**Multithreading**

**Assignment**

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

Ans: Modern operating system has the capability to execute several programs simultaneously i.e. at the same time. This ability is known as multitasking.In system’s terminology, it is called “Multithreading” .

Multithreading is a conceptual programming where a program is divided into two or more sub programs which can be implemented at the same time in parallel.

In most of our computer there is only on processor and hence, in reality, the processor is doing only one thing at a time. However, the processor switches between the processes so fast that is appears to human being that all of them are being done simultaneously.

1. **What are the benefits of using Multithreading?**

Ans: The benefits of using Multithreading are:

1. **Resource Sharing-**All the threads of a process share its resources such as memory, data, files etc. A single application can have different threads within the same address space using resource sharing.
2. **Responsiveness-**Program responsiveness allows a program to run even if part of it is blocked using multithreading. This can also be done if the process is performing a lengthy operation. For example - A web browser with multithreading can use one thread for user contact and another for image loading at the same time.
3. **Utilization of Multiprocessor Architecture**-In a multiprocessor architecture, each thread can run on a different processor in parallel using multithreading. This increases concurrency of the system. This is in direct contrast to a single processor system, where only one process or thread can run on a processor at a time.
4. **Economy-**It is more economical to use threads as they share the process resources. Comparatively, it is more expensive and time-consuming to create processes as they require more memory and resources. The overhead for process creation and management is much higher than thread creation and management
5. **What is Thread in Java?**

Ans: A thread in java is similar to a program that has a single flow of control. It has a beginning, a body and an end. All the program which we have done till now are called as flow of execution.

Java has a unique property that it supports multithreading. That is, java enables us to use multiple flows of control in developing program and each flow of control is considered as a separate program known as thread.

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

Ans: There are two ways of implementing thread in java:-

* 1. Extending the Thread class
  2. Implementing Runnable interface

**Extending the Thread class**

**Example:**

My thread is in running state.

class MultithreadingDemo extends Thread

{

  public void run()

 {

     System.out.println("My thread is in running state.");

 }

  public static void main(String args[])

 {

    MultithreadingDemo obj=new MultithreadingDemo();

        obj.start();

  }

}

//Output: My thread is in running state.

**Implementing Runnable interface**

Example:

My thread is in running state.

class MultithreadingDemo implements Runnable

{

   public void run()

 {

      System.out.println("My thread is in running state.");

  }

    public static void main(String args[])

 {

      MultithreadingDemo obj=new MultithreadingDemo();

      Thread Obj =new Thread(obj);

tobj.start();

 }

}

//Output: My thread is in running state.

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

Ans:

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Process** | **Thread** |
| 1. | Process means any program is in execution. | Thread means a segment of a process. |
| 2. | The process takes more time to terminate. | The thread takes less time to terminate. |
| 3. | It takes more time for creation. | It takes less time for creation. |
| 4. | It also takes more time for context switching. | It takes less time for context switching. |
| 5. | The process is less efficient in terms of communication. | Thread is more efficient in terms of communication. |
| 6. | Multiprogramming holds the concepts of multi-process. | We don’t need multi programs in action for multiple threads because a single process consists of multiple threads. |
| 7. | The process is isolated. | Threads share memory. |
| 8. | The process is called the heavyweight process. | A Thread is lightweight as each thread in a process shares code, data, and resources. |
| 9. | Process switching uses an interface in an operating system. | Thread switching does not require calling an operating system and causes an interrupt to the kernel. |
| 10. | If one process is blocked then it will not affect the execution of other processes | If a user-level thread is blocked, then all other user-level threads are blocked. |
| 11. | The process has its own Process Control Block, Stack, and Address Space. | Thread has Parents’ PCB, its own Thread Control Block, and Stack and common Address space. |
| 12. | Changes to the parent process do not affect child processes. | Since all threads of the same process share address space and other resources so any changes to the main thread may affect the behavior of the other threads of the process. |
| 13. | A system call is involved in it. | No system call is involved, it is created using APIs. |
| 14. | The process does not share data with each other. | Threads share data with each other. |

1. **How can we create daemon threads?**

Ans: A daemon thread is a low-priority thread whose purpose is to provide services to user threads. Since daemon threads are only required when the user threads are operating, they do not prevent the JVM from quitting after all the user threads have completed execution.

Infinite loops, which are common in daemon threads, do not cause issues because no code (including finally blocks) is executed until all the user threads have completed execution. Therefore, daemon threads should not be used for I/O activities.

We can use the setDaemon method of the Thread class to create a daemon thread.

Syntax:

public final void setDaemon(boolean on)

Parameters:

on: If the value is True, then the current thread is marked as a daemon thread.

Return value:

This method doesn’t return anything.

**Example:**

public class Main

{

    public static void main(String[] args) throws InterruptedException

{

        Runnable runnable = () ->

{

            if(Thread.currentThread().isDaemon())

{

                System.out.println(Thread.currentThread().getName() + " is a daemon thread");

            }

else

{

                System.out.println(Thread.currentThread().getName() + " is not a daemon thread");

            }

        };

        Thread daemonThread = new Thread(runnable);

        daemonThread.setName("daemon-thread");

        daemonThread.setDaemon(true);

        Thread notDaemonThread = new Thread(runnable);

        notDaemonThread.setName("not-daemon-thread");

        daemonThread.start();

        notDaemonThread.start();

        Thread.sleep(1000);

}

}

Output:

daemon-thread is a daemon thread

not-daemon-thread is not a daemon thread

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

Ans: **wait ():-**

* Wait() method belongs to Object class.
* Wait() method releases lock during Synchronization.
* Wait() should be called only from Synchronized context.
* Wait() is not a static method.
* Wait() Has Three Overloaded Methods:
* wait()
* wait(long timeout)
* wait(long timeout, int nanos)
* public final void wait(long timeout)

Example:

synchronized (monitor)

{

Thread. sleep (1000); Here Lock Is Held By The Current Thread

//after 1000 milliseconds, current thread will wake up, or after we call that is interrupt() method

}

**sleep():-**

* Sleep() method belongs to Thread class.
* Sleep() method does not release the lock on object during Synchronization.
* There is no need to call sleep() from Synchronized context.
* Sleep() is a static method.
* Sleep() Has Two Overloaded Methods:
* sleep(long millis)millis: milliseconds
* sleep(long millis,int nanos) nanos: Nanoseconds
* public static void sleep(long millis) throws Interrupted\_Execption

example: synchronized(monitor)

{monitor.wait() Here Lock Is Released By Current Thread }