République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie d'Oran - Mohamed Boudiaf

Faculté de Mathématique et d'Informatique

Département d'Informatique

Polycopié de Travaux Pratiques (TP) :

Algorithme et Système Réparti

Asmaa BOUGHRARA

2023-2024

Avant propos

Connaissances préalables (pour le TP)

Réseaux, Programmation Orienté Objet et Système d'exploitation 1 & 2.

Objectifs Pédagogiques

La conception du programme de ce TP repose sur plusieurs objectifs et considérations.

1. ***Maîtrise des concepts de la programmation répartie :*** Le choix de privilégier la programmation par rapport à l'utilisation d'un simulateur vise à permettre aux étudiants de développer une compréhension approfondie des concepts de la programmation répartie. Cela leur offre l'opportunité de mettre en pratique les principes théoriques liés à ce domaine.
2. ***Adaptation à un public varié :*** Étant donné que la matière est destinée à des étudiants de plusieurs spécialités et qu'il s'agit d'un semestre de tronc commun, il a été délibérément décidé de ne pas approfondir le concept de réseau. Ainsi, la conception du programme vise à offrir une approche équilibrée, tout en offrant des connaissances pertinentes à chacun.
3. ***Équilibre entre la théorie et la pratique :*** Le choix de ne pas utiliser un simulateur dans le cadre de ce programme vise à éviter que les étudiants se focalisent excessivement sur les algorithmes répartis, étant donné qu'il s'agit d'un tronc commun. Cela permet de maintenir un équilibre entre la théorie et la pratique, tout en offrant une introduction adéquate aux concepts de la programmation répartie.

Intitulé du Master : Réseaux et systèmes Distribués

Semestre *: 1*

Intitulé de l'UE : UEM1.1

Intitulé de la matière : Algorithmique et Systèmes Répartis

Crédits : 3

Coefficients : 2

Objectifs de l'enseignement : L'une des tendances majeures des systèmes informatiques est la répartition des traitements entre des "entités" coopératives. Celles-ci peuvent être soit des processeurs d'une machine multi-processeurs, soit des stations de travail d'un réseau local, soit des serveurs d'application connectés par l'Internet. L'objectif de ce cours est donc de décrire les principaux services nécessaires à la conception d'applications réparties.

Connaissances préalables recommandées

Connaissances acquises durant le cursus de formation de la licence : Systèmes informatiques (SI) ou Ingénierie des Systèmes d'Information et du Logiciel (ISIL)

Contenu de la matière

1- Généralités
   1- modèles de répartition,
   2- Scénario, Evaluation et vérification d'algorithmes répartis
2- La communication
   1- Contrôle de flux : le modèle producteur / consommateur et producteurs multiples
   2- Communication synchrone avec RdV
   3- Qualité de service : réseau Fifo
3- Le temps
   1- Temps horloge, temps environnement
   2- Environnement synchrone
   3- Temps physique
   4- Horloges physiques
4- Les algorithmes de concurrence
   1- Algorithme de Lamport
   2- Algorithme de Ricart et Agrawala
   3- Algorithme de Carvalho et Roucairol
5- L'observation
   1- Etat d'une application répartie
   2- Détection des propriétés d'une application stables et paisibles
6- Les algorithmes d'élection
   1- Algorithmes de Chang et Roberts
   2- Algorithme de Franklin
7- La mémoire virtuelle répartie et linéarisabilité
   1- Linéarisabilité par exclusion mutuelle
   2- Linéarisabilité avec gestionnaire statique
   3- Linéarisabilité avec gestionnaire dynamique
   4- Propagation des écritures
8- L'autostabilisation
   1- Routage auto-stabilisant
   2- Gestion de la mémoire virtuelle répartie
   3- Exclusion mutuelle

Mode d'évaluation : *Contrôle continu, examen,*

Références

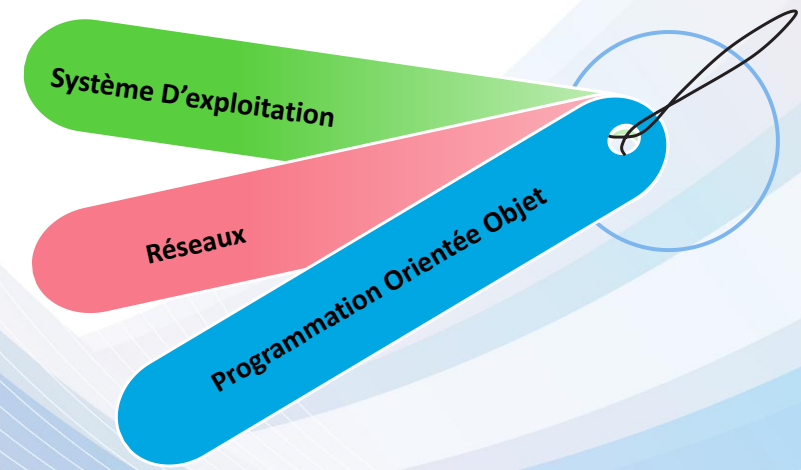
1- M. Raynal "Gestion de données réparties : problèmes et protocoles" Collection Direction des Etudes et des Recherches d'EDF n°82. Hermès. 1992 ISSN 0399-4198
2- A. Tanenbaum "Systèmes d'exploitation. Systèmes centralisés. Systèmes distribués" Troisième Edition Dunod – Prentice Hall. ISBN 2-10-004554-7. 1994
3- G. Tel "Introduction to Distributed Algorithms" Second Edition Cambridge University Press. ISBN 0-521-79483-8. 2000
4- B. Awerbuch "Complexity of network synchronization" JACM 32 (1985), 804-823.
5- L. Lamport "Time, clocks, and the ordering of events in a distributed system" Communications of the ACM 21 (1978) pp 558-564.

Université des **S**ciences et de la **T**echnologie d'**O**ran - **M**ohamed **B**oudiaf
Faculté des **M**athématiques et **I**nformatique
Département d'Informatique
1ème année Master Informatique (S1)

A.S.R : *TP*

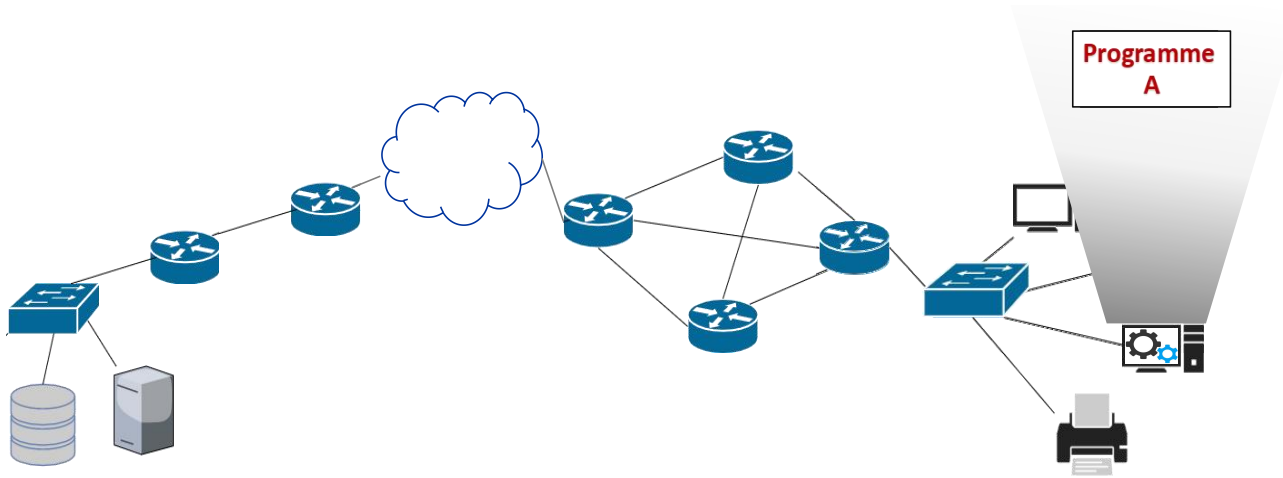
Boughrara Asmaa

Prérequis



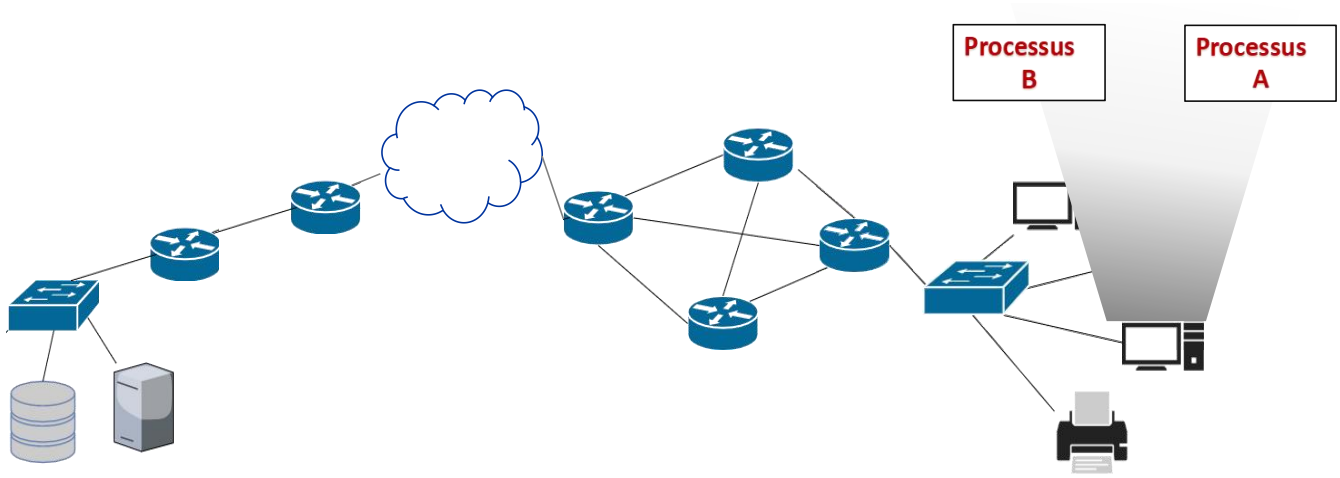
Introduction

Introduction

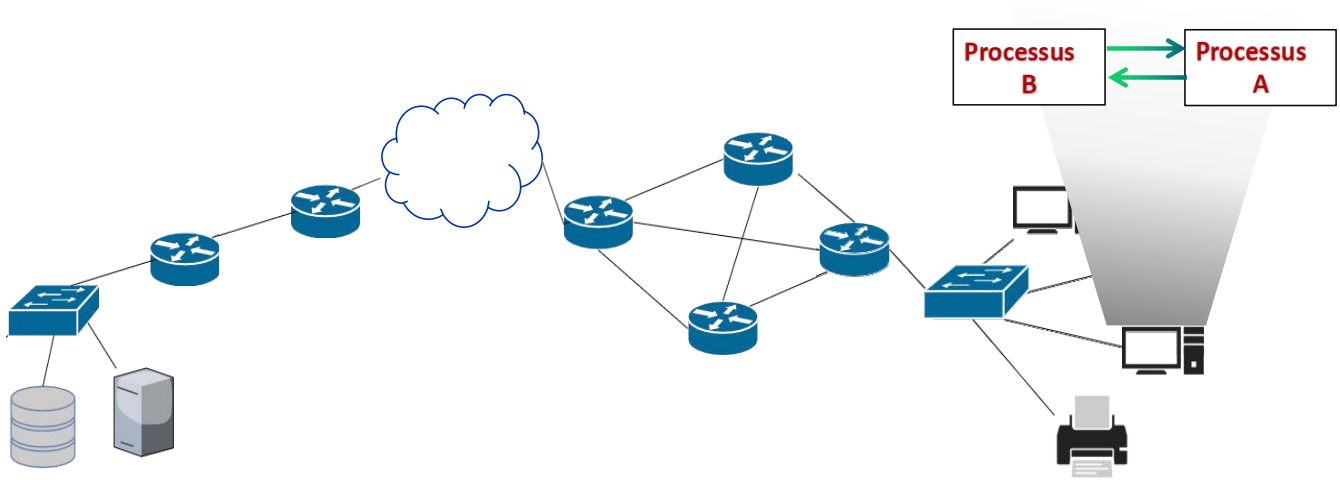


Introduction

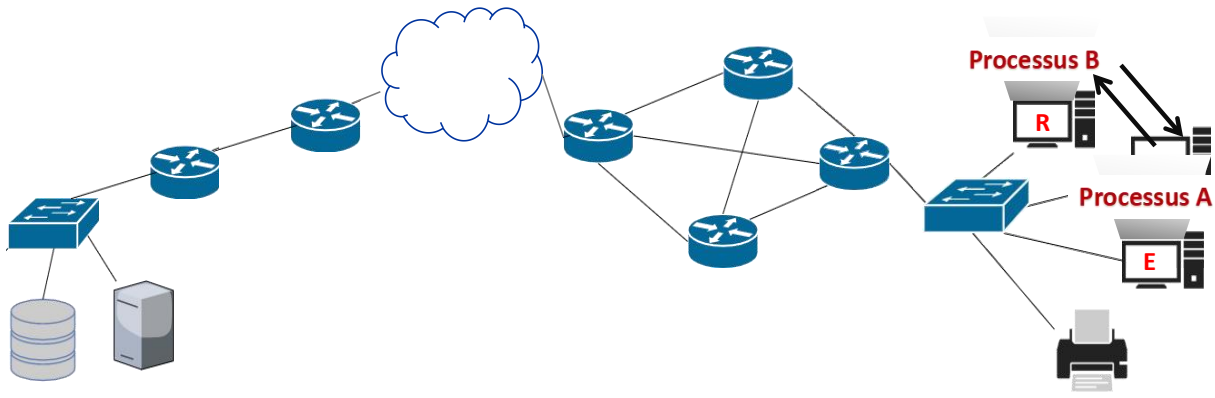
Multi-process -> Système d'exploitation 1 & 2



