Programmation réseau

Gestion de la concurrence

Préparé par : A. Begdouri
MST-SIR1

Gestion de la concurence

Bien concevoir son serveur

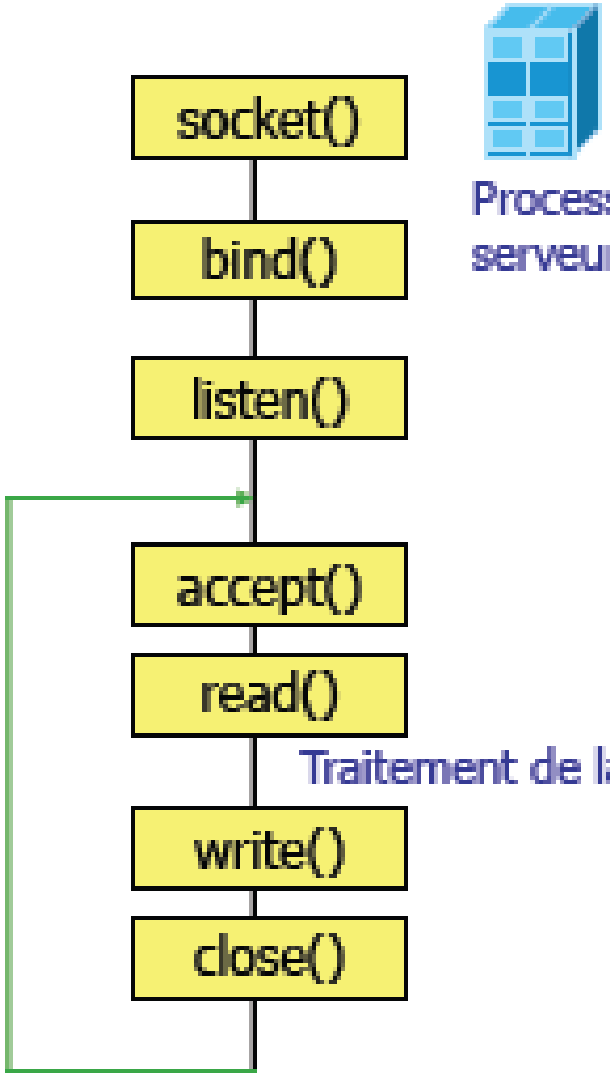
- En fonction de l'objectif métier du service et de l'environnement de l'application.

- Choix du serveur:
  - Serveur concurrent / itératif ?
  - Serveur avec état / sans état ?
  - Serveur multi-protocoles / multi-services ?

Serveur itératif / concurrent

- Serveur itératif
  - Traite séquentiellement les requêtes
- Serveur concurrent
  - Le serveur accepte les requêtes et les délègues à des processus fils

