**Unix operating System**

**Unit 1**

**Explain different computer system architecture**

Computer architecture means construction/design of a computer. A computer system may be organized in different ways. Some computer systems have single processor and others have multiprocessors. So based on the processors used in computer systems, they are categorized into the following systems.

1. Single-processor system

2. Multiprocessor system

3. Clustered Systems:

1. Single-Processor Systems:

Some computers use only one processor such as microcomputers (or personal computers PCs). On a single-processor system, there is only one CPU that performs all the activities in the computer system. However, most of these systems have other special purpose processors, such as I/O processors that move data quickly among different components of the computers. These processors execute only a limited system programs and do not run the user program. Sometimes they are managed by the operating system.

2. Multiprocessor Systems:

In multiprocessor system, two or more processors work together. In this system, multiple programs (more than one program) are executed on different processors at the same time. This type of processing is known as multiprocessing. Multiprocessor system is also known as parallel system. Mostly the processors of multiprocessor system share the common system bus, clock, memory and peripheral devices. This system is very fast in data processing.

Types of Multiprocessor Systems:

The multiprocessor systems are further divided into two types; (i). Asymmetric multiprocessing system

(ii). Symmetric multiprocessing system

(i) Asymmetric Multiprocessing System(AMS):

The multiprocessing system, in which each processor is assigned a specific task, is known as Asymmetric Multiprocessing System

(ii) Symmetric Multiprocessing System(SMP):

The multiprocessing system, in which multiple processors work together on the same task, is known as Symmetric Multiprocessing System.

3. Clustered Systems:

Clustered system is another form of multiprocessor system. This system also contains multiple processors but it differs from multiprocessor system. The clustered system consists of two or more individual systems that are coupled together. In clustered system, individual systems (or clustered computers) share the same storage and are linked together ,via Local Area Network (LAN).

Types of Clustered Systems:

Like multiprocessor systems, clustered system can also be of two types (i). Asymmetric Clustered System

(ii). aSymmetric Clustered System

In asymmetric clustered system, one machine is in hot-standby mode while the other machine is running the application. The hot-standby host machine does nothing.

(ii). Symmetric Clustered System:

In symmetric clustered system, multiple hosts (machines) run the applications. They also monitor each other.

**Operating system operations**

* A Multiuser System

UNIX is basically a multiprogramming system. Here, either

Multiple users can run separate jobs or

Singe user can run multiple jobs

many processes are running simultaneously. And, the resources like CPU, memory and hard disk etc are shared between all users. Hence, UNIX is a multiuser system as well.

* A Multitasking System

Unix is a multitasking system, wherein a single user can run multiple jobs concurrently. A

user may edit a file, print a document on a printer and open a browser etc – all at a time.

* Documentation

Unix provides a large set of documents to understand the working of every command and

feature of it.

* Pattern Matching

Unix has very sophisticated pattern matching features. The character like \* (known as a

metacharacter) helps in searching many files starting with a particular name.

* The UNIX Toolkit

Unix contains diverse set of applications like text manipulation utilities, compilers and

interpreters, networked applications, system administration tools etc.

* The Building-Block Approach

Unix is a collection of few hundred commands, each of which is designed to perform one

task. More than one command can be connected via the | (pipe) symbol to perform multiple

tasks.

**Explain different operating system services**

## Program execution

Operating systems handle many kinds of activities from user programs to system programs like printer spooler, name servers, file server, etc. Each of these activities is encapsulated as a process.

* Loads a program into memory.
* Executes the program.
* Handles program's execution.
* Provides a mechanism for process synchronization.
* Provides a mechanism for process communication.

**i/o operation:** An I/O subsystem comprises of I/O devices and their corresponding driver software. An Operating System manages the communication between user and device drivers.

* I/O operation means read or write operation with any file or any specific I/O device.
* Operating system provides the access to the required I/O device when required.

**File System manipulation**

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions

* Program needs to read a file or write a file.
* The operating system gives the permission to the program for operation on file.
* Permission varies from read-only, read-write, denied and so on.
* Operating System provides an interface to the user to create/delete files.

**Communication**:Multiple processes communicate with one another through communication lines in the network.

   
Two processes often require data to be transferred between them

 Both the processes can be on one computer or on different computers, but are connected through a computer network.

## Error handling

Errors can occur anytime and anywhere. An error may occur in CPU, in I/O devices or in the memory hardware.

* he OS constantly checks for possible errors.
* The OS takes an appropriate action to ensure correct and consistent computing.

## Resource Management

In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job

* The OS manages all kinds of resources using schedulers.
* CPU scheduling algorithms are used for better utilization of CPU.

## Protection

Considering a computer system having multiple users and concurrent execution of multiple processes, the various processes must be protected from each other's activities.

* he OS ensures that all access to system resources is controlled.
* The OS ensures that external I/O devices are protected from invalid access attempts

**What are system calls and explain different types of system calls**

 A system call is a way for programs to **interact with the operating system**. A computer program makes a system call when it makes a request to the operating system’s kernel. System call **provides** the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system

types of system calls −

### Process Control

These system calls deal with processes such as process creation, process termination etc.

### File Management

These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

### Device Management

These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

### Information Maintenance

These system calls handle information and its transfer between the operating system and the user program.

### Communication

These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

**Discuss different types of system programs**

**System Programming** can be defined as the act of building Systems Software using System Programming Languages

**File management** is defined as the process of manipulating files in the computer system, its management includes the process of creating, modifying and deleting files.

* It helps to create new files in the computer system and placing them at specific locations.
* It helps in easily and quickly locating these files in the computer system.
* It makes the process of sharing files among different users very easy and user-friendly.

**Status Information –**   
Information like date, time amount of available memory, or disk space is asked by some user . All this information is formatted and displayed on output devices or printed.