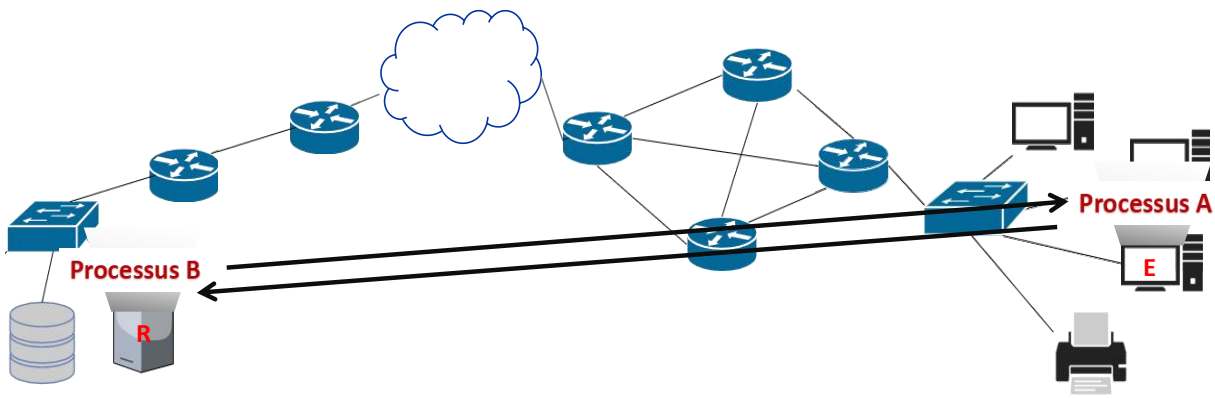
Communication



Communication



Communication



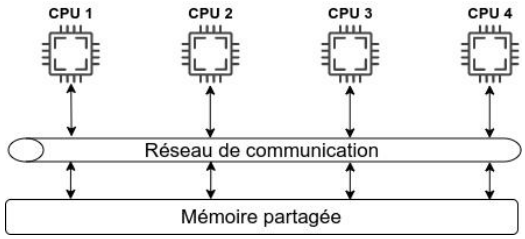
Système réparti

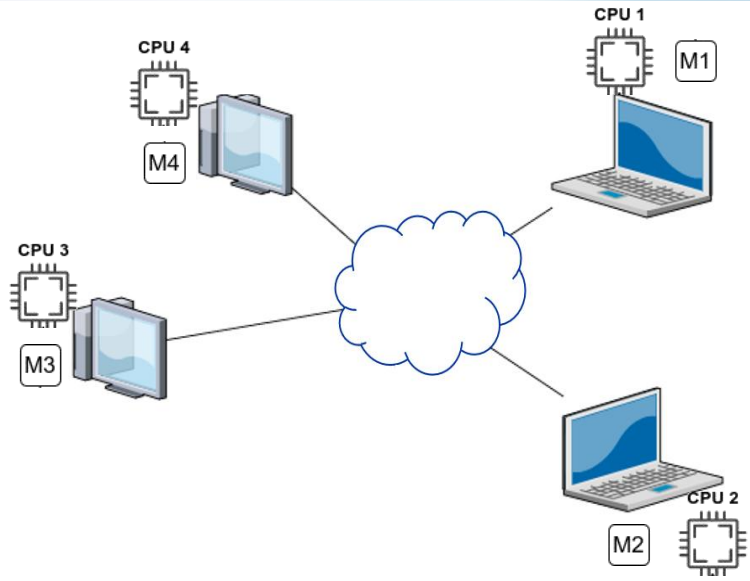
Problèmes liés à la gestion

Mémoire

1) Mémoire répartie (non partagée)

