Question DS écrit java

4x Expliquez ce qu'est un code ré entrant ou thread safe.

Un code thread safe est un code capable de fonctionner correctement alors qu'il est exécuté simultanément au sein du même espace d'adressage par plusieurs thread.

4x Donnez un exemple de code C ou en java qui ne soit pas ré entrant et pourquoi ?

```
package fr.esisar;
public class Somme
{
int c;
public int somme(int a, int b)
{
    c=a+b;
    System.out.println("c="+c);
    return c;
}
}
```

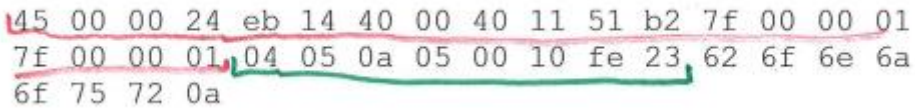
Non thread safe, car pas synchroniser

4x Donnez une solution pour rendre votre code thread safe

```
package fr.esisar;
public class Somme
{
int c;
synchronised public int somme(int a, int b)
{
    c=a+b;
    System.out.println("c="+c);
    return c;
}
}
```

Ajouter un synchronised

3x Un client envoie un datagramme UDP à un serveur. Vous avez réalisé une capture du paquet IP correspondant à cet envoi (capture en héxadécimal).



Sur quel port écoute le serveur (réponse en décimal) ? Justifiez votre réponse.
Quel est le port du client (réponse en décimal) ? Justifiez votre réponse.

Nous savons que la partie en rouge sur le datagramme correspond à l'entête IP composé de 20 octets. 7F 00 00 01 est l'IP source et 7F 00 00 01 est l'IP destinataire. La partie en vert correspond au 8 octets de l'entête UDP. 0405 correspond au port source et 0A05 correspond au port destinataire.

A05 = 2565 et 405 = 1029.

3x Quel est le numéro de port pour un serveur http ? Qui est 127.0.0.1 ?

Le numéro de port pour un serveur http est le port 80. 127.0.0.1 est l'adresse localhost ce qui signifie qu'il s'agit de l'interface de la machine local.

2x D'après la formule de Leibniz, on sait que :

$$\frac{\pi}{8} = \frac{1}{1\times 3} + \frac{1}{5\times 7} + \ldots + \frac{1}{(4k+1)(4k+3)} + \cdots$$

Sur une machine 4 processeur écrivez un programme en Java calculant cette suite le plus rapidement possible. Avec k = 100 000 000. Votre programme devra afficher la valeur de PI calculée, et le temps mis pour faire ce calcul.

```
package test;

public class pi32 extends Thread{
long start;
long end;
double res =0;
double tampon;
public pi32(long start, long end)
{
    this.start = start;
    this.end = end;
}

public void run()
{
    for(double i = start; i<=end;i++) { //boucle qui calcul la formule de leibnitz
        tampon = 1/((4*i+1)*(4*i+3));
        res = res+tampon;
    }
}

    public static void main(String[] args) throws InterruptedException{
        // TODO Auto-generated method stub
long start = 0;
double start_time =System.currentTimeMillis();
long end = 25000000;
double pi=0;

pi32[]arrayla = new pi32[4];

for(int i =0;i<4;i++) {
    arrayla[i]= new pi32(start,end);
    start=end+1;
    end=end+25000000; //création des 4 threads chaque thread s'occupe de 1/4 de k
}
for(int i=0;i<4;i++) {
    arrayla[i].start();
    arrayla[i].join();  //attend la fin d'un thread
}

for(int i=0;i<4;i++) {
    pi+=arrayla[i].res;
}

System.out.println("pi/8= "+pi);
    long now = System.currentTimeMillis();
    System.out.println("Le temps d'éxécution est de "+((now-start_time)/1000)+ "s");
    }
}
```

2x Expliquez dans quel cas un programme se termine avec une erreur « segmentation fault ».

Une erreur segmentation fault signifie qu'un processus essaie de lire ou d'écrire la mémoire d'un autre processus.

2x Nous considérons deux processus P1 et P2 communiquant entre eux par TCP. P1 est exécuté sur la machine A, P2 sur la machine B. P1 joue le rôle du serveur, P2 joue le rôle du client.