Serveur itératif en mode connecté



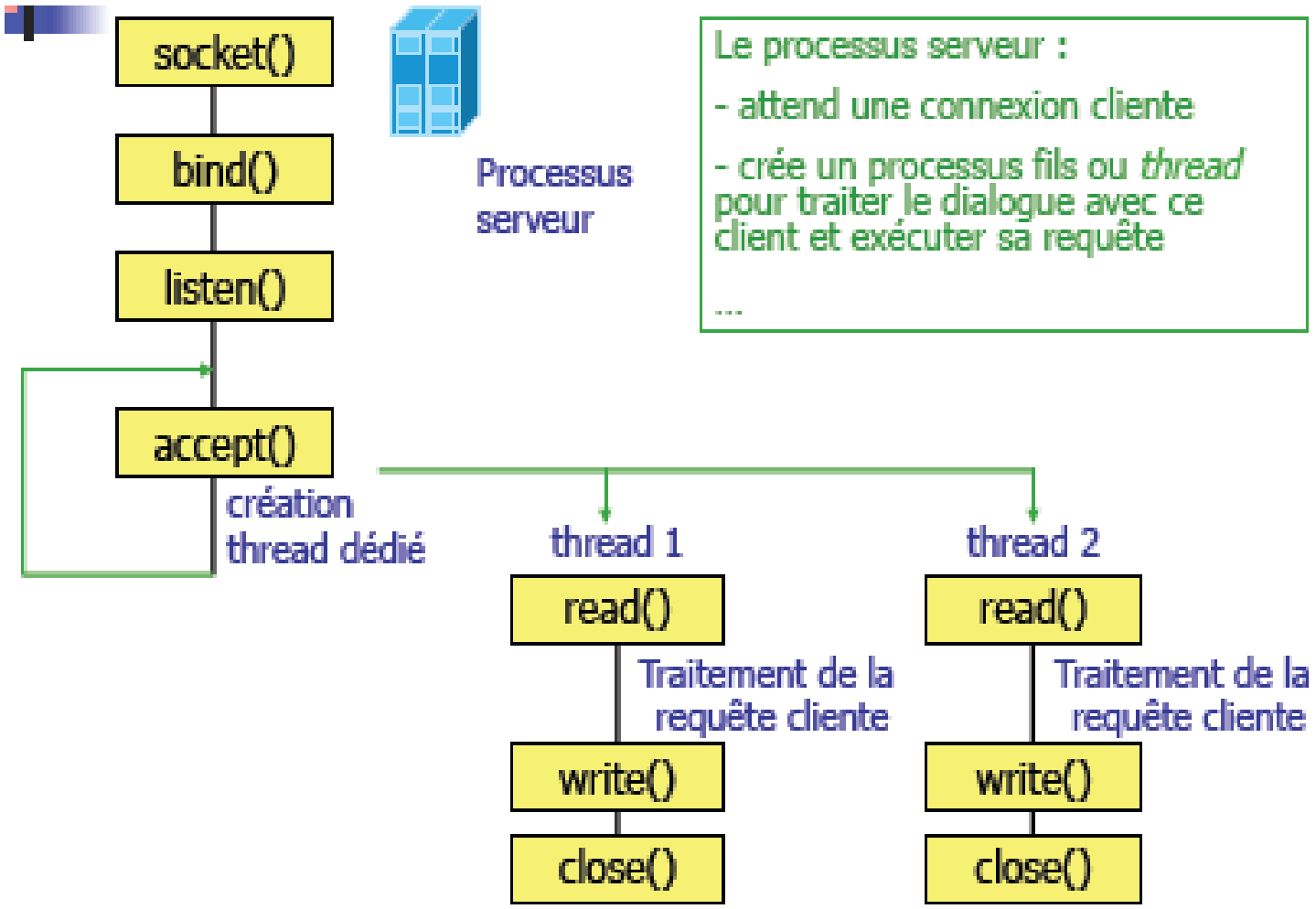
Processus serveur

Le processus serveur :

- attend une connexion cliente
- lit la requête
- traite la requête
- envoie la réponse
- ferme la connexion cliente

...

Serveur concurrent en mode connecté



Serveur itératif / concurrent

- Serveur itératif
  - Adapté aux requêtes qui peuvent s'exécuter rapidement
  - Souvent utilisé en mode non connecté (optimisation des performances)

- Serveur concurrent
  - Adapté aux requêtes qui demandent un temps de traitement
    - le cout du traitement est important pour que le coût de création d'un processus ne soit pas pénalisant
  - Souvent utilisé en mode connecté

Service avec ou sans état

- Service avec état
  - Le serveur conserve localement un état pour chacun des clients connectés: informations sur le client, les requêtes précédentes, etc.
- Service sans état
  - Le serveur ne conserve aucune information sur l'enchainement des requêtes, etc.

=> Incidence sur les performances et la tolérance aux pannes lorsque le client fait plusieurs requêtes successives:

- Maximiser les performances => utiliser un service sans état
- Maximiser la tolérance aux pannes => utiliser un service avec état

Serveur multi protocoles

- Serveur qui offre le même service en mode connecté et non connecté
- Le serveur écoute sur 2 sockets différentes pour rendre le même service
  - Exemple: le service DAYTIME (RFC 867) permettant de lire la date et l'heure sur le serveur
    - Port 13 sur TCP
      - La requête est implicite représentée par la demande de connexion du client qui déclenche la réponse du serveur
    - Port 13 sur UDP
      - Requiert une requête du client: datagramme arbitraire qui n'est pas lu par le serveur mais qui déclenche l'émission des données de la part du serveur

Serveur multi protocoles: Fontionnement

- Un seul processus utilisant des opérations non bloquantes de manière à gérer les communications en mode connecté et non connecté

