Examen – NFP111
Systèmes et applications répartis

Session du 14 septembre 2007

Aucune notes personnelles ou documents ne sont autorisés.

1) La transparence dans les systèmes répartis

1.1) Qu'appelle-t-on transparence dans un système réparti ?

Dans un système réparti, la transparence permet au système d'être encapsulé. Ainsi, l'utilisateur profite des services sans se soucier de qui fait quoi, ni comment, sans savoir où se trouvent les ressources.

C'est ce qui permettra d'obtenir une image système unique.

1.2) Comment peut-être obtenue cette transparence ?

La transparence peut être obtenue de deux manières :

1. *La plus simple en cachant ce qui est fait et comment c'est fait aux utilisateurs. Par exemple des compilations multiples et en parallèle sur plusieurs ordinateurs. L'utilisateur ne lance qu'une commande de make.*
2. *Plus difficile est d'assurer la transparence au niveau du programmeur.*

1.3) Quels sont les aspects liés à la transparence ?

Plusieurs aspects sont liés à la transparence d'un système réparti :

- *La transparence du lieu : l'utilisateur ne peut pas dire où se trouve la ressource.*
- *La transparence de migration : Les ressources peuvent migrer d'un système vers un autre.*
- *La transparence de réplication (copie) : L'utilisateur ne peut pas dire combien de copies de la ressource existent.*
- *La transparence de concurrence : Plusieurs utilisateurs peuvent automatiquement partager une même ressource.*
- *La transparence du traitement parallèle : Les activités peuvent se dérouler en parallèle sans que l'utilisateur ne le sache.*

2) Les RPCs

2.1) Quels sont les problèmes liés aux passages de paramètres dans le cadre des RPCs ?

Le passage de paramètres dans le cadre des RPCS n'est pas quelque chose de simple. Facile dans le cas où les machines client et serveur sont identiques. Dans les systèmes répartis, les machines peuvent être différentes. Des problèmes de conversion peuvent apparaître :

- *EBCDIC, ASCII.*
- *Format des entiers (poids forts poids faibles).*
- *Etc.*

La problématique va donc être de connaître le format du message et la plateforme cliente :

- *Le premier octet (Byte) du message détermine le format utilisé par le client.*
- *Le serveur compare cet octet avec le sien : Si c'est le même aucune transformation n'est nécessaire, sinon il effectue la transformation.*

Le passage de pointeurs ou de tableaux de longueur variable est aussi un des problèmes majeurs à résoudre.

2.2) Comment gérer le passage de pointeurs ?
Le pointeur (adresse) n'est connu que dans l'espace d'adressage du processus qui le crée. Cette adresse n'est valide que dans la machine où s'exécute le processus. Il va falloir que la souche client qui récupère un pointeur, copie le contenu de la zone adressée dans le message. Au retour elle place le résultat dans la zone.

2.3) Comment gérer le passage de tableaux de longueur variable ?
Le problème est fréquent avec des tableaux dont les tailles ne sont pas connues au préalable. Une des solutions est de coder en dur les longueurs de paramètres. Dans l'hypothèse où la longueur du paramètre est inférieure à l'espace réservé, on perd de l'espace. Si la longueur est supérieure, on a une erreur.

2.4) Qu'appelle-t-on chemin critique dans le cadre d'un appel RPC ?
La séquence d'instructions exécutées pour chaque RPC est appelée chemin critique. C'est le temps nécessaire et rajouté par la RPC vs si le traitement était effectué en local.
L'étude de ce chemin critique permet d'évaluer la charge d'une RPC pour un système (ainsi il adaptera la configuration en fonction de la performance nécessaire).

Ce chemin critique démarre quand le client appelle la souche client et se termine lorsque le résultat est renvoyé par le serveur.

3) Les transactions

3.1) Qu'appelle-t-on « commit à deux phases » ?