**File Modification –**   
For modifying the contents of files we use this. For Files stored on disks or other storage devices, we used different types of editors.

**Programming-Language support –**   
For common programming languages, we use Compilers, Assemblers, Debuggers, and interpreters which are already provided to users. It provides all support to users. We can run any programming language.

**Program Loading and Execution –**   
When the program is ready after Assembling and compilation, it must be loaded into memory for execution. A loader is part of an operating system that is responsible for loading programs and libraries

**Communications –**   
Virtual connections among processes, users, and computer systems are provided by programs. Users can send messages to another user on their screen, User can send e-mail, browsing

**6. Write a short note on Virtual machines and system boot.**

Virtualization is a technology that allows us to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illu- sion that each separate environment is running on its own private computer.

These environments can be viewed as different individual operating systems (for example, Windows and UNIX) that may be running at the same time and may interact with each other. A user of a virtual machine can switch among the various operating systems in the same way a user can switch among the various processes running concurrently in a single operating system.

**System boot** Booting the system is done by loading the kernel into main memory, and starting its execution. The CPU is given a reset event, and the instruction register is loaded with a predefined memory location, where execution starts. The initial bootstrap program is found in the BIOS read-only memory

]

**Discuss scheduling criteria**

1. **CPU utilisation –**   
   The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. Theoretically, CPU utilisation can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the load upon the system.
2. **Throughput –**   
   A measure of the work done by CPU is the number of processes being executed and completed per unit time. This is called throughput. The throughput may vary depending upon the length or duration of processes.
3. **Turnaround time –**   
   For a particular process, an important criteria is how long it takes to execute that process. The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time. Turn-around time is the sum of times spent waiting to get into memory, waiting in ready queue, executing in CPU, and waiting for I/O.
4. **Waiting time –**   
   A scheduling algorithm does not affect the time required to complete the process once it starts execution. It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue.
5. **Response time –**   
   In an interactive system, turn-around time is not the best criteria. A process may produce some output fairly early and continue computing new results while previous results are being output to the user

**Write a short note on multiple process scheduling**

In multiple-processor scheduling **multiple CPU’s** are available and hence **Load Sharing** becomes possible. However multiple processor scheduling is more **complex** as compared to single processor scheduling.

One approach is when all the scheduling decisions and I/O processing are handled by a single processor which is called the **Master Server** and the other processors executes only the **user code**. This is simple and reduces the need of data sharing. This entire scenario is called **Asymmetric Multiprocessing**.

A second approach uses **Symmetric Multiprocessing** where each processor is **self scheduling**. All processes may be in a common ready queue or each processor may have its own private queue for ready processes. The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.

**4. Write a short note on semaphore and explain the use of wait () and signal () operations**

Semaphore is simply a variable that is non-negative and shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment. Semaphores are of two types:

1. **Binary Semaphore –**   
   This is also known as mutex lock. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problems with multiple processes.
2. **Counting Semaphore –**   
   Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

**wait () and signal ()**

Two standard operations, wait and signal are defined on the semaphore. Entry to the critical section is controlled by the wait operation and exit from a critical region is taken care by signal operation. The semaphore operation are implemented as operating system services and so wait and signal are atomic in nature i.e. once started, execution of these operations cannot be interrupted.

. **Explain readers-writers problem**

Consider a situation where we have a file shared between many people.

* If one of the people tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her.
* However if some person is reading the file, then others may read it at the same time.

Precisely in OS we call this situation as the **readers-writers problem**

**Problem parameters:**

* One set of data is shared among a number of processes
* Once a writer is ready, it performs its write. Only one writer may write at a time
* If a process is writing, no other process can read it
* If at least one reader is reading, no other process can write
* Readers may not write and only read

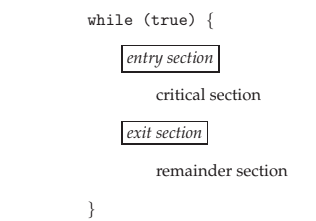
**6. Define critical section and write the general structure of critical section problem.**

**Explain the necessary conditions to satisfy critical section**

Each process has a segment of code, called a critical section, in which the process may be accessing — and updating — data that is shared with at least one other process. The important feature of the system is that,

when one process is executing in its critical section, no other process is allowed to execute in its critical section. The critical-section problem is to design a protocol that the processes can use to synchronize their activity so as to cooperatively share data.

**the general structure of critical section**



**Mutual exclusion**

By Mutual Exclusion, we mean that if one process is executing inside critical section then the other process must not enter in the critical section

**Progress**

Progress means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.

**Bounded Waiting**

We should be able to predict the waiting time for every process to get into the critical section. The process must not be endlessly waiting for getting into the critical section.

**7. Explain bounded buffer and dining philosopher problem as a classic problem of**

**Synchronization**

The bounded-buffer problems (aka the producer-consumer problem) is a classic example of concurrent access to a shared resource. A bounded buffer lets multiple producers and multiple consumers share a single buffer. Producers write data to the buffer and consumers read data from the buffer.

**The Dining Philosopher Problem –** The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.

**8. What is monitor with reference to synchronization? Explain with diagram.**

The monitor is one of the ways to achieve Process synchronization. The monitor is supported by programming languages to achieve mutual exclusion between processes.

1. It is the collection of condition variables and procedures combined together in a special kind of module or a package.
2. The processes running outside the monitor can’t access the internal variable of the monitor but can call procedures of the monitor.
3. Only one process at a time can execute code inside monitors.

**Unit -2**

1. **Define deadlock? Explain the necessary conditions for a deadlock**

***Deadlock***is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Necessary Conditions

**1. Mutual exclusion.** At least one resource must be held in a nonsharablemode; that is, only one thread at a time can use the resource. If anotherthread requests that resource, the requesting thread must be delayed untilthe resource has been released**.**

**2. Hold and wait**. A thread must be holding at least one resource and waiting to acquire additional resources that are currently being held by other threads.

