SYSTEM PROGRAMMING - LINUX

(Attempt all questions)

LINUX UTILITIES AND INSTALLATION

1. Explain the role of the bootloader in the Linux boot process. Provide a specific example of a common Linux bootloader. (4 marks)

2. Describe two common methods for accessing the command-line interface (CLI) in a typical Linux desktop environment. (4 marks)

3. Using the command line, how would you create a new directory named "reports" in your home directory? Also, how would you then navigate into this newly created directory? Provide the exact commands. (4 marks)

4. Explain the meaning of the following Linux file permissions: -rw-r–r–. Breakdown each part of this representation. (4 marks)

5. Write a single command that lists all files in the /etc directory that end with the .conf extension and displays them one page at a time. (4 marks)

WORKING WITH FILES AND DIRECTORIES

1. Imagine you are in your home directory;

   (a) Create a new directory named reports. (1 Mark)

   (b) Inside the reports directory, create two empty files named weekly.txt and monthly.csv. (2 Marks)

   (c) Move the monthly.csv file to your home directory. (1 Mark)

   (d) Rename the weekly.txt file in the reports directory to summary.txt. (1 Mark)

   (e) Finally, remove the empty reports directory. (1 Mark)

2. Explain the purpose and usage of the following Linux commands, providing a brief example for each:

   (a) ls -l

   (b) cd ..

3. Describe three different ways to view the content of a text file named notes.md in your current directory from the Linux command line. Briefly explain a key advantage of each method. (3 marks)

4. Consider a file named data.log in your current directory. Write the Linux command(s) to:

   (a) Display only the first 10 lines of the file. (2 Marks)

   (b) Count the total number of lines in the file. (2 Marks)

5. Explain the difference between an absolute path and a relative path in the Linux file system. Provide one example of each for a file named document.pdf located within a directory named documents inside a user's home directory. (3 Marks)

LINUX PROCESSES

1. Explain the concept of a process in Linux. What key information is typically associated with each running process? (6 marks)

2. Describe the function of the ps command. Provide an example of how you would use it to display all processes running under your current user. (6 marks)

3. What are process IDs (PIDs)? Explain why they are important in managing Linux processes. How can you find the PID of a specific running program (provide an example command)? (6 marks)

4. Explain the difference between a foreground and a background process in the Linux shell. How would you move a running foreground process to the background? (6 marks)

5. Describe two common signals used to control processes in Linux. For each signal, explain its purpose and provide a command-line example of how it might be sent to a process (you can assume you know the PID of the process). (6 marks)

REDIRECTION AND INTERPROCESS COMMUNICATION

1. Explain the fundamental concepts of standard input (stdin), standard output (stdout), and standard error (stderr) in the Linux environment. How are they typically associated with a running process? (6 marks)

2. Describe the different redirection operators (>, >>,<) in the Linux shell. Explain what each operator does and provide a simple command-line example for each. (6 marks)

3. What is a pipe in the UNIX shell? Explain its primary purpose and provide a practical example demonstrating how to use it to combine two basic commands. (6 marks)

4. Briefly explain the concept of named pipes (FIFOs) as a form of inter-process communication. What are their key characteristics and how are they typically created and used? (6 marks)

5. Describe the basic principles of using signals for inter-process communication in Linux. Provide an example of a common signal and explain how one process might use it to communicate with another. (6 marks)

Success in Your Examinations

MARKING GUIDE

LINUX UTILITIES AND INSTALLATION

1. Explain the role of the bootloader in the Linux boot process. Provide a specific example of a common Linux bootloader. (4 marks)

   The bootloader is the first software program that runs when a computer starts up. Its primary role is to initialize the hardware components and then load the operating system kernel into memory so that the OS can take control of the system. Without a bootloader, the operating system could not be started. Example: GRUB (Grand Unified Bootloader) is a very common bootloader used in many Linux distributions. Marking Guide:

