TD2

Exercice 1 : File d'attente bornée

Une file d'attente est une structure de stockage de données qui fonctionne selon une politique *FIFO* (First In First Out). On souhaite développer une classe File qui implante une file d'attente bornée i.e. de capacité limitée. La file peut être partagée entre différents programmes (threads) qui peuvent lui ajouter ou retirer des objets. Un thread qui tente d'ajouter un objet à une file pleine doit être suspendu tant que la file est pleine. De même, un thread qui souhaite retirer un objet d'une file vide doit être suspendu tant que la file est vide.

Donner le code de la classe File.

Développer une classe Producer qui permet d'instancier un thread qui dépose un nombre aléatoire de valeurs aléatoires dans une file d'attente. *Rappel. La fonction* random.randint(a,b) *retourne un entier aléatoire dans l'intervalle [a,b].*

Développer une classe Consumer qui permet d'instancier un thread qui lit et qui affiche sur l'écran les valeurs lues à partir d'une file d'attente. La lecture des valeurs se fait à un un rythme régulier.

Donner un programme de teste de fonctionnement des trois classes précédentes.

Exercice 2 : Simulation d'un commutateur Ethernet

Nous souhaitons développer un programme de simulation d'un commutateur Ethernet. Un commutateur est un équipement d'interconnexion de réseaux locaux qui opère au niveau 2. Un commutateur offre un nombre *pair* de ports de connexion. Une interface ethernet d'une machine peut être connectée à un seul port à la fois.

Une trame ethernet est composée d'un ensemble de champs dont les principaux sont :

- L'adresse MAC[1] source : une adresse MAC est définie par une suite de 6 octets donnés en notation hexadécimale. Par exemple, la chaîne de caractères suivante AABBCCDD0910 représente une adresse MAC valide. La chaîne AABBCCDD09 ne représente pas une adresse MAC valide comme elle est constituée de 10 chiffres hexadécimaux seulement. La chaîne AABBCCDD09XX n'est pas valide non plus comme elle contient des caractères non hexadécimaux!
- L'adresse MAC destination.
- Le champs de données : représenté par une liste d'objets.

La commutation est faite selon un algorithme simple : en recevant une trame Ethernet sur un port p, le commutateur associe l'adresse MAC source de la trame au port de réception. Cette association (adresse MAC, port) est sauvegardée dans une *table de commutation*. Pour chaque trame reçue sur un port p, le commutateur cherche dans la table de commutation le port associé à l'adresse destination de la trame. Trois cas peuvent avoir lieu :

- L'adresse de destination est associée à un port $q \neq p$. Dans ce cas la trame sera envoyée sur le port q.
- L'adresse destination est associée au port p lui même. Dans ce cas le commutateur ne fait rien.
- L'adresse destination n'est associée à aucun port. Dans ce cas la trame sera diffusée sur tous les ports (sauf p).

Les association entre une adresse MAC et un numéro de port ont une durée de vie t limitée. (i.e. l'association s'efface au bout t seconds).Naturellement, une adresse MAC ne peut pas être associée à plusieurs ports en même temps.

[1] . media access control address

Pour soucis de simplification on vous donne dans la suite le code des deux classes MAC et ThrameEthernet:

```
class MAC:
    def __init__(self,adr):
        assert isinstance(adr,str) and len(adr)==12 and all(c in
            string.hexdigits for c in adr)
        self.adr=adr
    def __cmp__(self,other):
        assert isinstance(other,MAC)
        if self.adr==other.adr:
            return 0
        else:
            return 1
    def __str__(self):
        return self.adr
    def __setattr__(self,att,val):
        if att=="adr":
            assert isinstance(val,str) and len(val)==12 and all(
                c in string.hexdigits for c in val)
            self.__dict__[att]=val
    def __hash__(self):
        return hash(self.adr)
```

Rappel : la méthode __hash__ détermine la fonction de *hashage* à utiliser. Nous utilisons une fonction de hashage sur la chaîne de caractères qui représente l'adresse MAC. (i.e. Utiliser la fonction prédéfinie hash()).

```
class TrameEthernet:
    def __init__(self,src,dst,data):
        assert isinstance(src,MAC)
        assert isinstance(dst,MAC)
        assert isinstance(data,list)

        self.src=src
        self.dst=dst
        self.data=data

    def getSrc(self):
        return self.src
    def getDst(self):
        return self.dst
    def getDate(self):
        return self.data
    def __str__(self):
        return "%s:%s"%(self.src,self.dst)
```

Proposer une classe Switch qui permet de simuler le fonctionnement d'un commutateur. Les ports d'un commutateur doivent pouvoir travailler d'une manière concurrente. Un port doit fournir les deux services suivants :

- send pour envoyer une trame ethernet du commutateur sur le port ;
- receive qui permet de recevoir une trame en entrée sur le port.

Pour chaque trame reçue sur un port, le programme doit afficher une ligne de texte de la forme :

- **Reception** *MAC source* **:** *MAC destination* **on Port :** *numéro du port*

Pour chaque émission d'une trame sur un port, le programme doit afficher une ligne

- **Send out** MAC source **:** MAC destination **on Port :** numéro du port

Question 1 : implémentez une classe ComTable pour gérer une table de commutation avec entrées éphémères

Question 2 : Implémentez les ports d'un commutateur sous forme de threads afin qu'on puisse les adresser d'une manière concurrente.

Question 3 : Programmez la classe switch

Question 4 : Ecrivez le programme de test permettant d'obtenir le résultat suivant

```
1  Reception AABBCCDD0910:AABBCCDD0911 on Port :  1
2  Send out AABBCCDD0910:AABBCCDD0911 on port : 0
3  Send out AABBCCDD0910:AABBCCDD0911 on port : 2
4  Send out AABBCCDD0910:AABBCCDD0911 on port : 3
5
6  Reception AABBCCDD0911:AABBCCDD0910 on Port :  0
7  Send out AABBCCDD0911:AABBCCDD0910 on port : 1
8
9  Reception AABBCCDD0910:AABBCCDD0911 on Port :  1
10 Send out AABBCCDD0910:AABBCCDD0911 on port : 0
```

```
class Switch:
    def __init__(self,nbPort,ttl):
        assert isinstance(nbPort,int) and nbPort >0 and (nbPort
            %2 ==0)
        assert isinstance(ttl,int) and ttl >0
        self.ports=[]
        self.ttl=ttl
        for i in range(nbPort):
            self.ports.append(Port(i,self))
        for i in range(nbPort):
            self.ports[i].start()
        self.comTable= ComTable(self.ttl)
        self.lock = RLock()

    def getPort(self,num):
        assert isinstance(num,int) and num >=0 and num <len(self
            .ports)
        return self.ports[num]


    def getNbPort(self):
        return(len(self.ports))

    def commute(self,trame,inPort):
        assert isinstance(trame,TrameEthernet)
        assert inPort in self.ports

        self.lock.acquire()
        print "\nReception {} on Port :  {}".format(trame,inPort
            )
        macSrc = trame.getSrc()
        macDst = trame.getDst()

        self.comTable.put(macSrc,inPort)
        if macDst in self.comTable.keys():
            if self.comTable.get(macDst)!=inPort:
                self.comTable.get(macDst).send(trame)
        else:
            self.broadcast(trame,inPort)

    def broadcast(self,trame,port):
        assert isinstance(trame,TrameEthernet)
        assert port in self.ports

        for p in self.ports:
            if p!=port:
                p.send(trame)
```