**3. No preemption.** Resources cannot be preempted; that is, a resource can be released only voluntarily by the thread holding it, after that thread has completed its task.

***Circular Wait:*** A set of processes are waiting for each other in circular form.

1. **Explain different methods for handling deadlock.**
2. **Methods for handling deadlock**   
   There are three ways to handle deadlock   
   1) Deadlock prevention or avoidance: The idea is to not let the system into a deadlock state.   
   One can zoom into each category individually, Prevention is done by negating one of above mentioned necessary conditions for deadlock.   
   Avoidance is kind of futuristic in nature. By using strategy of “Avoidance”, we have to make an assumption. We need to ensure that all information about resources which process will need are known to us prior to execution of the process. We use Banker’s algorithm (Which is in-turn a gift from Dijkstra) in order to avoid deadlock.
3. 2) Deadlock detection and recovery: Let deadlock occur, then do preemption to handle it once occurred.
4. 3) Ignore the problem altogether: If deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take.
5. **With a neat diagram explain resource-allocation graph**

the resource allocation graph is the pictorial representation of the state of a system. As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.

It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle. Let's see the types of vertices and edges in detail.

**4. Write and explain deadlock avoidance algorithm foe deadlock avoidance**.

5**. How can deadlock can be prevented? Describe them.**

Mutual Exclusion

The mutual-exclusion condition must hold. That is, at least one resource must be nonsharable. Sharable resources do not require mutually exclusive access and thus cannot be involved in a deadlock. Read-only files are a good example of a sharable resource.

Hold and Wait

To ensure that the hold-and-wait condition never occurs in the system, we must guarantee that, whenever a thread requests a resource, it does not hold any other resources. One protocol that we can use requires each thread to request and be allocated all its resources before it begins execution.

No Preemption

The third necessary condition for deadlocks is that there be no preemption of resources that have already been allocated. To ensure that this condition does not hold, we can use the following protocol. If a thread is holding some resources and requests another resource that cannot be immediately allocated

to it (that is, the thread must wait), then all resources the thread is currentlyholding are preempted.

**Eliminate Circular Wait**   
Each resource will be assigned with a numerical number. A process can request the resources increasing/decreasing. order of numbering.

1. **Explain deadlock detection algorithm for several instances of a resource**

* The wait-for graph scheme is not applicable in a system with several instance of each resource type.
* We have to use deadlock detection algorithms that can be used for such system.
* Such algorithm uses some time-variant data structures similar to those used in banker’s algorithm.
* **Request:** An ***n***X***m*** matrix to represent the current request of each process.
* **Allocation:** An ***n***X***m*** matrix to represent the number of resources of each type currently allocated to each process.
* **Available:**  A vector of length ***m***to represents the number of available resources of each type
* Execution of deadlock detection algorithm for every resource request will add extra overhead on the system.

**7. Banker’s algorithm problem to find the need matrix and safe state.(Problem)**

**1. Distinguish between paging and segmentation**

**Paging Segmentation**

|  |  |  |
| --- | --- | --- |
| 1. | In paging, program is divided into fixed or mounted size pages. | In segmentation, program is divided into variable size sections. |
| 2. | For paging operating system is accountable. | For segmentation compiler is accountable. |
| 3. | Page size is determined by hardware. | Here, the section size is given by the user. |
| 4. | It is faster in the comparison of segmentation. | Segmentation is slow. |
| 5. | Paging could result in internal fragmentation. | Segmentation could result in external fragmentation. |
| 6. | In paging, logical address is split into page number and page offset. | Here, logical address is split into section number and section offset. |
| 7. | Paging comprises a page table which encloses the base address of every page. | While segmentation also comprises the segment table which encloses segment number and segment offset. |
| 8. | Page table is employed to keep up the page data. | Section Table maintains the section data. |
| 9. | In paging, operating system must maintain a free frame list. | In segmentation, operating system maintain a list of holes in main memory. |
| 10. | Paging is invisible to the user. | Segmentation is visible to the user. |

1. **Define demand paging and explain the various steps in handling a page fault with a schematic diagram**

**Demand Paging:** In demand paging, a page is delivered into the memory on demand i.e., only when a reference is made to a location on that page. Demand paging combines the feature of simple paging and implement virtual memory as it has a large virtual memory. Lazy swapper concept is implemented in demand paging in which a page is not swapped into the memory unless it is required.

We check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an invalid memory access.

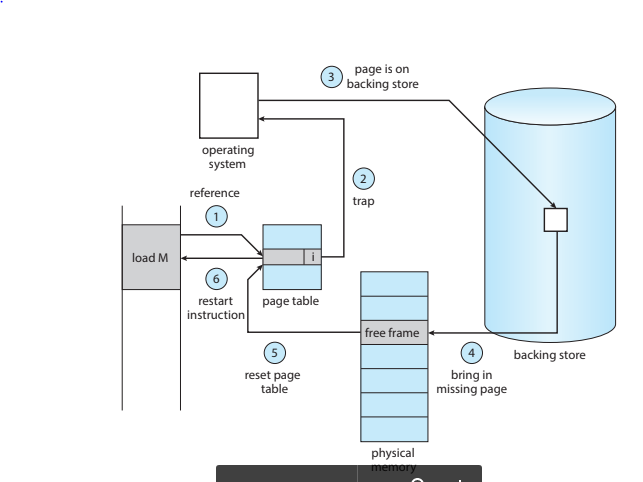
2. If the reference was invalid, we terminate the process. If it was valid but we have not yet brought in that page, we now page it in.

