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

here are two basic approaches to computer architecture: the von Neumann architecture and the Harvard architecture. Each has its own strengths and weaknesses, and the "best" one depends on the specific application and requirements.

Von Neumann Architecture:

- In the von Neumann architecture, both instructions and data are stored in the same memory space.
- The CPU fetches instructions and data from the same memory using a single bus system.
- This architecture is simple and easy to implement, making it suitable for general-purpose computing.
- However, it can lead to performance bottlenecks due to competition for memory access between instructions and data.
- This is well-suited for general-purpose computing tasks where the performance difference between instruction and data access is not critical. Many modern computers, including personal computers and servers, use variations of the von Neumann architecture.

Harvard Architecture:

- In the Harvard architecture, separate memory spaces are used for instructions and data.
- This allows simultaneous access to instructions and data, potentially improving performance.
- Harvard architecture can be more complex to implement and may require more physical hardware resources.
- It is commonly used in embedded systems and certain specialized applications where performance is crucial.
- This is particularly advantageous in applications that require high-speed and parallel processing, such as embedded systems, digital signal processing, and specialized hardware like GPUs (Graphics Processing Units). Harvard architecture can exploit parallelism more effectively due to separate memory spaces for instructions and data.

****

Task 2: Languages support auto garbage collection:

Jave is the first language that use auto garbage collection.

This related with memory management which the language can Get rid of the object have not used in memory Without doing this manually, this reduces the occurrence of errors such as out of memory , this made program fast and more efficient

#There are language use other ways to Get rid of the object such python

Python: Python has an automatic garbage collection system that uses reference **counting and a cycle-detecting garbage collector to clean up memory.**

****

BIOS (basic input/output system) is the program a computer's microprocessor uses to start the computer system after it is powered on. It also manages data flow between the computer's operating system (OS) and attached devices, such as the hard disk, video adapter, keyboard, mouse and printer.

Uses of BIOS

The main use of BIOS is to act as a middleman between OSes and the hardware they run on. BIOS is theoretically always the intermediary between the microprocessor and I/O device control information and data flow. Although, in some cases, BIOS can arrange for data to flow directly to memory from devices, such as video cards, that require faster data flow to be effective.

How does BIOS work?

BIOS comes included with computers, as firmware on a chip on the motherboard. In contrast, an OS like Windows or iOS can either be pre-installed by the manufacturer or vendor or installed by the user. BIOS is a program that is made accessible to the microprocessor on an erasable programmable read-only memory (EPROM) chip. When users turn on their computer, the microprocessor passes control to the BIOS program, which is always located at the same place on EPROM.

When BIOS boots up a computer, it first determines whether all of the necessary attachments are in place and operational. Any piece of hardware containing files the computer needs to start is called a *boot device*. After testing and ensuring boot devices are functioning, BIOS loads the OS -- or key parts of it -- into the computer's random access memory (RAM) from a hard disk or diskette drive (the boot device).

****

Task4 : Linux Vs Unix with example:

What is Linux?

Linux is an open-source operating system. This OS is supported on several computer platforms and includes multiple software features that handle computer resources, and allow you to do tasks. This operating system was launched by Linus Torvalds at the University of Helsinki in 1991.

What is Unix?

Unix is a powerful and multitasking operating system that behaves like a bridge between the user and the computer. It allows the user to perform specific functions. This operating system was launched in 1960, and was released by AT&T Bell Labs.

Features	Linux	Unix
Basic Definition	Linux is an open-source operating system. This OS is supported on several computer platforms and includes multiple software features that handle computer resources, and allow you to do tasks.	Unix is a powerful and multitasking operating system that behaves like a bridge between the user and the computer.
Launched by	This operating system was launched by Linus Torvalds at the University of Helsinki in 1991.	This operating system was launched in 1960 and released by AT&T Bell Labs.
OS family	It belongs to the Unix-like family.	It belongs to the Unix family.
Available in	It is available in multiple languages.	It is available in English.
Kernel Type	It is monolithic.	It can be microkernel, monolithic, and hybrid.
Written in	C and other programming languages.	C and assembly language.
File system support	It supports more file systems than Unix.	It also supports less than Linux.
Usage	It is used in several systems like desktop, smartphones, mainframes and servers.	Unix is majorly used on workstations and servers.
Examples	Some examples of Linux are: Fedora, Debian, Red Hat, Ubuntu, Android, etc.	Some examples of unix are IBM AIX, Darwin, Solaris, HP-UX, macOS X, etc.
Security	Linux provides higher security.	Unix is also highly secured.
Price	Linux is free and its corporate support is available at a price.	Unix is not totally free. There are some Unix versions that are free, other than that UNIX is expensive.

