1. What are the Functions of operating system?

Ans:- An operating system (OS) is a software program that manages computer hardware and software resources and provides common services for computer programs. Some of the main functions of an operating system include:

1. Resource management:
2. Process management:
3. Memory management:
4. Device management:
5. File management:
6. User interface:
7. Overall, the operating system serves as a bridge between computer hardware and software, providing a platform for programs to run on and enabling users to interact with the computer system.

2. What is kernel and it's types?

The kernel is a critical component of an operating system that manages system resources and provides a layer of abstraction between applications and hardware. It is responsible for managing system resources such as CPU, memory, and input/output (I/O) devices, and providing essential services to other parts of the operating system.

There are several types of kernels, including:

1. Monolithic kernel:
2. Microkernel
3. Hybrid kernel:
4. Exokernel:
   Nanokernel:
5. Overall, the choice of kernel type depends on the specific needs and requirements of the operating system and the applications it will run.

3. What is deadlock, which are the necessary conditions of deadlock?

In computer science, a deadlock is a situation where two or more processes are blocked and unable to proceed because they are waiting for each other to release resources. This creates a standstill, where none of the processes can progress, and the system becomes unresponsive.

There are four necessary conditions for a deadlock to occur, known as the Coffman conditions:

1. Mutual exclusion: At least one resource must be held in a non-sharable mode. This means that only one process can use the resource at a time, and other processes must wait for it to be released.
2. Hold and wait: A process must be holding at least one resource and waiting to acquire additional resources that are currently held by other processes.
3. No preemption: Resources cannot be forcibly removed from a process that is holding them. A process must release a resource voluntarily, by completing its task or by requesting to release the resource.
4. Circular wait: A cycle of processes exists, where each process is waiting for a resource that is being held by another process in the cycle.

If all four conditions are present, then a deadlock is possible. To prevent deadlock, at least one of these conditions must not hold. For example, if resources can be preempted, then a deadlock can be broken by forcibly removing a resource from a process that is holding it. Alternatively, if circular wait is not possible, then a deadlock cannot occur.

4. What is semaphore?

In computer science, a semaphore is a synchronization object that is used to control access to a shared resource in a multi-process or multi-threaded environment. It is a variable that is used to indicate the status of a shared resource, and it can be accessed and modified only through two atomic operations: wait (P) and signal (V).

The wait operation (also known as "down" or "acquire") decrements the value of the semaphore and blocks the calling process if the resulting value is negative. This indicates that the resource is currently being used by another process, and the calling process must wait until it becomes available.

The signal operation (also known as "up" or "release") increments the value of the semaphore and unblocks a waiting process, allowing it to acquire the resource. If there are multiple processes waiting, then the scheduler determines which process to unblock first.

There are two types of semaphores:

1. Binary semaphore: A binary semaphore has only two possible values, 0 and 1, and it is used to control access to a single resource. It can be used to implement mutual exclusion, where only one process can access the resource at a time.
2. Counting semaphore: A counting semaphore has an integer value that can be greater than or equal to zero, and it is used to control access to a pool of identical resources. It can be used to implement resource allocation, where multiple processes can access the resources as long as they are available.

Semaphores are commonly used in operating systems to implement process synchronization, mutual exclusion, and resource allocation. They are also used in concurrent programming, where multiple threads need to access shared resources in a coordinated manner.

5. **What is a thread process and child process?**

In operating systems, a process is an instance of a program that is running on a computer, and it consists of the program code, data, and resources (such as memory, files, and network connections). A process can be divided into one or more threads, which are individual execution units that share the same resources.

A thread is a lightweight process that can run concurrently with other threads within the same process. Each thread has its own stack and program counter, but it shares the same memory, file descriptors, and other resources with other threads in the same process. Threads can communicate and synchronize with each other using synchronization primitives such as mutexes and condition variables.

A child process is a process that is created by another process (known as the parent process). The child process inherits the resources and environment of the parent process, but it runs independently and can have its own threads and processes. The child process can also communicate with the parent process using inter-process communication mechanisms such as pipes or sockets.