Le commit à deux phases est un protocole qui a été conçu pour permettre à tous les participants d'abandonner une transaction.
Pour des raisons d'atomicité : si une partie de la transaction est abandonnée, la transaction complète sera abandonnée.

Les deux phases du protocole sont les suivantes :

- *1e Phase :Les participants votent pour que la transaction soit entérinée ou abandonnée. Lorsqu'un participant a voté pour son COMMIT, il ne peut plus revenir en arrière et ne pourra plus l'abandonner. (Tout doit être fait pour que ce soit le cas avec une sauvegarde sur mémoire stable).*
- *2e Phase : Tous les votes sont regardés, si tous sont COMMIT, la transaction est COMMIT, sin au moins un ABORT la transaction est ABORT.*

3.2) Qu'est-ce qu'une mémoire stable ?

Il existe trois catégories de mémoire :

1. *RAM (Random Access Memory) (par opposition, ROM = Read Only Memory), mémoires qui se désactivent à l'arrêt ou si la machine tombe en panne.*
2. *Les disques, qui peuvent perdre l'information si une tête s'écrase sur le support magnétique.*
3. *La mémoire stable, qui survivra à toutes les pannes même les tremblements de terre, les inondations ou autres calamités.*

Le stockage stable (mémoire stable), repose sur un ensemble de technologies :

- *L'utilisation de mémoire dynamique de type ECC, Chipkill, Miroir*
- *L'utilisation des disques avec des mécanismes de récupération d'erreurs (Checksum), de redondance (technologie RAID).*
- *Haute disponibilité des systèmes (HA).*
- *Disaster Recovery (DR).*

La tolérance aux pannes est ainsi presque parfaite.

4) Processeurs et processus

4.1) Quelles sont les différences essentielles entre les processus et les « threads » ?
Le processus *est l'unité de structuration d'architecture. Il a son espace d'allocation de ressources :*

- *Mémoire : Espace d'adressage (protégé).*
- *Ressources système : ports, sémaphores, canaux d'E/S, ...*
- *Threads (processus légers).*

La thread ou processus léger *est l'unité d'exécution.*
Son exécution a un caractère Séquentiel. Pas de parallélisme dans une thread.
Il peut y avoir plusieurs threads par processus.

4.2) Dans le cadre des processus légers, à quoi servent les techniques de gestion des zones critiques ?
Dans un système où un ensemble de processus cohabitent et accèdent aux mêmes ressources, les régions critiques sont essentielles. Lorsque des processus partagent les mêmes informations et que l'on veut s'assurer de l'exclusivité de l'accès, il peut utiliser l'exclusion mutuelle.

Vous avez 2h30.
Bon travail.

Corrigé examen – NFP111
Systèmes et applications répartis

Session du 9 mars 2007

Aucune notes personnelles ou documents ne sont autorisés.

1) Les concepts des systèmes répartis

Le matériel peut être organisé de différentes manières, Flynn a défini une taxinomie basée sur les instructions et les données. En se basant sur la taxinomie de Flynn :

1.1) Que signifient les acronymes suivants : SISD, SIMD, MISD, MIMD ? Pour chacun des acronymes donnez une brève description et un exemple l'illustrant.

1.2) Lequel des quatre acronymes peut se rapporter aux systèmes répartis ? Deux groupes distincts peuvent en dériver, les multi processeurs et les multi ordinateurs. À l'aide d'un graphe, présentez les différentes déclinaisons et caractéristiques de ces deux groupes. Vous en profiterez pour donner les définitions de faiblement et fortement couplé.

