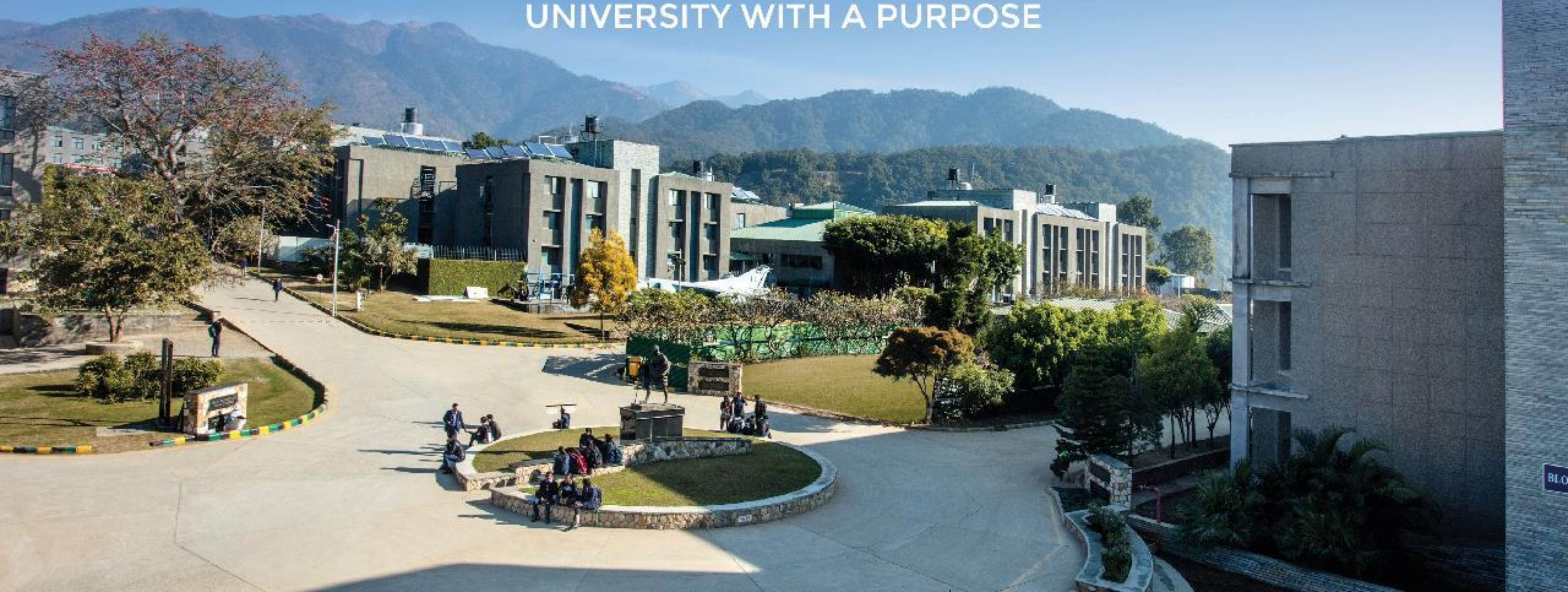
UPES

UNIVERSITY WITH A PURPOSE



Thread Scheduling and deadlock

Thread Scheduling

- The JVM schedules using a preemptive , priority-based scheduling algorithm.
- All Java threads have a priority and the thread with the highest priority is scheduled to run by the JVM.
- In case two threads have the same priority a FIFO ordering is followed.

- A different thread is invoked to run in case one of the following events occur:

1. The currently running thread exits the Runnable state i.e. either blocks or terminates.
2. A thread with a higher priority than the thread currently running enters the Runnable state. The lower priority thread is preempted and the higher priority thread is scheduled to run.

Java Thread Priority

- In a Multi threading environment, thread scheduler assigns processor to a thread based on priority of thread.
- Whenever we create a thread in Java, it always has some priority assigned to it. Priority can either be given by JVM while creating the thread or it can be given by programmer explicitly.
- Accepted value of priority for a thread is in range of 1 to 10. There are 3 static variables defined in Thread class for priority.
  - **public static int MIN_PRIORITY:** This is minimum priority that a thread can have. Value for this is 1.
  - **public static int NORM_PRIORITY:** This is default priority of a thread if do not explicitly define it. Value for this is 5.
  - **public static int MAX_PRIORITY:** This is maximum priority of a thread. Value for this is 10.

Java Thread Priority

Get and Set Thread Priority:

- **`public final int getPriority():`** `java.lang.Thread.getPriority()` method returns priority of given thread.
- **`public final void setPriority(int newPriority):`** `java.lang.Thread.setPriority()` method changes the priority of thread to the value newPriority. This method throws IllegalArgumentException if value of parameter newPriority goes beyond minimum(1) and maximum(10) limit.

