

Communications

mardi 24 septembre 2024 13:12

Quand les processus se trouvent sur deux sites distincts, la synchronisation se fait par **envoi** et **réception** de messages

Modèle de distribution :

- Un ensemble de sites
 - Chaque site possède sa propre mémoire non accessible aux autres sites
 - Chaque site dispose d'un identifiant unique
- Des lignes de communication
 - Bi-point : reliant 2 sites
 - Bi-directionnelle : l'échange est disponible dans les deux directions
 - Chaque direction est appelé "canal"
 - On considère comme une clique :

Chaque site est conçu selon le modèle en couche :

- Couche réseau
- Couche services
- Couche application

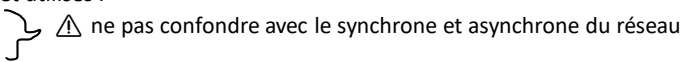
Chaque couche fournit un ensemble de services aux couches supérieures

La communication : (on garde la définition de la couche application)

Une communication est une suite de 3 actions :

- **l'envoi**
- Le transport (ne nous intéresse pas dans ce cours)
- **La réception**

On va voir les modèles les plus connus et utilisés :

- Communication synchrone
 - Communication asynchrone
 - Communication par rendez-vous
- 

Communication synchrone :

La communication est dite synchrone quand les actions d'envoi et de réception sont possibles que si :

- l'émetteur se trouve en état d'envoi
- Et le récepteur dans un état de réception

=> d'où la modélisation en utilisant un **canal de communication** => c'est une ligne de communication un à un dans lequel il n'y a que deux partenaires : un émetteur unique et un récepteur unique. L'émetteur ne fait que des émissions et le récepteur ne fait que des réceptions sur un canal. Si on veut par exemple que l'émetteur fasse des réceptions, il faut un autre canal.

La sémantique des actions de communication :

- Envoi : $c!e; P \xrightarrow{c!e} P'$
- Réception : $c?v; P \xrightarrow{c?v} P'$

Un processus qui est prêt à émettre/recevoir sur un canal est bloqué tant qu'aucun autre processus n'est capable d'effectuer une action complémentaire sur le même canal

Emulation java d'une communication synchrone :

Cas producteur/consommateur :

- Canal : mémoire tampon de taille 1
- Émetteur : le producteur
- Récepteur : le consommateur

Ce qui change :

Le producteur finit de produire que :

- s'il a fini de mettre l'objet dans le tampon
- **ET que le consommateur a consommé ce qui a été produit**

Le consommateur reste inchangé

Avantages et inconvénients :

- Communiquer est une action de synchronisation
- Un moyen efficace pour le contrôle d'exécution des applications distribuées => applications simples
- Contraignant quand la synchronisation n'est pas nécessaire, mais juste l'échange de donnée

- => beaucoup de synchrone tue la concurrence (synonyme d'efficacité)
- Une application mal conçue => risque de blocage
- Permet seulement une communication un à un

Communication asynchrone :

- l'action d'envoi n'est pas bloquante (peut l'être si le répondeur est plein)
- l'action de réception est bloquante ssi il n'y pas de message en attente
- Communication n-à-1

=> d'où l'utilisation de la notion de **port**

- Adresse d'écoute
- Doté d'une mémoire tampon

Un processus peut avoir un ensemble de ports modélisé par une suite d'expressions $p = \langle e_1, \dots, e_n \rangle$

$Q = [p_1 = \langle \dots \rangle, \dots, p_n]$

Emulation java d'une communication asynchrone :

- Plusieurs producteurs - un consommateur
- Une zone de tampon avec une taille illimitée (ou limitée, bornée)
- Les émetteurs : producteurs
- Le port : la zone bornée de taille n
- Le récepteur : consommateur

Communication rendez-vous :

J'envoie un message asynchrone pour une réception synchrone

J'envoie un message et j'attends un message en retour

La forme utilisée pour réaliser les systèmes Requête-Réponse pour supporter la communication **client-serveur**

C'est une forme qui mélange les deux premières formes

- Les clients envoient les demandes de RDV pour le traitement de requêtes
- Le serveur traite de manière asynchrone les requêtes. Une à la fois.
- Les réponses sont envoyés au client sur un canal qui lui est spécifique

Les **entrées** se comportent comme des ports pour stockés les requêtes client pour un traitement asynchrone en plus d'autres fonctionnalités

La méthode **appel** crée un canal et met le client en attente de la réponse sur ce canal. Le client à l'appel de **appel** se bloque tant qu'il n'a pas reçu ni la réponse ni un refus.

Distribution inter-applications et inter-machines :

Les sockets

Il faut :

- l'adresse internet de la machine
- Le numéro de port

Pour que deux machines puissent communiquer, il faut qu'une joue le rôle du client et l'autre le rôle du serveur

Notion de port :

- Correspond à un port logique et non physique
- Numéro allant de 1 à 65535
- Les ports de 1 à 1023 sont réservés aux services courants (finger, ftp, http(80), SMTP(25), ...)