*2) Mémoire partagée : **Problème de la section critique***

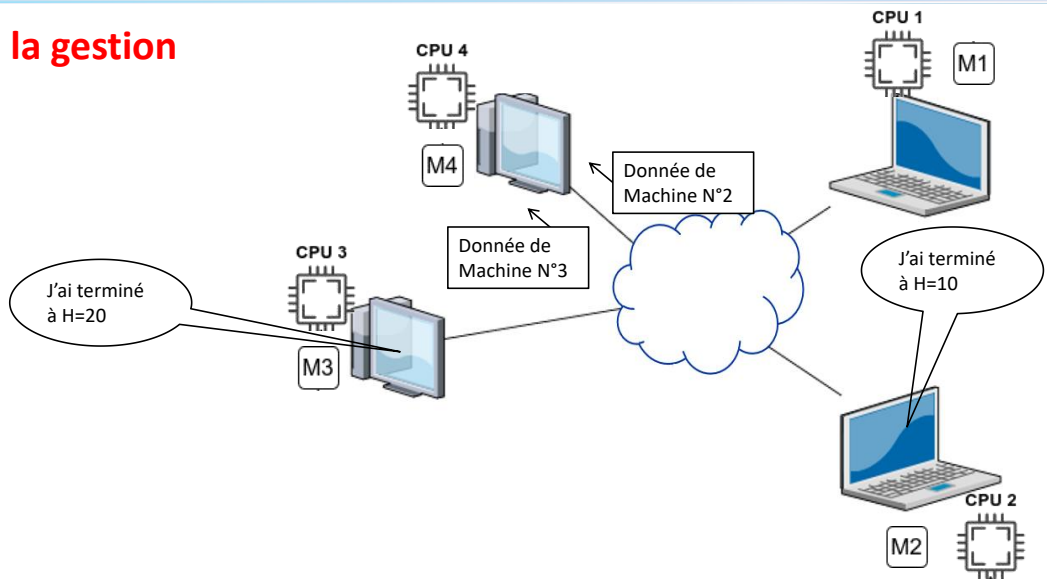




Système réparti

Problèmes liés à la gestion

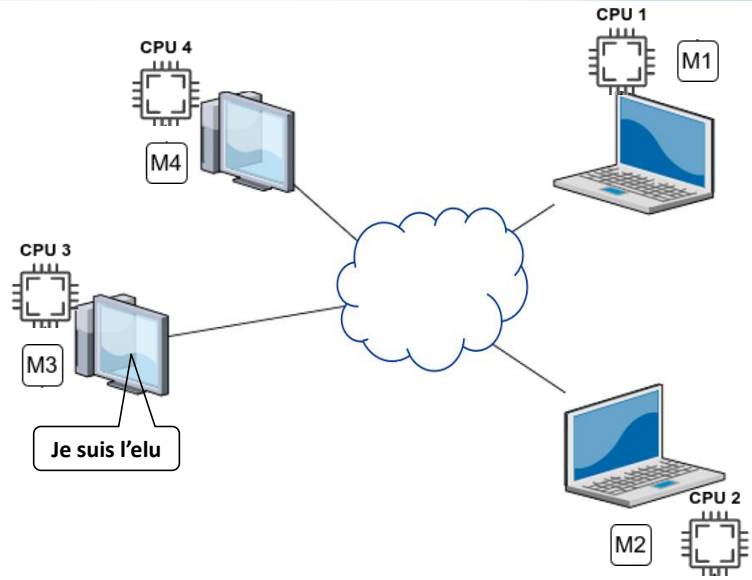
Mémoire **Temps**



Système réparti

Problèmes liés à la gestion

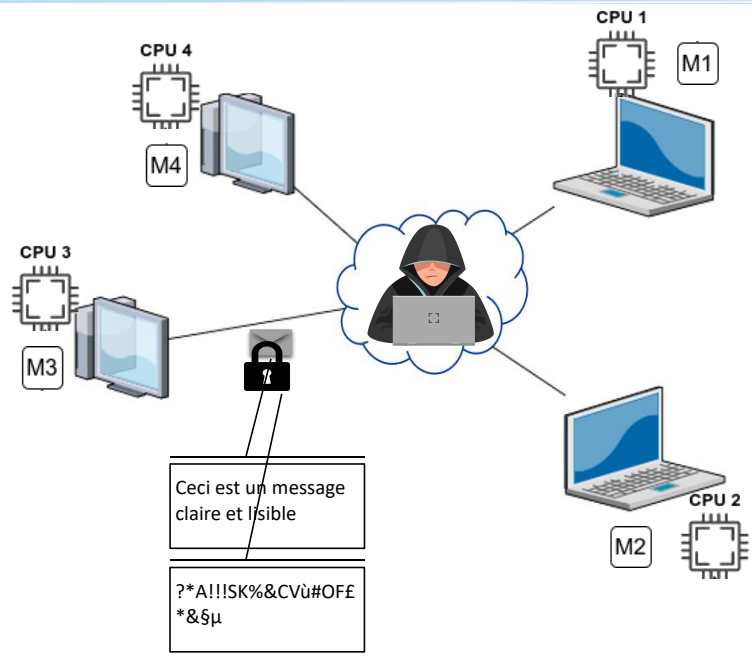
- Mémoire
- Temps
- Élection



Système réparti

Problèmes liés à la gestion

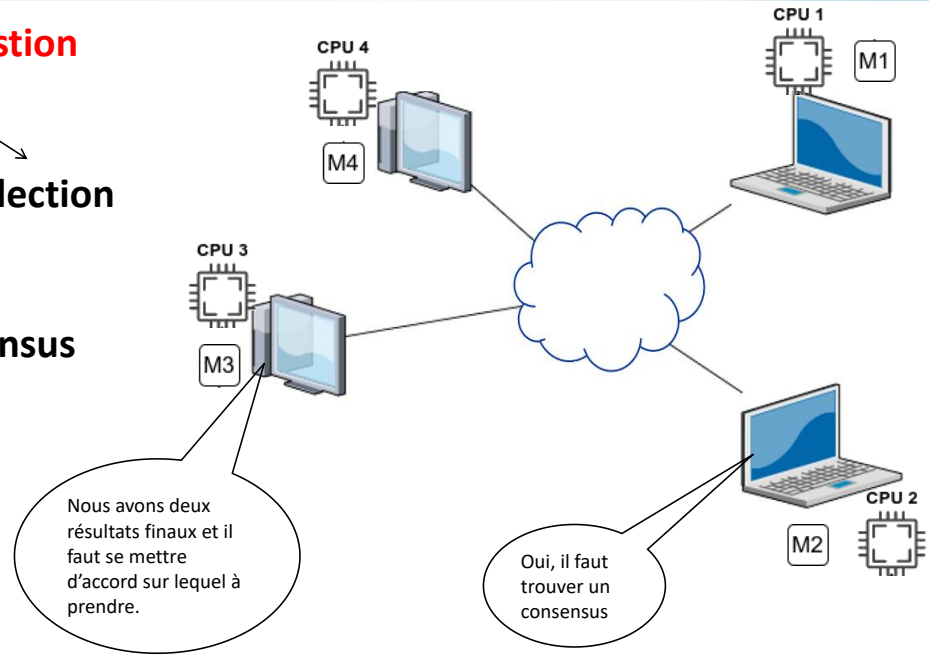
- Mémoire
- Temps
- Élection
- Sécurité



Système réparti

Problèmes liés à la gestion

- Mémoire
- Temps
- Élection
- Sécurité
- Consensus



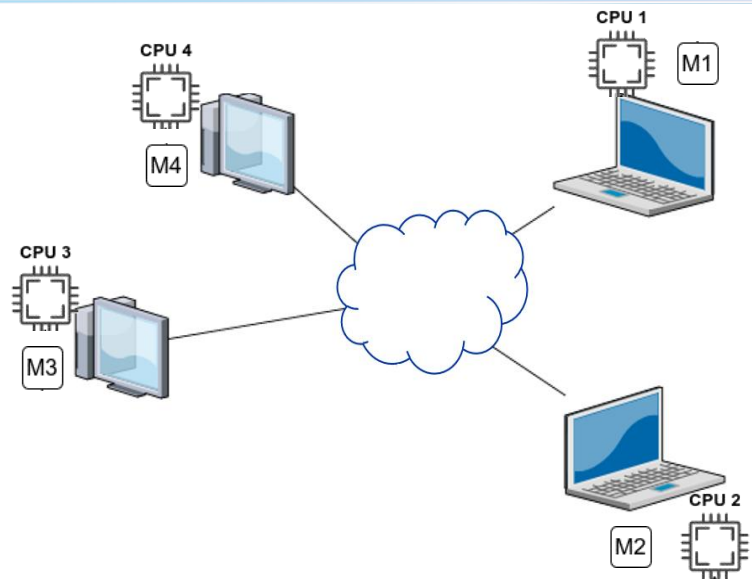
Système réparti

Problèmes liés à la gestion

Solutions

ASR

Algorithmes répartis



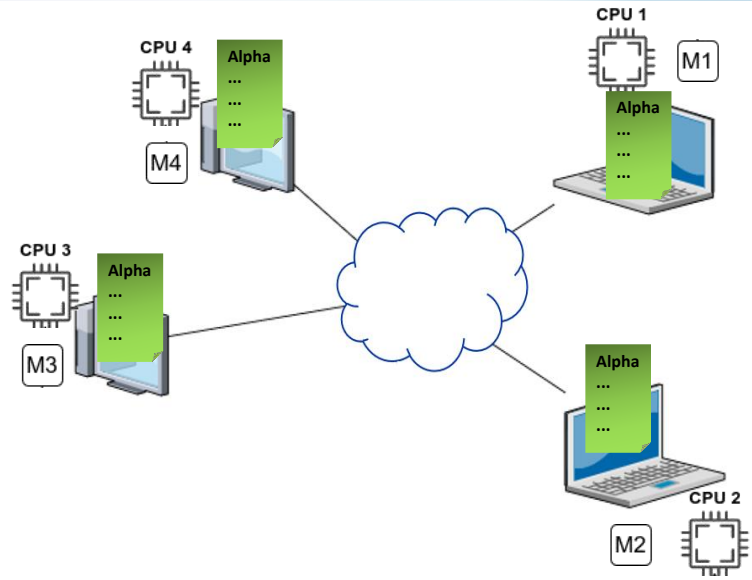
Système réparti

Problèmes liés à la gestion

Solutions

ASR

Algorithmes répartis



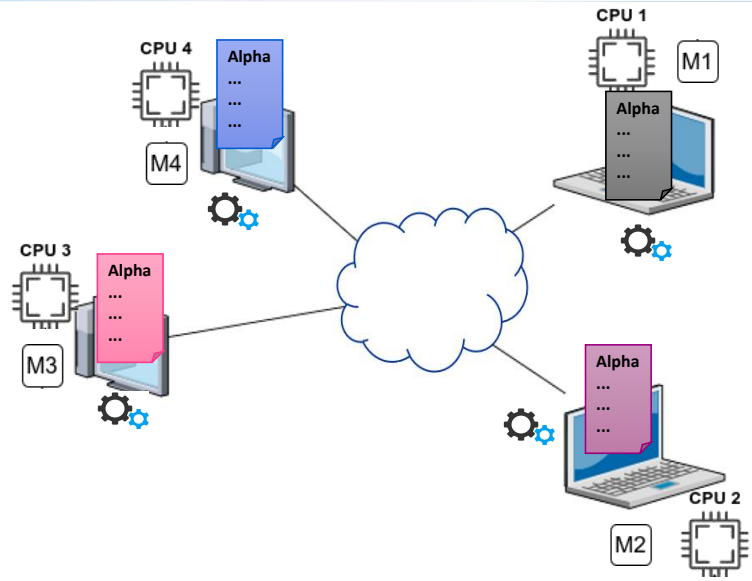
Système réparti

Problèmes liés à la gestion

Solutions

ASR

Algorithmes répartis



Asmaa Boughrara USTO - MI - M1

ASR

Intitulé de l'UE : UM311
Intitulé de la matière : Algorithmique et Systèmes Répartis
Crédits : 3
Coefficients : 2

Objectifs de l'enseignement : L'une des tendances majeures des systèmes informatiques est la répartition des traitements entre des "entités" coopératives. Celles-ci peuvent être soit des processeurs d'une machine multi-processeurs, soit des stations de travail d'un réseau local, soit des serveurs d'application connectés par l'Internet. L'objectif de ce cours est donc de décrire les principaux services nécessaires à la conception d'applications réparties.

Connaissances préalables recommandées
Connaissances acquises durant le cursus de formation de la licence : Systèmes informatiques (SI) ou Ingénierie des Systèmes d'Information et du Logiciel (ISIL)

Contenu de la matière

1- Généralités
   1- modèles de répartition,
   2- Scénario, Evaluation et vérification d'algorithmes répartis
2- La communication
   1- Contrôle de flux
   2- Communication synchrone avec RdV
   3- Qualité de service : réseau Fifo
3- Le temps
   1- Temps horloge, temps environnement
   2- Environnement synchrone
   3- Temps physique
   4- Horloges physiques
4- Les algorithmes de concurrence
   1- Algorithme de Lamport
   2- Algorithme de Ricart et Agrawala
   3- Algorithme de Carvalho et Roucairol
5- L'observation
   1- Etat d'une application répartie
   2- Détection des propriétés d'une application stables et paisibles
6- Les algorithmes d'élection
   1- Algorithmes de Chang et Roberts
   2- Algorithme de Franklin
7- La mémoire virtuelle répartie et linéarisabilité
   1- Linéarisabilité par exclusion mutuelle
   2- Linéarisabilité avec gestionnaire statique
   3- Linéarisabilité avec gestionnaire dynamique
   4- Propagation des écritures
8- L'autostabilisation
   1- Routage auto-stabilisant
   2- Gestion de la mémoire virtuelle répartie
   3- Exclusion mutuelle

	Cours	TP
Communication répartie	Différents types	Comment Implémenter
Mémoire	Comprendre les différents algorithmes	Comment Implémenter
Temps	Comprendre les différents algorithmes	Comment Implémenter
Élection	Comprendre les différents algorithmes	Comment Implémenter
…….		

TP ASR

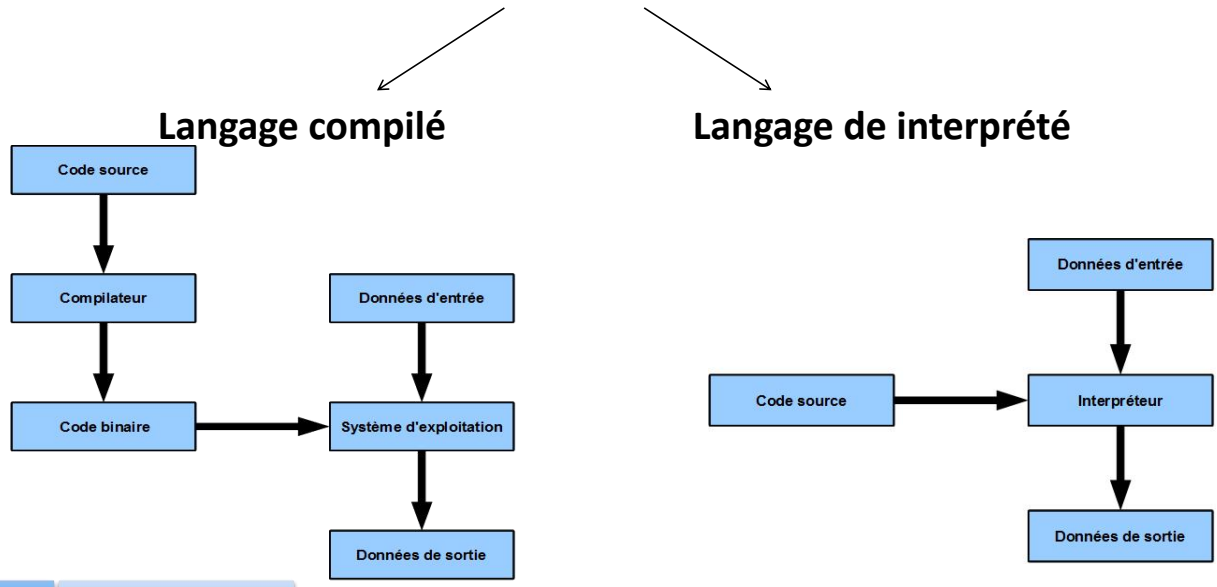
L'objectif de ce TP:

Savoir comment implémenter ses propres architectures, topologies et algoritmes répartis

	Cours	TP
Communication répartie	Différents types	Comment Implémenter
Mémoire	Comprendre les différents algorithmes	Comment Implémenter
Temps	Comprendre les différents algorithmes	Comment Implémenter
Élection	Comprendre les différents algorithmes	Comment Implémenter
…….		

TP ASR : Langage de programmation

Paradigme de mise en oeuvre



TP ASR : Langage de programmation

Paradigme de mise en oeuvre

Langage compilé **Langage de interprété**

Java, C, C++ PHP, Python ;

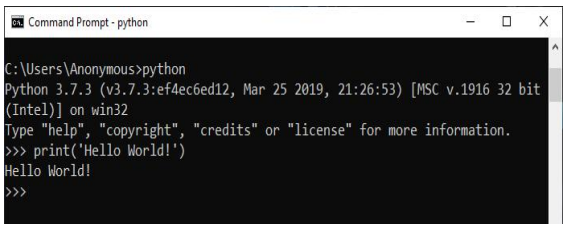
Pour connaître la version de python: depuis l'éditeur de commande taper la commande:
Sous Windows : **>python --version**
Sous Linux : **$ python3 --version**
Car on trouve deux versions pré-installée : version 2 et 3. Ainsi, sous Linux il faut toujours saisir la commande **python3** si on souhaite travailler avec la version 3.

Python 3.....

TP ASR : Python

Travailler avec Python

- **Mode interactif**
- **IDE**
- **Mode Script**

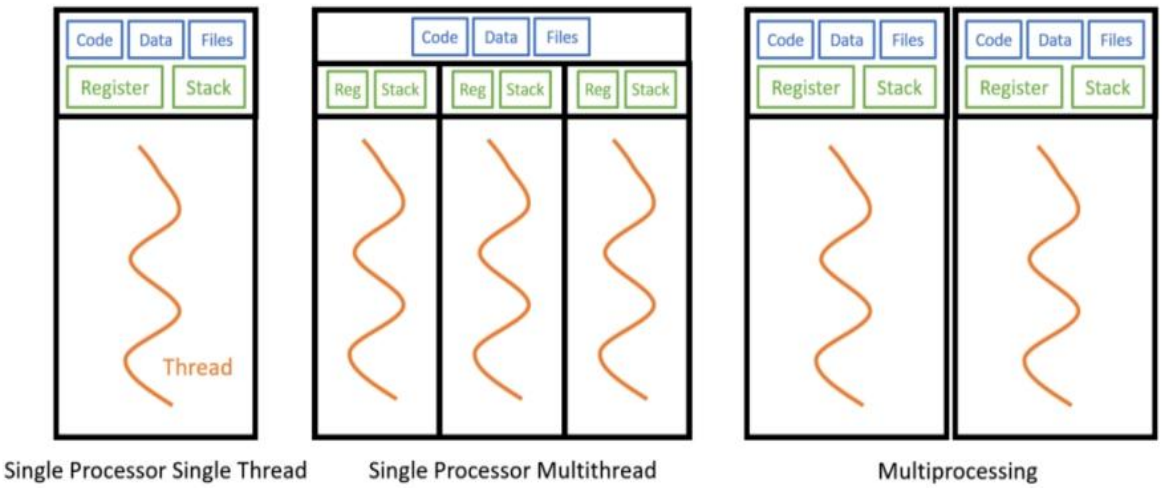


Programmation multitâche

Programmation multitâche

Le multitâche est la condition sine qua non de l'informatique distribuée.

Deux concepts : Le Multithreading et Le Multiprocessing (Multitraitement) qu'il faut distinguer.



Programmation multitâche

Multithreading	Multiprocessing
un processeur exécute plusieurs threads simultanément.	un système exécuterplusieurs processeurs simultanément, chaque processeur pouvant exécuter un ou plusieurs threads.
utile pour les processus liés aux E/S, tels que la lecture de fichiers à partir d'un réseau ou d'une base de données	utile pour calcul, car il bénéficiera de la présence de plusieurs processeurs

TP N°0 : Introduction

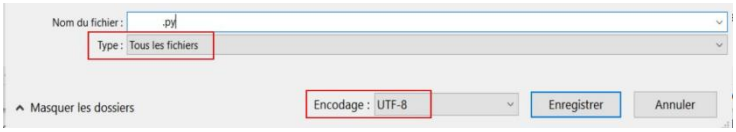
Objectif

S'initier à la programmation orientée-objet en python et le multithreading

Travailler avec Python

Le script Python est un fichier contenant du code écrit en Python avec extension **.py** ou **.pyw** (sous Windows).

Si vous travaillez avec l'éditeur «BlocNote», assurez-vous que le type soit «Tous les fichiers» et l'Encodage soit «UTF-8»



Pour l'exécuter, taper la commande : «python hello.py» (sous Windows) ou «python3 hello.py» (sous Linux).

Règles de nommage

- Les noms de variables, de fonctions et de modules sont en *Snake Case.* Ex : ma_variable, fonction_test_27(), mon_module
- Les constantes[1] sont écrites en *Screaming Snake Case.* Ex: MA_CONSTANTE, VITESSE_LUMIERE
- Les noms de classes sont en *Pascal Case.* Ex: MaClasse, MyException

Attention: Le langage python est sensible à la casse.

Exercice N°1: Programmation Orientée Objet

1) Créer un dossier «TP0».
2) Ouvrir un éditeur de texte et enregistrer le fichier sous le nom «TP_OO.py»
3) Écrire le script suivant, où la classe «Humain» possède deux attributs : «nom» et «age», qui seront initialisés lors de l'instanciation:

4) Méthode de constructeur

- La méthode constructeur doit obligatoirement être nommée **__init__** (deux caractères « souligné », le mot **init**, puis encore deux caractères « souligné »). Il doit accepter au moins un paramètre, dont le nom doit être **self**, et qui doit être le premier paramètre.

- Utiliser la touche tabulation pour l'identation

```
class Humain:
    def __init__(self,a,b=18):
        self.nom=a
        self.age=b
    def afficher(self):
        print("Nom= ",self.nom,", age= ",self.age,"ans")

personne1=Humain("Leila",20) #instancier la class
personne1.afficher()

personne2=Humain("Ali")
personne2.afficher()
```

5) Exécuter le script

[1] Python n'a pas de véritables constantes immuables, le Screaming Snake Case est utilisé par convention pour les variables que l'on considère comme des constantes, afin d'indiquer qu'elles ne doivent pas être modifiées après leur initialisation.

6) Héritage

a) Créer une nouvelles classe (appelée «Etudiant») qui hérite de la classe «Humain».