   - Explains the role of initializing hardware (1 mark)
   - Explains the role of loading the kernel (1 mark)
   - Explains the consequence of not having a bootloader (1 mark)
   - Provides a correct and specific example of a Linux bootloader (1 mark)

2. Describe two common methods for accessing the command-line interface (CLI) in a typical Linux desktop environment. (4 marks)

   The two common methods for accessing the command-line interface (CLI) in a typical Linux desktop environment are:

   (a) Using a Terminal Emulator Application: Most Linux desktop environments provide a graphical application (often called "Terminal," "Konsole," "xterm," etc.) that opens a window providing a shell interface. Users can launch this application from the desktop environment's menu or by using a keyboard shortcut (e.g., Ctrl+Alt+T).•

(b) Using a Virtual Console (TTY): Linux provides several virtual consoles that can be accessed using keyboard shortcuts. Typically, pressing Ctrl+Alt+F1 through Ctrl+Alt+F6 will switch to different text-based virtual consoles, providing a direct command-line interface without the graphical environment. To return to the graphical environment, you usually use Ctrl+Alt+F7 (though this might vary slightly depending on the distribution).

Marking Guide:

(a) Identifies and describes a terminal emulator application (2 marks)

(b) Identifies and describes the use of a virtual console (TTY) (2 marks)

3. Using the command line, how would you create a new directory named "reports" in your home directory? Also, how would you then navigate into this newly created directory? Provide the exact commands. (4 marks)

mkdir /reports The mkdir command is used to create a new directory. / is a shorthand notation that represents the user's home directory.

To navigate into the "reports" directory

cd /reports The cd command (change directory) is used to move into a different directory.

Again, / specifies the home directory, followed by the name of the directory to navigate into.

Marking Guide:

- Provides the correct mkdir command to create the directory (2 marks)
- Provides the correct cd command to navigate into the directory (2 marks)

4. Explain the meaning of the following Linux file permissions: -rw-r–r–. Breakdown each part of this representation. (4 marks)

The Linux file permissions -rw-r–r– indicate the access rights for the owner of the file, the group associated with the file, and other users on the system.

- The first character (-) indicates the file type. In this case, it's a regular file. (d would indicate a directory, l a symbolic link, etc.)
- The next three characters (rw-) represent the permissions for the owner of the file:

  r: read permission (allows the owner to view the contents of the file)

  w: write permission (allows the owner to modify the contents of the file)

  -: execute permission is not granted to the owner.

  The following three characters (r–) represent the permissions for the group associated with the file:

  r: read permission (allows members of the group to view the contents of the file)

  -: write permission is not granted to the group.

  -: execute permission is not granted to the group.

  The last three characters (r–) represent the permissions for other users on the system:

  r: read permission (allows other users to view the contents of the file)

  -: write permission is not granted to other users.

  -: execute permission is not granted to other users.

  Marking Guide:

  – Correctly identifies the file type indicator (1 mark)

  – Correctly explains the owner's permissions (rw-) (1 mark)

  – Correctly explains the group's permissions (r–) (1 mark)

  – Correctly explains the other users' permissions (r–) (1 mark)

5. Write a single command that lists all files in the /etc directory that end with the .conf extension and displays them one page at a time. (4 marks)

   ls /etc/*.conf | less

   ls /etc/*.conf: This part of the command uses the ls (list directory contents) utility to list all files within the /etc directory that match the pattern *.conf. The asterisk (*) is a wildcard that matches any sequence of characters, so this will find all files ending with .conf.

   |: This is the pipe symbol. It takes the standard output of the command on its left (ls /etc/*.conf) and redirects it as the standard input to the command on its right (less).

   less: This is a pager program that displays its input one screenful at a time, allowing the user to scroll through the output.

