Exercices SD

MaDSI 1 2020

Partage de ressources

Le but de cet exercice c'est de vous faire utiliser les éléments que vous connaissez pour imaginer des scénarios intuitives quotidiennes.

Questions

☞Donnez cinq types de ressources matérielles et logicielles qui peuvent être utilement partagées dans un réseau.

☞Donnez des exemples de leur partage tel qu'il se produit dans la pratique. Proposez des schémas illustratifs.

Partage de ressources

Considérons des stratégies de mise en œuvre d'une base de données multi-utilisateurs réparties dans différentes zones géographiques.

Questions

☞Quels avantages voyez-vous à adopter une approche à serveur unique pour représenter l'état d'une BD multi-utilisateurs?

☞Quels problèmes pouvez-vous identifier et comment pourraient-ils être résolus?

Partage de ressources

Une utilisatrice arrive dans une gare qu'elle n'a jamais visitée auparavant, avec une tablette équipée de dispositif sans fil. Elle voudrait se connecter localement à l'application de la station, installée sur sa tablette, afin de connaitre l'heure d'arrivée du train. L'application est hébergée dans le serveur de la station. La station est équipée d'un Wifi public.

Questions

☞Suggérer les mécanismes techniques nécessaires à la connexion de l'utilisatrice à l'application de la station.

☞Quels défis techniques doivent être surmontés?

☞Au cas où l'utilisateur utiliserait l'application à distance, qu'est-ce qui se passerait?

Cloud vs Client-serveur

☞Comparer le Cloud avec le modèle client-serveur traditionnel?

☞Qu'est-ce qui est nouveau dans le Cloud computing en tant que concept?

Hétérogénéité

Un programme serveur écrit dans un langage (par exemple C ++) fournit l'implémentation d'un objet (décrivant un ensemble de méthodes) qui est destiné à être accédé par des clients qui peuvent être écrits dans un langage différent (par exemple Java, ou C). Les ordinateurs client et serveur peuvent avoir un matériel différent, mais tous sont connectés à Internet.

Questions

- ☞Décrivez les problèmes dus à l'hétérogénéité qui doivent être résolus pour permettre à un objet client d'appeler une méthode sur l'objet serveur.
- ☞Discuter des problèmes probables pouvant découler de la centralisation de données dans une architecture client-serveur, lorsque le nombre de clients est supposé augmenter de manière considérable au fil du temps.
- ☞Proposer des mécanismes de solution pour chacun de ces problèmes énoncés.

Processus

Considérons des comptes bancaires modélisés comme des objets et que chaque objet de compte bancaire est identifié par son numéro.

Supposons que les transferts d'argent d'un compte à un autre se produisent simultanément et soient donc implémentés en tant que processus.

Questions

1. Quels problèmes peuvent survenir si deux threads accèdent au même compte?
2. Comment éviteriez-vous ce problème?
3. Décrivez comment cette solution peut provoquer des blocages.
4. Décrivez une technique permettant d'éviter ces blocages.

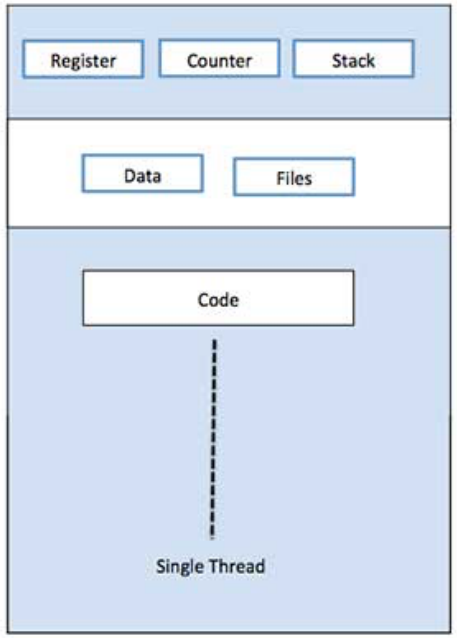
Processus

Un serveur de fichiers utilise la mise en cache et atteint un taux de réussite de 80%. Les opérations sur les fichiers dans le serveur coûtent 5 ms de temps sur le processeur lorsque le serveur trouve le bloc demandé dans le cache, et prennent 15 ms supplémentaires de temps d'E/S disque sinon.