3. We find a free frame (by taking one from the free-frame list, for example).

4. We schedule a secondary storage operation to read the desired page into the newly allocated frame.

5. When the storage read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.

6. We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.



3. **Explain the with the help of supporting diagram TLB improves the performance of**

**demand paging**

4. What is memory allocation? Explain the strategies most commonly used to select a

free hole from the set of available holes

1. **What is paging**

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non – contiguous.. The paging is always performed between active pages.

1. **Explain the concept of swapping**

Swapping is a process of swapping a process temporarily to a secondary memory from main memory which is fast as compared to secondary memory. But as [RAM](https://www.geeksforgeeks.org/different-types-ram-random-access-memory/) is of less size so the process that is inactive is transferred to secondary memory. The main part of swapping is transferred time and the total time directly proportional to the amount of memory swapped. lthough the process of swapping affects the performance of the system, it helps to run larger and more than one process

* Swap-out is a method of removing a process from RAM and adding it to the hard disk.
* Swap-in is a method of removing a program from a hard disk and putting it back into the main memory or RAM.

**7. Explain paging hardware with TLB**

**9. Explain different Page replacement algorithms to find page faults for the given set of**

**input stream.**

**Unit 3**

**1 Define the file system in unix? Explain the different types of file categories**

Unix file system is a logical method of organizing and storing large amounts of information in a way that makes it easy to manage. All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system.

Three types of file system are

Ordinary (Regular) File

It is a most common type of file that contains only data as a stream of characters. An ordinary file

can be one among these –

 Text file: contains only printable characters. Source codes of programming languages like C, Java etc

Binary file: contains both printable and non-printable characters covering entire ASCII (0 – 255) set. The object codes, executable files etc. created by compiling C language are binary files.

2 .Directory File

A directory contains no data, but it keeps some information about the files and subdirectories that it contains. The UNIX file system is organized with a number of directories and subdirectories. A user also can create them, as and when required. Usually, a group of related files are kept in a single directory.

A directory file contains an entry for every file and subdirectory it has. Each such entry has two components viz. –

* The filename
* A unique identification number for the file or directory (called as the inode number)

One cannot write into a directory file. But, the actions like creating a file, removing a file etc. makes kernel to update the corresponding directory by creating/removing filename and inode number associated with that file.

3. Device files

Device filenames are generally found inside a single directory structure, /dev. A device file is not a stream of characters. In fact, it does not contain anything. The operation of a device is completely managed by the attributes of its associated file. The kernel identifies a device from its attributes and then uses them to operate the device.

The activities like printing files, installing softwares from CD-ROM, taking backup of files into a tape/drive etc.

**2 Discuss the parent-child relationship  with a neat figure**

All files in UNIX are related to each other. The file system in UNIX is a collection of all types (ordinary, directory and device files) related files organized in a hierarchical structure. A UNIX file system has root represented by /, which serves as reference point for all files.

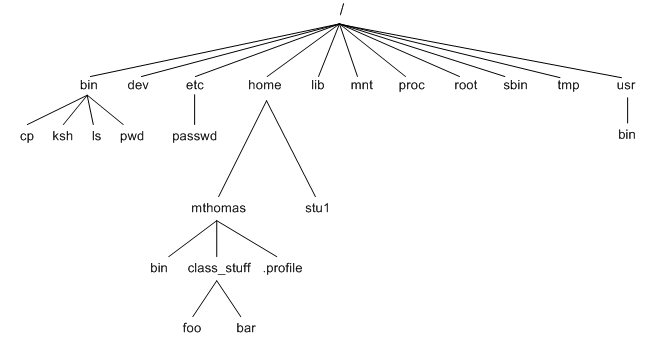
The root directory (/) has a number of subdirectories under it. These subdirectories in turn have

more subdirectories and files under them.

Every file, apart from root, must have a parent, and there will be a parent-child relationship path

from root to file. In the Figure 2.1, cp is child of bin and bin is child of /. That is, / is grandparent

of cp. Note that, in a parent-child relationship, parent is always a directory.



**3.Explain the command**

**HOME :** When a user logs into the system, UNIX places him into a directory called as home directory. It is created by the system when a user account is opened. If you have logged in with a user name john, then, your home directory would be /home/john. This can be viewed using the shell variable

$echo $HOME

/home/john # the first / represents root directory

**Pwd: CHECKING YOUR CURRENT DIRECTORY**

Once a user logs in to the UNIX system, it places him in a specific directory of the file system. Though a user can move from one directory to other, for a given moment of time, he will be in one directory, known as current directory. To know current directory, the pwd (print working directory) command is used.

$pwd/home/john

**Mkdir**: A directory can be created using the command mkdir.

The name of the directory to be created has to be given as argument. For example,

$mkdir docs

* One can create more than one directory with a single mkdir command as below

$mkdir docs progs db # three directories created

* A directory tree can be created as

$mkdir test test/prgms test/data

**rmdir: REMOVING DIRECTORIES**

To remove (or delete) a directory, the rmdir command is used. Few important points about this command are . A directory has to be empty before removing it. That is, it should not contain any

files or subdirectories.