   Marking Guide:

   - Correctly uses ls with the wildcard to list the specific files (2 marks)
   - Correctly uses the pipe (|) to redirect output (1 mark)
   - Correctly uses a pager command like less (or more) to display output page by page (1 mark)

   WORKING WITH FILES AND DIRECTORIES

   (a) Imagine you are in your home directory;

      i. Create a new directory named reports. (1 Mark)

         To create a new directory named reports in the home directory, use the mkdir command as follows

         mkdir reports

      ii. Inside the reports directory, create two empty files named weekly.txt and monthly.csv. (2 Marks)

To create two empty files, weekly.txt and monthly.csv, inside the reports directory, first navigate into the directory using cd, and then use the touch command.

cd reports

touch weekly.txt monthly.csv

Alternatively, you can create them directly from the home directory:

touch reports/weekly.txt reports/monthly.csv

iii. Move the monthly.csv file to your home directory. (1 Mark)

To move the monthly.csv file to the home directory, use the mv command:

mv reports/monthly.csv /

iv. Rename the weekly.txt file in the reports directory to summary.txt. (1 Mark)

To rename the weekly.txt file to summary.txt within the reports directory, use the mv command:

mv reports/weekly.txt reports/summary.txt

v. Finally, remove the empty reports directory. (1 Mark)

To remove the empty reports directory, use the rmdir command:

rmdir reports

(b) Explain the purpose and usage of the following Linux commands, providing a brief example for each:

i. ls -l

This command lists the files and directories in the current directory in a long listing format. The -l option provides detailed information about each item, including permissions, number of links, owner, group, size, and modification time. Example:

ls -l

This would output a detailed listing of the contents of the current directory.

ii. cd ..

This command changes the current working directory to its parent directory.

The .. represents the parent directory.

Example:

If the current directory is /home/user/documents, then running:

cd .. will change the current directory to /home/user.

(c) Describe three different ways to view the content of a text file named notes.md in your current directory from the Linux command line. Briefly explain a key advantage of each method. (3 marks)

Three different ways to view the content of notes.md:

1. cat notes.md: This command concatenates and displays the entire content of the file to the standard output.

Advantage: Simple and quick for viewing the whole file.

2. less notes.md: This command displays the file content page by page, allowing you to navigate through the file using the arrow keys, Page Up, Page Down, etc.

Advantage: Useful for viewing large files as it doesn't load the entire file into memory at once and provides navigation capabilities.

3. head notes.md: This command displays the first few lines (by default, 10) of the file.

Advantage: Useful for quickly inspecting the beginning of a file without having to view the entire content. You can also specify the number of lines to view using the -n option (e.g., head -n 20 notes.md).

(d) Consider a file named data.log in your current directory. Write the Linux command(s) to:

i. Display only the first 10 lines of the file. (2 Marks)

To display only the first 10 lines of the file:

head -n 10 data.log

ii. Count the total number of lines in the file. (2 Marks)

To count the total number of lines in the file:

wc -l data.log

This command will output the number of lines followed by the filename. If you only want the line count, you can pipe the output to awk:

wc -l data.log | awk '{print $1}'

(e) Explain the difference between an absolute path and a relative path in the Linux file system. Provide one example of each for a file named document.pdf located within a directory named documents inside a user's home directory. (3 Marks)

An absolute path specifies the location of a file or directory starting from the root directory (/). It provides a complete and unambiguous way to locate a file or directory, regardless of the current working directory.

Example of an absolute path:

/home/user/documents/document.pdf

A relative path specifies the location of a file or directory relative to the current working directory. It doesn't start with the root directory.

Example of a relative path (assuming the current directory is /home/user):

documents/document.pdf

Example of a relative path (assuming the current directory is /home/user/another_directory):

../documents/document.pdf (where .. refers to the parent directory).

LINUX PROCESSES

(a) Explain the concept of a process in Linux. What key information is typically

associated with each running process? (6 marks)

In Linux, a process is an instance of a running program. It represents a program in execution, including the program code, its current activity, processor registers, program counter, stack, heap, and all other resources associated with it.

Key Information Associated with a Process:

Process ID (PID): A unique numerical identifier for the process.