In summary, a process is an instance of a program that can have one or more threads, while a thread is a lightweight process that can run concurrently with other threads within the same process. A child process is a process that is created by another process, and it inherits the resources and environment of the parent process

Thread processes are commonly used in applications that require parallelism and multitasking, such as web servers, multimedia applications, and scientific simulations.

Child processes can be used for a variety of purposes, such as executing a separate program, handling a specific task, or running in the background.

6. **What are the File Locking systems using Semaphore?**

In operating systems, file locking is a mechanism that prevents multiple processes or threads from accessing the same file simultaneously. It is used to ensure data consistency and avoid conflicts when multiple processes are reading or writing to the same file.

One way to implement file locking is to use semaphores. Semaphores are synchronization objects that can be used to control access to a shared resource. In the context of file locking, semaphores are used to coordinate access to a file between multiple processes or threads.

There are two common file locking systems that use semaphores:

1. Record locking: In record locking, the file is divided into multiple records, and each record is locked independently. A semaphore is used to coordinate access to each record. When a process wants to access a record, it waits for the semaphore associated with that record to become available. Once it has the semaphore, it can read or write to the record. When it is done, it releases the semaphore so that another process can access the same record.
2. File locking: In file locking, the entire file is locked instead of individual records. A semaphore is used to indicate whether the file is currently locked or not. When a process wants to access the file, it first checks the semaphore. If the semaphore is available, it locks the semaphore and accesses the file. When it is done, it releases the semaphore so that another process can access the file.

Function :-

In addition, functions can be used to abstract away implementation details and expose only the interface to the caller. This makes it easier for the caller to use the function and hides the complexity of the implementation. Functions can also be used to encapsulate state and behavior, providing a clean separation between different parts of the program.

Library Function:-

In computer science, a library function (also known as a built-in function or standard function) is a pre-written block of code that is part of a software library and performs a specific task or set of tasks. Library functions are provided by the operating system or by third-party libraries, and they are designed to be reusable across different programs.

Library functions are often used to perform common operations such as input/output, string manipulation, math operations, and memory management. Examples of library functions include printf() for printing output to the console, strcpy() for copying strings, and malloc() for allocating memory dynamically.

System Call

In computer science, a system call is a request made by a program to the operating system kernel for a service or resource that it cannot access on its own. System calls provide an interface between the user space and kernel space of an operating system, allowing user-level programs to interact with the hardware and resources of the system.

Examples of system calls include reading or writing to files, creating or deleting processes, allocating memory, and controlling devices such as printers and network interfaces. System calls are typically invoked by programs through high-level language constructs such as functions or methods.

When a program makes a system call, the operating system switches the program's execution from user mode to kernel mode, which provides privileged access to the system's resources. The operating system then executes the requested service or resource and returns control to the user program.

System calls are a critical component of an operating system's functionality, allowing programs to interact with the system's hardware and resources in a controlled and secure manner. They provide a level of abstraction that shields the user program from the complexities of the underlying hardware and operating system, allowing programs to be written in high-level languages and run on a wide range of systems.

Overall, system calls are a fundamental concept in operating systems and computer science, providing a bridge between user-level programs and the kernel, enabling the creation of complex and powerful software systems.

Kernel Mode

In case you meant "kernel mode", it refers to a privileged mode of operation in a computer's CPU where the operating system kernel runs. In this mode, the kernel has direct access to the computer's hardware and can perform operations that are not permitted in user mode. Kernel mode is used to implement critical functions such as device drivers, memory management, and process scheduling.

Kernel mode provides a secure and controlled environment in which the operating system can execute and manage the computer's hardware and resources. Access to kernel mode is restricted to trusted system processes, ensuring that only authorized programs can access and modify the system's critical resources.

Fork System call

In computer science, the fork system call is a mechanism provided by the operating system that allows a process to create a copy of itself. When a process calls the fork system call, the operating system creates a new process, called the child process, that is an exact copy of the parent process, including all of its memory, file descriptors, and other resources.