* To remove one directory, use statement like

$rmdir test #removes the directory test

* More than one directory (even in tree structure) can be removed at a time. For example,

$mkdir test/prgms test/data test

**cd: CHANGING THE CURRENT DIRECTORY**

The cd command is used to move around the file system by changing the directory. This

command can be used in three different ways –

* If the user john is in his home directory and would like to move to subdirectory called as progs, then the command should be given as –

$ cd progs # user is moved to progs directory now

$ pwd # verify this using pwd

/home/john/progs

* If the user would like to shift some other directory (but not his own subdirectory),then absolute path name can be given. For example,

$ pwd #check current directory

/home/john/progs

$ cd /bin # change directory to /bin

$ pwd # verify new location

/bin

**4.Write and explain the concept of absolute pathnames and relative pathnames**

ABSOLUTE PATHNAMES

A path of a file (or directory) which specifies the complete hierarchy of that file starting from the root (/) directory is called as absolute pathname of that file (or directory). Most of the UNIX commands that take file or directory name as arguments will assume that the specified file exists in the current working directory. That is, if you use the command

$cat test.sh

it is assumed that the file test.sh is in current directory. If you would like to access the file in some other directory, you have to give the command by specifying absolute pathname of that file as –

$ cat /home/john/test.sh

RELATIVE PATHNAMES

A relative path is defined as the path related to the present working directory. For example, if we are currently in /home/john and would like to move to /home/john/test, then we can just give the command as

$cd test #this is relative path usage

We can also give the command as

$ cd /home/john/test #this is absolute path usage

So, one can say that, if the pathname starts with / (root), we can say that it is absolute path. Otherwise, it is relative path.

**5 Discuss the Basics of vi editor**

* The Repeat Factor:

vi provides repeat factor in command and input mode commands. Command mode command k moves the cursor one line up. 10k moves cursor 10 lines up.

* Saving Text and Quitting

The ex Mode When you edit a file using vi, the original file is not distributed as such, but only a copy of it that is placed in a buffer. From time to time, you should save your work by writing the buffer contents to disk to keep the disk file current.

the input mode commands are:

Command Action

:W saves file and remains in editing mode

:x saves and quits editing mode

:wq saves and quits editing mode

:w save as

* Navigation Mode A command mode command doesn’t show up on screen but simply performs a function. To move the cursor in four directions,

k moves cursor up

j moves cursor down

h moves cursor left

l moves cursor right

* Absolute Movement The editor displays the total number of lines in the last line Ctrl-g to know the current line number

40G goes to line number 40

1G goes to line number 1

G goes to end of file

* Editing Text The editing facilitates in vi are very elaborate and invoke the use of operators. They use operators, such as,

d delete

y yank (copy)

* Deleting Text

x deletes a single

character dd delete

entire line

yy copy entire line

6dd deletes the current line and five lines below

1. **Explain ls options in detail**

* [ls -a](https://www.javatpoint.com/linux-ls#linux-ls-a) : In Linux, hidden files start with . (dot) symbol and they are not visible in the regular directory. The (ls -a) command will enlist the whole list of the current directory including the hidden files

$ ls –a

. .. .exrc Thesis

.emacs .gnome2 Shell1.sh

* (–F) : The ls command displays files as well as directories. To know which of them are directories and executable files, one can use –F option.

$ ls –Fx

Thesis/ Shell1.sh\* Shell2.sh\* ShellPgms/

Emp.txt cmd.c helpdir/

The \* indicates that the file contains executable code, and / refers to a directory.

* (–R): This option lists all files and subdirectories in a directory tree. That is, contents of subdirectories also will be displayed recursively till thereis no subdirectory is left out.

ls –R

Thesis :

Shell1.sh Shell2.sh ShellPgms

Emp.txt cmd.c

./Thesis:

Chap1.aux Chap1.bib Chap1.tex Chap1.pdf

Annex.aux Annex.pdf

* (–x) : When there are many files, it is better to display them in multiple columns.

$ ls –x

Thesis Shell1.sh Shell2.sh ShellPgms

Emp.txt cmd.c helpdir

* The –l option of ls command is used for listing the various attributes like permissions, size ownership etc. of a file. The output of ls –l is referred to as the listing.

$ ls -l

total 144

-rw-rw-r—1 john john 280 Jan 30 09:56 caseEx.sh

-rw-rw-r-- 1 john john 104 Feb 3 06:40 cmdArg.sh

* If we want to list the attributes of only the directory, but not its contents, we can use –d option as below –

$ ls –d myDir

drwxrwxr-x 2 john john 4096 Feb 6 05:48 myDir

* ls –g :If you don't want to display the owner information in your list, then you can exclude this column with the help of this command.

**7 What are file permissions? What are the different ways of setting file permissions (chmode both relative and absolute permission**

UNIX has a simple and well-defined system of assigning permissions to files. UNIX follows three-tiered file protection system to determine a file’s access rights. Example:

$ ls –l chap02 dept.lst shell1.sh

-rwxr-xr-- 1 john richard 20500 Sep 29 11:53 chap02

-rwxr-xr-x 1 john richard 850 Oct 02 10:12 dept.lst

-rw-rw-rw- 1 john richard 48 Nov 07 08:03 shell1.sh

Every group contains any of the characters r, w, x and -. The meaning of these is –

* r: indicates read permission – means, cat command can display the file
* w: indicates write permission – file can be edited with an editor
* x: indicates execute permission – the file can be executed as a program
* -: indicates absence of the corresponding permission

The chmod (change mode) :command is used for assigning/removing different permissions to/from category (user, group, others). This command can be run only by the user (owner) and the super-user (admin). The chmod command can be used in two ways –

* In a relative manner by specifying the changes to the current permissions
* In an absolute manner by specifying the final permissions

1 Relative Permissions

When changing permissions in a relative manner, chmod changes only the permissions specified in the command line and leaves the other permissions unchanged. The syntax is–

*chmod category operation permission filenames*

The argument for chmod is an expression consisting of some letters and symbols describing user category and type of permission being assigned/removed.

