability (24x7), high scalability (many clients, huge data), fault tolerance (any computer may fail).
- The requests are redirected to the computer having less load using "load balancing" techniques.
- The set of computers connected together for a certain task is called as "cluster". Examples: Linux.

Handheld systems

- OS installed on handheld devices like mobiles, PDAs, iPODs, etc.
- Challenges:
  - Small screen size
  - Low end processors
  - Less RAM size
  - Battery powered
- Examples: Symbian, iOS, Linux, PalmOS, WindowsCE, etc.

Realtime systems

- The OS in which accuracy of results depends on accuracy of the computation as well as time duration in which results are produced, is called as "RTOS".
- If results are not produced within certain time (deadline), catastrophic effects may occur.
- These OS ensure that tasks will be completed in a definite time duration.
- Time from the arrival of interrupt till begin handling of the interrupt is called as "Interrupt Latency".
- RTOS have very small and fixed interrupt latencies.
- RTOS Examples: uC-OS, VxWorks, pSOS, RTLinux, FreeRTOS, etc.

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).

User interfacing

- UI of OS is a program (Shell) that interface between End user and Kernel.
- Shell -- Commmand interpreter
  - End user --> Command --> Shell --> Kernel
- User interfacing (Shell)
  - Graphical User Interface (GUI)
  - Command Line Interface (CLI)

Example shells

- Windows
  - GUI shell: explorer.exe
  - CLI shell: cmd.exe, powershell.exe
- DOS
  - CLI shell: command.com
- Unix/Linux
  - CLI shell: bsh, "bash", ksh, csh, zsh, ...
    - ls /bin/*sh
    - echo $SHELL
  - shell of current user can be changed using "chsh" command.
- GUI shell/standards
  - GNOME: GNU Network Object Model Environment (e.g. Ubuntu, Redhat, CentOS, ...)
  - KDE: Kommon Desktop Environment (e.g. Kubuntu, SuSE, ...)

Linux File Structure

- Linux follows "/" (root) file system.
- "/" is a starting point of Linux file system.

- All your data is stored in this partition.
- / contains boot, bin, sbin, etc, root, home, dev, proc, mnt, media, opt
- In Linux everything is a file.
- Mainly there are two types of files in Linux
  - File
  - Directory (Folder)
- Linux Directories
  - boot - files related to booting
    - vmlinuz - kernel Image
    - grub - boot loader
    - config - kernel configuration
    - initrd/initramfs - initail root file system
  - bin - user commands in binary format
  - sbin - all admin/system commands in binary format
  - etc - configuration files
  - root - home directory of root user
  - home - it contains sub directories for each user with its name
    - devendra -> /home/devendra
    - sunbeam -> /home/sunbeam
    - osboxes -> /home/osboxes
  - dev - it contains all device related files
  - lib - shared program libraries required by kernel
  - mnt - it is temporary mount point
  - media - it is mount point for media eg cdrom
  - opt - stores optional files of large softwares
  - proc - virtual file system - it contains information about system or processes
  - sys - entries of each block devices, subdirectories for each physical bus type supported, every device class registered with the kernel, global device hierarchy of all devices
  - tmp - temporary files that may be lost on system shutdown
  - usr - read only directory that stores small programs and files accessible to all users

Path

- It is a unique location of any file in the file system.
- It is represented by character strings with few delimiters ("/", "\", ":")
- Types of path

  - There are two types of paths in linux

  - Absolute path

    - Path which starts with "/" is called as absolute path.
    - E.g. /home/devendra/MyData/Demos/demo01.sh

  - Relative path

    - Path with respect to current directory is called as relative path
    - E.g. MyData/Assignments/assign02.pdf

Types of files

- Regular file (-)
- Directory file (d)
- Link file (l)
- Socket file (s)
- Pipe file (p)
- Character Special file (c)
- Block Special file (b)