**OOPS weekly Assignment**

**ABHISHEK SHARMA**

**R171218005**

**500067644**

Q.1 Differentiate between process and thread?

1. A process is an executing program whereas, the thread is a small part of a process.
2. Each process has its own address space whereas, the threads of the same process share the address space as that of the process.
3. In process-based multitasking, more than two processes can run at the same time whereas, in thread-based multitasking, more than two thread can run at the same time.
4. Inter-process communication between two processes is costlier than inter-thread communication.
5. Context switching between two processes is expensive and limited as compared to context switching between two threads.

Q.2 What is Thread Scheduler in java?

Thread scheduler in java is the part of the JVM that chooses which thread should run. There is no assurance that which runnable thread will be picked to run by the thread scheduler. Only one thread at a time can run in a single process. Thread scheduler internally works in multi-threading.

Under pre-emptive scheduling, the most elevated need task executes until it enters the pausing or dead states or a higher need task appears. The scheduler then determines which task should execute next, based on priority and other factors.

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

When more than one thread attempts to access a common resource, we have to guarantee that resource will be utilized by just each thread in turn. The process by which this is accomplished is called synchronization. when we don't utilize synchronization, and let at two or more threads get to a mutual resource simultaneously, it will prompt distorted outcomes.

We can synchronize our code in either of two ways. Both involve the use of the synchronized keyword.

1. **Using Synchronized Methods**: To make a method synchronized, simply add the synchronized keyword to its declaration
2. **Using Synchronized Statement or Block**: to synchronize access to objects of a class that does not use synchronized methods, put calls to the methods defined by this class inside a synchronizedblock.

Q.4 What about the daemon threads?

* Daemon thread is a low priority thread that runs in background to perform tasks such as garbage collection.
* It doesn’t prevent the JVM from exiting once all user threads have finished their execution.
* It is an utmost low priority thread whose only role is to provide services to user threads.

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.

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.io.BufferedReader;

class searchWordThread extends Thread

{

String fname;

String wordSearch;

searchWordThread(String nm,String wd){

fname=nm;

wordSearch=wd;

}

synchronized public void run(){

try {

if(search())

{System.out.println("found word '"+wordSearch+"' in "+fname);

}

else{

System.out.println("'"+wordSearch+"' not found in "+fname);

}

} catch (Exception e) {

//TODO: handle exception

}

}

public boolean search() throws IOException

{

String line;

// check if File exists or not

FileReader fr=null;

BufferedReader br=null;

try

{

fr = new FileReader(fname);

br = new BufferedReader(fr);

}

catch (FileNotFoundException fe)

{

System.out.println("File not found!");

}

// read from FileReader till the end of file

String word[];

while ((line=br.readLine())!=null)

{

word=line.split(" ");

for(String w : word)

{

if(w.equals(wordSearch))

{//System.out.println("found "+w);

fr.close();

br.close();

return true;

}

}

}

// close the file

fr.close();

br.close();

return false;

}

}

class WordCountThread extends Thread

{

String fname;

String wordSearch;

WordCountThread(String nm,String wd){

fname=nm;

wordSearch=wd;

}

synchronized public void run(){

try {

// Thread.sleep(10);

System.out.println("Word frequency: "+count());

} catch (Exception e) {

//TODO: handle exception

}

}

public int count() throws IOException

{

String line;

FileReader fr=null;

BufferedReader br=null;

try // check if File exists or not

{

fr = new FileReader(fname);

br = new BufferedReader(fr);

}

catch (FileNotFoundException fe)

{

System.out.println("File not found!");

}

String word[];

int count=0;

while ((line=br.readLine())!=null)

{

word=line.split(" ");

for(String w : word)

{

if(w.equals(wordSearch))

{

count++;

}

}

}

// close the file

fr.close();

br.close();

return count;

}

}

class weekly\_access\_1

{

public static void main(String args[])

{

try {

searchWordThread mt1 = new searchWordThread("doc1.txt","THE");

searchWordThread mt2 = new searchWordThread("doc2.txt","abcd");

WordCountThread mt3=new WordCountThread("doc3.txt","THE");

WordCountThread mt4=new WordCountThread("doc3.txt","war");

Thread t1=new Thread(mt1);

Thread t2=new Thread(mt2);

Thread t3=new Thread(mt3);

Thread t4=new Thread(mt4);

t1.start();

t2.start();

t2.join();

System.out.println("\nFor doc 3: ");

System.out.println("WORD | FREQUENCY");

System.out.println("--------------------");

System.out.print("THE | ");

t3.start();

t3.join();

System.out.print("war | ");

t4.start();

} catch (Exception e) {

//TODO: handle exception

}

}

}