The child process and parent process then continue to execute independently, with separate memory spaces and execution paths. The child process is assigned a new process ID (PID) and inherits its parent's working directory, environment variables, and open file descriptors.

The fork system call is commonly used in Unix-like operating systems to create new processes and perform tasks such as parallel processing and daemonization. After forking, the child process can execute a different program or code segment from the parent process using the exec system call.

Procedural call

The procedural call mechanism involves several steps. First, the program pushes the current execution state (including the current instruction pointer and any relevant data) onto a call stack. Then, it updates the instruction pointer to point to the entry point of the procedure or function being called. The program may also pass parameters or arguments to the procedure or function at this time.

When the procedure or function has completed its task, it returns control to the calling program by popping the saved execution state

from the call stack and updating the instruction pointer to the location immediately following the original procedural call.

System Program

In computer science, a system program is a type of software that is designed to help manage and control the operation of a computer system. Unlike application programs, which are designed to perform specific tasks for end users, system programs work behind the scenes to support and enhance the functionality of the operating system and hardware.

There are many types of system programs, including:

1. Operating system utilities: These are programs that are part of the operating system and are used to manage system resources such as memory, processes, and files.
2. Device drivers: These are programs that enable the operating system to communicate with hardware devices such as printers, scanners, and storage devices.
3. System libraries: These are collections of prewritten code that are used by application programs to perform common tasks such as input/output operations, string manipulation, and networking.
4. System maintenance and administration tools: These are programs that are used to manage and monitor the operation of the system, including tasks such as backups, performance monitoring, and security management.

API

In computer science, an Application Programming Interface (API) is a set of protocols, routines, and tools for building software

applications. An API specifies how software components should interact, and it allows developers to build applications by using pre-built components or services.

APIs can be thought of as a bridge that connects software components, allowing them to interact and exchange data in a standardized way. For example, an API may define how an application can access a web service, retrieve data from a database, or communicate with a hardware device.

APIs can be classified into several types, including:

1. Web APIs: These APIs are used to build web applications and typically use HTTP protocols to enable communication between web services.
2. Operating System APIs: These APIs are used to build applications that interact with the operating system, including file management, networking, and graphics.
3. Database APIs: These APIs are used to build applications that interact with databases, allowing developers to perform operations such as querying, inserting, updating, and deleting data.
4. Library APIs: These APIs are used to build applications that use pre-built code libraries or frameworks to perform specific tasks.

Write down the uses of pipe command and write a c program that illustrates how to execute two commands concurrently with a command pipe.

The pipe command in Unix/Linux is used to connect the output of one command to the input of another command. It allows two or more commands to be executed concurrently, with the output of one command becoming the input of the next command.
For example, the command "ls l grep txt" would first execute the "ls" command to list all files in the current directory, and then pipe the

output to the "grep txt" command, which would filter the output to only show files containing the string “txt".

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
   int fd[2];
   pid_t pid;

   char *cmd1[] = {"ls", "-l", NULL};
   char *cmd2[] = {"grep", ".txt", NULL};

   if (pipe(fd) == -1) {
      fprintf(stderr, "Pipe failed");
      return 1;
   }

   pid = fork();

   if (pid < 0) {
      fprintf(stderr, "Fork failed");
      return 1;
   }

   if (pid == 0) {
      // Child process - execute cmd2
      close(fd[1]); // Close write end of pipe
      dup2(fd[0], STDIN_FILENO); // Redirect input from read end of
pipe
      close(fd[0]); // Close read end of pipe
      execvp(cmd2[0], cmd2);
      fprintf(stderr, "Command execution failed");
      exit(1);
   } else {
      // Parent process - execute cmd1
```