Parent Process ID (PPID): The PID of the process that created the current process.

User ID (UID) and Group ID (GID): The user and group associated with the process, determining its permissions.

CPU Usage: The amount of CPU time the process has consumed.

Memory Usage: The amount of RAM the process is currently using.

State: The current state of the process (e.g., running, sleeping, stopped, zombie).

Priority: The scheduling priority of the process, influencing how much CPU time it gets.

Command: The name of the executable and its arguments that started the process.

Marking Guide:

- Defines a process as an instance of a running program (2 marks)
- Lists and briefly explains at least four key pieces of information associated with a process (1 mark per correct and relevant piece of information = 4 marks)

(b) Describe the function of the ps command. Provide an example of how you would use it to display all processes running under your current user. (6 marks)

The ps command (process status) is used to display information about the cur-

rently running processes. It provides a snapshot of the active processes on the system.

Example to display all processes running under the current user:

ps aux — grep "$(whoami)"

ps aux: This part of the command displays a comprehensive list of all processes running on the system, showing information about the user, PID, CPU and memory usage, etc.

|: This is the pipe symbol, redirecting the output of ps aux to the grep command.

grep "$(whoami)": The grep command filters the output it receives. $(whoami) is a command substitution that executes the whoami command (which prints the current username) and uses its output as the search pattern for grep. This effectively filters the process list to show only those processes owned by the current user.

Marking Guide:

- Correctly states the function of the ps command (2 marks)
- Provides the correct ps aux command (or a similar valid option like ps -u $USER) (2 marks)
- Correctly uses grep and whoami (or a similar method) to filter by the current user (2 marks)

(c) What are process IDs (PIDs)? Explain why they are important in managing Linux processes. How can you find the PID of a specific running program (provide an example command)? (6 marks)

Process IDs (PIDs) are unique numerical identifiers assigned by the Linux kernel to each running process. They are crucial for managing processes because they provide a way to specifically identify and refer to individual processes for various

operations.

Importance of PIDs:

Process Control: PIDs are used by commands like kill, renice, and top to target specific processes for actions such as terminating them, changing their priority, or monitoring their resource usage.

System Monitoring: System administrators and monitoring tools use PIDs to track the activity and resource consumption of individual processes.

Inter-Process Communication (IPC): Some IPC mechanisms rely on PIDs to identify the communicating processes.

Logging and Auditing: PIDs are often included in system logs to track which process performed certain actions.

Example command to find the PID of a specific running program (e.g., firefox): pgrep firefox or ps aux — grep firefox (The ps aux method would require you to identify the PID from the output.)

Marking Guide:

- Defines Process IDs (PIDs) as unique numerical identifiers (2 marks)
- Explains at least two valid reasons why PIDs are important for process management (1 mark per reason = 2 marks)
- Provides a correct example command using pgrep or ps with grep to find the PID of a specific program (2 marks)

(d) Explain the difference between a foreground and a background process in the Linux shell. How would you move a running foreground process to the background? (6 marks)

Foreground Process: A foreground process is one that is started and runs directly in the current terminal session. The shell waits for the foreground process to

complete before it returns the command prompt to the user, meaning the user cannot interact with the shell until the process finishes.

Background Process: A background process is one that is started and runs independently of the current terminal session. The shell returns the command prompt immediately after starting a background process, allowing the user to continue executing other commands while the background process runs.

Moving a running foreground process to the background:

While the foreground process is running, press Ctrl + Z. This sends a SIGTSTP signal to the process, which typically suspends its execution.

Then, use the bg command to resume the suspended process in the background. You can optionally specify a job ID (e.g., %1) if multiple processes are suspended. If only one is suspended, simply typing bg will suffice.

