Task 5

Task 1: two basic ways of computer architecture, and which one is the best?

- There are two basic computer architecture approaches: CISC (Complex Instruction Set Computing) and RISC (Reduced Instruction Set Computing). Neither is definitively the best; the choice depends on specific needs. CISC has complex, versatile instructions, while RISC has simpler, faster instructions. Use CISC for versatility, RISC for speed and efficiency.

Task 2: Languages support auto garbage collection?

- Many programming languages support automatic garbage collection, which is a memory management feature that helps developers avoid memory leaks and manually freeing memory. Some popular programming languages with built-in automatic garbage collection include: -

1- Java
2- C#
3- Python
4- Ruby
5- C++
6- JavaScript

Task 3: What is BIOS? Why do we use it?

- BIOS stands for "Basic Input/Output System." It is a firmware that is embedded on a computer's motherboard and serves as the initial software that runs when the computer is powered on or restarted. BIOS plays a crucial role in the boot-up process and provides essential functions for hardware initialization, system configuration, and low-level interaction between the operating system and the computer's hardware components. boot-up process and provides essential functions for hardware initialization, system configuration, and low-level interaction between the operating system and the computer's hardware components.

- reasons for using BIOS include:

1- System Boot-Up.
2- Hardware Initialization.
3- System Configuration.
4- Low-Level Access.
5- Compatibility and Legacy Support.

Task 4: Linux Vs Unix with example?

- Linux and Unix are both operating systems, but there are significant differences between them. Linux is a Unix-like operating system that was developed as an open-source alternative to proprietary Unix systems. Let's explore some key differences and provide an example of each:

Origin and Licensing:

Unix: Unix is a family of proprietary operating systems that originated at AT&T Bell Labs in the 1970s. Different versions of Unix, such as AIX, HP-UX, and Solaris, have been developed by various companies. Unix systems were historically sold as commercial products.

Linux: Linux is a free and open-source operating system kernel that was created by Linus Torvalds in 1991. It is based on Unix principles but developed independently. Various Linux distributions (distros) combine the Linux kernel with software from other sources to create complete operating systems.

Licensing Model:

Unix: Traditionally, Unix systems were proprietary and required licensing fees to use. Some versions of Unix have transitioned to more open licensing models in recent years.

Linux: Linux is released under the GNU General Public License (GPL) and other open-source licenses. This means that the source code is freely available, and users are free to modify and distribute it.

Community and Development:

Unix: Development of Unix systems is typically controlled by the companies that produce them, with limited community involvement.

Linux: Linux has a vibrant and diverse community of developers and contributors from around the world. This community-driven model has led to rapid development and innovation.

Examples:

Unix Example: IBM's AIX is a version of Unix used on IBM's hardware platforms. It provides enterprise-level features and is commonly used in large-scale computing environments.

Linux Example: Ubuntu is a popular Linux distribution that is known for its user-friendliness and community support. It is used for desktop, server, and cloud computing.

Kernel and Compatibility:

Unix: Different Unix variants have variations in their kernels and system utilities. While they share some common features and APIs, compatibility can be an issue when moving between different Unix systems.

Linux: The Linux kernel is at the core of all Linux distributions. While there may be some differences between Linux distributions, they generally adhere to common standards and APIs, making software and applications more portable.

Market and Usage:

Unix: Historically, Unix systems were prevalent in enterprise environments and research institutions.

Linux: Linux has gained widespread adoption across various domains, including web servers, mobile devices, embedded systems, desktops, and cloud computing.

Task 5: What is fragmentation?

- Fragmentation is a computer issue where data or memory becomes scattered in small pieces instead of being in one place. It can slow down performance and cause inefficiencies.

Task 6: Compare among all scheduling algorithms [Round robin - Priority - First come first serve]?

1- Round Robin (RR):

- Each process gets a fixed time slice (quantum) to execute before being moved to the back of the queue.
- Suitable for time-sharing systems and ensures fairness among processes.
- May lead to high context-switching overhead due to frequent process switches.
- Good for preventing starvation as all processes get a chance to run.
- May not be optimal for long-running or CPU-intensive processes.

2- Priority Scheduling:

- Processes are assigned priorities, and the highest priority process executes first.
- Suitable for real-time systems where certain tasks need immediate attention.
- Can lead to starvation of lower-priority processes if not managed properly.
- Requires a mechanism to handle priority inversions (when lower-priority processes hold resources needed by higher-priority processes).
- Can be dynamic (priorities change over time) or static (priorities assigned and fixed).

3- First Come First Serve (FCFS):

- Processes are executed in the order they arrive in the ready queue.
- Simple and easy to implement.
- Can lead to the "convoy effect" where short processes are delayed behind long processes.
- Not suitable for time-sensitive systems as it doesn't consider process priority.
- Can cause long waiting times for higher-priority processes if they arrive later.

Each scheduling algorithm has its strengths and weaknesses, making it suitable for different scenarios. Round Robin is good for sharing resources fairly, Priority Scheduling is useful for real-time tasks, and FCFS is straightforward but might not be efficient in certain situations. The choice of algorithm depends on the system's requirements and goals.

Task 7: Parallel processing Vs Threads?

Parallel processing and threads are concepts related to achieving concurrent execution in computing, but they have distinct characteristics and purposes:

1- Parallel Processing:

- Parallel processing refers to the execution of multiple tasks or processes simultaneously to improve overall system performance and efficiency.
- In parallel processing, multiple independent tasks are divided into smaller subtasks that can be executed concurrently on separate processors or cores.
- Parallel processing is typically used to tackle computationally intensive tasks, such as scientific simulations, data analysis, and rendering, where dividing the workload can significantly speed up execution.
- Examples of parallel processing architectures include multi-core processors and clusters of computers working together.

2- Threads:

- Threads are smaller units of a process that can be scheduled independently for execution within the same process.
- Threads within a process share the same memory space, allowing them to communicate and interact more efficiently than separate processes.
- Threads are lighter weight than processes and can be created, switched, and managed more quickly.
- Threads are commonly used for tasks that require concurrent execution and communication, such as GUI applications, web servers, and multi-threaded games.

Task 8: Languages support multithreading?

- Many programming languages support multithreading, which allows developers to create concurrent threads of execution within a single process. Here are some popular programming languages that provide built-in support for multithreading:

1- Java
2- C#
3- C++
4- Python
5- Ruby
6- Scala

Task 9: Clean Code principles?

- Clean Code principles are guidelines for writing code that is easy to understand, maintain, and collaborate on. It emphasizes meaningful names, small functions, avoiding duplication, clear comments, consistent formatting, and other practices that enhance code quality. Following Clean Code principles helps create readable and efficient code.