```
        close(fd[0]); // Close read end of pipe
        dup2(fd[1], STDOUT_FILENO); // Redirect output to write end of
pipe
        close(fd[1]); // Close write end of pipe
        execvp(cmd1[0], cmd1);
        fprintf(stderr, "Command execution failed");
        exit(1);
    }

    return 0;
}
```

What is Shell?

In computer science, a shell is a command-line interface that allows users to interact with an operating system. A shell provides a command prompt where users can enter commands to perform various tasks, such as navigating the file system, launching applications, and executing scripts.

The shell is often referred to as a "command interpreter" because it interprets the commands entered by the user and translates them into actions that can be performed by the operating system. The shell also provides various features such as scripting, environment variables, and command history, which make it a powerful tool for managing and automating system tasks.

There are several types of shells available, including:

1. Bourne shell (sh): The original Unix shell, which provides a simple and efficient command-line interface.
2. C shell (csh): A shell with a syntax similar to the C programming language, which provides additional features such as history, command-line editing, and job control.
3. Bourne-Again shell (bash): A popular Unix shell that is an extension of the Bourne shell, providing additional features such as command-line editing, shell scripting, and job control.

4. Z shell (zsh): A Unix shell that is an extension of the Bourne shell, providing additional features such as advanced command-line editing, spelling correction, and globbing.

What is shell scripting

Shell scripting is the process of creating scripts or programs using a shell, which is a command-line interface for interacting with an operating system. Shell scripts are essentially text files containing a series of commands that can be executed by the shell, either manually or as part of an automated process.

Shell scripting is commonly used for system administration tasks, such as managing files and directories, configuring system settings, and performing backups and maintenance tasks. It is also used for software development tasks, such as building and testing software, deploying applications, and managing software environments.

Some of the key features of shell scripting include:

1. Command execution: Shell scripts can execute any command that can be run from the command line, including system commands, shell commands, and other scripts.
2. Variables: Shell scripts can define and use variables to store and manipulate data.
3. Control structures: Shell scripts support a range of control structures, such as loops, conditionals, and functions, that allow for more complex scripting tasks.
4. I/O redirection: Shell scripts can redirect input and output streams to and from files or other commands, allowing for more flexible data processing.
5. Error handling: Shell scripts can handle errors and exceptions using conditional statements and exit codes.

Shell programs are stored in which file?

Shell programs are typically stored in text files with a ".sh" file extension. This is not a requirement, however, as shell scripts can be stored in files with any extension or even no extension at all, as long as the file has execute permissions and begins with the appropriate "shebang" line.

The "shebang" line is a special directive at the beginning of a shell script that tells the operating system which interpreter to use to execute the script. For example, the shebang line "#!/bin/bash" at the beginning of a script tells the system to use the Bash shell to execute the script.

Once the script is saved with the appropriate file extension and the shebang line is added, it can be executed by typing the name of the script file into the shell prompt or by specifying the path to the script file if it is not in the current working directory.

What are the different types of Shells available?

There are several types of shells available in Unix-based operating systems. Some of the most common shells are:

1. Bourne Shell (sh): This is the original Unix shell, and it provides a simple and efficient command-line interface. It is the basis for many other shells, such as Bash.
2. Bourne-Again Shell (bash): This is the most popular Unix shell and an extension of the Bourne shell, providing additional features such as command-line editing, shell scripting, and job control. It is the default shell in most Linux distributions.
3. C Shell (csh): This shell has a syntax similar to the C programming language and provides additional features such as history, command-line editing, and job control.
4. Korn Shell (ksh): This shell is an extension of the Bourne shell, providing additional features such as command-line editing, shell scripting, and job control.
5. Z Shell (zsh): This shell is similar to Bash but provides additional features such as advanced command-line editing, spelling correction, and globbing.
6. Fish Shell (fish): This shell is designed to be user-friendly and easy to use, providing features such as syntax highlighting, autosuggestions, and abbreviated commands.

What are the advantages of C Shell over Bourne Shell?

The C Shell (csh) and the Bourne Shell (sh) are two popular Unix shells, each with its own advantages and disadvantages. Some of the advantages of C Shell over Bourne Shell are:

1. Command history: C Shell has built-in support for command history, allowing you to easily recall and reuse previously executed commands.