- 2 implémentations possibles:
  - En mode itératif
  - En mode concurrent

Serveur multi-protocoles: Mode itératif

- Le serveur ouvre une socket UDP et une socket TCP.
  - Il boucle sur des appels non bloquants à receivfrom() et accept() sur chacune des sockets.
- Si une requête TCP arrive
  - Le serveur utilise accept() pour créer une nouvelle socket de service servant la communication avec le client
  - Lorsque la communication avec le client est terminée, la socket de service correspondante est fermée
  - Le serveur réitère son attente sur les deux sockets initiales.
- Si une requête UDP arrive
  - Le serveur reçoit et émet des messages avec le client
  - Lorsque les échanges sont terminés, le serveur réitère son attente sur les deux sockets initiales

Serveur multi-protocoles: Mode concurrent

- Un automate gère l'arrivée des requêtes par des primitives non bloquantes
- Création d'un nouveau processus fils pour toute nouvelle connexion TCP
- Traitement de manière itérative des requêtes UDP
  - Elles sont traitées en priorité
  - Pendant ce temps, les demandes de connexion sont mises en attente

Serveurs multi-services

- Si une machine doit abriter plusieurs serveurs beaucoup de ressources doivent être mobilisées:
  - Un certain nombre de processus nécessaires
  - Consommation des capacités de calcul et de stockage qui leur sont associées

=> Serveur multi-services: un serveur qui répond à plusieurs services (une socket par service)

- Avantages:
  - Le code réalisant les services n'est présent que lorsqu'il est nécessaire
  - La maintenance se fait à base du service et non du serveur: l'administrateur peut facilement activer ou désactiver un service.