Réponse : la déclarer comme une simple classe tout en mettant le nom de la classe mère entre les parenthèse:

b) "Surcharger" la méthode afficher()

c) Exécuter le script

```
class Etudiant(Humain):
    def __init__(self,x,y,z):
        Humain.__init__(self,x,y)
        self.niveau=z

personne3=Etudiant("Fatima",19,"Licence")
personne3.afficher()
```

Exercice N°2: Multithreading (as a Python Function)

1) Ouvrir un éditeur de texte et enregistrer le fichier sous le nom «TP_threads.py» dans le dossier «TP0»

2) Implémenter deux fonctions : fonctions1 et fonction 2 qui affichent des valeurs jusqu'à 4 et 6 respectivement. En utilisant la boucle for.

3) Lancer l'exécution.

Question: Comment se fait l'affichage des valeurs?

```
def fonction1():
    for i in range(5):
        print("i = ",i)

def fonction2():
    for j in range(7):
        print("j = ",j)
fonction1()
fonction2()
```

4) Ajouter le sleep() après chaque affichage (pour pouvoir visualiser l'affichage)

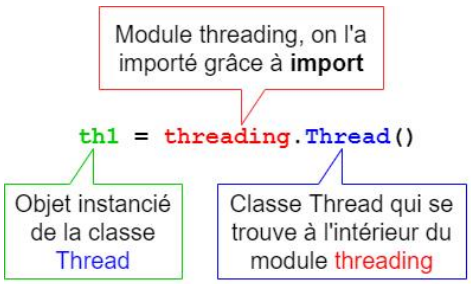
```
import time

time.sleep(0.4)
```

5) Implémenter un Thread

a) Importer le module «threading»

b) Un thread est un objet instancié depuis la classe «Thread» "(le «T» est en majuscule) qui se trouve dans le module «threading» (le «t» est en minuscule)



Remarque:

Il est possible de remplacer les étapes «a» et «b» par:

```
from threading import Thread
Th1=Thread()
```

c) Exécuter les deux fonctions dans des threads

Le constructeur de la classe «Thread» prend plusieurs paramètres. Ce qui nous concerne dans cette étape c'est la cible d'exécution (sur quoi il va se lier). Dans notre cas c'est la fonction «fonction1» et «fonction2». Ainsi, on écrit:

```
th1=threading.Thread(target=fonction1)
th2=threading.Thread(target=fonction2)
```

d) Lancer l'exécution.

Question: Qu'est ce que vous remarquez?

e) Démarrer les threads

```
th1.start()
th2.start()
```

f) Lancer l'exécution.

Questions: Qu'est ce que vous remarquez?

g) Ajoutez une instruction d'affichage "FIN" après les deux instructions de l'étape «e»
Lancer l'exécution.

Question: qu'est ce que vous remarquez après exécution du script?

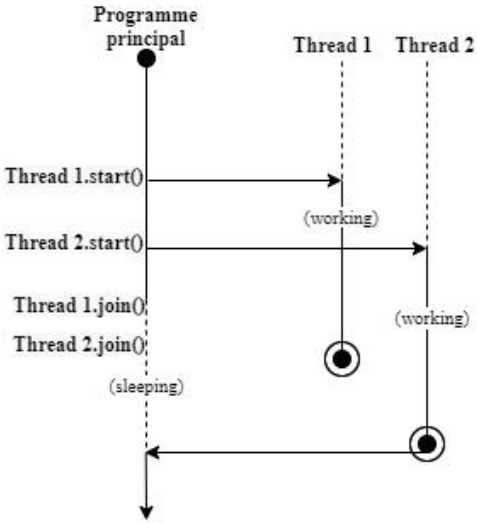
h) Bloquer le programme principal avec la méthode «join()»

Question: Quel effet a la méthode join() sur le programme principal?

Explication

```
th1.join()
th2.join()

print("C'est fini")
```



6) Personnaliser le thread lors de l'appel:

Nous allons maintenant appeler une seule fonction (fonction1) par deux threads différents. Pour faire:
- Le nombre d'itération devient dynamique. La valeur sera passée comme argument de la fonction;
- Chaque thread devra porter un nom pour pouvoir identifier lors de l'exécution.

La syntaxe de classe «Thread» est la suivante:

```
Thread(group=None, target=None, name=None, args=(), kwargs={}, *, daemon=None)
```

7) Afficher les informations concernant le thread:

- `threading.current_thread()` → afficher les informations concernant le thread
- `threading.current_thread().name` → affiche uniquement le nom
- `threading.get_ident()` → affiche uniquement l'identifiant du thread
- `Th1.ident` → attribue de l'objet «th1», représente l'identifiant de thread. Il est égale à None si le thread n'a pas été démarré.

Ainsi:

```
1   import time
2   import threading
3   def fonction1(N):
4       for i in range(N):
5           print("i = ",i, "|",threading.current_thread())
6           time.sleep(0.4)
7
8   th1=threading.Thread(target=fonction1, name="alpha", args=(5,))
9   th2=threading.Thread(target=fonction1, name="beta", args=(7,))
10  th1.start()
11  th2.start()
12  th1.join()
13  th2.join()
14  print("FINI")
```

Exercice N°3: Multithreading (as a Python class)

1) Importer le module threading;
2) Créer une classe nommé «Class1» de type thread qui hérite de la classe mère Thread.
3) Surcharger le constructeur pour qu'il récupère deux arguments : nb et name. Ne pas oublier d'appeler le constructeur threading.Thread.__init__

```
class Class1(threading.Thread):
    def __init__(self,a,b):
        threading.Thread.__init__(self)
        self.nb=a
        self.name=b #attribut par défaut de la class Thread
```

4) Définir la méthode «run()»**:** c'est le code que devra exécuter le thread,

```
    def run(self):
        x=0
        for i in range (self.nb):
            print("x=", x, "|",threading.current_thread().name )
            x=x+1 # semblant d'un traitement
```

5) Créer une instance de la nouvelle classe .

```
th_cl1=Class1(6,"C-Alpha")
th_cl2=Class1(5,"C-beta")
```

Question: En implémentant le thread comme classe, est-il nécessaire de faire appel aux méthodes start et join? Justifiez votre réponse?
Que se passerait-il si je remplace th1.start() par th1.run()? Justifier votre réponse.

Exercice (Advanced)[2]

Définissez une classe **Domino()** qui permet d'instancier deux objets simulant les pièces d'un jeu de dominos. Le constructeur de cette classe initialisera les valeurs des points présents sur les deux faces A et B du domino (valeurs par défaut = 0).

Deux autres méthodes seront définies :
- une méthode **affiche_points()** qui affiche les points présents sur les deux faces;
- une méthode **valeur()** qui renvoie la somme des points présents sur les 2 faces.

Réaliser ce travail pour une exécution séquentielle, par la suite simultanée.

[2] Un exercice dit «advanced» est destiné aux étudiants qui ont terminé la fiche avant la fin de la séance de TP

Université des **S**ciences et de la **T**echnologie d'**O**ran - **M**ohamed **B**oudiaf
Faculté des **M**athématiques et **I**nformatique
Département d'Informatique
1ème année Master Informatique (S1)

A.S.R : *TP*

Boughrara Asmaa

Université des **S**ciences et de la **T**echnologie d'**O**ran - **M**ohamed **B**oudiaf
Faculté des **M**athématiques et **I**nformatique
Département d'Informatique
1ème année Master Informatique (S1)

TP N°1
Communication inter-process

Boughrara Asmaa

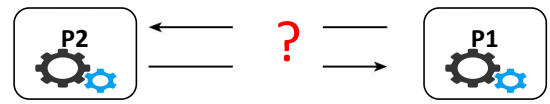
USTO - MI - M1

Communication répartie

Interaction Homme / Machine

- ***Lecture*** : readln (c), cin(c++)
- ***Ècriture*** : println (c), cout(c++)

Communication répartie



Services de communications:

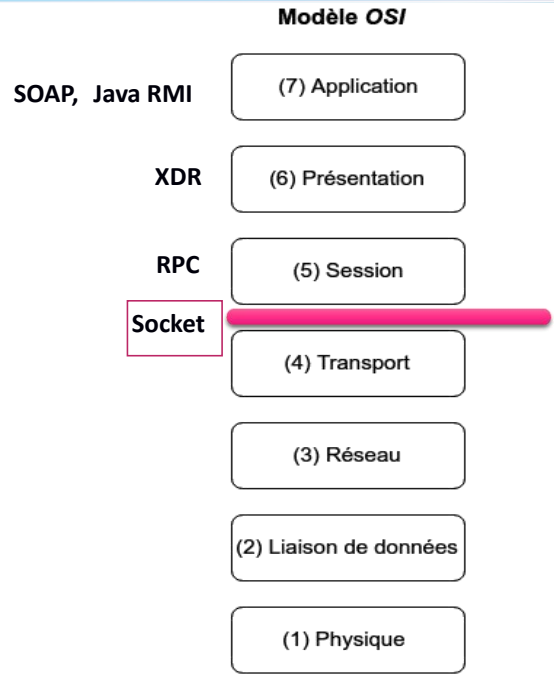
1) Envoi de messages : **Socket;**
2) Appel de procédure : **RPC**, **XDR;**
3) Invocation de méthode : **Java RMI**, **CORBA;**
4) Invocation de service : **SOAP;**

... etc.

Services de communications

Niveaux d'abstraction :

Bas Niveau : **Socket**



Socket ... késako ?

Socket

En 1982;
Chercheurs de l'université Berkeley l'ont proposé pour leur système de type UNIX :
-> Berkeley Software Distribution (abr. BSD)

BSD Socket

Les constructeurs de stations de travail comme HP, SUN, IBM, SGI, ont adopté ces sockets,

Socket

"Prise de courant"
"Prise de réseau" ou «prise de communication».

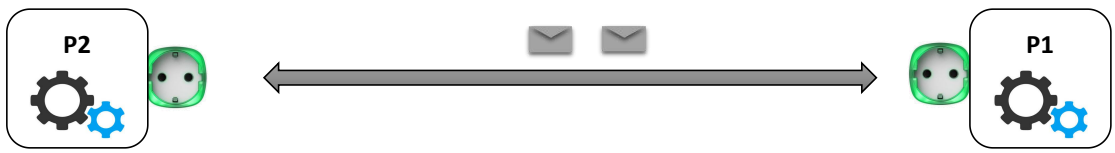
Il s'agit d'un point d'entrée/sortie permettant aux processus de communiquer entre eux (**IPC** - Inter Process Communication) aussi bien sur une même machine qu'à travers un réseau.

Socket

API (*Application Program Interface*)

Un ensemble de primitives qui permettent de gérer l'échange de données entre processus.

Socket



Une communication entre deux processus, suit le format général suivant :

1) Ouverture : *création de socket et le cannal de communication*
2) Échange : e*nvoi et réception*
3) Fermeture

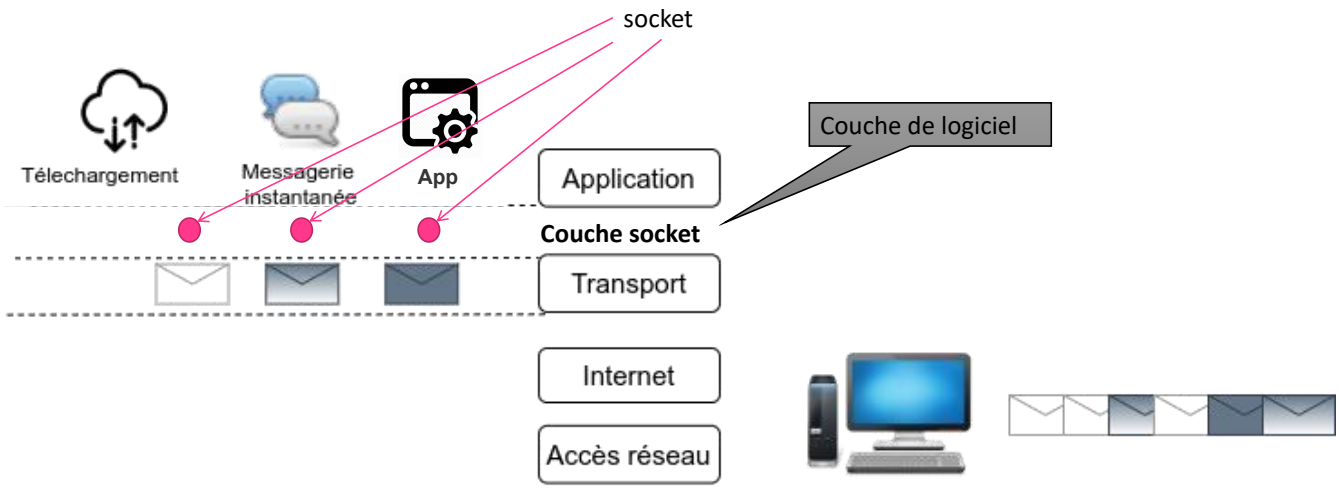
Socket

Paradigme Cient-Serveur

- **Client**: application qui prend l'initiative du lancement de la communication, c'est à dire demande l'ouverture de connexion, l'envoi d'une requête, l'attente de la réponse à la requête, reprise de l'exécution du programme.