2. Command-line editing: C Shell provides advanced command-line editing capabilities, allowing you to easily edit and manipulate command lines before executing them.
3. Interactive shell scripting: C Shell is designed to be more interactive and user-friendly, making it easier to write and test shell scripts.
4. Job control: C Shell provides powerful job control features, allowing you to easily manage and manipulate background processes.
5. Aliases: C Shell allows you to define aliases, which are shorthand commands that can be used to execute longer or more complex commands.
6. Environment variables: C Shell provides more flexible and powerful environment variable handling, allowing you to easily customize and configure your shell environment.

What is the importance of writing Shell Scripts?

Shell scripts are a series of commands and instructions that are written in a scripting language and executed by a shell. They are important for a variety of reasons, including:

1. Automation: Shell scripts can automate repetitive tasks and reduce the amount of time and effort required to perform them. This is especially useful for tasks that need to be performed regularly or for complex tasks that require multiple steps.
2. Efficiency: Shell scripts can increase efficiency by allowing users to perform multiple tasks with a single command. This is

useful when working with large data sets or when performing complex calculations.
3. Portability: Shell scripts can be executed on different operating systems and platforms without the need for modification, which makes them highly portable. This is especially useful in multi-platform environments.
4. Debugging: Shell scripts can be used for debugging and testing code. This is because they allow users to easily run and test individual commands or sections of code.
5. Customization: Shell scripts can be customized to meet the specific needs of the user. This means that users can create scripts that perform tasks in a way that is most efficient and effective for their workflow.

In a typical UNIX environment how many kernels and shells are available?

In a typical UNIX environment, there is typically only one kernel, which is the core of the operating system that manages system resources and communicates with hardware. However, there can be multiple shells available to the user. The shell is a command-line interface that allows users to interact with the system by entering commands.

There are many different shells available for UNIX-based systems, but some of the most common ones are:

1. Bourne shell (sh)
2. Bourne-Again shell (bash)
3. C shell (csh)
4. Korn shell (ksh)
5. Z shell (zsh)

Different shells have different features, syntax, and capabilities, so users can choose the one that best suits their needs and preferences. In addition, users can switch between shells as needed or use multiple shells simultaneously.

Is separate compiler required for executing a shell program?

No, a separate compiler is not required for executing a shell program. A shell program is a script written in a shell scripting language such as Bash, and it can be executed directly by the shell interpreter without the need for compilation.

When a user runs a shell program, the shell interpreter reads and executes the commands in the script line by line, in the same way that a user would enter commands directly into the shell.

This is one of the advantages of shell scripting: it allows users to automate tasks and perform complex operations without the need for compiling, linking, or building an executable binary file.

How many shell scripts come with UNIX operating system?

There are approximately 280 shell scripts that come with the UNIX operating system.

When should shell programming/scripting not be used?

- When the task is very much complex like writing the entire payroll processing system.
- Where there is a high degree of productivity required.
- When it needs or involves different software tools.

Basis of shell program relies on what fact?

The basis of shell programming relies on the fact that the UNIX shell can accept commands not just only from the keyboard but also from a file.

What are the default permissions of a file when it is created?

666 i.e. rw-rw-rw- is the default permission of a file, when it is created.

What can be used to modify file permissions?

File permissions can be modified using **umask**.

How to accomplish any task via shell script?

Any task can be accomplished via shell script at the dollar ($) prompt and vice versa.

What is umask?

The umask is a value that determines which permission bits are turned off by default for newly created files or directories. The default umask value is typically 022 for user accounts, which means that the write permission is turned off for the group and others, while all permissions are enabled for the user who created the file.

What are Shell Variables?

Shell variables are the main part of shell programming or scripting. They mainly provide the ability to store and manipulate information within a shell program.

What are the two types of Shell Variables? Explain in brief.

The two types of shell variables are:

#1) UNIX Defined Variables or System Variables – These are standard or shell defined variables. Generally, they are defined in CAPITAL letters.

Example: SHELL – This is a Unix Defined or System Variable, which defines the name of the default working shell.

#2) User Defined Variables – These are defined by users. Generally, they are defined in lowercase letters

Example: $ a=10 –Here the user has defined a variable called ‘a’ and assigned value to it as 10.

