Linux Internals

1 Ans In a shell script, you can read i/p from the user using the 'read' command. The 'read' command allows you to prompt the user for input and store their response in a variable.

Eg :- #!/bin/bash

read -p "Enter your name: " user_name

echo "Hello, $user_name! Welcome to the shell scripting world."

2 Ans You can access the command line arguments when executing a Bash script using special variables. These variables are named '$1', '$2', '$3' and so on, where '$1' represents the first argument, '$2' the second, and so forth. You can also use '$#' to get the total number of arguments and '$@' to represent all the arguments as an array.

Eg :- script named - args_example.sh

#!/bin/bash

echo "The name of the script is : $0"

echo "The first argument is : $1"

echo "Second " " : $2"

echo "third " " : $3"

echo "Total number of arguments: $#"

echo "All arguments as an array : $@"

Linux Internals

1 Ans In a shell script, you can read i/p from the user using the 'read' command. The 'read' command allows you to prompt the user for input and store their response in a variable.

Eg :- #!/bin/bash

read -p "Enter your name: " user_name

echo "Hello, $user_name! Welcome to the shell scripting world."

2 Ans You can access the command line arguments when executing a Bash script using special variables. These variables are named '$1', '$2', '$3' and so on, where '$1' represents the first argument, '$2' the second, and so forth. You can also use '$#' to get the total number of arguments and '$@' to represent all the arguments as an array.

Eg :- script named - args_example.sh

#!/bin/bash

echo "The name of the script is : $0"

echo "The first argument is : $1"

echo "Second " " : $2"

echo "third " " : $3"

echo "Total number of arguments : $#"

echo "All arguments as an array : $@"

3 Ans You can redirect the output of a script to a file using '>' operator in the command line. The '>' operator is used to redirect standard output to a file. If the file already exists, it will be overwritten. If the file doesn't exists, it will be created.

Eg:→ Script name - 'my_script.sh'

```
#!/bin/bash
echo "This is some output from my script"
echo "Redirecting this to a file".
```

→ redirect the script's o/p to a file named output.txt

```
./my_script.sh > output.txt
```

→ You can view the contents of 'output.txt' with a command like "Cat output.txt".

4 Ans Environment variables in Bash scripting are variables that are set in the shell's environment and can be accessed by processes and scripts running within that environment. These variables store information about system, user preferences, and other runtime settings. Environmental variables are typically in uppercase letters by convention.

→ Commonly used environment variables are:

PATH
HOME
USER
SHELL
PWD

Eg:- #!/bin/bash

```
command_to_check="ls"
if command -v $command_to_check &> /dev/null; then
    echo "$command_to_check is available in your PATH."
else
    echo "$command_to_check is not found in your PATH."
fi
```

5-Ans

Process	Threads
* Program under execution is called process. A process is an independent program with its own memory space & resources.	* A thread is a smallest unit of a process, & it shares the same memory space and resources with other threads within same process.
* Process have higher overhead in terms of memory & resources since each process has its own memory space and resources.	* Threads have lower overhead because they share the same memory space and resources within the parent process.
* Creating and terminating processes is slower and more resource intensive compared to threads.	* Creating and terminating threads is faster and more efficient.
* Processes are more fault-tolerant because a failure in one process is less likely to affect others.	* Threads are less fault-tolerant because a bug in one thread can potentially crash the entire process.

6 Ans

User level threads	Kernel level threads
* Managed by the user-space and not the operating system Kernel	* Managed by operating System Kernel.
* Thread management is performed using a user-level thread library without Kernel support	* Thread management is performed by the Kernel support
* Light weight & efficient, as thread creation, synchronization and scheduling are handled at user level.	* More heavy weight compared to ULTs because the OS has more control over their execution

7 Ans The 'lseek' function in C is used to set the file offset for a given file descriptor. To set the file offset to the beginning of a file, you should use an offset of 0.

8 Ans The lseek function is used to set the file offset for a given file descriptor. It allows you to move read/write position within a file. The function is defined as follows:

→ off_t lseek (int fd, off_t offset, int whence);

fd - identifies the open file

offset - specified how much to move the file offset, relative to the reference point.

Whence - determines the reference point for the offset movement.

Ans File descriptors are a fundamental concept in Unix-like operating systems, including Linux. They play a fundamental role in I/O operations. They are closely related to the 'open', 'read' & 'write' system calls.

Open - used to obtain a file descriptor when opening or creating file.

read and write system calls to perform input and output operations on file associated with that descriptor.

Ans

Process	Threads
1) Any program is in execution	1) Thread means a segment of process
2) The process takes more time to terminate	2) Thread takes less time to terminate
3) It takes more time for Creation	3) It takes less time for Creation
4) It takes more time for Context Switching	4) It takes less time for Context Switching
5) The process is isolated	5) Thread share Memory
6) Process is called heavy - weight process	6) Thread is lightweight
7) System call is involved in it	7) No system call is involved, It is created using API
8) The process doesnot share data with each other	8) Thread share data with each other

11) Ans :- A Makefile automates the compilation process. It defines rules for compiling source files into Object files and linking them into an executable. Developers can simply run it to build the entire project; makefile ensures that only the necessary files are recompiled when source code changes

12) A :- A Makefile Consists of three sections target, dependencies & rule

The target is normally either an executable or Object file name. The dependencies are source code or other things needed to make the target. The rules are the command needed to make the target

Components of the Makefile

1) Explicit rules
2) Implicit rules
3) definitions
4) Directives
5) comments

13) A :- 13) Describe the purpose of the make clean target in a makefile and How is it used clean a Project directory

Ans :- It is allowed you to type make clean at the command line to get rid of your Object & executable files. Sometimes the Compiler will link or Compile file incorrectly and the only way to get a fresh start is to remove all the Object & Executable files.