## Absolute Permissions

Irrespective of existing permissions for a file, we may need to assign a new set of permissions. That is, we wish to set all nine permission bits explicitly. This is known as *absolute permissions*. For this purpose, ***chmod*** uses a string of three octal  numbers. Various permissions are given a specific digit as below –

1. Read permission – 4
2. Write permission – 2
3. Execute permission – 1

* Assigning read and write(4+2=6) permissions to all –

$ chmod 666 test

$ ls –l test

-rw-rw-rw- 1 john richard 853 Sep 5 23:38 test

* To remove the write permission from group and others:

$ chmod 644 test

$ ls –l test

-rw-r--r-- 1 john richard 853 Sep 5 23:38 test

**8 Differentiate between Hard links and soft links**

A link in UNIX is a pointer to a file. Creating links is a kind of shortcuts to access a file. Links allow more than one file name to refer to the same file, elsewhere.

There are two types of links :

1. Soft Link or Symbolic links
2. Hard Links

Hard Links

* Each hard linked file is assigned the same Inode value as the original, therefore they reference the same physical file location. Hard links more flexible and remain linked even if the original or linked files are moved throughout the file system, although hard links are unable to cross different file systems.
* ls -l command shows all the links with the link column shows number of links.
* Links have actual file contents
* Removing any link, just reduces the link count, but doesn’t affect other links.
* Even if we change the filename of the original file then also the hard links properly work.
* We cannot create a hard link for a directory to avoid recursive loops.
* If original file is removed then the link will still show the content of the file.
* The size of any of the hard link file is same as the original file and if we change the content in any of the hard links then size of all hard link files are updated.
* The disadvantage of hard links is that it cannot be created for files on different file systems and it cannot be created for special files or directories.
* Command to create a hard link is:

$ ln [original filename] [link name]

Soft Links

* A soft link is similar to the file shortcut feature which is used in Windows Operating systems. Each soft linked file contains a separate Inode value that points to the original file. As similar to hard links, any changes to the data in either file is reflected in the other. Soft links can be linked across different file systems, although if the original file is deleted or moved, the soft linked file will not work correctly (called hanging link).
* ls -l command shows all links with first column value l? and the link points to original file.
* Soft Link contains the path for original file and not the contents.
* Removing soft link doesn’t affect anything but removing original file, the link becomes “dangling” link which points to nonexistent file.
* A soft link can link to a directory.
* Size of a soft link is equal to the name of the file for which the soft link is created. E.g If name of file is file1 then size of it’s soft link will be 5 bytes which is equal to size of name of original file.
* If we change the name of the original file then all the soft links for that file become dangling i.e. they are worthless now.
* Link across file systems: If you want to link files across the file systems, you can only use symlinks/soft links.
* Command to create a Soft link is:

$ ln -s [original filename] [link name]

**9 Explain chown, chgrp, find, Umask commands**

**chgrp command** in Linux is used to change the group ownership of a file or directory. All files in Linux belong to an owner and a group. You can set the group by the “chgrp” command.

**Syntax:**

chgrp [OPTION]… GROUP FILE…

chgrp [OPTION]… –reference=RFILE FILE…

To change the group ownership of a file.

sudo chgrp geeksforgeeks abc.txt

**chown** command is used to change the file Owner. Whenever you want to change ownership you can use chown command. Syntax

chown [OPTION]… [OWNER][:[GROUP]] FILE…

chown [OPTION]… –reference=RFILE FILE…

**Example:** To change owner of the file:

chown owner\_name file\_name

**find:** It can be used to find files and directories and perform subsequent operations on them. It supports searching by file, folder, name, creation date, modification date, owner and permissions

Syntax: $ find [where to start searching from]

[expression determines what to find] [-options] [what to find]

Example: $ find ./GFG -name sample.txt

**Umask :** Umask is a C-shell built-in command which allows you to determine or specify the default access (protection) mode for new files you create

umask [ value ]

example: umask 007

**10 \*\*Discuss the use and syntax of shell programming**

**11 Explain pattern matching and wild cards in shell using examples**

A pattern is framed using ordinary characters and a metacharacter (like \*) using well-defined rules. The pattern can then be used as an argument to the command, and the shell will expand it suitably before the command is executed.

The metacharacters that are used to construct the generalized pattern for matching filenames belong to a category called wild-card

|  |  |  |  |
| --- | --- | --- | --- |
| Wild Card | | Matches |  |
| \* | | Any number of characters including none | |
|  | ? | A single character | |
|  | [ijk] | A single character – either an i, j or k | |

Examples:

To list all files that begin with *chap*, use

***$ ls chap\****

The character class

You can frame more restrictive patterns with the character class. The character class comprises a set of characters enclosed by the rectangular brackets, [ and ], but it matches a single character in the class

 Examples:

***$ls chap0[124]***

Matches chap01, chap02, chap04 and lists if found.

**11. Explain how redirection is accomplished in unix**

Redirection can be defined as changing the way from where commands read input to where commands sends output. You can redirect input and output of a command.For redirection, meta characters are used. Redirection can be into a file or a program

The bash shell has three standard streams in I/O redirection:

standard input (stdin) : The stdin stream is numbered as stdin (0). The bash shell takes input from stdin. By default, keyboard is used as input.

standard output (stdout) : The stdout stream is numbered as stdout (1). The bash shell sends output to stdout. Output goes to display.

standard error (stderr) : The stderr stream is numbered as stderr (2). The bash shell sends error message to stderr. Error message goes to display.  
Each stream uses redirection commands. Single bracket '>' or double bracket '>>' can be used to redirect standard output. If the target file doesn't exist, a new file with the same name will be created.

Overwrite Commands with a single bracket '>' overwrite existing file content.

> : standard output

< : standard input

2> : standard error

Syntax:

cat > <fileName>

Example:

cat > sample.txt

**12. Discuss two special files /dev/nul and div/tty with examples**

**/dev/null**: If you would like to execute a command but don’t like to see its contents on the

screen, you may wish to redirect the output to a file called /dev/null. It is a special file that can accept any stream without growing in size. It’s size is always zero.