Marking Guide:

- Clearly explains the characteristics of a foreground process (2 marks)
- Clearly explains the characteristics of a background process (2 marks)
- Correctly describes the steps (using Ctrl+Z and the bg command) to move a foreground process to the background (2 marks)

(e) Describe two common signals used to control processes in Linux. For each signal, explain its purpose and provide a command-line example of how it might be sent to a process (you can assume you know the PID of the process). (6 marks)

Two common signals used to control processes in Linux are: SIGTERM (Signal 15): This is the standard signal used to request a process to terminate gracefully. It gives the process a chance to clean up resources (e.g., save data, close files) before exiting.

Command-line example: kill <PID> (sends SIGTERM by default) or kill -15

<PID>

SIGKILL (Signal 9): This signal forcefully terminates a process immediately. The process does not get a chance to perform any cleanup operations. It should be used as a last resort if a process is unresponsive.

Command-line example: kill -9 <PID>

- Marking Guide:
  - Correctly identifies and explains the purpose of SIGTERM (2 marks)
  - Provides a correct command-line example for sending SIGTERM (1 mark)
  - Correctly identifies and explains the purpose of SIGKILL (2 marks)
  - Provides a correct command-line example for sending SIGKILL (1 mark)

REDIRECTION AND INTERPROCESS COMMUNICATION

(a) Explain the fundamental concepts of standard input (stdin), standard output (stdout), and standard error (stderr) in the Linux environment. How are they typically associated with a running process? (6 marks)

Standard Input (stdin): This is the standard way for a program to receive input. By default, it's connected to the keyboard, meaning a program typically reads input from what the user types.

Standard Output (stdout): This is the standard way for a program to send regular output. By default, it's connected to the terminal screen, so normal program output is displayed there.

Standard Error (stderr): This is the standard way for a program to send error messages and diagnostic information. By default, it's also connected to the terminal screen, allowing error messages to be displayed to the user.

Association with a Running Process: When a process is started, the operating system typically sets up these three standard streams. Each stream is represented

by a file descriptor: 0 for stdin, 1 for stdout, and 2 for stderr. Programs can then read from file descriptor 0 and write to file descriptors 1 and 2 without needing to know the underlying devices (keyboard or screen).

Marking Guide:

- Clearly explains the concept of standard input (2 marks)
- Clearly explains the concept of standard output (2 marks)
- Clearly explains the concept of standard error (2 marks)

(b) Describe the different redirection operators (>, >>,<) in the Linux shell. Explain what each operator does and provide a simple command-line example for each. (6 marks)

> (Output Redirection): This operator redirects the standard output of a command to a file. If the file exists, its contents are overwritten. If the file does not exist, it is created.

Example: ls -l > file_list.txt (This command lists the files in the current directory and saves the output to a file named file_list.txt, overwriting it if it already exists).

>>(Append Output Redirection): This operator also redirects the standard output of a command to a file. However, if the file exists, the output is appended to the end of the file, preserving its existing content. If the file does not exist, it is created.

Example: echo "Adding more info" >> file_list.txt (This command adds the text "Adding more info" to the end of the file_list.txt file).

< (Input Redirection): This operator redirects the standard input of a command to come from a file instead of the keyboard. The command reads its input from the specified file.

Example: sort <names.txt (This command takes the content of the names.txt file

as input and sorts it, displaying the sorted output on the standard output).

Marking Guide:

- Correctly explains the function of the > operator and provides a valid example (2 marks)
- Correctly explains the function of the >> operator and provides a valid example (2 marks)
- Correctly explains the function of the < operator and provides a valid example (2 marks)

(c) What is a pipe (|) in the UNIX shell? Explain its primary purpose and provide a practical example demonstrating how to use it to combine two basic commands. (6 marks)

A pipe (|) in the UNIX shell is a mechanism for inter-process communication that allows the output of one command to be directly used as the input of another command. It creates a temporary, unidirectional data channel between two or more processes.

Primary Purpose: The main purpose of a pipe is to combine the functionality of multiple smaller, specialized commands to achieve more complex tasks. It allows you to chain commands together, where the output of one becomes the input of the next, enabling powerful data processing in a concise way.