****

Task5: What is fragmentation?

What is Fragmentation?

Fragmentation is an unwanted problem in the operating system in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused. It is also necessary to understand that as programs are loaded and deleted from memory, they generate free space or a hole in the memory. These small blocks cannot be allotted to new arriving processes, resulting in inefficient memory use.

The conditions of fragmentation depend on the memory allocation system. As the process is loaded and unloaded from memory, these areas are fragmented into small pieces of memory that cannot be allocated to incoming processes. It is called **fragmentation**.

****

Task6: Compare among all scheduling algorithms [Round robin - Priority - First come first serve]?

Criteria	Round Robin Scheduling	Priority Scheduling	First Come First Serve (FCFS)
Time Quantum	Fixed time quantum	-	-
Preemptive	Yes	Yes	No
Fairness	Fair	Depends on priority	Not inherently fair
Throughput	Moderate	Depends on priority	Lower
Response Time	Fair	Depends on priority	Poor
Turnaround Time	Higher due to context switches	Depends on priority	Depends on arrival order
Context Switching	Frequent	Possible with priorities	Fewer
Starvation	Minimal risk	Possible for low-priority	Possible for low-priority
Suitability	Time-sharing, interactive	Real-time, varying priority	Batch processing

****

Task7 : Parallel processing Vs Threads

1. Threading: CPU switches between different threads really fast, giving a falsehood of concurrency. Keypoint: **only one thread is running at any given time.** When one thread is running, others are blocked. You might think, how is this any useful than just running procedurally? Well, think of it as a priority queue. Threads can be scheduled. CPU scheduler can give each thread a certain amount of time to run, pause them, pass data to other threads, then give them different priorities to run at a later time. It's a must for **not instant running processes that interact with each other**. It's used in servers extensively: thousands of clients can request something at the same time, then getting what they requested at a later time (If done procedurally, only one client can be served at a time). **Philosophy: do different things together.** It doesn't reduce the total time (moot point for server, because one client doesn't care other clients' total requests).
2. Parallelism: threads are running parallel, usually in different CPU core, true concurrency. Keypoint: **mlutiple threads are running at any given time.** It's useful for heavy computations, **super long running processes**. Same thing with a fleet of single core machines, split data into sections for each machine to compute, pool them together at the end. Different machines/cores are hard to interact with each other. **Philosophy: do one thing in less time.**

Task8: Languages support multithreading.

- Java
- C++
- Go (Golang)
- Erlang
- Scala
- Ada
- Haskell
- Rust

These languages provide built-in features for creating and managing thr

****

Task9 : Clean Code principles:

- **Meaningful Names**: Use clear and descriptive names for variables, functions, and classes.
- **Functions and Methods**: Keep functions short, focused, and with a single purpose.
- **Comments**: Aim for self-explanatory code; use comments sparingly for complex parts.
- **Formatting**: Maintain consistent formatting and adhere to a coding style guide.
- **Single Responsibility**: Design classes with a single, well-defined responsibility.
- **DRY (Don't Repeat Yourself)**: Eliminate duplication by reusing code and abstractions.
- **OCP (Open/Closed Principle)**: Design for extension without modifying existing code.
- **Avoid Magic Numbers**: Replace magic values with named constants.
- **Error Handling**: Handle errors gracefully and use exceptions for exceptional cases.
- **Testing**: Write automated tests to ensure code correctness and robustness.
- **Dependency Injection**: Inject dependencies to improve modularity and testability.
- **Small Commits**: Make small, focused commits with clear messages.
- **Simplicity**: Prioritize simplicity over unnecessary complexity.
- **Refactoring**: Continuously improve code quality through small refactorings.
- **Readability**: Prioritize code readability for humans.

Adhering to these principles leads to cleaner, more maintainable, and reliable code.

Regenerate