what is scheduling algorthims and how it works?

Scheduling algorithms are a crucial aspect of operating systems, responsible for efficiently managing the execution of processes or tasks. Imagine a computer system with multiple processes competing for the CPU's attention—scheduling algorithms determine which process gets to run when and for how long.

There are various scheduling algorithms, each with its own approach. Here are a few common ones:

First-Come, First-Served (FCFS): This is a simple approach where the first process that arrives is the first to be executed. It's like standing in a queue.

Shortest Job Next (SJN) or Shortest Job First (SJF): The process with the shortest burst time (time required to complete) is scheduled next. It aims to minimize waiting time and increase system throughput.

Priority Scheduling: Each process is assigned a priority, and the one with the highest priority gets scheduled first. It can be either preemptive (can be interrupted) or non-preemptive.

Round Robin (RR): Each process gets a fixed time slot or quantum to execute. If the process doesn't finish in its time slot, it's put back in the queue, and the next process gets a turn.

Multilevel Queue Scheduling: Processes are divided into different priority levels, and each queue has its own scheduling algorithm. Processes move between queues based on their behavior and priority.

Multilevel Feedback Queue Scheduling: Similar to multilevel queue scheduling, but processes can move between queues based on their past behavior. For example, a process that uses too much CPU time may be moved to a lower-priority queue.

The goal of these algorithms is to optimize the use of system resources, reduce waiting times, and enhance overall system performance. Each has its strengths and weaknesses, and the choice of algorithm depends on the specific requirements and characteristics of the system.

what are 13 rules of a clean code?

1. Meaningful Names
2. Don't Repeat Yourself
3. Single Responsibility Principle (SRP)
4. Small Functions/Methods
5. Comments for Why, not What

6. **Consistent Formatting**
7. **Avoid Magic Numbers and Strings**
8. **Error Handling Is Part of the Code**
9. **Unit Tests**
10. **Refactor Regularly**
11. **Keep It Simple, Stupid**
12. **You Ain't Gonna Need It**
13. **Readability Matters**