MultiThreading

Assignment Questions:

1. What do you mean by MultiThreading? Why it is important?

Ans → Multithreading is a programming concept that allows multiple threads (i.e., sequences of instructions that can be executed concurrently) to run within a single program. Each thread operates independently of the others and can access shared resources such as memory and CPU time.

Multithreading is important because it can improve the performance and responsiveness of programs, especially in systems that have multiple CPUs or cores. By dividing a program into smaller, independent threads that can run in parallel, it's possible to utilize the available hardware resources more effectively and achieve faster execution times.

1. What are the benefits of using Multithreading?

Ans → Using multithreading in software development can offer several benefits, including:

1. Improved performance: By dividing a program into smaller, independent threads that can run in parallel, it's possible to utilize the available hardware resources more effectively and achieve faster execution times.
2. Better resource utilization: Multithreading allows programs to make better use of available system resources, such as multiple CPUs or cores, by distributing processing across them.
3. Improved responsiveness: By running computationally expensive operations in a separate thread, the user interface thread can remain responsive and continue to process user input while the operation is running.
4. Simplified code: Multithreading can simplify the code of certain programs by allowing for more streamlined, parallel execution of code.
5. What is thread in Java?

Ans → In Java, a thread is a lightweight process that enables concurrent execution within a single program. Each thread has its own call stack and program counter, allowing it to execute code independently of other threads running within the same program.

1. What are the two ways of implementing thread in Java?

Ans →

a. Extending the Thread class: In this approach, you create a subclass of the Thread class and override the run() method to define the code that the thread will execute. You then create an instance of the subclass and call its start() method to begin the thread's execution.

class MyThread extends Thread {

public void run() {

// code to be executed by the thread

}

}

MyThread thread = new MyThread();

thread.start();

b. Implementing the Runnable interface: In this approach, you create a class that implements the Runnable interface and defines the thread's code in the run() method. You then create an instance of the class and pass it to a new Thread object, which you then start.

class MyRunnable implements Runnable {

public void run() {

// code to be executed by the thread

}

}

MyRunnable runnable = new MyRunnable();

Thread thread = new Thread(runnable);

thread.start();

1. What’s the difference between thread and process?

Ans →

1. Resource utilization: A process is a separate instance of a program that runs independently and has its own memory space and system resources, such as CPU time, memory, and files. A thread, on the other hand, shares the memory and resources of the process that created it.
2. Creation time and overhead: Creating a new process is more resource-intensive and time-consuming than creating a new thread because a new process requires its own memory space and system resources, while a new thread shares the resources of its parent process.
3. Communication and synchronization: Communication and synchronization between processes are more complex than between threads because processes operate in separate memory spaces and must use inter-process communication (IPC) mechanisms to share data and synchronize execution. Threads, on the other hand, can communicate and synchronize directly through shared memory and variables.
4. How can we create daemon thread in Java?

Ans →In Java, a daemon thread is a thread that runs in the background and does not prevent the JVM from exiting when the main thread has finished executing. Here's how you can create a daemon thread in Java:

1. Extend the Thread class and override the run() method with your thread's code.

class MyThread extends Thread {

public void run() {

// thread code here

}

}

1. Set the daemon flag of the thread to true using the setDaemon() method.

MyThread thread = new MyThread();

thread.setDaemon(true);

1. Start the thread using the start() method.

thread.start();

1. What are the wait() and sleep() methods?

Ans →In Java, both the wait() and sleep() methods are used for delaying the execution of a thread, but they differ in their behavior and usage:

1. wait() method: The wait() method is defined in the Object class and is used for inter-thread communication. When a thread calls the wait() method, it releases the lock it holds on the object's monitor and waits until another thread notifies it by calling the notify() or notifyAll() method on the same object.
2. sleep() method: The sleep() method is defined in the Thread class and is used to pause the execution of a thread for a specified amount of time. When a thread calls the sleep() method, it does not release any locks or monitors and simply waits for the specified amount of time before resuming its execution.