To use make clean to clean a project directory to follow these steps

1) To create a Makefile
2) Define the target file
3) Run make clean
4) Customize the clean target

16) What is the difference between SIGTERM and SIGKILL signal

A

SIGTERM	SIGKILL
1) Gracefully kills the process	1) Kills the process immediately
2) Can be handled, ignored & blocked	2) Cannot be handled or blocked
3) Does not kill the child process	3) Kills the child process as well
4) To send SIGTERM signal use the kill command kill <process_id>	4) To send SIGKILL signal use option -9 with the kill command kill -9 <process-id>

17) What signal is typically generated when a program encounters a segmentation fault and why it is important

A) When a program encounters a segmentation fault that typical signal that is generated is 'SIGSEGV'.
A segmentation fault occurs when a program tries to access a memory location that it is not allowed to access

The importance of SIGSEGV in this context lies in its role as its safety Mechanism for detecting & handling memory access.

1) Safety
2) Diagnosis
3) Preventing Crashes
4) Predictable Termination

18) What is the 'SIGINT' signal, and how it is commonly generated?

Ans The 'SIGINT' signal short for "Interrupt signal" It has the signal number 2. The primary purpose of 'SIGINT' signal is to interrupt or terminate a running process in a controlled and graceful manner

The most common way to generate the 'SIGINT' signal is by pressing Cntl+C when a user presses Cntl C the terminal it sends the 'SIGINT' signal to that process. Resulting in the interruption (or) termination of process.

19) What is the TCP/Stream Socket?

A TCP (Transmission Control Protocol) Stream Socket is a type of network socket used in Computer Networking to establish a reliable, connection oriented communication channel between two devices over a tcp network, such as Internet. This type of socket is commonly used for data exchange between a client and a server.

Characteristics of TCP/Stream Socket.

1) Reliability
2) Connection Oriented
3) Full duplex Communication
4) Stream based
5) Flow Control
6) Port number
7) widely used

15) What is a signal in the Context of Unix-like Operating System & show is it typically generated.

A) A signal is used to notify a process of an synchronous or asynchronous event. when a signal is sent, the Operating System interrupts the target process normal flow of execution to deliver the signal.

Signals are typically generated in several ways:

1. User-initiated actions
2. Error conditions
3. System calls and functions
4. Timers and Alarms
5. External hardware events
6. other processes.

10) Explain the ... O_TRUNC flags when using the open system call.

* The O_CREAT flag is used to create a new file if it does not exist. If the file already exists, it has no effect on the existing file content.

* The O_TRUNC (Truncate) flag is used to truncate an existing file, removing its content, or to create a new file & open it for writing, effectively starting with an empty file.

C - programs

① Decimal to Binary

```
#include <stdio.h>
void dec to Binary (int n)
{
 int binaryNum[32];
 int i=0;
 while (n>0) {
  binaryNum[i] = n%2;
    n = n/2;
    i++;
  }
 for int (j=i-1; j>=0; j--)
 Printf ("%d", binaryNum[j]);
}
 int main()
 {
  int n=19;
```

10) Explain the role of the O_CREAT and O_TRUNC flags when using the open system call.

* The O_CREAT flag is used to create a new file if it does not exist. If the file already exists, it has no effect on the existing file content.

* The O_TRUNC (Truncate) flag is used to truncate an existing file, removing its content, or to create a new file & open it for writing, effectively starting with an empty file.

C - programs

(11) # Decimal to Binary

```
#include <stdio.h>
void dec to Binary (int n)
{
 int binaryNum[32];
 int i=0;
 while (n>0) {
   binaryNum[i] = n%2;
    n = n/2;
    i++;
  }
 for int (j=i-1; j>=0; j--)
  Printf ("%d", binaryNum[j]);
}
 int main()
 {
  int n=13;
```

10) Explain the role of the [redacted] and O_TRUNC flags when using the open system call.

* The O_CREAT flag is used to create a new file if it does not exist. If the file already exists, it has no effect on the existing file content.

* The O_TRUNC (TRuncate) flag is used to truncate an existing file, removing its content, or to create a new file & open it for writing, effectively starting with an empty file.

C - programs

(10) Decimal to Binary

```
#include <stdio.h>
Void dec to Binary (int n)
{
 int binaryNum[32];
 int i=0;
 while (n>0) {
  binary Num[i] = n%2;
    n = n/2;
    i++;
  }
 for int (j=i-1; j>=0; j--)
  Printf ("%d", binaryNum[j]);
}
 int main()
 {
```