What are the problems occurs in multithreading?

1. Deadlock

2. starvation

3. livelock

Deadlock - when multiple threads are depending on each other resource(single instance).

Thread1 - Resource1

Thread2- Resource2

Thread1 tries to access Resource2

Thread2 tries to access Resource1

Jebus - Pencil

Prakash - Compass

Both are waiting for each other resources.

Starvation - thread is unable to gain the regular access to the shared resources and unable to make progress.

Livelock - thread often act in response to other threads.

thread 1 - execute and give the output1

thread 2 - gets the output from thread1 and give the output2

thread 3 - gets the output from thread and give output3

========================================================

int i;

Thread t1=new Thread();

Thread t2=new Thread();

Thread t2=new Thread();

Whether all these thread Object will share same copy of i variable or different copy?

different copy..

if we want the same copy to used by all thread Objects.

volatile int i;

=======================================================

==========================

Concurrency :

===========

ChandraKumar, Arun and Vaibhav - they have to create a PPT

on JAVA Technology.

Chandrakumar - Introduction

Arun - main content

Vaibhav - advantages and disadvantages.

Chandrakumar - 15 minutes to create the PPT.

Then Arun creates the PPT - 25 minutes.

Vaibhav - creates the PPT - 10 minutes.

Total time spent = 50 minutes.

Chandrakumar, Arun and Vaibhav - all creates PPT at the same and then combine

it.

Total time spent = 20 or 25 minutes.

doing the task simultaneously to save the time.

to print the prime numbers from 1 to 10,000

processor 1 - alone is doing the activity, it will take time.

processor 1 - 1 to 1000

processor 2 - 1001 to 5000

processor 3 - 5001 to 8000

processor 4 - 8001 to 10000

== these kind of activities are achieved using the package

java.util.concurrent package.

1. java.util.concurrent.atomic package.

- atomic - act a single unit.

AtomicInteger a=new AtomicInteger(10);

thread1 = 10

thread2 = 10

thread3 = 10

==it is used to maintain the values across the threads.

2. java.util.concurrent.locks package.

write.lock();

//code

write.unlock();

read.lock();

//code

read.unlock();

### ReadWriteLock

The java.util.concurrent.locks package also contains a ReadWriteLock interface (and ReentrantReadWriteLock implementation) which is defined by a pair of locks for reading and writing, typically allowing multiple concurrent readers but only one writer. Example of using an explicit ReentrantReadWriteLock to allow multiple concurrent readers:

public class Statistic {

private final ReadWriteLock lock = new ReentrantReadWriteLock();

private int value;

public void increment() {

lock.writeLock().lock();

try {

value++;

} finally {

lock.writeLock().unlock();

}

}

public int current() {

lock.readLock().lock();

try {

return value;

} finally {

lock.readLock().unlock();

}

}

}

3. java.util.concurrent.synchronized<T> package

Synchronization :

=============

when the activities are happening at the same time.

Five classes of Synchronizer:

======================

1. Semaphore

2. CountDownLatch

3. Phaser

4. Exchanger

5. Cyclic Barrier.

**Semaphore** - is a classic concurrency tool , maintains a set of permits and may block until other threads release permits.

* Used to restrict access to shared resource

Acquire () and release() methods are used

Thread1 - resouce

Thread 2 - wants the resource [Thread2 has to wait until thread1 completes it]

===============================================================

**CountDownLatch** - allows one or more threads to wait until the completion of the countdown.(until set of opeartions is performed)

getCount()-Retrieve count

await()- outs thread to sleep

countDown()-reduce count

when it reaches zero all sleeping threads released

Thread1 - has to start after 5 seconds.

Thread2 - has to start after 15 seconds.

================================================================

Phaser –

useful where independent threads need to work in phases or steps in order to complete task.

threads can register/de-register over time, causing no. of threads require before advancement to change.

- Register(),

arriveAnd Deregister(),

arriveAndAwaitadvance()

onAdvance(),

There are two candidates waiting for the interview.

candidate 1 has to enter first, and then candidate-2..

candidate 1 has got an emergency and went out without attending interview.

candidate 2 can attend interview..

t1 - is taking to much time to execute, put this in queue

t2 - can start executing..

t3

================================================================

Exchanger - allows one or more threads to swap a pair of Objects, blocking until the exchange take place.

Wait for common pointof exchange

Ram and Ravi. Ram has got Chhapati , Ravi has got Veg rice.

Ram and Ravi can exchange their foods.

Thread 1- Resource1, Thread2 - Resource2

Thread can exchange resource , based on the requirement

exchange() methods

================================================================

CyclicBarrier –multiple threads are required to wait at predefined execution point called barrier.

created with the party count. after the number of parties have called await (block) , then barrier will be released.

Project - 5 threads.

Thread 1 and Thread 2 - are currently in use.

Thread 3 will be called.

=================================================================

Ram has got two set of Trousers. he is planning to wear one trouser as of now.

he wants to use trouser2 once the trouser1 gets old.

ans: Cyclic Barrier.

Question 2:

Ram has used these two trousers. he felt it is old. he wanted to give this to some one else .

ans: Semaphore

Question 3:

Mom,Dad and son plans to go for a movie. Dad got stuck in traffic while returning back to home. so Son had told Dad to come to theatre directly ..

ans: Phaser

==============================================================

Cyclic Barrier Example:

==================

import java.util.concurrent.\*;

class Demo23

{

public static void main(String[] args)

{

final CyclicBarrier c=new CyclicBarrier(2);

new Thread()

{

public void run()

{

try{

System.out.println("before thread1 call...");

c.await();

System.out.println("Thread1");

System.out.println("after thread1 call..");

}

catch(Exception e){}

}

}.start();

new Thread()

{

public void run()

{

try{

System.out.println("before thread2 call...");

c.await();

System.out.println("after thread2 call..");

}

catch(Exception e){}

}

}.start();

}

}

High level Threading Techniques:

=========================

option to create a group - facebook/wats app

to get some information, address all the same time, to interact with many people..

create a pool of threads.

pool of 5 threads - t1,, t2, t3, t4 and t5

do a task - printing even numbers from 1 to 500.

t1, t3 and t4

t1 - find even no from 1 to 100 - completed

t2 - even no from 101 to 250

t3 - even no from 251 to 400

t1 -- can do from 401 to 500

These kind of task are called as ExecutorService. which allows to create and reuse a group of thread objects by storing them into a pool.

Fork and Join Frameworks:

=====================

used in Divide and Conquer approach.

splitting the complex task into a simpler component modules.

demo(5) demo(6) demo(7) demo(8)................ etc

thread1 thread2 thread3 thread4

module module2 module3 module4

class calculator -> void add(){}

void sub(){}

void mul(){}

void div(){}

the reason why do we split methods are to make the task simpler.

Fork - divide the task , do the task and compute the result.

Join- combine the results of all the task and gives the output.

ForkJoinPool f = new ForkJoinPool(4);

Even task1=new Even(100);

Even task2=new Even(200);

add(task1);

add(task2);

task.fork();

task.join();

Executor Service:

demo(5) =10

demo(6)=15

demo(8)=28

demo(7)=21

find even numbers from 1 ..100

thread 1 - evenn from 1 to 50

thread2 - even from 51 to 100

thread1 = 2 4 6 8 10.........50

thread 2 = 52 54 56...........100

join() = 2 4 6 8 10.................................100