Practical Example: Let's say you want to find out how many files in the /etc directory contain the word "password". You can use a pipe to combine the ls and grep commands:

ls /etc | grep password

ls /etc: This command lists all the files and directories within the /etc directory, sending the output to its standard output.

|: The pipe takes the standard output of ls /etc and redirects it as the standard input to the grep password command.

grep password: This command reads its standard input (which is the output of ls /etc) and filters it, displaying only the lines that contain the string "password".

Marking Guide:

- Clearly explains what a pipe is and its function of connecting command output to command input (2 marks)
- Explains the purpose of combining simpler commands for complex tasks (2 marks) Provides a clear and functional example using the pipe to combine two basic commands (2 marks)

(d) Briefly explain the concept of named pipes (FIFOs) as a form of inter-process communication. What are their key characteristics and how are they typically created and used? (6 marks)

Named pipes, also known as FIFOs (First-In, First-Out), are special file types in Linux that provide a mechanism for inter-process communication between unrelated processes. Unlike regular pipes, which only exist while the connected processes are running, named pipes persist in the file system even after the processes using them have terminated.

Key Characteristics:

File System Entry: They have a name and reside in the file system, allowing unrelated processes to access them.

FIFO Behavior: Data written into a FIFO is read out in the same order it was written (first-in, first-out).

Blocking Operations: If a process tries to read from an empty FIFO, it will block until data is available. Similarly, if a process tries to write to a full FIFO (if it has

a capacity limit, though usually not enforced strictly), it will block until space becomes available.

Creation and Usage:

Creation: Named pipes are created using the mkfifo command followed by the desired name of the FIFO.

mkfifo my_fifo

Usage: One or more processes can open the FIFO for reading, and one or more processes can open it for writing. Data written to the FIFO by one process can be read by another. For example:

Process 1 (Writer): echo "Hello from Process 1" > my_fifo

Process 2 (Reader): cat < my_fifo (This would output "Hello from Process 1")

Marking Guide:

- Explains the concept of named pipes (FIFOs) and their persistence in the file system (2 marks)
- Describes at least two key characteristics of named pipes (e.g., file system entry, FIFO behavior, blocking) (1 mark per characteristic = 2 marks)
- Correctly describes how to create a named pipe using mkfifo and provides a basic example of how two processes might use it for communication (2 marks)

(e) Describe the basic principles of using signals for inter-process communication in Linux. Provide an example of a common signal and explain how one process might use it to communicate with another. (6 marks)

Signals in Linux provide a basic mechanism for inter-process communication by allowing one process to notify another process of the occurrence of a specific event. Signals are asynchronous, meaning they can be sent to a process at any time, regardless of what the process is currently doing.

Basic Principles:

Signal Generation: Signals can be generated by various events, such as user actions (e.g., pressing Ctrl+C), hardware errors, software conditions, or explicitly sent by other processes using system calls like kill.

Signal Delivery: When a signal is generated for a process, the kernel delivers the signal to that process.

Signal Handling: A process can either use the default action associated with a signal (e.g., termination), ignore the signal, or register a signal handler function to be executed when the signal is received.

Example of a Common Signal: SIGINT (Signal 2). This signal is typically generated when a user presses Ctrl+C in the terminal. Its default action is to terminate the foreground process.

Example of Communication: If you have a process with PID 1234 running in the foreground, and you want to interrupt it, you can press Ctrl+C in the terminal where it's running. This will send the SIGINT signal to process 1234, likely causing it to terminate. Another process could also explicitly send this signal using the kill command:

kill -SIGINT 1234

or

kill -2 1234

Marking Guide:

- Explains the basic principle of signals as asynchronous notifications of events (2 marks)
- Describes the concepts of signal generation, delivery, and handling (1 mark) Provides a correct example of a common signal (e.g., SIGINT, SIGTERM)

and explains its purpose (2 marks)

- Provides a valid example of how one process (e.g., via user input or the kill command) can send the signal to another process (1 mark)