Java Thread Priority

```
// Java program to demonstrate getPriority() and setPriority()
import java.lang.*;
class ThreadDemo extends Thread {
          public void run() {
                    System.out.println("Inside run method"); }
          public static void main(String[]args) {
                    ThreadDemo t1 = new ThreadDemo();
                    ThreadDemo t2 = new ThreadDemo();
                    ThreadDemo t3 = new ThreadDemo();

                    System.out.println("t1 thread priority : " + t1.getPriority()); // Default 5
                    System.out.println("t2 thread priority : " + t2.getPriority()); // Default 5
                    System.out.println("t3 thread priority : " + t3.getPriority()); // Default 5

                    t1.setPriority(2);
                    t2.setPriority(5);
                    t3.setPriority(8);
                    // t3.setPriority(21); will throw IllegalArgumentException
                    System.out.println("t1 thread priority : " + t1.getPriority()); //2
                    System.out.println("t2 thread priority : " + t2.getPriority()); //5
                    System.out.println("t3 thread priority : " + t3.getPriority());//8
                    // Main thread
                    System.out.print(Thread.currentThread().getName());
                    System.out.println("Main thread priority : " + Thread.currentThread().getPriority());
                    // Main thread priority is set to 10
                    Thread.currentThread().setPriority(10);
                    System.out.println("Main thread priority : " + Thread.currentThread().getPriority());
          }
}
```

Java Thread Priority

Note:

- Thread with highest priority will get execution chance prior to other threads. Suppose there are 3 threads t1, t2 and t3 with priorities 4, 6 and 1. So, thread t2 will execute first based on maximum priority 6 after that t1 will execute and then t3.
- Default priority for main thread is always 5, it can be changed later. Default priority for all other threads depends on the priority of parent thread.

Java Thread Priority

```
// Java program to demonstrate that a child thread gets same priority as parent
import java.lang.*;

class ThreadDemo extends Thread
{
        public void run()
        {
                System.out.println("Inside run method");
        }

        public static void main(String[]args)
        {
                // main thread priority is 6 now
                Thread.currentThread().setPriority(6);

                System.out.println("main thread priority : " +
                                Thread.currentThread().getPriority());

                ThreadDemo t1 = new ThreadDemo();

                // t1 thread is child of main thread
                // so t1 thread will also have priority 6.
                System.out.println("t1 thread priority : " + t1.getPriority());
        }
}
```

Deadlock

- Deadlock occurs when two threads have a circular dependency on a pair of synchronized objects.
- For example, suppose one thread enters the monitor on object X and another thread enters the monitor on object Y. If the thread in X tries to call any synchronized method on Y, it will block as expected. However, if the thread in Y, in turn, tries to call any synchronized method on X, the thread waits forever, because to access X, it would have to release its own lock on Y so that the first thread could complete.
- It may involve more than two threads and two synchronized objects. (That is, deadlock can occur through a more convoluted sequence of events than just described.)

Deadlock Example

Example:

```
class A {
synchronized void foo(B b) {
String name = Thread.currentThread().getName();
System.out.println(name + " entered A.foo");
try {
Thread.sleep(1000);
} catch(Exception e) {
System.out.println("A Interrupted");}
System.out.println(name + " trying to call B.last()");
b.last();       }
synchronized void last() {
System.out.println("Inside A.last");    }               }
class B {
synchronized void bar(A a) {
String name = Thread.currentThread().getName();
System.out.println(name + " entered B.bar");
try {
Thread.sleep(1000);
} catch(Exception e) {
System.out.println("B Interrupted");}
System.out.println(name + " trying to call A.last()");
a.last();       }
synchronized void last() {
System.out.println("Inside A.last");
}
}
class Deadlock implements Runnable {
A a = new A();
B b = new B();
Deadlock() {
Thread.currentThread().setName("MainThread");
Thread t = new Thread(this, "RacingThread");
t.start();
a.foo(b); // get lock on a in this thread.
System.out.println("Back in main thread");
}
public void run() {
b.bar(a); // get lock on b in other thread.
System.out.println("Back in other thread");
}
public static void main(String args[]) {
new Deadlock();
}
}
```

References

1. http://journals.ecs.soton.ac.uk/java/tutorial/java/threads/states.html
2. https://www.geeksforgeeks.org/java-thread-priority-multithreading/
3. Schildt, H. (2014). Java: the complete reference. McGraw-Hill Education Group..

THANK YOU

UPES

UNIVERSITY WITH A PURPOSE