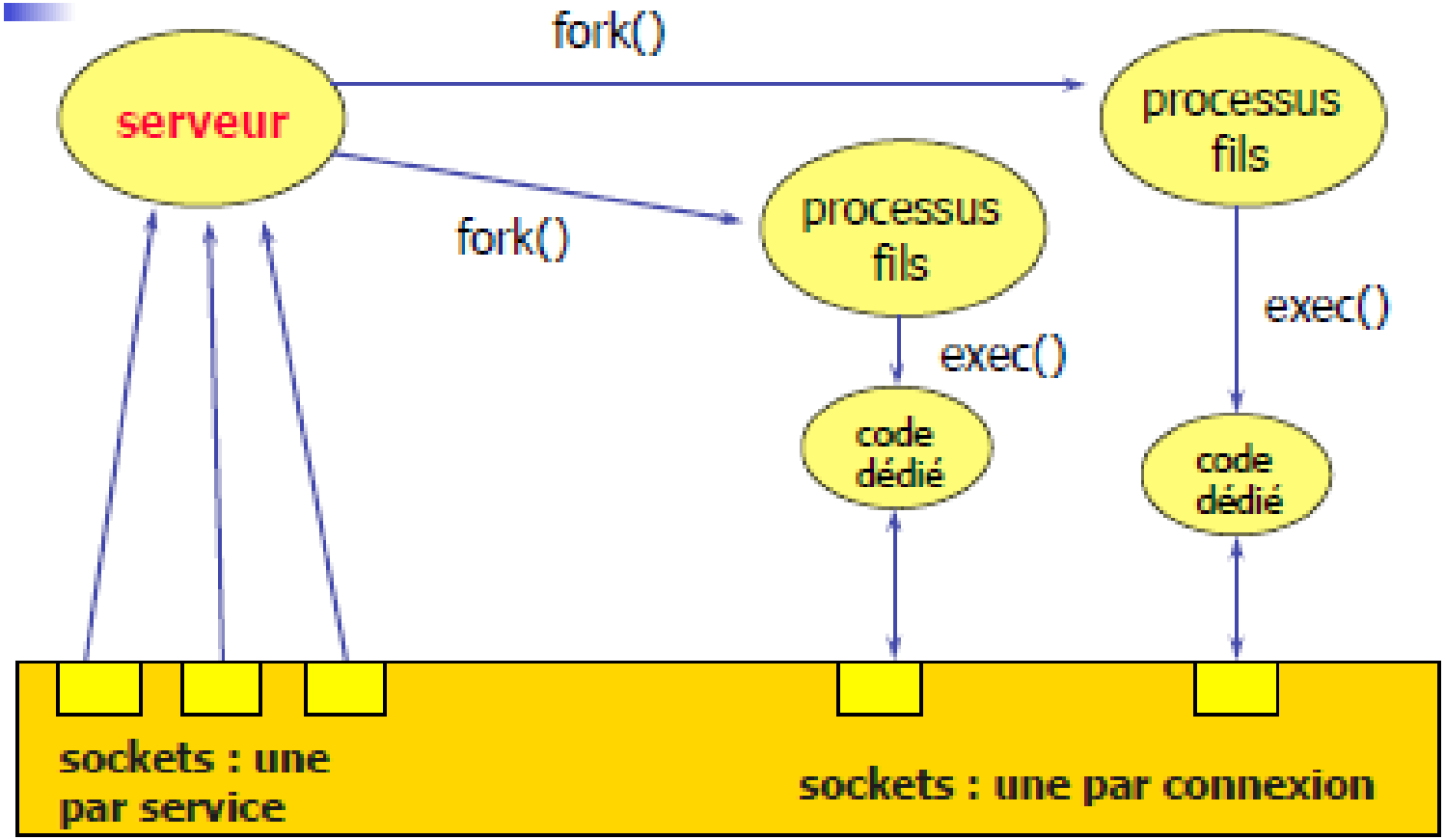
Serveur multi-services: Fonctionnement

- Le serveur ouvre une socket par service offert
- Il attend une connexion entrante sur l'ensemble des sockets ouvertes
- Lorsqu'une connexion entrante arrive, le serveur crée un processus fils qui prend en compte la connexion
- Le processus fils exécute un programme dédié réalisant le service demandé

Serveur multi-services: Fonctionnement

- Le serveur ouvre une socket par service offert
- Il attend une connexion entrante sur l'ensemble des sockets ouvertes
  - **Comment peut-il attendre une connexion sur plusieurs sockets? => scrutation de plusieurs sockets**
- Lorsqu'une connexion entrante arrive, le serveur crée un processus fils qui prend en compte la connexion
- Le processus fils exécute un programme dédié réalisant le service demandé:
  - **via excec() sous UNIX**

Serveur multi services



Scrutation de plusieurs sockets

- Pb:
  - Une attente de connexion accept() sur l'une des sockets empêche l'acceptation sur les autres
  - Pb lié au caractère bloquant des primitives

=> Scrutation:

  - Mécanisme permettant l'attente d'un événement (lecture, connexion, etc.) sur plusieurs points de communications
  - La scrutation est nécessaire dans le cas de serveurs multi-services et multi-protocoles

Scrutation de plusieurs sockets

- Première solution:
  - Rendre les primitives non bloquantes à l'ouverture de la socket
  - Inconvénient: attente active dans une boucle
- Deuxième solution:
  - Création d'un processus fils par socket pour la scrutation d'un service
  - Inconvénient: lourd et gaspillage des ressources
  - Mais l'avantage de l'activation à la demande est conservé

Scrutation de plusieurs sockets

- Troisième solution: la primitive select()

- Permet de réaliser un multiplexage d'opérations bloquantes
- Scrutation sur un ensemble de descripteurs passés en argument:
  - Descripteurs sur lesquels réaliser une lecture
  - Descripteurs sur lesquels réaliser une écriture
  - Descripteurs sur lesquels réaliser un test de condition exceptionnelle (arrivée d'un caractère urgent)
- Un argument permet de fixer un temps maximal d'attente avant que l'une des opérations souhaitées ne soit possible

Scrutation de plusieurs sockets

- La primitive select() rend la main quand une de ces conditions se réalise:
  - L'un des événements attendus sur un descripteur de l'un des ensembles se réalise
  - Le temps d'attente maximum s'est écoulé
  - Le processus a capturé un signal (provoque la sortie de select()).

Les processus démon

- L'invocation d'un service Internet standard
  - (FTP, TELNET, RLOGIN, SSH, RPC, etc.)
- nécessite la présence côté serveur d'un processus serveur:
  - Qui tourne en permanence
  - Qui est en attente des requêtes clientes

➔ **On parle alors de démon**

- A priori, il faudrait un démon par service
  - Or, pb: multiplication des services => multiplication du nombre de démons

=> Sous UNIX, un supér-démon: inetd

Le démon inetd

- C'est un super-serveur
  - Un processus multi-services multi-protocoles
  - Un serveur unique qui reçoit les requêtes
  - Active les services à la demande
  - Evite d'avoir un processus par service, en attente de requêtes
  - Interface de configuration (fichier inetd.conf)
    - Permet à l'administrateur système d'ajouter ou retirer de nouveaux services sans lancer ou arrêter un nouveau processus
- Le processus inetd attend les requêtes à l'aide de la primitive select() et crée un nouveau processus pour chaque service demandé (excepté certains services UDP qu'il traite lui-même)