How are shell variables stored? Explain with a simple example.

Shell variables are stored as string variables.

Example: $ a=10

In the above statement a=10, the 10 stored in ‘a’ is not treated as a number, but as a string of characters 1 and 0.

What is the lifespan of a variable inside a shell script?

The lifespan of a variable inside shell script is only until the end of execution.

How to make variables as unchangeable?

Variables can be made unchangeable using **readonly**. For instance, if we want variable '**a**' value to remain as **10** and not change, then we can achieve this using **readonly**.

Example:
$ a=10
$ readonly a

How variables can be wiped out?

Variables can be wiped out or erased using the **unset** command.

Example:
$ a =20
$ unset a

Upon using the above command the variable '**a**' and its value **20** get erased from shell's memory.

What are positional parameters? Explain with an example.

Positional parameters are the variables defined by a shell. And they are used whenever we need to convey information to the program. And this can be done by specifying arguments at the command line.

There is a total of 9 positional parameters present i.e. from $1 to $9.

Example: $ Test Indian IT Industry has grown very much faster
In the above statement, positional parameters are assigned like this.
$0 -> Test (Name of a shell program/script)
$1 ->Indian
$2 -> IT and so on.

What does the. (dot) indicate at the beginning of a file name and how should it be listed?

A file name that begins with a. (dot) is called as a hidden file. Whenever we try to list the files it will list all the files except hidden files.

But, it will be present in the directory. And to list the hidden file we need to use –a option of ls. i.e. $ ls –a.

What is a file system?

The file system is a collection of files that contain related information of the files.

Explain about file permissions.

There are 3 types of file permissions as shown below:

Permissions	Weight
r – read	4
w – write	2
x - execute	1

The above permissions are mainly assigned to owner, group and to others i.e. outside the group. Out of 9 characters first set of 3 characters decides/indicates the permissions which are held by the owner of a file. The next set of 3 characters indicates the permissions for the other users in the group to which the file owner belongs to.

And the last 3 sets of characters indicate the permissions for the users who are outside the group. Out of the 3 characters belonging to each set, the first character indicates the “read” permission, the second character indicates “write” permission and the last character indicates “execute” permission.

What are the different blocks of a file system? Explain in brief.

- **Super Block(2nd block)**: This block mainly tells about a state of the file system like how big it is, maximum how many files can be accommodated, etc.
- **Boot Block(1st)**: This represents the beginning of a file system. It contains the bootstrap loader program, which gets executed when we boot the host machine.
- **Inode Table(3rd)**: As we know all the entities in a UNIX are treated as files. So, the information related to these files is stored in an Inode table.
- **Data Block(4th)**: This block contains the actual file contents.

What are the three modes of operation of vi editor? Explain in brief.

The three modes of operation of **vi editors** are,

1. **Command Mode**: In this mode, all the keys pressed by a user are interpreted as editor commands.
2. **Insert Mode**: This mode allows for the insertion of a new text and editing of an existing text etc.
3. **The ex-command Mode**: This mode allows a user to enter the commands at a command line.

What is the alternative command available to echo and what does it do?

tput is an alternative command to **echo**.

Using this, we can control the way in which the output is displayed on the screen.

Q #33) How to find out the number of arguments passed to the script?

The number of arguments passed to the script can be found by the below command.

echo $ #

What are control instructions and how many types of control instructions are available in a shell? Explain in brief.

Answer: Control Instructions are the ones, which enable us to specify the order in which the various instructions in a program/script are to be executed by the computer. Basically, they determine a flow of control in a program.

There are 4 types of control instructions that are available in a shell.

- **Sequence Control Instruction**: This ensures that the instructions are executed in the same order in which they appear in the program.
- **Selection or Decision Control Instruction**: It allows the computer to take the decision as to which instruction is to be executed next.
- **Repetition or Loop Control Instruction**: It helps a computer to execute a group of statements repeatedly.
- **Case-Control Instruction**: This is used when we need to select from several alternatives.

What is IFS?

Answer: IFS stands for Internal Field Separator. And it is one of the system variables. By default, its value is space, tab, and a new line. It signifies that in a line where one field or word ends and another begins.

What are Metacharacters in a shell? Explain with some examples.

Answer: Metacharacters are special characters in a program or data field which provides information about other characters. They are also called, regular expressions in a shell.

Example:

ls s* – It lists all the files beginning with character 's'.

Execution over Shell Interpreter/Editor