**/dev/tty:** This file indicates one’s terminal. In a shell script, if you wish to redirect the output of some select statements explicitly to the terminal. In such cases you can redirect these explicitly to /dev/tty inside the script.

**13. Explain pipes, tee and command substitution in unix**

**Pipe** is used to combine two or more commands, and in this, the output of one command acts as input to another command, and this command’s output may act as input to the next command and so on. This direct connection between commands/ programs/ processes allows them to operate simultaneously and permits data to be transferred between them continuously rather than having to pass it through temporary text files or through the display screen.Pipes are unidirectional **i.e data flows from left to right through the pipeline.**

**Syntax :**

command\_1 | command\_2 | command\_3 | .... | command\_N

**Example :**  
**1. Listing all files and directories and give it as input to more command.**

$ ls -l | more

**tee** is an external command that handles a character stream by duplicating its input. It saves one copy in a file and writes the other to standard output. It is also a filter and hence can be placed anywhere in a pipeline.

**Example** $ who | tee users.lst

**Command substitution** is the mechanism by which the shell performs a given set of commands and then substitutes their output in the place of the commands.

Example:

###### **$ echo Current date and time is `date`**

**Unit 4**

**1. What is a process and explain the mechanism of creating a process in Unix**

 A process is a program in execution. A process is said to be born when the program starts execution and remains alive as long as the program is active. After execution is complete, the process is said to die. The kernel is responsible for the management of the processes.

Two important attributes of a process are:

1. The Process-Id (PID): Each process is uniquely identified by a unique integer

called the PID, that is allocated by the kernel when the process is born. The PID can be used to control a process.

1. The Parent PID (PPID): The PID of the parent is available as a process attribute.

There are three distinct phases in the creation of a process and uses three important system calls , fork, *exec*, and wait

**Fork**: A process in UNIX is created with the fork system call, which creates a copy of the process that invokes it.

**Exec:** The forked child overwrites its own image with the code and data of the new program. This mechanism is called exec, and the child process is said to *exec* a new program, using one of the family of exec system calls.

**Wait:** The parent then executes the wait system call to *wait* for the child to complete. It picks up the exit status of the child and continues with its other functions

**2 Explain the following commands**

**Cron:** The **cron** is a software utility, offered by a Linux-like operating system that automates the scheduled task at a predetermined time. It is a ***daemon process***, which runs as a background process and performs the specified operations at the predefined time when a certain event or condition is triggered without the intervention of a user

**Syntax:**

cron [-f] [-l] [-L loglevel]

**Options:**

* **-f :** Used to stay in foreground mode, and don’t daemonize.
* **-l :** This will enable the LSB compliant names for /etc/cron.d files

following values:

* **1 :**It will log the start of all cron jobs.
* **2 :** It will log the end of all cron jobs.

**Batch:** The batch command lets the operating system decide an appropriate time to run a process. When you schedule a job with batch, UNIX starts and works on the process whenever the system load isn’t too great

###### $ batch

sort /usr/sales/reports/\* | lp

echo “Files printed, Boss!” | mailx -s”Job done” boss

**.3 Explain the internal and external commands**

**Internal Commands :** Commands which are built into the shell. For all the shell built-in commands, execution of the same is fast in the sense that the shell doesn’t have to search the given path for them in the PATH variable, and also no process needs to be spawned for executing it.  
Examples: source, cd, fg, etc.

**External Commands :** Commands which aren’t built into the shell. When an external command has to be executed, the shell looks for its path given in the PATH variable, and also a new process has to be spawned and the command gets executed. They are usually located in /bin or /usr/bin. For example, when you execute the “cat” command, which usually is at /usr/bin, the executable /usr/bin/cat gets executed.  
Examples: ls, cat etc.

**4 How to runs jobs in background, what is the use**

There are two ways of starting a job in the background – with the shell’s & operator and the

nohupcommand.

**nohup (No Hang Up):** is a command in Linux systems that runs the process even after logging out from the shell/terminal.

Usually, every process in Linux systems is sent a **SIGHUP (Signal Hang UP)** which is responsible for terminating the process after closing/exiting the terminal. Nohup command prevents the process from receiving this signal upon closing or exiting the terminal/shell.

**Syntax:**

nohup command [command-argument ...]

##### **&: No Logging out**

Ordinarily, when the shell runs a command for you, it waits until the command is completed. During this time, you cannot communicate with the shell. You can run a command that takes a  
long time to finish as a background job, so that you can be doing something else. To do this, use the & symbol at the end of the command line to direct the shell to execute the command in the background.

**Syntax : $ sort –o emp.dat emp.dat &**

**5. Explain various ps commands with example**

**ps command** is used to list the currently running processes and their PIDs along with some other information depends on different options

**Syntax –**

**ps [options]**

* **Simple process selection :** Shows the processes for the current shell –

[root@rhel7 ~]# ps

PID TTY TIME CMD

12330 pts/0 00:00:00 bash

21621 pts/0 00:00:00 ps

* **View Processes :** View all the running processes use either of the following option with ps –

[root@rhel7 ~]# ps -A

[root@rhel7 ~]# ps -e

* **View Processes not associated with a terminal :** View all processes except both session leaders and processes not associated with a terminal.

[root@rhel7 ~]# ps -a

PID TTY TIME CMD

27011 pts/0 00:00:00 man

27016 pts/0 00:00:00 less

27499 pts/1 00:00:00 ps

* **View all the running processes :**

[root@rhel7 ~]# ps -r

* **5 View all processes owned by you :** Processes i.e same EUID as ps which means runner of the ps command, root in this case –

[root@rhel7 ~]# ps –x

* **.6 View all the processes except session leaders :**

[root@rhel7 ~]# ps –d

**6 Explain different environment variables**

There are two types of shell variables: local and environment variables. PATH, HOME, SHELL etc. are environment variables, because they are available in user’s environment.