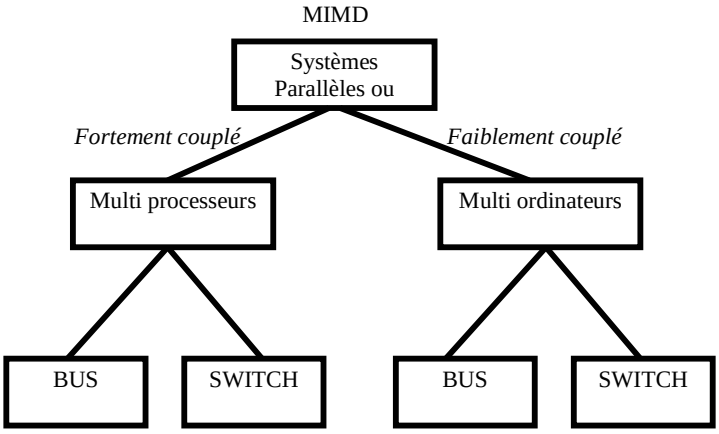
Ces concepts ont été définis par Flynn.

- ***SISD:*** *Single Instruction Single Data. (Mono processeur)*
- ***SIMD:*** *Single Instruction Multiple Data. (1 instruction et donnée traitées en parallèle).*
- ***MISD:*** *Multiple Instruction Single Data. (N'existe pas).*
- ***MIMD:*** *Multiple Instruction Multiple Data.*

D'un point de vue matériel, on suppose qu'on dispose de plusieurs processeurs ou plusieurs ordinateurs. On se place donc dans la classification de Flynn dans des architectures MIMD.

On peut distinguer dans ce contexte deux types d'architectures :

1. *Fortement couplée (tightly coupled) s'ils partagent de la mémoire vive.*
2. *Faiblement couplée (loosely coupled) s'ils ne partagent pas de mémoire vive.*



---

2) La synchronisation dans les systèmes répartis

2.1) Décrivez l'algorithme de Christian et l'algorithme de Berkeley pour la gestion des horloges physiques. Montrez les différences majeures entre les deux algorithmes. Expliquez pourquoi ces algorithmes (Christian et Berkeley) sont des algorithmes centralisés.
2.2) Quelle pourrait être une solution pour les rendre distribués.

Algorithme de Christian

Une machine reçoit le TUC, les autres devront rester synchroniser avec elle. Un temps de transaction de référence est établi : le temps de transaction, envoi-réponse, entre chaque machine et la machine référence. Il est préférable d'utiliser une approche statistique (valeur moyenne et sigma).
Les machines émettrices gardent trace des temps de référence et font des moyennes (corrigent si problème réseau).

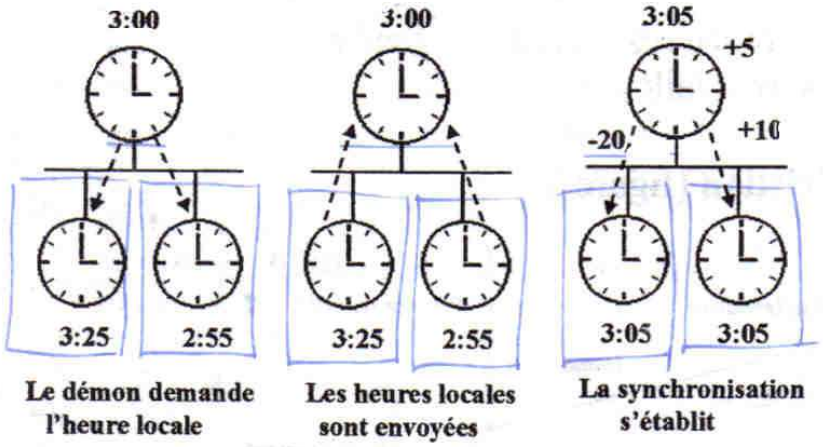
Régulièrement chaque machine envoie un message à la machine de référence et compare le temps de la transaction avec le temps de référence, si le temps dépasse la fenêtre admise (moyenne, +- sigma), la machine corrigera, progressivement de préférence, l'horloge logicielle.