Le scénario est le suivant :

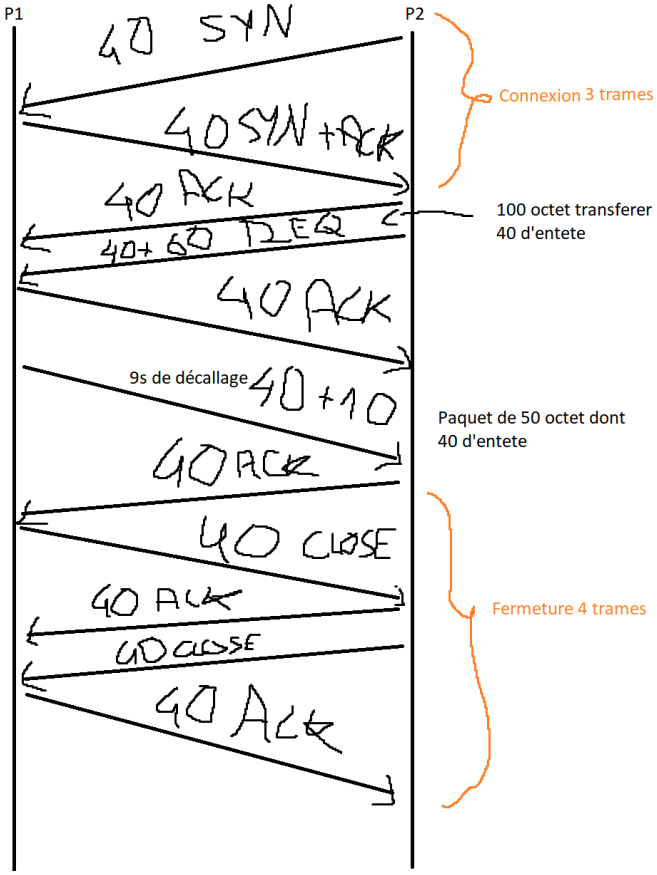
- P2 initie la connexion
- P2 envoie une demande à P1 avec 100 octets au niveau applicatif
- P1 a besoin de 9 secondes pour calculer sa réponse
- P1 répond avec 50 octets au niveau applicatif
- la connexion est ensuite fermée par P1 puis P2

Décrivez tous les paquets IP échangés entre P1 et P2 (taille, contenu). Faites un calcul de l'overhead apporté par les couches 3 et 4 du modèle OSI.

Entête TCP 20 octets et IP 20 octet et UDP 8 octets

40 -> 20 octet entete IP + 20 entete TCP

Overhead = 11*40/70 =628%

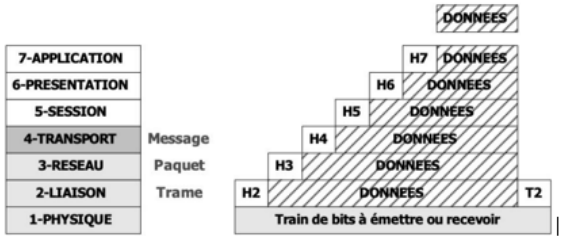


2x Indiquez par un dessin les 7 couches du modèle OSI.
Pour les couches 1 à 4 donnez un exemple d'implémentation

Modèle OSI

Pour Le Réseau Tout Se Présente Automatiquement

Couches matérielles	Couches Hautes
1 Physique(BIT)/câble 2 Liaison(TRAME)/Ethernet 3 Réseau (Paquet)/IP	4 Transport (Segment)/TCP-UDP 5 Session (Donnée)/RPC 6 Présentation (Donnée)/ASCII 7 Application (Donnée)/HTTP



1. Trains de bits (BITS) wifi
2. Adresse physique (TRAME) can
3. Détermine le parcours et l'adressage logique IP (Paquet) IP
4. Connexion de bout en bout et contrôle de flux TCP (Segment) TCP UDP

1x Expliquez le fonctionnement de l'API des sockets pour TCP

1. Le serveur se déclare auprès de la couche transport
2. Phase de connexion entre le premier client et le serveur
3. Acceptation du premier client par le serveur
4. Echange de données bidirectionnelle entre le 1er client et le serveur
5. Terminaison de la connexion entre le 1er client et le serveur

1x Donnez 5 techniques permettant d'implémenter un serveur parallèle. Donnez des exemples de logiciels utilisant ces techniques pour au moins 3 techniques. Détaillez le fonctionnement de chaque technique.