```
/u/user1/Shell_Scripts2017> ls s*
```

Output:

```
script1  script2
```

$ cat script1 > script2 – Here output of cat command or script1 will go to a script2.

Execution over Shell Interpreter/Editor

```
/u/user1/Shell_Scripts2017> cat script1 > script2
```

Output:

```
#!/bin/bash
# script1
# Usage: script1
echo Hello !!
echo How are you?
script2  #here it calls the script2
pwd
```

$ ls; who – This will execute ls first and then who.

Execution over Shell Interpreter/Editor

```
/u/user1/Shell_Scripts2017> ls; who
```

Output:

```
script10                  script5
Script2                   script6
```

```
tibadm      pts/1         Sep 14 08:22
crmadm      pts/2         Sep 14 08:43
```

How to execute multiple scripts? Explain with an example.

Answer: In a shell, we can easily execute multiple scripts i.e. one script can be called from the other. We need to mention the name of a script to be called when we want to invoke it.

Example: In the below program/script upon executing the first two echo statements of script1, shell script executes script2. Once after executing script2, the control comes back to script1 which executes a **pwd** command and then terminates.

Code for script1

```
#!/bin/bash
# script1
# Usage: script1
echo Hello !!
echo How are you?
script2  #here it calls the script2
pwd
```

Code for script2

```
#script2
echo Software testing is an interesting job.
```

Execution of script1 over Shell Interpreter/Editor

```
/u/user1/Shell_Scripts2017> script1
```

Output displayed on the Editor upon executing script1

```
Hello !!
How are you?
Software testing is an interesting job.
/u/user1/Shell_Scripts_2017
```

Q #41) Which command needs to be used to know how long the system has been running?

Answer: uptime command needs to be used to know how long the system has been running.

Example: $ uptime

On entering the above command at shell prompt i.e. $ uptime, the output should look like this.

9:21am up 86 day(s), 11:46, 3 users, load average: 2.24, 2.18, 2.16

Execution over Shell Interpreter/Editor

```
/u/user1/Shell_Scripts_2017> uptime
```

Output:

```
9:21am  up 86 day(s), 11:46,  3 users,  load average: 2.24, 2.18, 2.16
```

Q #42) How to find the current shell which you are using?

Answer: We can find the current shell that we are using with echo $SHELL.

Example: $ echo $SHELL

Execution over Shell Interpreter/Editor

```
$ echo $SHELL
```

Output:

```
/bin/bash
```

Q #43) How to find all the available shells in your system?

Answer: We can find all the available shells in our system with $ cat /etc/shells.

Example: $ cat /etc/shells

Execution over Shell Interpreter/Editor

```
$ cat /etc/shells
```

Output:

```
/bin/sh
/bin/bash
/sbin/nologin
/bin/ksh
/bin/dash
/bin/tcsh
/bin/csh
```

Q #44) How to read keyboard inputs in shell scripts?

Answer: Keyboard inputs can be read in shell scripts as shown below,

Script/Code

```
#!/bin/bash
#script6
read name
echo "Hello $name"
```

Execution over Shell Interpreter/Editor

```
/u/user1/Shell_Scripts_2017> script6
```

Output:

```
Mahesh
Hello Mahesh
```

Q #45) How many fields are present in a crontab file and what does each field specify?

Answer: The **crontab** file has six fields. The first five fields tell **cron** when to execute the command: minute(0-59), hour(0-23), day(1-31), month(1-12), and day of the week(0-6, Sunday = 0).

And the sixth field contains the command to be executed.

Q #46) What are the two files of crontab command?

Answer: Two files of crontab command are:

- **cron.allow** – It decides which users need to be permitted from using crontab command.
- **cron.deny** – It decides which users need to be prevented from using crontab command.

Q #47) What command needs to be used to take the backup?

Answer: tar is the command which needs to be used to take the backup. It stands for tape archive. The **tar** command is mainly used to save and restore files to and from an archive medium like tape.

Q #48) What are the different commands available to check the disk usage?

Answer: There are three different commands available to check the disk usage.

They are:

- **df** – This command is used to check the free disk space.
- **du** – This command is used to check the directory wise disk usage.
- **dfspace** – This command is used to check the free disk space in terms of MB.

Q #49) What are the different communication commands available in Unix/ Shell?

Answer: Basically, there are 4 different communication commands available in Unix/Shell. And they are mail, news, wall & motd.

Q #50) How to find out the total disk space used by a specific user, say for example username is John?

Answer: The total disk space used by John can be found out as:

du –s/home/John

Q #51) What is Shebang in a shell script?

Answer: Shebang is a # sign followed by an exclamation i.e. !. Generally, this can be seen at the beginning or top of the script/program. Usually, a developer uses this

to avoid repetitive work. Shebang mainly determines the location of the engine which is to be used in order to execute the script.

Here '#' symbol is called hash and '!' is called a bang.

Example: #!/bin/bash

The above line also tells which shell to use.

Q #52) What is the command to be used to display the shell's environment variables?

Answer: Command to be used to display the shell's environment variables is **env** or **printenv**.

Q #53) How to debug the problems encountered in shell script/program?

Answer: Though it depends on the type of problem encountered. Given below are some common methods used to debug the problems in the script.

- Debug statements can be inserted in the shell script to output/display the information which helps to identify the problem.
- Using "set -x" we can enable debugging in the script.

How to open a read-only file in Unix/shell?

Answer: Read-only file can be opened by:
vi –R <File Name>

Q #57) How can the contents of a file inside jar be read without extracting in a shell script?

Answer: The contents of the file inside a jar can be read without extracting in a shell script as shown below.
tar –tvf <File Name>.tar

Q #58) What is the difference between diff and cmp commands?

Answer: diff – Basically, it tells about the changes which need to be made to make files identical.

cmp – Basically it compares two files byte by byte and displays the very first mismatch.

Q #59) Explain in brief about sed command with an example.

Answer: sed stands for **stream editor**. And it is used for editing a file without using an editor. It is used to edit a given stream i.e. a file or input from a pipeline.

Syntax: sed options file

Example:

Execution over Shell Interpreter/Editor

```
/u/user1/Shell_Scripts_2017> echo "Hello World" | sed 's/Hello/Hi/'
```

Here '**s**' command present in **sed** will replace string **Hello** with **Hi**.

Output:

```
Hi World
```

Q #60) Explain in brief about awk command with an example.

Answer: awk is a data manipulation utility or command. Hence, it is used for data manipulation.

Syntax: awk options File Name

Example:

Script/Code

```
/u/user1/Shell_Scripts_2017> cat > script10
```

```
Hello John
Hello Richard
Hello Kevin
Hello Mike
Hello Robert
```

awk utility/command assigns variables like this.
$0 -> For whole line (e.g. Hello John)
$1 -> For the first field i.e. Hello
$2 -> For the second field

Execution over Shell Interpreter/Editor

```
awk '{print $0}' script10
```

The above script prints all the 5 lines completely.

Output:

```
Hello John
Hello Richard
Hello Kevin
Hello Mike
Hello Robert
```

Execution over Shell Interpreter/Editor

```
awk '{print $1}' script10
```

The above script prints only the first word i.e. Hello from each line.

Output:

```
Hello
Hello
Hello
Hello
Hello
```