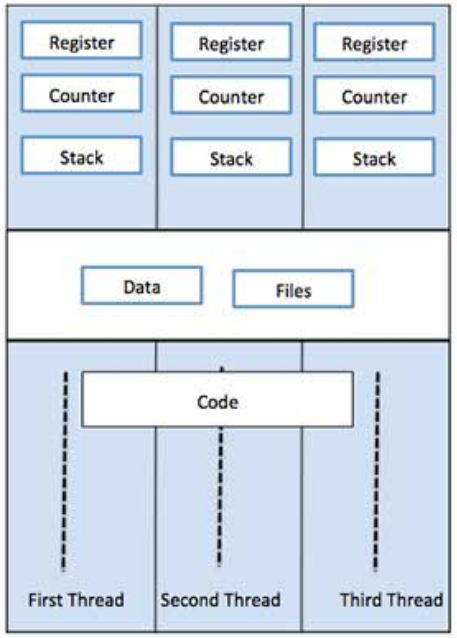
Questions

En expliquant toutes les hypothèses que vous faites, estimez la capacité de débit du serveur (demandes moyennes/seconde) si elle est:

1. simple filetage;
2. deux threads, fonctionnant sur un seul processeur;
3. deux threads, fonctionnant sur un ordinateur à deux processeurs.



Single Process P with single thread



Single Process P with three threads

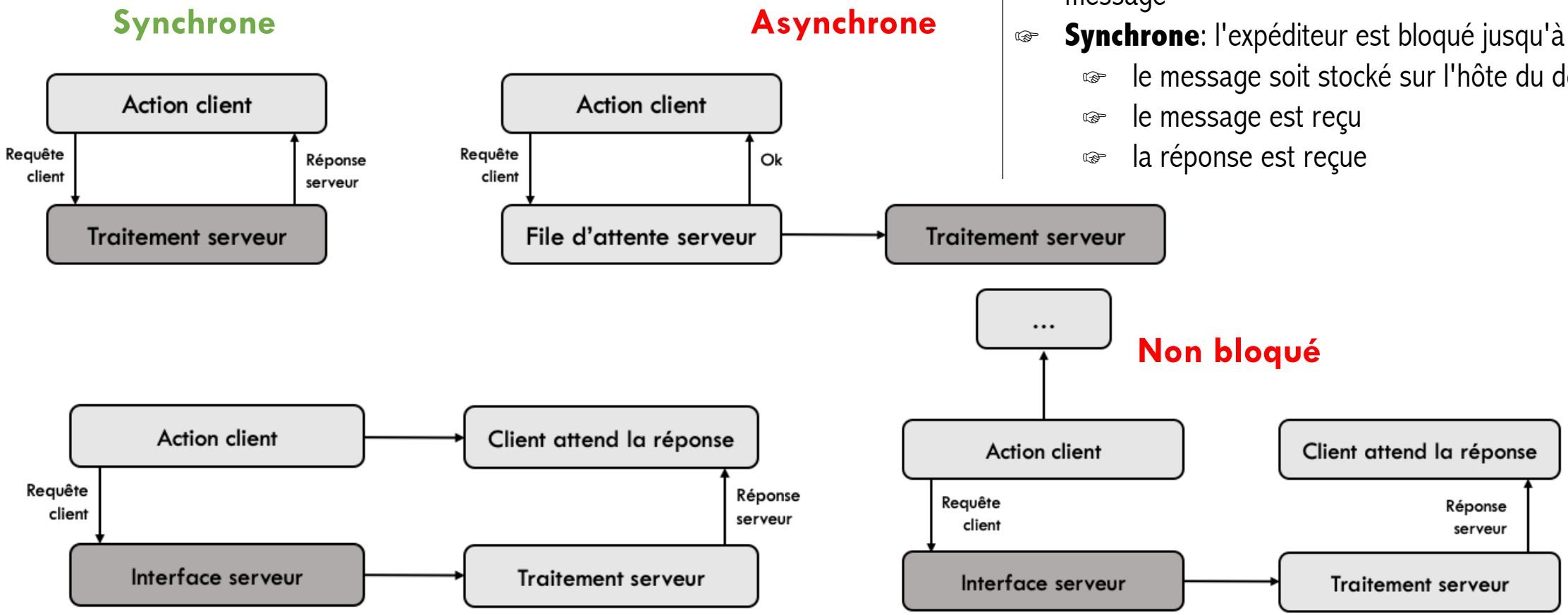
La principale différence entre un thread unique et les multi-threads (comme en Java) est que le thread unique exécute les tâches d'un processus tandis qu'en multi-thread, plusieurs threads exécutent les tâches d'un processus. Un processus est un programme en cours d'exécution. Lorsqu'il y a plusieurs threads dans un processus, cela s'appelle une application multi-thread.

