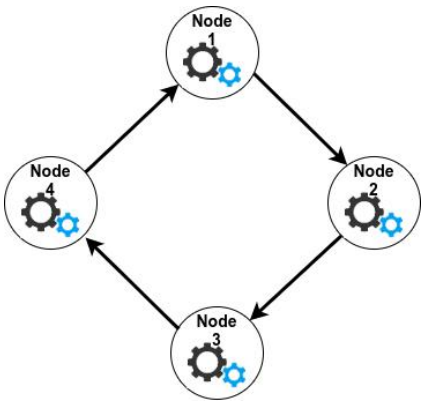
TP N°3 : Topologie Token Ring

Objectif

Dans ce TP, vous allez mettre en place un réseau en anneau (*Ring*) avec jetons (*Token*) en utilisant l'API BSD sockets en mode UDP.



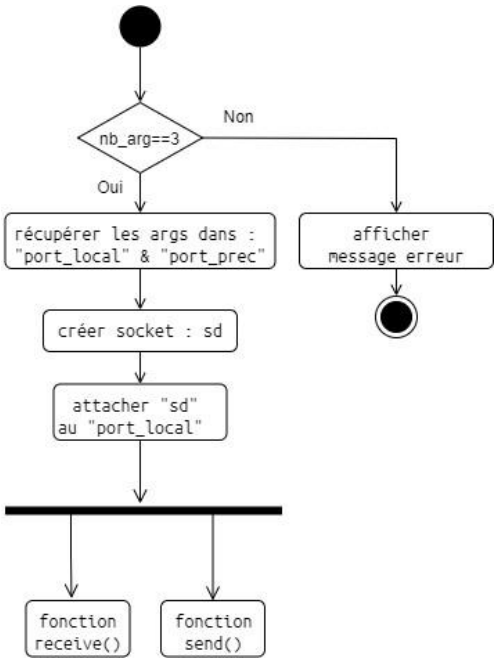
Principe de la topologie

Dans un réseau en anneau à jetons (Token Ring), les nœuds sont connectés de manière unidirectionnelle et séquentielle. Par exemple, si le nœud 1 souhaite envoyer un message au nœud 3, il ne peut pas lui envoyer directement. Il doit d'abord envoyer le message au nœud 2 qui le transmettra ensuite au nœud 3.

Même si le flux est bidirectionnel avec UDP, il sera utilisé, dans ce TP, uniquement dans un seul sens

Exercice N°1:

Le script principal s'appelle «node.py» représentant un nœud avec socket. À chaque fois qu'on exécute «node.py», dans un terminal, cela représente le lancement d'un nœud et le lier à un autre nœud précédemment exécuté.



Le ou les numéros de port devra (devront) être dynamique(s):
Lancer le programme comme suivant : **`>python node.py port_local port_prec`**

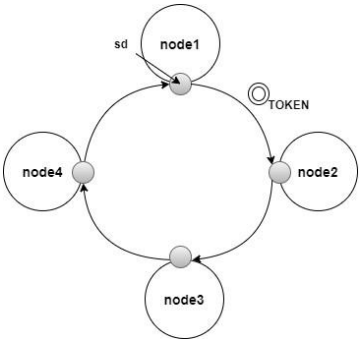
Fonction send() : donner la main à l'utilisateur pour décider du contenu de message et vers quelle destination.

Fonction receive() : l'instruction recvfrom s'exécute à l'infini. À chaque réception, la fonction affiche le contenu du message et l'adresse de destination.

1) Implémenter le script «node.py»

Exercice N°2: Ajouter Token

Dans le principe de la topologie token-ring, le nœud n'a pas le droit d'envoyer quoique se soit, s'il ne détient pas le jeton.



1) Ajouter une variable «token_held» booléenne qui indique si le nœud détient le jeton.
2) Optimiser le code pour que lorsqu'on exécute la commande suivante : **`>python node.py port_local port_prec`**, on aura l'option d'ajouter un 4ème argument "jeton" qui fera comprendre au programme que ce nœud possède le jeton. Si on ne met pas cet argument alors par défaut le nœud ne possède pas le jeton.
3) Ajouter les instructions nécessaires pour qu'au niveau de la réception, il faut distingué entre recevoir n'importe quel message ou recevoir un jeton.
    a) *Exécuter le script.*
    b) *Qu'est ce que vous remarquez?*
    c) *Comment vous expliquez cette situation ?*

4) Remplacer la fonction «send» par une fonction «menu_send». Elle permet de demander à l'utilisateur à chaque fois s'il préfère envoyer un message (à n'importe quel numéro de port) ou le jeton au successeur (libérer jeton). Soit:
    a) *Exécuter le script.*
    b) *Qu'est ce que vous remarquez?*
    c) *Comment vous expliquez cette situation ?*

5) Si tout le programme fonctionne correctement, exécuter le choix N°3.
    Question : Qu'est ce que vous remarquez?