Algorithme de Berkeley

Même principe que Christian, avec les différences suivantes :

- *Pas de TUC.*

- *Le serveur de référence appelle les autres machines (inverse de Christian), il utilise un daemon.*
- *Le daemon reçoit les réponses.*
- *Le daemon calcule un temps moyen, serveur compris, il ajuste le serveur si nécessaire et demande aux autres machines de s'ajuster en donnant les directives (+t1, t2, etc.).*



Solution pour rendre l'algorithme distribué
Christian et Berkeley sont des algorithmes centralisés. Il existe néanmoins des algorithmes répartis basés sur le fait que toutes les machines « broadcast » leurs temps régulièrement et chaque machine reçoit le temps de toutes les autres (ou une partie des autres), fait sa moyenne et calcule son temps.

---

3) Les transactions

3.1) Dans le cadre des transactions, qu'appelle-t-on concurrence optimiste ? À quoi sert cette approche ?

3.2) Quelles sont les trois phases de l'approche concurrence optimiste ? Quelles sont les règles à appliquer pour la phase de validation ?

Les protocoles de concurrence *cherchent à sérialiser les transactions dans leurs accès aux objets. Trois alternatives sont considérées :*

1. ***L'utilisation des verrous (lock)*** *: pour trier les transactions qui veulent accéder au même objet en se basant sur leur ordre d'arrivée (FIFO).*
2. ***La concurrence optimiste*** *: Elle permet aux transactions d'effectuer leur traitement jusqu'à ce qu'elles soient capables d'entériner leur traitement*

(COMMIT). Ensuite un contrôle est effectué pour voir si les opérations conflictuelles ont été effectuées.

3. ***L'utilisation des timestamps** : pour ordonner les transactions qui accèdent à un même objet en se basant sur leur temps de départ.*

La concurrence optimiste est un protocole de concurrence qui se base sur l'hypothèse que la plupart des applications clientes ont des chances faibles d'accéder aux mêmes objets.

L'approche se fait en trois phases :

1. ***La phase de traitement (Working phase)** :*
   - *Pendant la phase de traitement, les transactions ont une copie de la version entérinée (COMMIT) la plus récente de l'objet qu'elle veut modifier. Cette copie s'appelle **version tentative**.*
   - *Les demandes de lecture sont effectuées immédiatement sur la version entérinée (COMMIT) la plus récente de l'objet ou sur la version tentative de la transaction si elle existe.*
   - *Les écritures se font sur la version tentative de la transaction. Ainsi, chaque transaction a une copie de l'objet qu'elle veut modifier et les écritures peuvent se faire de manière concurrente sur le même objet. Dès lors plusieurs versions tentative de l'objet cohabitent.*
   - *De plus, un enregistrement est effectué pour chaque transaction, gardant trace de l'ensemble des objets lus et des objets écrits.*

   ***Note :** Comme les lectures se font sur des versions entérinées (COMMIT) ou des copies de ceux-ci, il ne peut pas y avoir de problème de lecture. Les problèmes de récupérations inconsistantes ne peuvent pas apparaître.*
2. ***La phase de validation (Validation phase)** :*
   - *La phase de validation démarre lorsque l'ordre de terminaison de la transaction (CloseTxn) est reçu.*
   - *Validation de la transaction par recherche des conflits.*
   - *Si validation OK alors COMMIT sinon ABORT.*
   - *L'ABORT conduit à la réémission de la transaction par le client.*
3. ***La phase de mise à jour (Update phase)** :*
   - *Si la transaction est validée, les versions tentatives sont entérinées et deviennent permanentes.*

La validation des transactions

Le mécanisme de validation utilise les règles conflictuelles de lecture écriture pour s'assurer que le déroulement d'une transaction est sériellement équivalent et n'impacte les transactions qui se déroulent en même temps.

Le tableau ci-dessous résume les règles à respecter pour la validation d'une transaction.