Types de communication

- **Asynchrone**: l'expéditeur continue après l'envoi d'un message
- **Synchrone**: l'expéditeur est bloqué jusqu'à ce que
  - le message soit stocké sur l'hôte du destinataire
  - le message est reçu
  - la réponse est reçue

Types de communication

- **Asynchrone**: l'expéditeur continue après l'envoi d'un message
- **Synchrone**: l'expéditeur est bloqué jusqu'à ce que
  - le message soit stocké sur l'hôte du destinataire
  - le message est reçu
  - la réponse est reçue



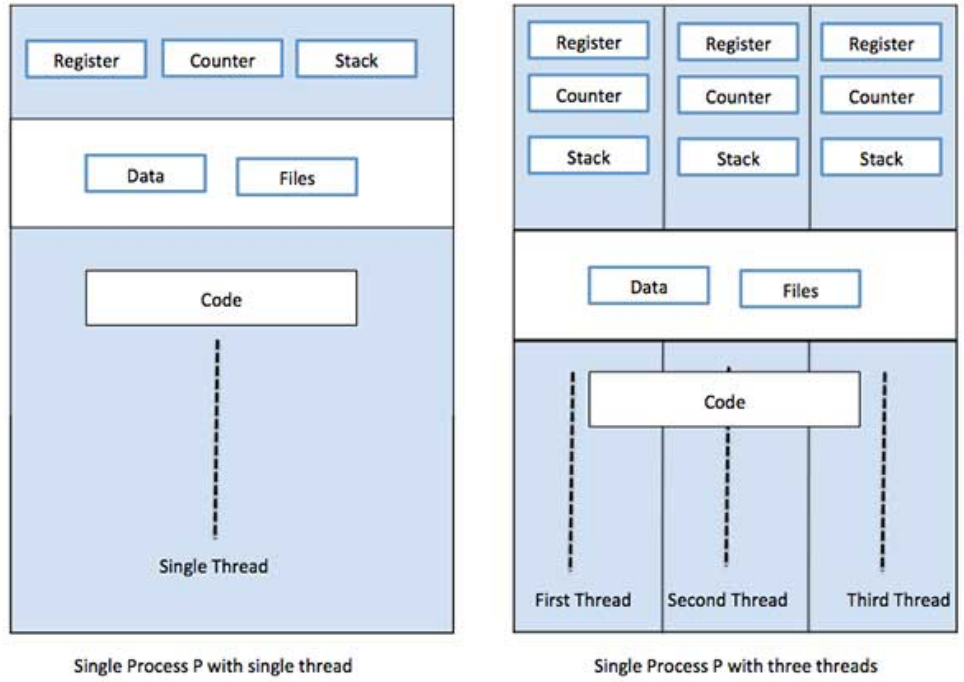
Threads

Un serveur de fichiers utilise la mise en cache et atteint un taux de réussite de 80%. Les opérations sur les fichiers dans le serveur coûtent 5 ms de temps sur le processeur lorsque le serveur trouve le bloc demandé dans le cache, et prennent 15 ms supplémentaires de temps d'E/S disque sinon.

Questions

En expliquant toutes les hypothèses que vous faites, estimez la capacité de débit du serveur (demandes moyennes/seconde) si elle est:

1. simple filetage;
2. deux threads, fonctionnant sur un seul processeur;
3. deux threads, fonctionnant sur un ordinateur à deux processeurs.



Single Process P with single thread

Single Process P with three threads

La principale différence entre un thread unique et les multi-threads (comme en Java) est que le thread unique exécute les tâches d'un processus tandis qu'en multi-thread, plusieurs threads exécutent les tâches d'un processus. Un processus est un programme en cours d'exécution. Lorsqu'il y a plusieurs threads dans un processus, cela s'appelle une application multi-thread.

Protocole requête/réponse (1)

Fondamentalement, toutes les communications client/serveur suivent le modèle d'un protocole « requête/réponse » :

- ☛ le client envoie la requête sous forme de message de demande,
- ☛ le serveur exécute l'opération demandée,
- ☛ le serveur répond avec un message de réponse.