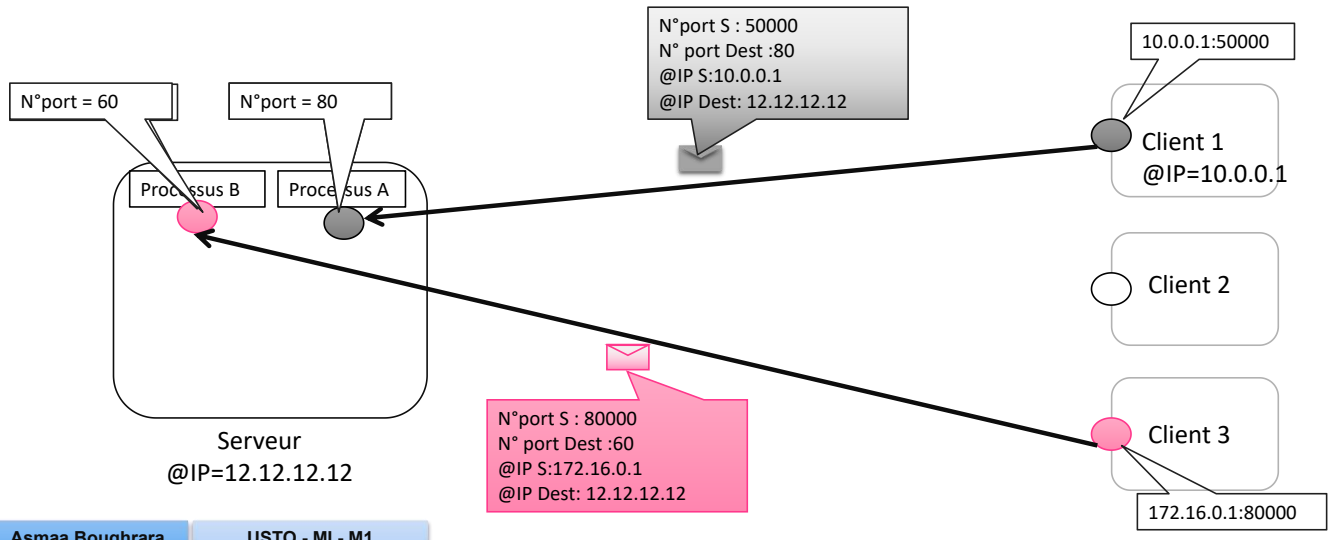
- **Serveur**: application qui attend la communication, c'est à dire l'attente d'une demande d'ouverture de connexion, la reception d'une requête et l'envoi d'une réponse.

Socket Vs TCP/IP



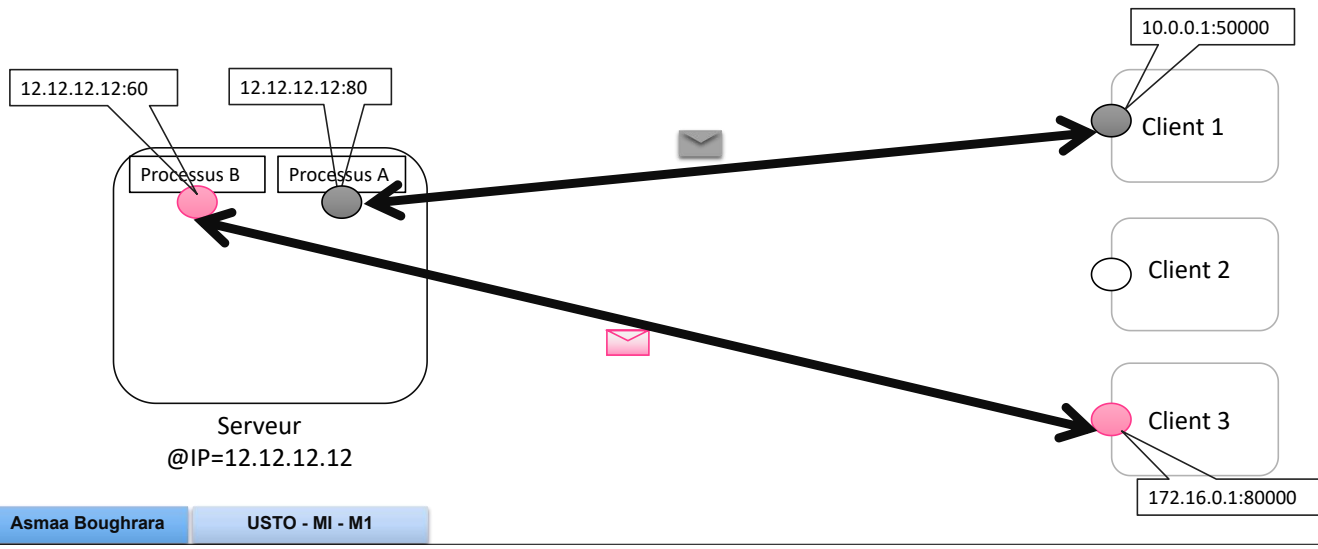
Socket: *autres définitions*

1) Socket pour décrire l'adresse d'un processus TCP ou UDP.
C'est à dire une interface de connexion : **adresse IP + N° port**

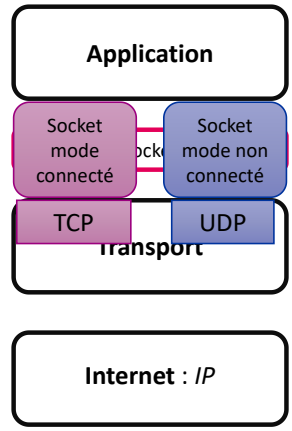


Socket: *autres définitions*

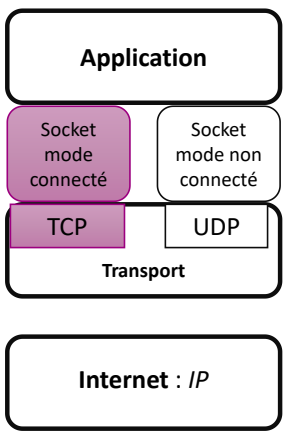
2) Le terme socket désigne aussi un canal de communication par lequel 2 processus peuvent communiquer sur un réseau.

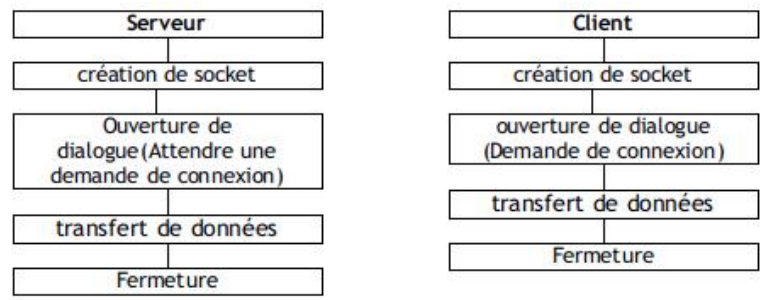


Socket Vs TCP/IP

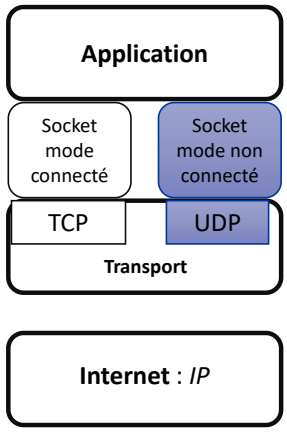


Socket Vs TCP/IP

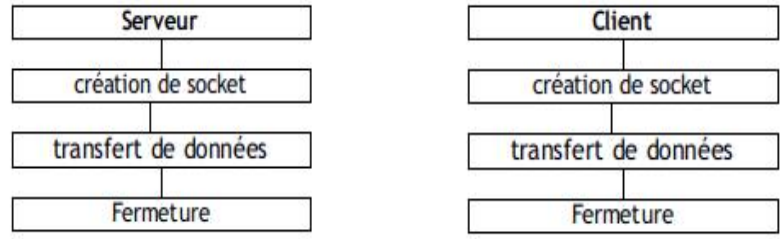




Socket Vs TCP/IP



Aucune liaison n'est établie. Les messages sont échangés individuellement.



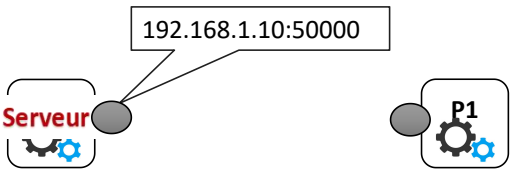
Socket : langage de programmation

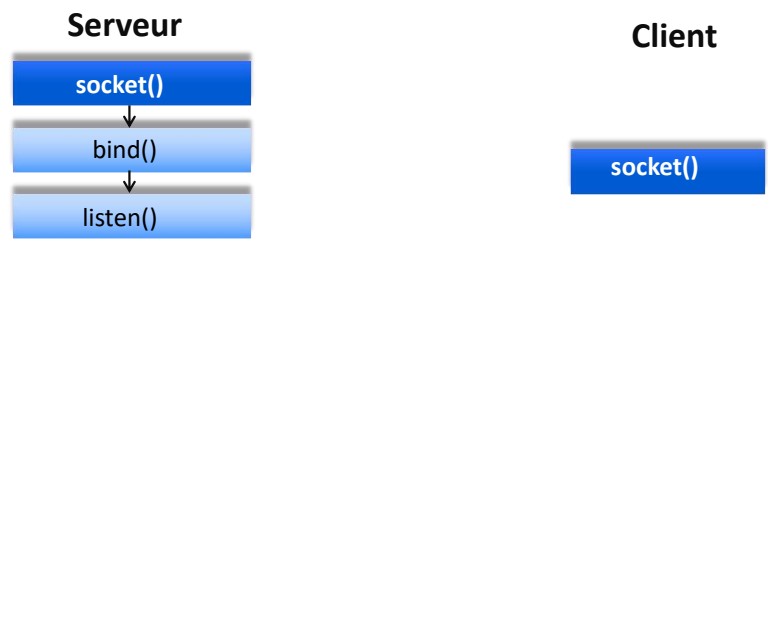
Langage de programmation

- c/c++
- C#
- Java
- JavaScript
- Python
- Php
- Perl
- ...etc.

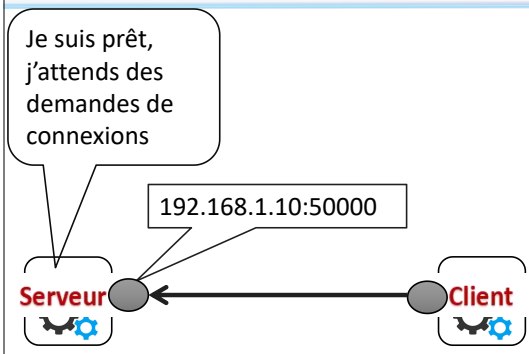
Socket : mode connecté (TCP)

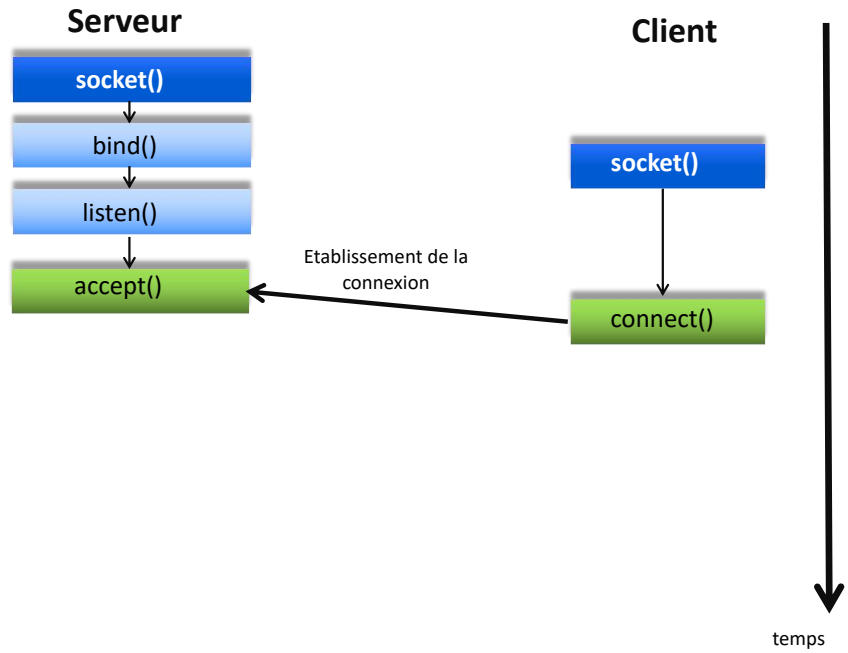
Mode connecté (TCP)



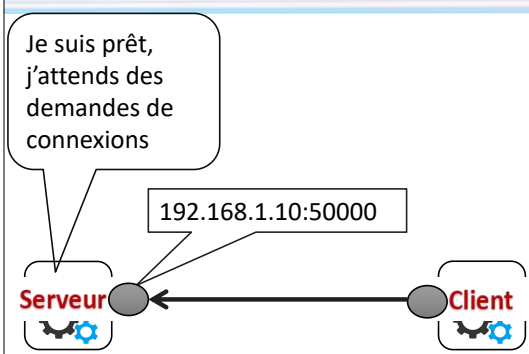


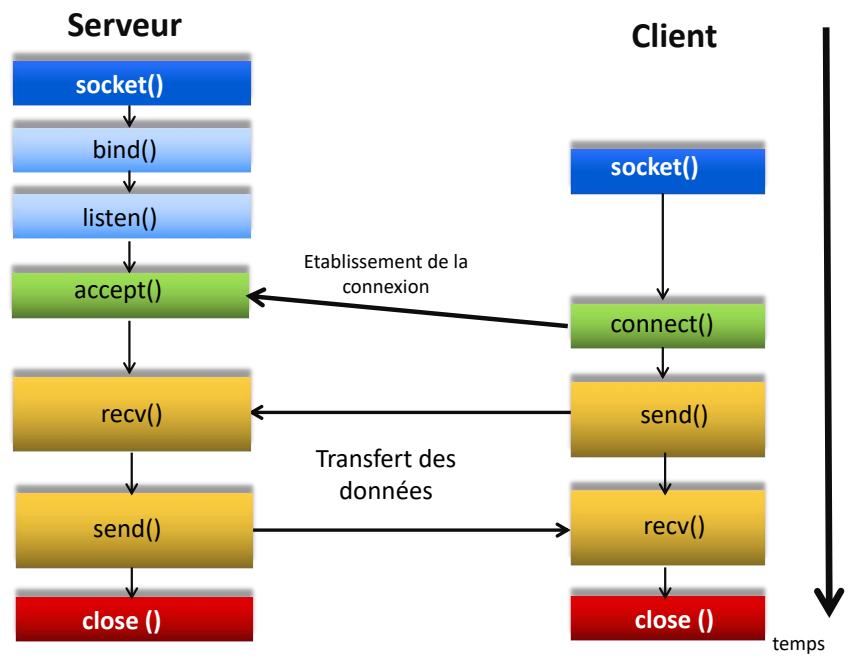
Mode connecté (TCP)