1. **Serveur multi processus** -> Le travail est délégué à un processus fils crée via fork().
2. **Serveur multi processus avec réservoir de processus** -> processus père gère un réservoir de processus fils avec une stratégie de nombre moyen de fils. Le travail est délégué au fils inoccupé.
3. **Serveur multi threadé standard** -> Utilise des threads à la place des processus. Le travail est délégué à un thread fils crée via new Thread().
4. **Serveur multi threadé avec réservoir de processus** -> le thread père gère un réservoir de thread fils avec une stratégie de gestion de fils moyen. Le travail est délégué au fils inoccupé. **Exemple : TOMCAT**
5. **Serveur parallèle classique** -> Une tâche attend les demandes de connexion qui les vérifie valide la demande puis la transfère à une tâche fille.

1x Quelle est l'année d'invention du WEB ?

Le WEB a été inventé en 1989.

1x Expliquez les termes little endian et big endian

Little endian correspond à l'écriture de l'octet de poids faible à l'adresse la plus petite (envoie de l'octet de poids faible en premier).

Big endian correspond à l'écriture de l'octet de poids fort à l'adresse la plus petite (envoie de l'octet de poids fort en premier).

1x Expliquez comment est codé le nombre 770 dans un entier 32bits dans chaque cas (little et big endian) Pour chaque octet vous donnerez sa valeur en décimal.

Décomposition de 770 en puissance de 2 pour le big endian. On décompose donc 1*2^1+1*2^8+1*2^9 ce qui nous donne en HEXA 0x302. Pour passer de big endian à little endian il suffit de permuter le premier et le dernier octet. Nous obtenons donc 0x203 ce qui correspond à 770 en décimale.

1x Que signifie « network byte ordre » ? Pourquoi a-t-on du inventer cette notion ?

Network bit order correspond au format big endian du TCP IP. Nous avons dû inventer cette notion, car les deux standards existent en informatique (big et little endian) et pour éviter des problèmes de communications il est important que tout le monde parle le même langage.

1x Si un paquet IP est perdu lors d'un échange par UDP, quel est l'impact sur votre programme ? En tant que programmeur, qu'avez-vous à faire par rapport à la perte d'un paquet IP si vous utilisez UDP ?

Le paquet est perdu et ne sera pas envoyé renvoyer. Il faut donc implémenter une gestion d'exception qui demande le renvoi de la trame.

1x Si un paquet IP est perdu lors d'un échange par TCP, quel est l'impact sur votre programme ? En tant que programmeur, qu'avez-vous à faire par rapport à la perte d'un paquet IP si vous utilisez TCP ?

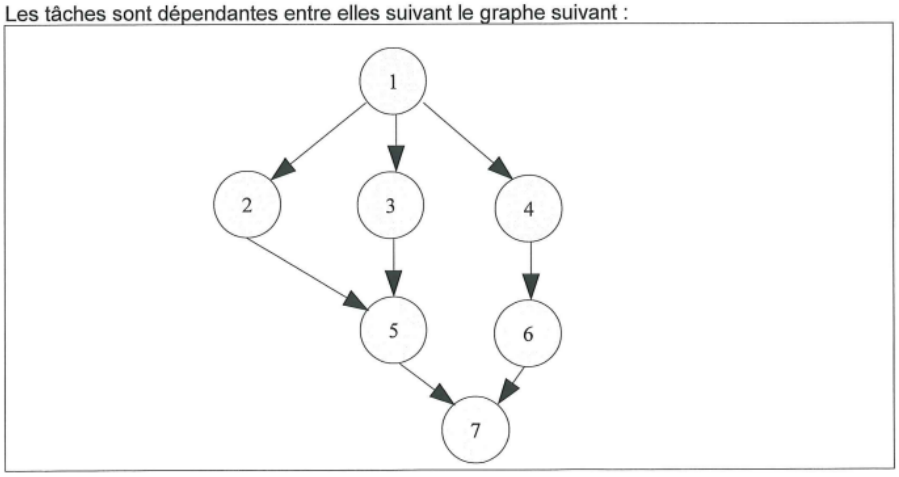
Le paquet sera envoyé dans une trame ultérieure. Il faut donc vérifier que la trame soit complète, si ce n'est pas le cas il faut attendre la réception de la fin de la trame avant de traiter les données reçues.

