Object Oriented Programming

Aman Kumar Gupta

CSE DevOps 18

R171218016-500067759

Q.1 Differentiate between process and thread?

	Process	Thread
1	An executing instance of a program is called a process.	A thread is a subset of the process.
2	Some operating systems use the term ‘task‘ to refer to a program that is being executed.	It is termed as a ‘lightweight process’, since it is similar to a real process but executes within the context of a process and shares the same resources allotted to the process by the kernel.
3	A process is always stored in the main memory also termed as the primary memory or random access memory	Usually, a process has only one thread of control – one set of machine instructions executing at a time.
4	Therefore, a process is termed as an active entity. It disappears if the machine is rebooted.	A process may also be made up of multiple threads of execution that execute instructions concurrently.
5	Several process may be associated with a same program.	Multiple threads of control can exploit the true parallelism possible on multiprocessor systems.
6	On a multiprocessor system, multiple processes can be executed in parallel.	On a uni-processor system, a thread scheduling algorithm is applied and the processor is scheduled to run each thread one at a time.
7	On a uni-processor system, though true parallelism is not achieved, a process scheduling algorithm is applied and the processor is scheduled to execute each process one at a time yielding an illusion of concurrency.	All the threads running within a process share the same address space, file descriptors, stack and other process related attributes.
8	Executing multiple instances of the ‘Calculator’ program. Each of the instances are termed as a process.	Since the threads of a process share the same memory, synchronizing the access to the shared data within the process gains unprecedented importance.

On your computer, open Microsoft Word and web browser. We call these two *processes*.

In Microsoft word, you type some thing and it gets automatically saved. Now, you would have observed editing and saving happens in parallel - editing on one thread and saving on the other thread.

Q.2 What is Thread Scheduler in java?

Thread scheduler in java is the part of the JVM that decides which thread should run. The scheduler maintains a pool of threads. When Java thread is started calling **start()** method, it joins the pool of waiting threads.

For deciding processor allocation for each waiting thread, the scheduler takes many aspects into consideration.

- Priority of thread
- Waiting time of thread
- Nature of thread

The JVM is based on **preemptive and priority based** scheduling algorithm. The thread with more priority is given first preference than the thread with less priority. The thread with more priority relinquishes (empties) the thread with less priority that is being executed. If the threads of equal priority are in the pool, the waiting time is taken in consideration. Nature of threads sometimes affects. The **daemon threads** are given less importance and are executed only when no other thread is available for execution.

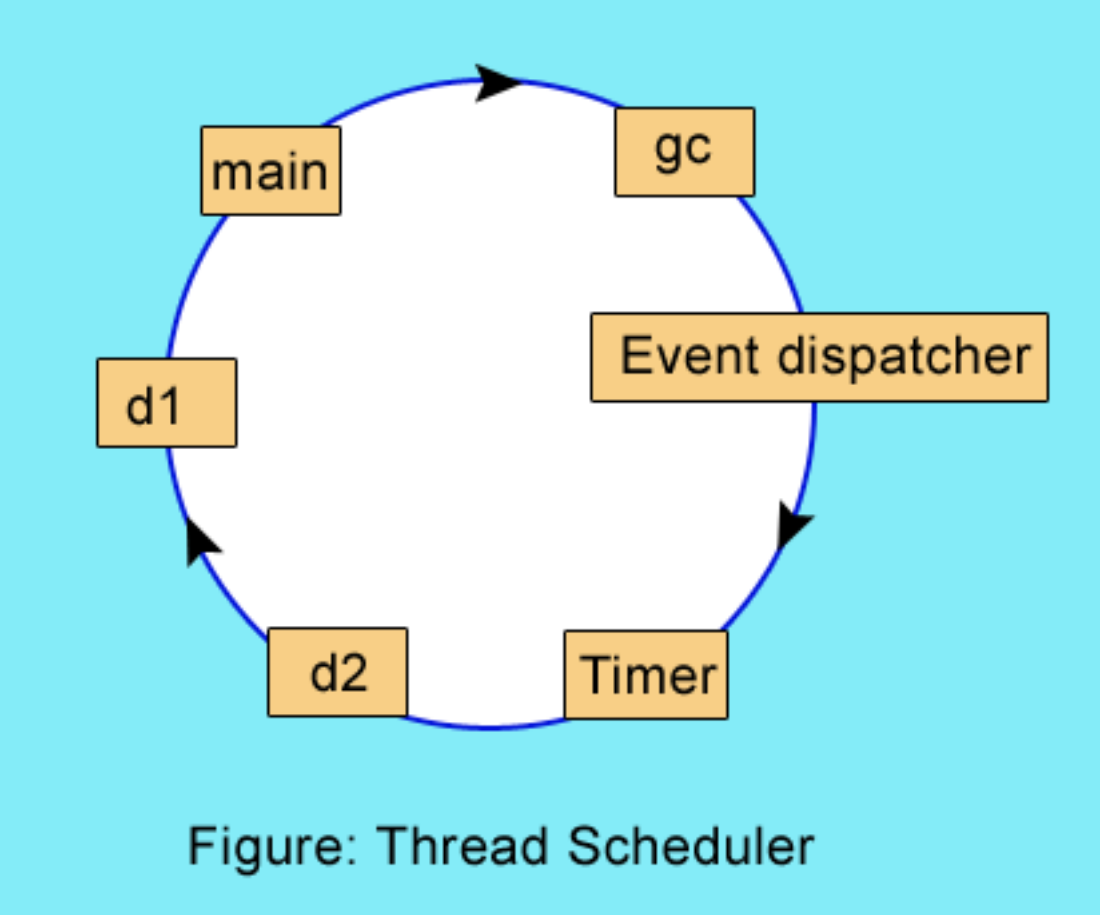


Figure: Thread Scheduler

Q.3 What is the synchronization? How to achieve it in Java Multithreading?