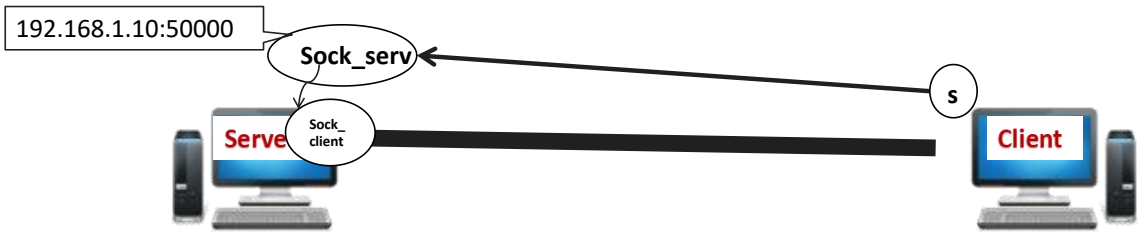


Mode connecté (TCP)





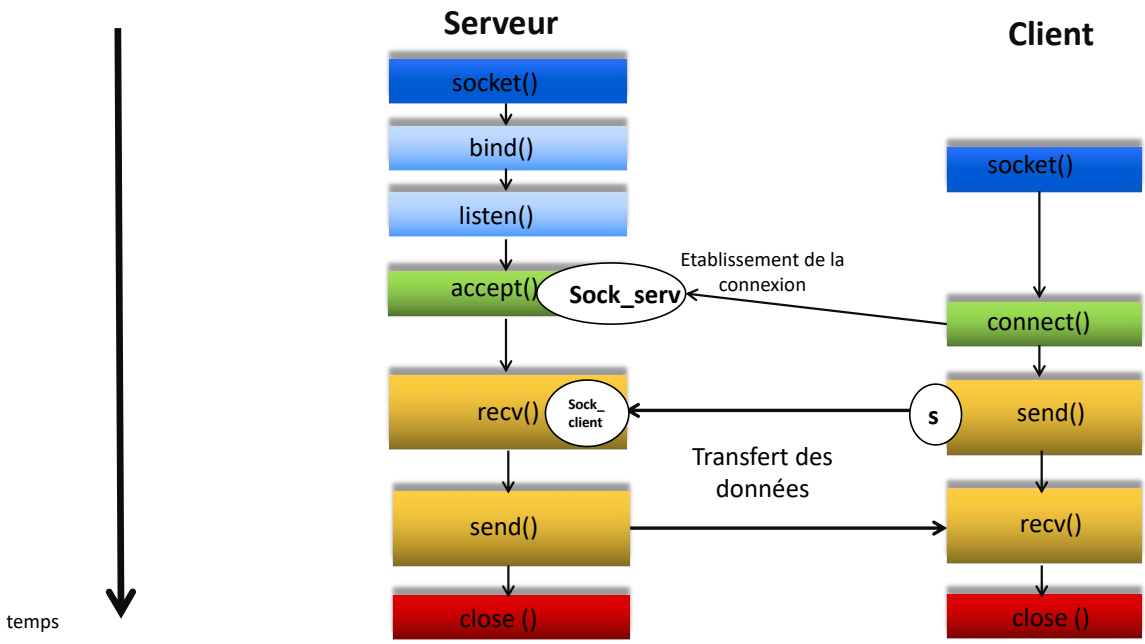
Mode connecté (TCP)



socket()
bind()
listen()
accept() : **bloquante**

socket()
connect()

Mode connecté (TCP)



TP N°1 : Communication inter-process (TCP)

Objectif

Implémenter une communication inter-process en utilisant l'API socket en mode TCP.

Exercice 1:

Dans cet exercice, nous allons implémenter deux scripts, un représentant le client et l'autre le serveur (mode TCP).
On procède en plusieurs étapes, décrites schématiquement ci-après :

Partie 1 : Côté Serveur

1) Créer un dossier «TP1»;
2) Ouvrir un éditeur de texte et enregistrer le fichier sous le nom «serveur.py»**;**
3) Importer le module «socket», qui contient toutes les méthodes et les classes nécessaires;

4) Création de socket :

```
Sock_serv=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

- *AF_INET* : le type d'adresses : AF_INET (pour IPv4) et AF_INET6 (pour IPv6);
- *SOCK_STREAM* : le type de service : flux d'octets (pour TCP), SOCK_DGRAM (pour UDP).

5) Associer le socket à une adresse IP et un numéro de port :

```
Sock_serv.bind(('127.0.0.1',60000))
```

Les valeurs passées à «bind()» est un «tuple[1]» à deux : (hôte, port).

6) Mettre le socket en mode écoute :

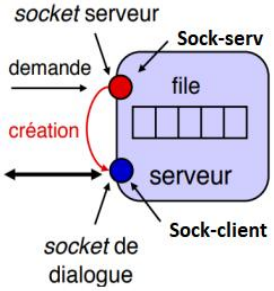
```
Sock_serv.listen()
```

- La fonction listen() ne s'utilise qu'en mode connecté (donc avec le protocole TCP).
- La fonction correspond à une «ouverture passive»: il n'y a pas de création effective d'un canal de communication mais le serveur affiche sa présence pour -sur demande - créer un tel canal.
- L'argument (qui est optionnel) qu'on lui transmet indique le nombre maximum de connexions à accepter en parallèle.

7) Accepter les demandes de connexions:

```
Sock_client, add_client=Sock_serv.accept()
```

- **`Sock_client`** : est la référence d'un nouvel objet de la classe «socket()»;
- **`add_client`** : un tuple contenant l'adresse IP et le n° de port du client.



[1] Les tuples ressemblent aux listes, mais on ne peut pas les modifier une fois qu'ils ont été créés.

8) Réception des données:

```
msgClient=Sock_client.recv(1024)
```

«1024» indique le nombre maximum d'octets à réceptionner en une fois. Si la taille du message reçu dépasse la taille du buffer, les octets surnuméraires sont mis en attente dans un tampon. Ils sont transmis lorsque la même méthode «recv()» est appelée à nouveau.

9) Émettre des données:

```
Sock_client.send(b"Welcome to this server")
```

- La méthode «send()» renvoie le nombre d'octets expédiés;
- Les données à envoyer en utilisant la fonction «send» doivent être de type «bytes» (le b devant une chaîne de caractères);
- Ou faire appel à la fonction «encode()» pour convertir une chaîne de caractère en bytes :

```
Sock_client.send("Welcome to this server".encode())
```

10) Afficher les données reçues

Vous remarquez la lettre b à côté du message reçu. Car le message est de type bytes.
Donc on utilise la méthode «decode()» pour convertir la donnée de «bytes» en une chaîne de caractère.

```
print("Serveur a reçu : ",msgClient.decode()," -- du client : ",Sock_client)
```

11) Fermeture

```
Sock_client.close()

Sock_serv.close()
```

Partie 2 : Côté client

1) Ouvrir un éditeur de texte et enregistrer le fichier sous le nom «client.py»
2) Importer le module «socket»
3) Création du socket :
   Créer un socket au niveau du client, comme cela a été fait au niveau du serveur et l'appeler «s»

4) Demande de connexion au serveur :

```
s.connect(('127.0.0.1', 60000))
```

La fonction «connect()» ne s'utilise qu'en mode connecté (donc avec le protocole TCP).
Il est possible d'utiliser «'localhost'» à la place du «'127.0.0.1'»

5) Émettre des données avec la méthode «send()»;
6) Réception des données grâce à la méthode «recv()»;
7) Fermeture du socket client avec la méthode «close()».

Exécution

1) Démarrer l'invite de commandes de votre machine;
2) À l'aide de la commande **cd** entrer dans le répertoire où se trouve le script «serveur.py»
3) Utiliser la commande **dir** (sous Windows) et **ls** (sous Linux) pour vérifier que vous êtes dans le bon répertoire;
4) Exécuter le script «serveur.py» à l'aide de la commande python:

```
>python serveur.py
```

5) Démarrer un deuxième invite de commandes;
6) À l'aide de la commande **cd** entrer dans le répertoire où se trouve le script «client.py»
7) Exécuter le script «client.py» à l'aide de la commande python:

```
>python client.py
```

Questions:

1) Pourquoi l'adresse IP utilisée dans ce TP est 127.0.0.1?
2) *Quelle est la condition imposée sur le choix de la valeur du numéro de port?*

Exercice 2:

1) Ajouter une instruction pour permettre l'affichage du message reçu au niveau du client
2) Ré-exécuter le script serveur et avant de lancer le client, utiliser la commande **netstat** pour afficher les connexions tcp ouvertes. Pour plus de détails sur cette commande, utiliser le Help.
3) Ré-exécuter le script client. Une fois les deux scripts terminés, lancer de nouveau la commande **netstat** pour voir si les sockets sont toujours ouverts.
4) Changer les deux scripts pour que le message envoyé et le numéro de port du serveur deviennent dynamiques (C'est à l'utilisateur de saisir le message à envoyer et le numéro de port).
5) Utiliser la méthode «getsockname()» de la classe «socket» pour que le script «client» affiche l'adresse IP et numéro de port attribués au socket du client.
6) Utilisez la méthode «bind()» pour attribuer une adresse IP et N° port statique au client (N°port = 62000).
   a) *Lancer le script client pour qu'il envoie un message au serveur*
   b) *Lancer de nouveau le script client.*
   c) Qu'est ce que vous remarquez?
7) Modifier le script du serveur pour que ce dernier une fois il a terminé la communication avec le client , il revient pour accepter de nouvelle demande de connexion. Ce travail se répète à l'infini.

Exercice 3:

1) Modifier le script client pour qu'il puisse maintenir un dialogue avec le serveur (envoyer plusieurs messages)

2) Modifier la boucle sans fin (au niveau du serveur) pour maintenir le dialogue jusqu'à ce que le client décide d'envoyer le mot « fin » ou une simple chaîne vide. Les écrans des deux machines afficheront chacune l'évolution de ce dialogue.
Utiliser la méthode «lower()» pour la comparaison. Exemple:

```
s1 = 'STRIng'
if s1.lower()=='string':
    print('They are equal')
```

Exercice 4:

1) Changer le type d'adresse utilisé de IPv4 en IPv6
2) Exécuter le script serveur et après la commande **netstat**
3) *Qu'est ce que vous remarquez?*
4) Exécuter le script client sans changer le type d'adresse
5) *Qu'est ce que vous remarquez?*
6) Changer le type d'adresse du client et exécuter le script client
7) *Qu'est ce que vous remarquez?*

TP N°2 : Communication inter-process (UDP)

Objectif

Implémenter une communication inter-process en utilisant l'API socket en mode non connecté ou non fiable.

Excercie 1 :

1) Implémenter une communication inter-process avec les sockets en mode non connecté («serveur_udp.py» et «client_udp.py»). Sachant que la procédure est similaire à celle du mode connecté. Enfin à certaine exceptions:

a) Pour la création de socket, on modifie l'argument qui défini le type de service par SOCK_DGRAM;
b) Dans le mode non connecté, l'établissement de connexion n'existe pas. Ainsi, les méthodes : «listen()», «accept()» et «connect()» n'existent pas;
c) C'est au moment de l'envoie qu'il faut préciser l'adresse de destination. Pour faire, on utilise la méthode «sendto()» de la classe socket :

```
sendto(bytesToSend, serverAddressPort)
```

- bytesToSend : le message à envoyer, de type byte
- serverAddressPort : tuple contenant l'adresse IP et le numéro de port

d) Pour la réception des données, on utilise la méthode «recvfrom(bufferSize)» qui possède la même syntaxe que «recv()» en mode connecté.

2) Remplacer la méthode «recvfrom()» par «recv()» :

a) Est il possible de recevoir le message ?
b) Pourquoi il est recommandé d'utiliser «recvfrom()».

Exercice 2 : Passage d'arguments de la ligne de commande

On souhaite récupérer la valeur du message à envoyer et N° port du serveur depuis la ligne de commande. Exemple:

```
>python script.py Bonjour 60000
```

- Pour cela on utilise le module **sys :** il fournit un accès à certaines variables utilisées et maintenues par l'interpréteur. Pour pouvoir l'utiliser, il faut importer le module **sys**
- **sys.argv :** représente la liste des arguments (type string) de la ligne de commande passés à un script Python. argv[0] est le nom du script, argv[1] vaut la chaîne 'Bonjour' et argv[2]=='**60000**'.

```
message = sys.argv[1]
port_distant = int(sys.argv[2])

s.sendto(message.encode(), ('localhost', port_distant))
```

Exercice 3: Gestion des erreurs

Il est hautement recommandable de placer le code :
- la partie servant à établir la connexion (en mode TCP)
- la partie servant à rattacher le socket à une adresse IP et N°port

à l'intérieur d'un gestionnaire d'exceptions «try-except».

1) Améliorer vos deux scripts, en utilisant le gestionnaire d'exceptions.