Menu:
1. Envoyer un message
2. Libérer le jeton
3. Quitter

Exercice N°3: Gestion de fichier texte

L'objectif de cette exercice est d'optimiser ce code, pour qu'on puisse lancer la commande suivante : **`>python node.py port_local jeton(option)`**. Au début le programme récupère le port_précédent depuis un fichier text (le dernier enregistré) et enregistre le «port_local» dans le même fichier texte.

Aussi récupérer depuis le fichier texte le numéro de port suivant pour l'utiliser dans le processus de libération du jeton.

1) Ouvrir un fichier:

La meilleure façon de faire est d'utiliser l'instruction **with**. Cela garantit que le fichier est fermé lors de la sortie du bloc à l'intérieur de **with**.

```
with open("etudiants.txt", "a", encoding = 'utf-8') as f: # f: descripteur
   # traitments sur
   # le fichier f
```

Mode	*Description*
'r'	Ouvrir un fichier en lecture. (par défaut)
'w'	Ouvrir un fichier pour l'écriture. Dans ce mode, si le fichier spécifié n'existe pas, il sera créé. Si le fichier existe, alors ses données sont détruites.
'x'	Ouvrir un fichier pour une création exclusive. Si le fichier existe déjà, l'opération échoue.
'a'	Ouvrir un fichier en mode ajout. Si le fichier n'existe pas, ce mode le créera. Si le fichier existe déjà, les nouvelles données seront ajoutées à la fin du fichier.
'+'	Ouvrir un fichier pour la mise à jour (lecture et écriture)

Il ne faut pas compter sur le codage par défaut, sans quoi le code se comportera différemment selon les plateformes. Sous Windows, il s'agit de "cp1252" mais de "utf-8" sous Linux.

2) Écrire dans un fichier

```
with open("etudiants.txt", "a", encoding = 'utf-8') as f:
   f.write("Mostafa \n ")
   f.write("Ismail \n ")
   f.write("Dounia \n ")
   # autres instruction
```

3) Lire un fichier

Pour lire un fichier, il faut l'ouvrir au moins en mode "r". En plus de cela, il faut également vous assurer que le fichier existe déjà car, en mode "r", la fonction «open()» génère une erreur «*FileNotFoundError*» si elle ne parvient pas à trouver un fichier.

Pour tester si un fichier existe ou non, nous pouvons utiliser la fonction «isfile()» du module «os.path».

```
isfile(path)
```

La méthode «read(size)» permet de lire un nombre défini(size) de données . Si le paramètre size n'est pas spécifié, il lit et retourne jusqu'à la fin du fichier.

```
with open("etudiants.txt", "r", encoding='utf-8') as f:
   print("4 premières données : ", f.read(4))
   print("4 données suivantes : ", f.read(4))
   print("le reste du fichier : ", f.read())
   print("lecture supplémentaire ", f.read())
```

La méthode «read()» renvoie newline sous la forme '\ n'.

Il est possible de changer le curseur de fichier actuel (position) en utilisant la méthode «seek()». De même, la méthode «tell()» renvoie la position actuelle (en nombre d'octets).

```
with open("etudiants.txt", "r", encoding='utf-8') as f:
   print("4 premières données ", f.read(4))
   print("position actuelle : ", f.tell())
   print("4 données suivantes : ", f.read(4))
   f.seek(0)  # position le curseur au début
   print("4 données : ", f.read(4))
```

Lire un fichier ligne par ligne :

```
with open("etudiants.txt", "r", encoding='utf-8') as f:
   for ligne in f:
       print(ligne)
```

La méthode «strip()» enlève le "\n" ou un espace vide

```
line.strip()
```

La méthode «readlines()» représente une liste de toutes les lignes du fichier:

```
with open("etudiants.txt", "r", encoding='utf-8') as f:
    lines = f.readlines()
    for i, line in enumerate(lines):
       print(i,line)
```

La fonction «enumerate()» renvoie un objet itérable qui produit des paires (indice, élément)

Question : Trouver-vous que la topologie proposée et réalisée dans ce TP est totalement décentralisée?