At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called synchronization.

The synchronization keyword in java creates a block of code referred to as critical section.

General Syntax:

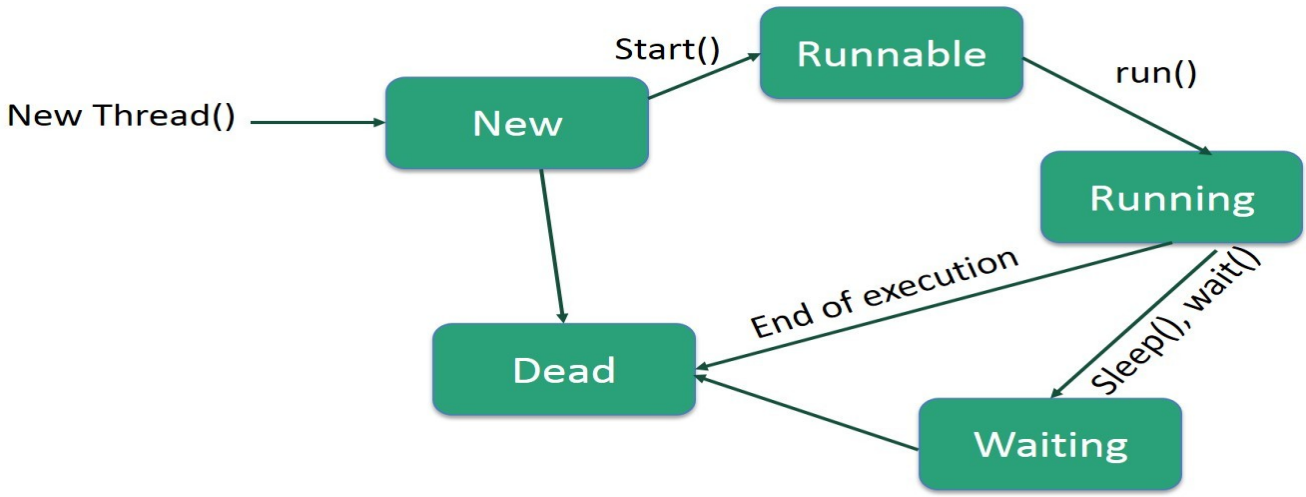
```
synchronized (object)
{
//statement to be synchronized
}
```

Every Java object with a critical section of code gets a lock associated with the object. To enter critical section a thread need to obtain the corresponding object's lock.

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms :

- Implementing the Runnable Interface
- Extending the Thread class



Q.4 What about the daemon threads?

Daemon thread is a low priority thread (in context of JVM) that runs in background to perform tasks such as garbage collection etc., they do not prevent the JVM from exiting (even if the daemon thread itself is running) when all the user threads (non-daemon threads) finish their execution.

JVM terminates itself when all user threads (non-daemon threads) finish their execution, JVM does not care whether Daemon thread is running or not, if JVM finds running daemon thread (upon completion of user threads), it terminates the thread and after that shutdown itself.

Properties:

- They can not prevent the JVM from exiting when all the user threads finish their execution.
- JVM terminates itself when all user threads finish their execution
- If JVM finds running daemon thread, it terminates the thread and after that shutdown itself. JVM does not care whether Daemon thread is running or not.
- It is an utmost low priority thread.

Q.5 Write a program in java having two different thread classes;

1. for searching a word in a given text file that prints whether the word is present in the document or not,
2. will calculate the frequency of each word present in the document.

Create following threads:

1. A thread to search a word in Document1.txt
2. A thread to search a word in Document2.txt
3. A thread to print a table of word and corresponding frequency available in Document3.txt

Start all three threads in main class and capture the output.
Note: Use File I/O to read the document.

```
import java.io.File;
import java.util.Scanner;
import java.lang.*;
class SearchWord extends Thread{
    File  Obj;
    String word;
    Boolean present=false;
    SearchWord(String word, String fileName){
        this.word = word;
        Obj = new File(fileName);
    }
    public void run(){
        try{
        Scanner sc = new Scanner(Obj);
        while (sc.hasNextLine()) {
        String data = sc.nextLine();
        String sp[] = data.split(" ");
        for(int i=0; i<sp.length ; i++ ){
            if(sp[i].contains(word))
                present = true;
        }
        if(present)
            break;
        }
if(present){
        System.out.println("The word "+word+" is present");}
else {System.out.println("The word "+word+" is not present ");}
    }catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
}
public class Main {
    public static void main(String args[]){
        SearchWord t1 = new SearchWord("Java","Doc1.txt");
        SearchWord t2 = new SearchWord("Cpp","Doc2.txt");
        t1.start();
        t2.start();
    }
}
```

```
aman@amanlaptop: ~/old/backup/h/StudyMaterial/2ndYear/Java/WeeklyAssignment/ASS1
aman@amanlaptop:~/old/backup/h/StudyMaterial/2ndYear/Java/WeeklyAssignment/ASS1$ javac Main.java
aman@amanlaptop:~/old/backup/h/StudyMaterial/2ndYear/Java/WeeklyAssignment/ASS1$ java Main
The word Cpp is present
The word Java is present
aman@amanlaptop:~/old/backup/h/StudyMaterial/2ndYear/Java/WeeklyAssignment/ASS1$
```