2023 Exam unix

(a) Basic features of operating system.
(b) Processing time vs. turnaround time.
(c) Program vs. process.
(d) Advantages of multi-threading.
(e) Text files vs. binary files.
(f) Absolute path vs. relative path.
(g) 'mv' command vs. 'ep' command.
(h) Zombie state of a process

(a) Basic Features of an Operating System
An operating system (OS) serves as an intermediary between computer hardware and the user. Its main functions and features include:

1. **Process Management**: The OS manages processes in the system, including process creation, scheduling, and termination. It ensures that processes run efficiently and that resources are allocated properly.
2. **Memory Management**: It handles the allocation and deallocation of memory to various processes. It ensures that each process gets the required memory and prevents memory leaks or illegal access to memory.
3. **File System Management**: The OS manages files and directories, handling the storage, retrieval, and organization of data on disk drives. It provides services like file access, naming, and protection.
4. **Device Management**: The OS controls hardware devices like printers, disks, and input/output devices. It provides device drivers and ensures efficient communication between hardware and software.
5. **Security and Access Control**: The OS ensures the security of data and resources by enforcing user authentication, access control lists, encryption, and other security protocols.
6. **User Interface**: The OS provides a way for users to interact with the system, typically through a command-line interface (CLI) or graphical user interface (GUI).
7. **Resource Allocation**: It allocates and manages system resources such as CPU time, memory, and disk space among

competing processes.

(b) Processing Time vs. Turnaround Time

- **Processing Time** (or **CPU Time**): This refers to the actual time the CPU spends executing a particular process. It does not include the time spent waiting for I/O operations or other events.
- **Turnaround Time**: This is the total time taken to complete a process, from the time it enters the system (arrival time) to the time it finishes execution. It includes:
  - Waiting time (time the process spends in the ready queue),
  - Processing time (time spent executing on the CPU),
  - I/O time (time spent waiting for or performing I/O operations).

Formula for **Turnaround Time**:

Turnaround Time
=
Completion Time
−
Arrival Time

Turnaround Time=Completion Time−Arrival Time

Example:

If a process arrives at time 0, starts execution at time 3, and completes at time 10, the processing time is 7 units (from 3 to 10), but the turnaround time is 10 units (from arrival at 0 to completion at 10).

(c) Program vs. Process

- **Program**: A program is a static set of instructions or code written to perform a specific task. It is just a collection of instructions stored in a file (e.g., a text file or binary file). It does not do anything until it is executed.
- **Process**: A process is a program in execution. It is an active entity, which includes:
  - Program code (text section),
  - Current activity (registers, program counter),
  - Stack and heap memory,
  - Resources (open files, I/O devices).

A program becomes a process when it is loaded into memory and begins execution.

(d) Advantages of Multi-Threading

Multi-threading allows a single process to have multiple threads of execution, enabling concurrent execution of tasks. Some advantages include:

1. **Improved CPU Utilization**: By allowing multiple threads to run simultaneously, multi-threading can utilize the CPU more effectively, especially when one thread is waiting for I/O operations (e.g., disk or network access).
2. **Better Performance**: In multi-core processors, threads can be executed on different cores simultaneously, leading to better performance and faster processing times.
3. **Responsive User Interfaces**: In applications with graphical user interfaces (GUIs), multi-threading can allow the UI to remain responsive while other threads perform background tasks (e.g., file loading, data processing).
4. **Resource Sharing**: Threads within the same process share resources like memory, which is more efficient than allocating separate memory for each task.
5. **Simplified Program Structure**: Multi-threading can simplify the design of programs that need to perform several tasks concurrently, like web servers or databases, by dividing work into smaller threads.

(e) Text Files vs. Binary Files

- **Text Files**:
  A text file contains human-readable characters encoded using standard character encodings (like ASCII or UTF-8). It consists of a sequence of characters (letters, digits, punctuation) that can be interpreted as plain text by any text editor. Text files are often used for configuration files, source code, and documentation.
  Characteristics:
  - Human-readable.
  - Easy to create and edit using simple text editors.
  - Can be large due to redundant encoding of characters.
- **Example**: .txt, .html, .csv, .xml
- **Binary Files**:
  A binary file contains data in a format that is not intended to be human-readable. It stores raw data in binary (0s and 1s),

which is interpreted by programs. These files are used for storing data like images, videos, executables, and other media.
  Characteristics:
  - Not human-readable.
  - More compact than text files because they store data in a more efficient format.
  - Requires specific programs or software to open and interpret.
- **Example**: .exe, .jpg, .mp3, .dat

(f) Absolute Path vs. Relative Path

Both **absolute** and **relative** paths are ways of specifying the location of a file or directory in a file system. The key difference lies in how they are defined:

Absolute Path

- **Definition**: An absolute path provides the complete path to a file or directory starting from the root directory (/ in UNIX-based systems, or a drive letter like C: in Windows).
- **Characteristics**:
  - It is always the same, regardless of your current working directory.
  - It begins from the root or base of the file system.
  - It specifies the entire location of the file or directory, providing a full address from the root to the target.
- **Example (Linux/Unix)**:
  arduino
  Copy code

  ```
  /home/user/documents/report.txt
  ```
-

  Here, /home/user/documents/report.txt specifies the file report.txt in the documents directory, which is in the user directory, starting from the root /.
- **Example (Windows)**:

  makefile
  Copy code

  ```
  C:\Users\John\Documents\report.txt
  ```
-

Relative Path

- **Definition**: A relative path specifies the location of a file or directory relative to the current working directory.
- **Characteristics**:
  - It depends on where you are in the file system.
  - You don’t need to include the full path from the root directory; it is assumed to be relative to the current location.
  - It uses . for the current directory and .. for the parent directory.
- **Example (Linux/Unix)**: If you're in /home/user/ and want to refer to the file report.txt in /home/user/documents/, you can use:
  bash
  Copy code

  ```
  documents/report.txt
  ```
-

  If you're in /home/user/documents/ and want to move up to the parent directory and then access report.txt, you can use:
  bash
  Copy code

  ```
  ../report.txt
  ```
-

- **Example (Windows)**: If you’re in C:\Users\John and want to refer to Documents\report.txt, you can use:

Copy code

```
Documents\report.txt
```

-

(g) 'mv' Command vs. 'cp' Command

Both mv and cp are used for file operations in Unix-like operating systems (Linux, macOS), but they have different functions.

mv Command (Move)

- **Function**: The mv command is used to **move** files or directories from one location to another. It can also **rename**files.
- **Characteristics**:
  - When moving files within the same filesystem, it simply updates the file’s directory entry and does not copy the data.
  - When moving files across filesystems, it copies the data to the target location and then deletes the original.
  - It can be used to rename files or directories.
- **Example (Move a file)**:
  bash
  Copy code

  ```bash
  mv file1.txt /home/user/documents/
  ```

-

  This command moves file1.txt to the /home/user/documents/ directory.
- **Example (Rename a file)**:
  bash
  Copy code

```
mv oldfile.txt newfile.txt
```

-

This command renames oldfile.txt to newfile.txt.

cp Command (Copy)

- **Function**: The cp command is used to **copy** files or directories.
- **Characteristics**:
  - It creates a new copy of the file at the target location, leaving the original file intact.
  - You can use options to copy directories recursively (-r) or to preserve file attributes like timestamps (-p).
- **Example (Copy a file)**:
  bash
  ```
  cp file1.txt /home/user/documents/
  ```
- This command copies file1.txt to the /home/user/documents/ directory.
- **Example (Copy a directory)**:
  bash
  Copy code
  ```
  cp -r dir1 /home/user/documents/
  ```
- This command recursively copies the dir1 directory and its contents to the target directory.

Key Differences:

- mv moves or renames files, while cp copies files.
- mv removes the original after moving (if within the same filesystem), while cp keeps the original and creates a copy.

(h) Zombie State of a Process

A **zombie process** is a process that has **terminated** but still has an entry in the process table. This happens when the process has finished its execution, but the parent process has not yet read its exit status (via a system call like wait()), so the process's entry remains in the process table.

Key Characteristics of a Zombie Process:

- **Completion of Execution**: A process enters the zombie state after it has finished execution but before its parent process has collected its exit status.

- **Resource Cleanup**: The kernel does not fully clean up the process resources (like the process ID) until the parent process acknowledges the termination.
- **Non-Running**: A zombie process is **not running**. It is simply a dead process that remains in the process table.
- **Process Table Entries**: The system keeps a small entry in the process table to allow the parent process to collect the status of the terminated process.

Why Zombies Occur:

- When a process terminates, it sends an exit status to its parent process. The parent process is expected to call wait() or waitpid() to retrieve this status. If the parent process fails to do this (for example, if it is not running or never calls wait()), the child process remains in the **zombie state**.

Example:

- A typical situation where a zombie might appear is in a parent-child process relationship, where the child process finishes execution but the parent doesn't read the exit status.

How to Identify a Zombie Process:

You can check for zombie processes using the ps command. A zombie process is usually marked with a Z in the process status column (STAT):

perl

Copy code

```
ps aux | grep Z
```

How to Remove a Zombie Process:

- Normally, a zombie process will be cleaned up automatically once the parent process collects its exit status.
- If the parent process is not responding, the operating system may eventually clean it up.
- If a zombie process does not disappear, terminating or restarting the parent process usually clears it.