* Accessing Variable values

In order to determine value of a variable, use the command

echo $VARIABLE

* Set New Environment Variables

You can create your own user defined variable, with syntax

VARIABLE\_NAME= variable\_value

* Deleting Variables

syntax can be used to remove a Variable from the system.

unset variablename

1. **Explain read, command line and exit status of a command**

**Read:** The read statement is the shell’s internal tool for making scripts interactive.  It is used with one or more variables

**Syntax:**

read

**example :**echo "what is your name..?";read name;echo "hello $name"

**command line** The Unix shell is used to run commands, and it allows users to pass run time arguments to these commands.These arguments, also known as command line parameters, that allows the users to either control the flow of the command or to specify the input data for the command.

**exit status of a command:** To terminate a program exit is used. Nonzero value indicates an error condition. Example 1:

$ cat foo

1. **Explain Set, shift and trap commands**

**Set:** The set statement assigns positional parameters $1, $2 and so on, to its arguments. This is used for picking up individual fields from the output of a program

Example 1:

$ set 9876 2345 6213

**Shift:** is a builtin command in bash which after getting executed, shifts/move the command line arguments to one position left. The first argument is lost after using shift command. This command takes only one integer as an argument.

**Syntax:**

shift

**Trap : Trap allows you to catch signals and execute code when they occur**. Signals are asynchronous notifications that are sent to your script when certain events occurs

*Unit 5*

1. **Explain the structure of awk and built-in variables and functions of awk**

Awk is a scripting language used for manipulating data and generating reports.

Structuare of awk

**Syntax:**

**awk options 'selection \_criteria {action }' input-file > output-file**

**Options:** 

-f program-file : Reads the AWK program source from the file program-file, instead of from the first command line argument.

-F fs : Use fs for the input field separator

**Example:**

$cat > employee.txt

**BEGIN block** The BEGIN block gets executed at program start-up. It executes only once. This is good place to initialize variables.

**Syntax**

BEGIN {awk-commands}

**Body Block:** The body block applies AWK commands on every input line. By default, AWK executes commands on every line. We can restrict this by providing patterns.

**Syntax**

/pattern/ {awk-commands}

**END Block** The END block executes at the end of the program. END is an AWK keyword and hence it must be in upper-case

**Syntax**

END {awk-commands}

**Built in variebles**

*ARGC*

It implies the number of arguments provided at the command line.

Example

[jerry]$ awk 'BEGIN {print "Arguments =", ARGC}' One Two Three Four

*ENVIRON*

It is an associative array of environment variables.

Example

[jerry]$ awk 'BEGIN { print ENVIRON["USER"] }'

*FS*

It represents the (input) field separator and its default value is space. You can also change this by using -F command line option.

Example

[jerry]$ awk 'BEGIN {print "FS = " FS}' | cat -vte

*NF*

It represents the number of fields in the current record. For instance, the following example prints only those lines that contain more than two fields.

Example

[jerry]$ echo -e "One Two\nOne Two Three\nOne Two Three Four" | awk 'NF > 2'

*NR*

It represents the number of the current record. For instance, the following example prints the record if the current record number is less than three.

Example

[jerry]$ echo -e "One Two\nOne Two Three\nOne Two Three Four" | awk 'NR < 3'

*OFS*

It represents the output field separator and its default value is space.

Example

[jerry]$ awk 'BEGIN {print "OFS = " OFS}' | cat -vte

**Built-in functions**

**Awk int(n) function:** int() function gives us the integer part of the given argument.

**awk log(n) function:** log() function provides natural logarithmic(with base e) of given amount n

**awk sqrt(n) function:** sqrt() function gives the positive root for the given integer n. This function also accepts the positive numbe

**wk length(string) Function:** The length() function calculates the length of a string.

**awk substr(s, p, n) Function:** The length() function is used to extract substring function from a string.

**awk tolower(s) Function:** Translate all uppercase characters in string s to lowercase and returns the new string.

1. **Explain the commands export, eval and exec**

**eval** is a built-in Linux command which is used to execute arguments as a shell command. It combines arguments into a single string and uses it as an input to the shell and execute the commands.

**Syntax**

eval [arg ...]

The **export** command is a built-in utility of Linux Bash shell. It is used to ensure the environment variables and functions to be passed to child processes. It does not affect the existing environment variable.

Syntax:

export [-f] [-n] [name[=value] ...] or export -p

**exec** command in Linux is used to execute a command from the bash itself. This command does not create a new process it just replaces the bash with the command to be executed. If the exec command is successful, it does not return to the calling process.

1. **Syntax:**
2. exec [-cl] [-a name] [command [arguments]] [redirection ...]
3. **Explain if and for control structure of awk**

**If statement**

It simply tests the condition and performs certain actions depending upon the condition. Given below is the syntax of if statement −

**Syntax**

if (condition)

action

***example***

[jerry]$ awk 'BEGIN {num = 10; if (num % 2 == 0) printf "%d is even number.\n", num }'

**If Else Statement**

In if-else syntax, we can provide a list of actions to be performed when a condition becomes false.

**Syntax**

if (condition)

action-1

else

action-2

*Example*

[jerry]$ awk 'BEGIN {

num = 11; if (num % 2 == 0) printf "%d is even number.\n", num;

else printf "%d is odd number.\n", num

}'

**For Loop**

The syntax of for loop is −

***Syntax***

for (initialization; condition; increment/decrement)

action

*Example*

[jerry]$ awk 'BEGIN { for (i = 1; i <= 5; ++i) print i }'

1. **Explain associative array in awk**
2. **Explain awk filter**
3. **Explain Built in operators used in awk**
4. **Explain Begin and End section of awk**