```
try:
    ####demande de connexion de serveur ou se rattacher à une adresse IP/Port
except:
     #### Message d'erreur à afficher
     sys.exit(0) #### nécessaire pour quitter le programme
```

2) Utiliser la méthode «system('cls')» du module «os» pour effacer l'écran de la console dans un environnement Windows ou «system('clear')» sous linux et macOS.

Exercice (Advanced) : Interface Graphique

Améliorer ce TP, en ajoutant une interface graphique.

Tkinter: Il existe beaucoup de modules pour construire des applications graphiques. Par exemple : Tkinter, wxpython, PyQt, PyGObject, etc. Tkinter est un module de base intégré dans Python , normalement vous n'avez rien à faire pour pouvoir l'utiliser. Tkinter permet de piloter la bibliothèque graphique Tk (Tool Kit), Tkinter signifiant tk interface.

1) Créer une Fenêtre

Nous allons importer le package Tkinter, créer une fenêtre et lui attribuée une dimension un titre.

```
from tkinter import *
fenetre = Tk()
fenetre.geometry('370x400')
fenetre.title("Client")
# Placer les widgets
#....
fenetre.mainloop()
```

La méthode mainloop() est la plus importante. Cette fonction fait appelle à une boucle infinie. Une boucle d'événements indique essentiellement au code de continuer à afficher la fenêtre jusqu'à ce que nous la fermions manuellement.
De plus, aucun code ne s'exécutera après la boucle, jusqu'à ce que la fenêtre sur laquelle elle est appelée soit fermée.

2) Les widget Tkinter

Pour créer un logiciel graphique vous devez ajouter dans une fenêtre des éléments graphiques que l'on nomme widget. Ce widget peut être tout aussi bien une liste déroulante que du texte.

2.1) Les labels

Les labels sont des espaces prévus pour afficher du texte.

```
label = Label(fenetre, text="PORT_Server")
label.place(x=10,y=5)
```

Le widget label possède des options en plus du text: *bg* (background color)*, font, justify*, ...etc

En créant des interfaces graphiques avec Tkinter, vous aurez certainement besoin de définir les emplacements de vos widgets. En effet, il existe trois méthodes principales : pack(), grid() et place().

Method place() : Ce gestionnaire de géométrie organise les widgets en les plaçant dans une position spécifique dans le widget parent.

Syntaxe : `widget.place( place_options )`
Il existe une liste des options possibles. Ce qui nous intérésse au décalage horizontal et vertical en pixels (x, y).

2.2) Entrée / input

```
W_Port = Entry(fenetre)
W_Port.place(x=100,y=5)
```

Récupérer la valeur d'un input : Pour récupérer la valeur d'un input il vous faudra utiliser la méthode **get()** :

2.3) Text

Les widgets de texte offrent des fonctionnalités avancées qui vous permettent de modifier un texte multiligne.
Syntaxe: `w = Text ( master, option, ... )`
master : Ceci représente la fenêtre
En plus des mêmes options du label (citées en haut), le widget text possède d'autres options comme *highlightcolor* (couleur de surbrillance)

```
W_Message = Text(fenetre, width=30, height=18, font=("Arial", 12))
W_Message.place(x=10,y=50)
```

Récupérer la valeur d'un input :
Le widget Text possède aussi la méthode **get()** pour récupérer la valeur d'un Texte saisi par l'utilisateur, qui a un argument de position de départ, et un argument de fin facultatif pour spécifier la position de fin du texte à récupérer.

Syntaxe : `get(start, end=None)`
Si 'end' n'est pas indiqué, un seul caractère spécifié à la position de départ 'start' sera renvoyé.

```
msg = W_Message.get("1.4","end")
```

Récupérer le texte de la première ligne, à partir du 4ème caractère j'usqu'à la fin.
Insérer une valeur dans un widget text :
Syntaxe : `insert(index, [,string]…)`
Les chaînes passées en deuxième argument sont insérées à la position spécifiée par l'index passé en premier argument.

```
msg=s.recv(1024)
W_Message.tag_config('r', background="lightsteelblue", foreground="royalblue")
W_Message.insert(END,"\n")
W_Message.insert(END,msg.decode()+"\n", 'r')
```

2.4) Les boutons

Les boutons permettent de proposer une action à l'utilisateur. Dans l'exemple ci-dessous, on lui propose de fermer la fenêtre.

```
B_connect = Button(fenetre, text="connect", command=lambda: get_connect(W_Port))
B_connect.place(x=300,y=5)
```

Ajouter un bouton fonctionnel
Il est possible d'attribuer une action au bouton grâce à l'option **command**. Dans l'exemple, on fait appel à la fonction **get_connect()**.

Passage de paramètre lors de clic d'un bouton
Pour faire passer des arguments à la fonction associée à l'option **command**, on ajout la fonction ***lambda***.

Désactiver un bouton Tkinter en Python.
Un bouton Tkinter a trois états: ***active, normal, disabled***.
disabled : pour griser le bouton et le rendre insensible.

```
def get_connect(str):
    s.connect(('127.0.0.1', int(str.get())))
    B_connect['state'] = DISABLED
```

Attribuer une image à un bouton
Il est possible d'attribuer une image à un bouton ou un label, en utilisant la fonction **PhotoImage()** :

```
send_img = PhotoImage(file="image_send.png")
B_send.config(image=send_img)
```

Il est possible que le type de l'image en question ne soit pas reconnu par interpréteur. Il faut s'assurer que l'image est vraiment ***png.***

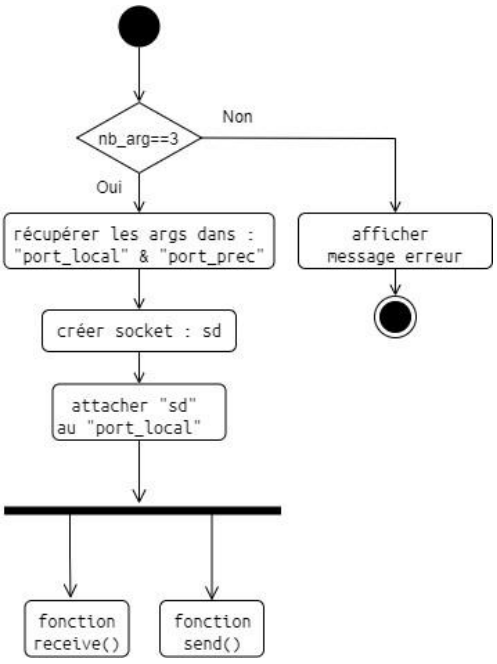
TP N°3 : Topologie Token Ring

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.

Exercice 1: Environnement de base

Implémenter le script «node.py» représentant un nœud avec socket. À chaque fois qu'on exécute «node.py», dans un terminal, cela représente le lancement d'un nouveau nœud dans la topologie.



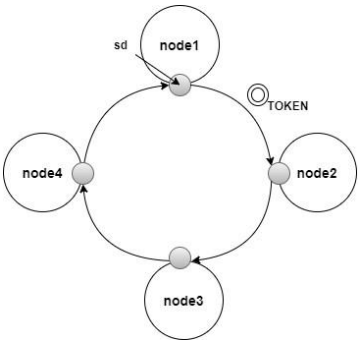
Le ou les numéros de port devra (devront) être passé(s) comme arguments ç-à-d pouvoir lancer le script comme suivant : **`>python node.py port_local port_prec`**

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.

Exercice 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.



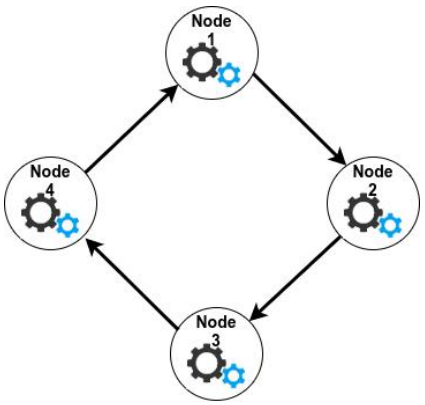
1) Ajouter une variable «token_held» booléenne qui indique si le nœud détient le jeton.
2) Optimiser le code pour que lorsqu'on exécute la commande suivante : **`>python node.py port_local port_prec`**, on aura l'option d'ajouter un 4ème argument "jeton" qui fera comprendre au programme que ce nœud possède 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).

Exercice 3: Gestion du nœud suivant

Optimiser le script pour que le nœud arrive à déterminer quel est son nœud suivant sans que l'utilisateur ne donne l'information.

Exercice (Advanced) :

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.

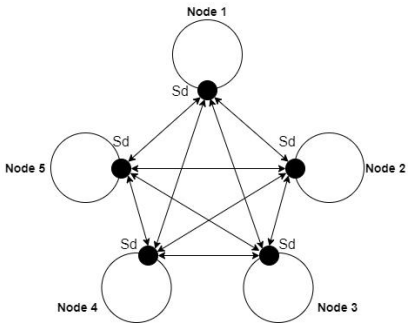


Optimiser le script pour que même les messages seront envoyés suivant la topologie en anneau.

TP N°4 : Topologie Maillée

Objectif

Dans ce TP, vous allez mettre en place un réseau maillé (*mesh*) en utilisant l'API BSD sockets mode UDP.



Principe de la topologie

Dans un réseau maillé (*mesh*), les nœuds sont tous connectés entre eux. Sans faire appel à une entité intermédiaire.

Dans ce TP, on utilise les sockets en mode UDP. Chaque nœud possède un seul socket pour l'entrée et sortie. Le but est de réaliser un implémentation simplifier.

Gestion de fichier texte

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.

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

```
isfile(path)
```

2) Écrire dans un fichier

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

3) Lire un fichier

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())
```