Questions

1. Donnez un exemple de protocole requête/réponse dans le monde réel.
2. Discutez des avantages/inconvénients des types de communications synchrone et asynchrone pour un protocole de requête/réponse.
3. Dans quelles conditions la communication synchrone est-elle préférable? Et quand est-ce que la communication asynchrone est préférable?
4. Proposez un sketch de schéma de requête/réponse.

Protocole requête/réponse (2)

Imaginez un protocole requête-réponse où une requête est une invocation d'une méthode d'un objet distant. Supposons que les demandes aient le format suivant:

- ☛ messageType: 0 (0=requête 1=réponse)
- ☛ requestID int
- ☛ objectReference RemoteObjectRef
- ☛ methodId int
- ☛ arguments int t[32]

Questions

5. Dans quelles circonstances un **requestID** est-il nécessaire?
   (Astuce 1: N'oubliez pas que les messages sont envoyés via un réseaux
   Astuce 2: rappelez-vous la question 2 de la page précédente)
6. Quels éléments doit contenir un message de réponse?

Protocole requête/réponse (3)

Suite des questions

5. Quel protocole de transport serait le plus approprié pour implémenter un protocole de requête-réponse, UDP ou TCP?

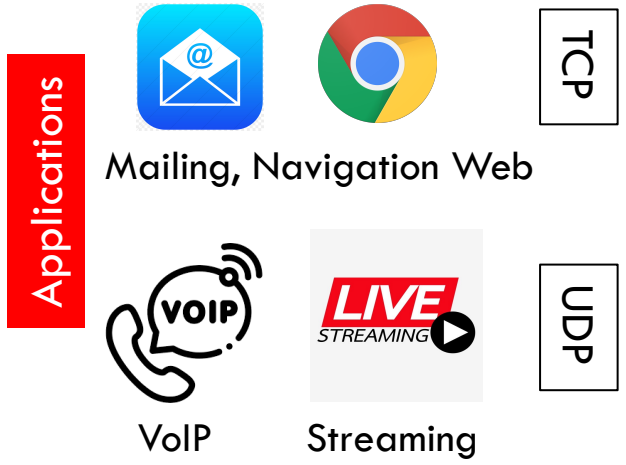
6. La réponse dépend-elle de circonstances supplémentaires?

7. Sur quel protocole au niveau du transport le protocole de requête-réponse que vous avez mentionné comme réponse à la question 1 est-il implémenté?

TCP vs UDP

TCP (Transmission Control Protocol) est orienté connexion, tandis que UDP (User Datagram Protocol) est sans connexion. TCP suit toutes les données envoyées, nécessitant un accusé de réception pour chaque octet (généralement).

UDP est efficace pour la transmission réseau de type diffusion et multidiffusion.
TCP est fiable car il garantit la livraison des données jusqu'à destination.
UDP est plus rapide, plus simple et plus efficace que TCP.



Protocole requête/réponse (4)

Suite des questions

Supposons qu'un protocole de demande-réponse soit implémenté à l'aide d'UDP.

8. Qu'est-ce qui peut changer par rapport à l'échange de messages?

Décrivez 3 problèmes possibles.

☞ **Astuces** (Pensez à la perte de messages et à ses conséquences.)

9. Pour chaque problème identifié, décrivez un raffinement (simple) du protocole de requête-réponse de base qui le surmonte.

Protocole requête/réponse (5)

Pour les protocoles de requête-réponse, on distingue trois types de sémantiques qui se distinguent par les garanties données par rapport à l'exécution du fonctionnement du serveur:

- ☞ **Peut-être**: le serveur peut exécuter la demande **une fois, plusieurs fois ou pas du tout**
- ☞ **Au moins une fois**: le serveur exécute la demande **au moins une fois**, mais **peut l'exécuter plusieurs fois** jusqu'à ce que le client reçoive une réponse
- ☞ **Au plus une fois**: pour chaque requête, le serveur exécute l'opération **au plus** une fois; l'application appelante reçoit le résultat ou une exception

Questions

10. Expliquer pour laquelle des trois sémantiques est-il nécessaire que le client transmette les demandes plus d'une fois?

11. Expliquer pour laquelle des trois sémantiques est-il nécessaire que le serveur se souvienne des requêtes auxquelles il a répondu?

12. Pour chacune des trois sémantiques, expliquer ce que le serveur doit faire lorsqu'il reçoit une demande.

Astuce: Vérifier si le serveur doit se souvenir de quelque chose et si oui, quoi.