1x On vous fournit le code suivant :

```
public class Job
{
	public void t1() {
		// ... Réalise des calculs ...
	}

	public void t2() {
		// ... Réalise des calculs ...
	}

	public void t3() {
		// ... Réalise des calculs ...
	}

	public void t4() {
		// ... Réalise des calculs ...
	}

	public void t5() {
		// ... Réalise des calculs ...
	}

	public void t6() {
		// ... Réalise des calculs ...
	}

	public void t7() {
		// ... Réalise des calculs ...
	}
}
```

Cette classe contient 7 fonctions, chaque fonction correspond à une tâche à réaliser. La durée d'exécution de chaque fonction n'est pas connue.

Les tâches sont dépendantes entre elles suivant le graphe suivant :



Explication :

- la tâche 2 peut se faire uniquement si la tâche 1 est terminée
- la tâche 5 peut se faire si les tâche 2 et 3 sont terminées
- ...

Donnez le code d'une classe JobManager, qui va exécuter les 7 tâches le plus rapidement possible (c'est à dire en parallélisant les tâches si cela est possible) et qui respecte les contraintes de séquencement.

```java
package test3;

public class job{

    public void t1() {
        System.out.println("t1");

    }
    public void t2() {
        System.out.println("t2");

    }
    public void t3() {
        System.out.println("t3");

    }
    public void t4() {
        System.out.println("t4");

    }
    public void t5() {
        System.out.println("t5");

    }
    public void t6() {
        System.out.println("t6");

    }
    public void t7() {
        System.out.println("t7");

    }
}
```

```java
package test3;
import java.util.Random;

public class JobManager extends Thread{
    Integer id, rands, state;
    JobManager prev1;
    JobManager prev2;
    job job2;

    public JobManager(job job2, int id, JobManager prev1, JobManager prev2)
    {
        this.id = id;
        this.prev1 = prev1;
        this.prev2 = prev2;
        this.state = 2;
        this.job2 = job2;
        Random rand = new Random();
        rands = rand.nextInt(6000);
    }
    public JobManager(job job2, int id, JobManager prev1)
    {
        this.id = id;
        this.prev1 = prev1;
        this.state = 1;
        this.job2 = job2;
        Random rand = new Random();
        rands = rand.nextInt(6000);
    }
    public JobManager(job job2, int id)
    {
        this.id = id;
        this.state = 0;
        this.job2 = job2;
        Random rand = new Random();
        rands = rand.nextInt(6000);
    }
    @Override
```

```java
    @Override
    public void run() {
        try {
            switch(state)
            {
            case 0:
                break;
            case 1:
                prev1.join();
                break;
            case 2:
                prev1.join();
                prev2.join();
                break;
            }
            switch(id) {
            case 1:
                job2.t1();
                break;
            case 2:
                job2.t2();
                break;
            case 3:
                job2.t3();
                break;
            case 4:
                job2.t4();
                break;
            case 5:
                job2.t5();
                break;
            case 6:
                job2.t6();
                break;
            case 7:
                job2.t7();
                break;
            }
            System.out.println("Thread en cours: "+this.id.toString());
            sleep(rands);

        } catch (InterruptedException e1) {
            e1.printStackTrace();
        }
    }
}
```

```java
package test3;


public class TDM7
{
    public static void main(String[] args) throws InterruptedException
    {

        //public JobManager(job job2, int id, JobManager prev1, JobManager prev2, JobManager state)
        //public JobManager(job job2, int id, JobManager prev1)
        //public JobManager(job job2, int id, job job2)

        job job1 = new job();

        JobManager t1 = new JobManager(job1,1);
        JobManager t2 = new JobManager(job1,2,t1);
        JobManager t3 = new JobManager(job1,3,t1);
        JobManager t4 = new JobManager(job1,4,t1);
        JobManager t5 = new JobManager(job1,5,t2,t3);
        JobManager t6 = new JobManager(job1,6,t4);
        JobManager t7 = new JobManager(job1,7,t5, t6);

        t1.start();
        t2.start();
        t3.start();
        t4.start();
        t5.start();
        t6.start();
        t7.start();
        t7.join();


        System.out.println("fini");
    }
}
```