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("position actuelle : ", f.tell())
   f.seek(0)  # position le curseur au début
```

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)

Exercice 1:

- Le script principale s'appelle «node.py» représentant un nœud avec socket qui devra s'exécuter de la manière suivante: **`>python node.py port_local`**
- L'envoie et la réception exécutent simultanément.
- Ne pas oublier d'entourer l'instruction «bind()» d'un bloc «try-except».

- Utiliser les outils de gestion de fichier pour permettre à un nœud d'enregistrer son numéro de port dans un fichier texte et récupérer les ports enregistrés:

  ◆La fonction «verif_port_exist» vérifie si le port_local existe déjà dans «BDD.txt»
  ◆La fonction «creat_my_list» : récupérer les numéros de ports enregistrés dans «BDD.txt» et les enregistrer dans une structure de données de type liste et nommée «my_list».
  ◆«my_list» est une variable globale.
  ◆La fonction «add_port» ajoute le «port_local» dans «BDD.txt»
  ◆La fonction «introduce_self» permet au socket d'envoyer un message pour se faire connaître aux prêts des autres sockets.
  ◆La fonction «registry_in_list» permet d'enregistrer le nouveau numéro de port reçu dans «my_list»

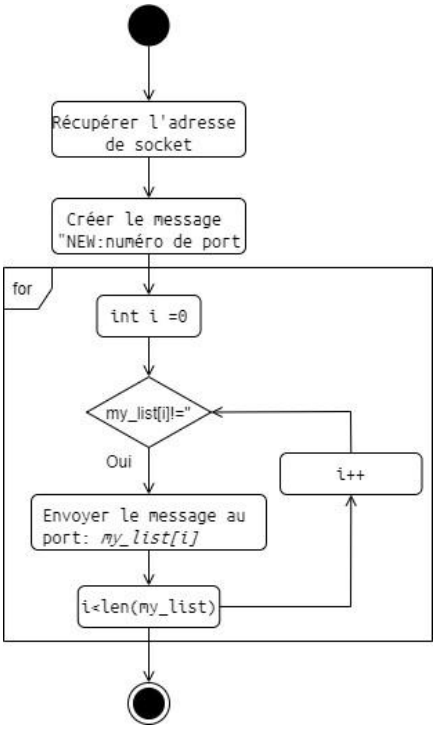


Figure 1-Diagramme d'activité de la fonction "introduce_self"

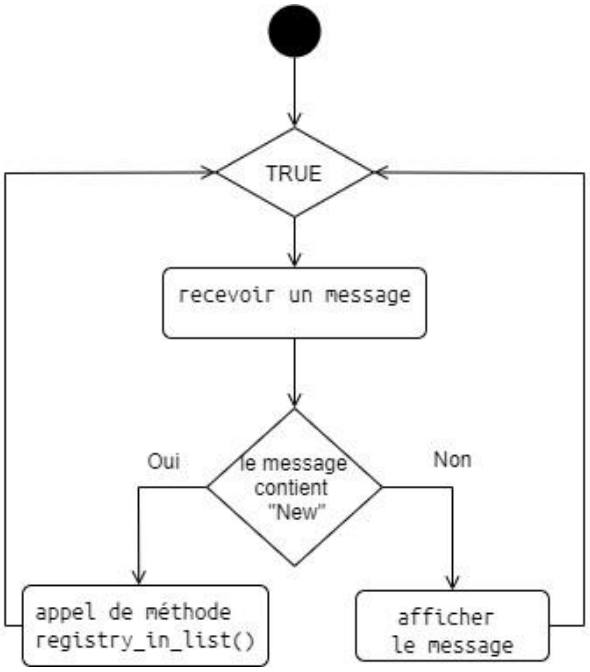


Figure 2-Diagramme d'activité de la méthode "handle_reception

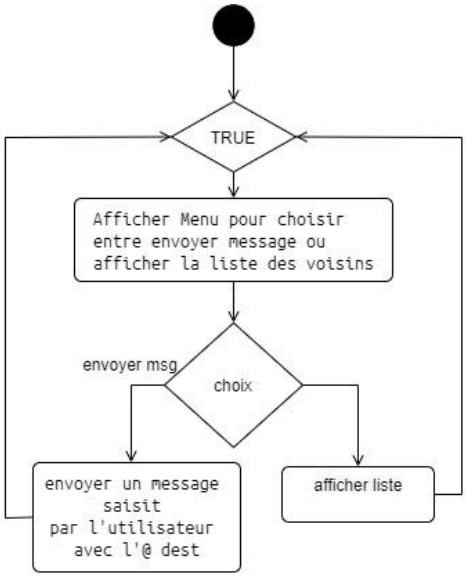


Figure 3-Diagramme d'activité de la méthode "handle_treatment"

Exercice 2:

Ajouter une fonction «broadcast» permettant à un nœud d'envoyer un message à tous ses voisins.

Exercice 3:

Optimiser le code de TP3 avec les outils de gestion de fichier pour se débarrasser des arguments passés au niveau de la ligne de commande.

TP N°5 : Mémoire partagée

Objectif

Dans ce TP, vous allez implémenter une version simplifiée de l'algorithme producteurs_consommateur.

Introduction

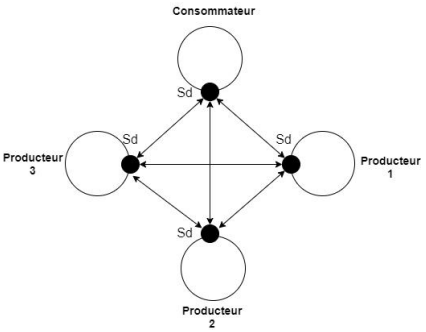
Le concept de producteurs-consommateur est connu chez les étudiants en licence. Il a été abordé dans le cours de Système d'Exploitation. Où plusieurs processus, dans la même machine, souhaitent accéder à une ressource en commun (ex: un tableau afin d'empiler ou dépiler) appelée « section critique ». L'algorithme en question était là pour gérer cet accès à travers des sémaphores (les opérations wait et signal).

Mais qu'en est-il des processus qui se trouvent dans des sites (nœuds) distincts et seule une connexion réseau les connectes?

Comment l'algorithme de producteurs-consommateur a été adapté dans le cas d'un système réparti?

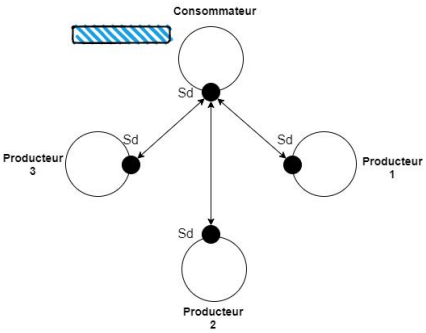
Principe

L'algorithme de producteur_consommateur est implémenter dans un système réparti où les nœuds sont interconnectés à travers un réseau maillé. On a «p» nœuds de production P_1 ,…, P_p et 01 seul nœud de consommation C .



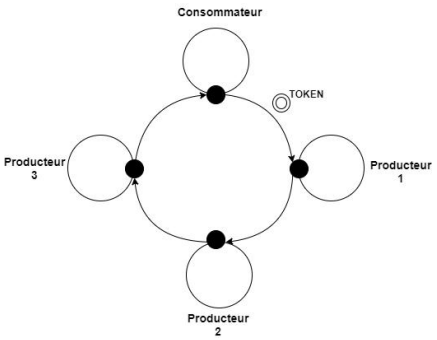
Chaque producteur a le droit de produire plusieurs messages.

Aussi, les producteurs sont aussi connectés directement au consommateur pour pouvoir envoyer les messages produits.



Pour géré l'accès à la mémoire partagée, le concept de sémaphore ou verrou a été remplacé par le principe de token ring dans le système réparti. Où seul le nœud qui possède le jeton (token) aura le droit d'envoyer ses messages produits : remplir le tableau de stockage du consommateur.

Ainsi, la section critique est représentée par un tableau déclaré au niveau du consommateur. Le nœud qui va initialiser le Token sera le consommateur.



Producteur

- Le nœud doit identifier le port du consommateur;
- Chaque producteur P_i n'a le droit d'envoyer un message au consommateur pour qu'il soit stocker au niveau de se dernier que s'il possède le jeton. Il peut envoyer autant de messages qu'il souhaite tant qu'il reçoit pas le mot : «memory full».

Consommateur

- Quand le consommateur reçoit un message produit, il le stocke dans son tampon. On peut dire que le producteur a accédé à la section critique.
- Lorsque le consommateur réalise que le tampon est plein, il vide le tampon (consommer tous les messages).

Exercice 1:

Suivant ces explications et ce qui a été réalisé au niveau des précédents TPs, proposer une implémentation de l'algorithme producteurs-consommateur.

Exercice 2:

En programmation concurrente avec les threads, nous avons parfois besoin de coordonner les threads avec une variable booléenne. Python fournit un objet événement via la classe **threading.Event**.

Un objet **threading.Event** encapsule une variable booléenne qui peut être définie «*set*» (True) ou (False). Le **threading.Event** fournit un moyen simple de partager cette variable entre les threads qui peuvent servir de déclencheur pour une action. C'est l'un des mécanismes les plus simples de communication entre les threads.

```
flag = threading.Event() #l'événement sera dans l'état "non défini"

flag.clear() # Set to False => le thread est bloqué

flag.wait()  # wait for the event to be set to True
             #   => Débloquer le thread jusqu'à ce que flag soit True

flag.set()   # Set to True => le thread est débloqué
```

1) Au niveau de chaque nœud, le thread responsable de l'envoi (pour le producteur) et celui de consommer (pour le consommateur) devra être bloqué (en pause) jusqu'à la réception du «Token». Utiliser la classe «Event()» pour bloquer et débloquer les threads.

TP N°6 : Horloge logique

Objectif

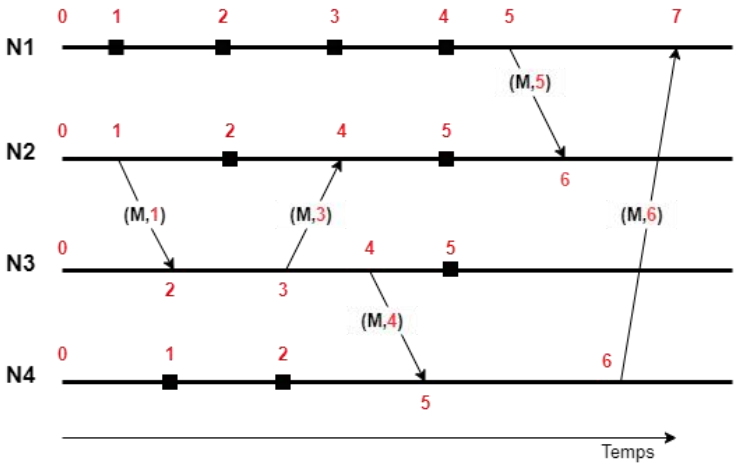
Dans ce TP, vous allez implémenter l'algorithme des horloges logiques de Lamport. Le travail en question ce base sur la topologie en maille précédemment réalisée.

Principe de l'algorithme

Chaque processus **Ni** possède sa propre horloge **HLi** à valeurs entières. Un événement **e** qui se produit est daté par la valeur courante de **HLi** .

Les règles d'évolution des horloges au niveau du processus **Pi**:

- Initialisation : **HL =0**
- Événement local : **HL =HL +1** .
- Envoi d'un message **M** : envoyer un message **M** estampillé avec la valeur de l'horloge **(M, H)**, après avoir incrémenter la valeur de l'horloge.
- Réception d'un message **(M, H) : HL=max(HL, H) +1** .



Les valeurs en rouge représentent la valeur de l'horloge

■ : événement local

Implémentation

Dans l'algorithme des horloges, les messages sont estampillés avec la valeur d'horloge. Il est possible d'envoyer le message concaténé avec la valeur d'horloge. Cependant, cette solution n'est pas scalable[1] (non évolutif). Ainsi, il est plus judicieux de définir le message comme un objet dont les attributs sont «clock» et «content». «content» est le contenu du message.
Pour envoyer des objets à travers les sockets, il faut faire appel à la **sérialisation/désériablisation**.

La sérialisation est le processus de conversion d'un objet en flux d'octets (*bytes*) pour stocker l'objet ou le transmettre à la mémoire, une base de données ou un fichier. Son principal objectif est d'enregistrer l'état d'un objet afin de pouvoir le recréer si nécessaire. La sérialisation est aussi connu sous les termes de "pickling", "marshalling" ou encore de "flattening". Il y a deux modules pour ça: «pickle» et «json».

Pickle Vs JSON:https://docs.python.org/fr/3/library/pickle.html?highlight=getattr-comparison-with-json

- pickle est un format binaire, tandis que JSON est un format textuel (constitué de caractères Unicode et généralement encodé en UTF-8) ;
- JSON peut être lu par une personne, contrairement à pickle ;
- JSON offre l'interopérabilité avec de nombreux outils en dehors de l'écosystème Python, alors que pickle est propre à Python ;
- Par défaut, JSON n'est capable de sérialiser qu'un nombre limité de types natifs Python, contrairement à pickle.

Module «pickle»

- pickle.dumps : Renvoie la représentation sérialisée de l'objet sous forme de bytes. À ne pas confondre avec la méthode «pickle.dump» qui permet d'écrire la représentation sérialisée de l'objet dans un fichier

```
s.sendto(pickle.dumps(msg), ('localhost',int(my_list[i])))
```

- pickle.loads : renvoie l'objet reconstitué à partir de la représentation sérialisée data. À ne pas confondre avec la méthode «pickle.load»

```
msg, addr=s.recvfrom(1024)
msgD = pickle.loads(msg)
if "New" in msgD.content:
```

Exercice 1 :

En se basant sur le TP «mesh» :
1) définir le message à échanger comme un objet dont les attributs sont «content» et «clock» .
2) développer les fonctions nécessaires à l'algorithme lamport. Soit:
   - lamport_when_receive()
   - lamport_when_send()
   - lamport_local_event()

Exercice 2 :

Implémenter les horloges vectorielles.

Exercice (Advanced) :

Installer le module JSON et utiliser le pour sérialiser et désérialiser les messages.

---

[1] La scalabilité est un terme employé dans le domaine de l'informatique matérielle et logicielle, pour définir la faculté d'un produit informatique à s'adapter aux fluctuations de la demande en conservant ses différentes fonctionnalités.

TP N°7 : Élection

Objectif

Dans ce TP, vous allez implémenter l'algorithme d'élection de *Chang et Roberts*.

Principe de l'algorithme

- L'algorithme en question s'applique sur la topologie en mesh.
- Chaque nœud possède un identifiant ID. Générer par lui même de manière aléatoire.
- **Type de message:** Le message utilisé pour l'élection est un objet dont les attributs sont :

  «Type» = toujours contient la chaîne de caractère «ELECT»
  «Id_elect» : contient l'identifiant du nœud élu
  «Port_elect» : contient le numéro de port du nœud élu.

- Un nœud possède deux variables globales: «Leader_ID» et «Leader_Port».
- On propose l'algorithme suivant pour le nœud :

Initialisation de l'élection :

```
1. générer une valeur aléatoire représentant l'identifiant du nœud. La valeur
   est stockée dans la variable «ID»
2. Leader_ID ← ID
3. Leader_Port  ← PORT_In (Numéro de port)
```

Réception du message M:

```
1. si (M.Id_elect > Leader_ID) alors :
        Leader_ID ← M.Id_elect
        Leader_Port ← M.Port_elect
   sinon  :
        M.Id_elect  ← Leader_ID
        M.Port_elect ← Leader_Port

2. envoyer au nœud suivant le message (M)
```

Exercice 1 :

1) Implémenter l'algorithme d'élection de *Chang et Roberts* pour d'élire le nœud avec le plus grand identifiant (mais qui ne le sait pas au départ.). Ce nœud sera considéré comme le leader.

2) Ajouter les instructions d'affichage au niveau de l'éditeur de commande pour voir le ID du nœud généré puis les valeurs de «Leader_ID» et «Leader_Port» après élection;

3) Le processus de réception se répète jusqu'à ce que chaque nœud reçoive la même valeur Leader_ID et Leader_Port 2 fois.

TP N°8 : Sécurité des données

Objectif

Dans ce TP, vous allez vous initier à la sécurité des données en utilisant un algorithme de cryptage. L'algorithme en question s'applique sur la topologie en mesh.

Implémentation

Python ne dispose pas d'un module intégré pour le cryptage de messages via des sockets UDP. Cependant, vous pouvez utiliser des bibliothèques tierces pour le cryptage. L'une des bibliothèques de cryptage les plus couramment utilisées en Python est «cryptography».
Pour l'installer :

```
> pip install cryptography
```

Le module «cryptography» contient plusieurs algorithmes de cryptage. Dans ce TP, nous allons utiliser l'algorithme symétrique «Fernet» :

```python
from cryptography.fernet import Fernet

# Générez une clé de cryptage
key = Fernet.generate_key()
cipher_suite = Fernet(key)

# Message à envoyer
message_to_send = "Message secret à envoyer".encode()
encrypted_message = cipher_suite.encrypt(message_to_send)

# Envoi du message crypté
sock.sendto(encrypted_message, (UDP_IP, UDP_PORT))

# Réception du message crypté
data, addr = sock.recvfrom(1024)
decrypted_message = cipher_suite.decrypt(data)
print("Message reçu :", decrypted_message.decode())
```

- En se basant sur ces instructions, optimiser le script «node.py» pour pouvoir envoyer des messages cryptés.
- Ne pas oublier que tous les nœuds doivent avoir la même clé de cryptage.
- Utiliser l'outil wireshark pour visualiser les messages échangés.