Adresses internet :

- Connexion internet :
 - Adresse internet de la machine
 - Numéro : 193.49.192.193
 - Il existe des adresse IP accessibles par le réseau local et des adresses IP accessibles par l'extérieur
- Désignation par des noms symboliques
 - Les noms symboliques sont associés aux adresses numériques
 - DNS (Domain Name Server)
 - Ex: lil.univ-littoral.fr : 193.49.192.193

Le client ouvre la connexion avec le serveur avant de pouvoir lui adresser des appels, puis ferme la connexion

Un serveur peut répondre à des demandes de services de plusieurs clients

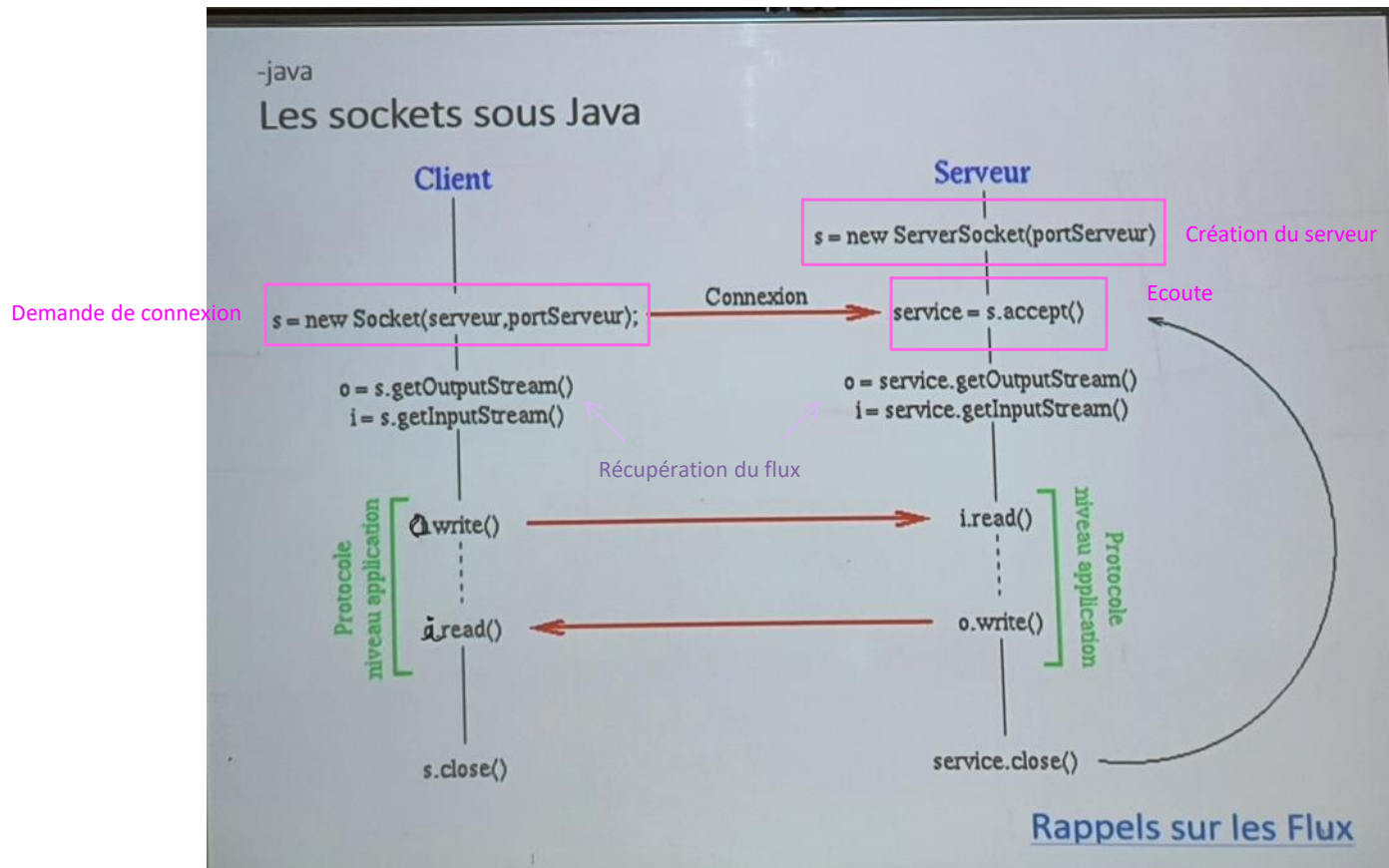
La méthode `fork()` permet de créer un sous serveur qui va s'occuper des demandes d'un client et le serveur principal n'a donc plus la main sur le client

La communication sous java :

- `java.net.Socket`
 - Côté client (mais aussi côté serveur)
 - Permet une communication 1-1 (1 client - 1 serveur)
 - c'est un couple de canaux (asynchrone)
- `java.net.ServerSocket`
 - Utilisé uniquement côté serveur
 - c'est un point d'entrée
 - Req : demande d'établissement d'une connexion
 - Rep : établissement d'une socket entre le serveur et le client

IN pour envoyer des données

OUT pour recevoir des données



Envoie d'une donnée avec la méthode `write()`

Lecture d'une donnée avec la méthode `read()` -> cette méthode est synchrone

Si on a plusieurs clients, on utilise les processus (Threads)