Tv	***Ti***	***Règles***
Ecriture	*Lecture*	***1)*** *Ti ne doit pas lire d'objets écrits par Tv.*
Lecture	*Ecriture*	***2)*** *Tv ne doit pas lire d'objets écrits par Ti.*
Ecriture	*Ecriture*	***3)*** *Ti ne doit pas écrire d'objets écrits par Tv.* *Tv ne doit pas écrire d'objets écrits par Ti.*

Vu que les phases de validation et de mise à jour sont courtes comparées à la phase de traitement. On peut décider qu'une seule transaction ne peut se trouver dans ses phases au même moment. Il suffit de déclarer ces deux phases dans une zone critique. Dans ce cas la règle 3 est toujours respectée.

---

4) Les systèmes de fichiers distribués

Dans le cadre de l'amélioration des performances d'un système de fichier réparti, l'implémentation d'un cache sur le client est essentielle. Afin de s'assurer de la cohérence des informations et des temps d'accès acceptables, quatre méthodes ou algorithmes de gestion de cache client peuvent être considérés.

4.1) Quels sont ces quatre algorithmes ?

4.2) Pour chacun d'eux vous donnerez une description, ses avantages et ses inconvénients.

Le cache client introduit de l'inconsistance et peut aboutir à rendre le système incohérent. Si deux clients lisent simultanément le même fichier et si les deux le modifient, plusieurs problèmes peuvent survenir.

La gestion de la cohérence de cache client peut être effectuée selon quatre méthodes ou algorithmes :

1. *Ecriture immédiate (**Write through**).*
2. *Ecriture différée (Delayed write ou **write back**).*
3. *Ecriture à la fermeture (Write on close).*

4. *Contrôle centralisé (Centralized control).*

Ecriture immédiate

Dès qu'une entrée dans le cache est modifiée (fichier ou bloc), la nouvelle valeur est gardée dans le cache mais renvoyée ***immédiatement*** *au serveur.*
Un autre client qui lit le fichier voit ainsi la valeur la plus récente.

Des problèmes subsistent (un exemple) :

- *C1 lit le fichier FIC1.*
- *C1 termine, mais FIC1 reste dans le cache.*
- *C2 lit FIC1.*
- *C2 modifie FIC1 et l'écrit sur le serveur.*
- *C1 lit le fichier FIC1, il l'a dans le cache. Il ne va donc pas le chercher sur le serveur.*
- ***La version de FIC1 est obsolète pour C1.***

Une des façons de régler le problème et de demander au serveur un contrôle sur les dates (ou la version, ou checksum) avant d'utiliser un fichier.
Note : *même si la taille de l'information échangée est faible, cela demande une RPC et allonge le temps d'accès.*

Ecriture différée

Au lieu d'envoyer les informations immédiatement après la modification, le client note les modifications et toues les x secondes (30 secondes par exemple), envoie en une fois toutes les modifications au serveur.

Cette méthode est très efficace si l'on a : Lecture – Modification – Destruction d'un fichier avant l'envoi au serveur. En effet, le serveur n'aura dans ce cas rien à faire.

Ecriture à la fermeture

L'écriture ne se fait qu'à la fermeture du fichier.
On ne s'affranchit pas du problème du dernier qui écrit.
En fait ce n'est pas pire que dans un système mono processeur lorsque deux processus écrivent en même temps.

Contrôle centralisé

C'est une approche totalement différente qui utilise un algorithme centralisé dont le fonctionnement est le suivant :

- *Les ouvertures de fichier sont demandées au serveur de fichiers.*
- *Le serveur de fichier tient une table à jour en répertoriant qui utilise quoi.*
- *Toutes les demandes de lecture sont acceptées.*
- *Les demandes d'écriture sont rejetées et mises dans une file d'attente.*
- *Le serveur peut envoyer un message pour dire au client d'invalider un fichier de leur cache.*

Avec cet algorithme, plusieurs écritures et lectures peuvent être effectuées concurremment avec des résultats ni meilleurs ni pires que pour un système mono-processeur.

Méthode	
Ecriture immédiate	*Fonctionne mais sans amélioration pour les écritures.*
Ecriture différée	*Meilleure performance mais des possibles ambiguïtés sémantiques.*
Ecriture à la fermeture	*Respecte la sémantique session.*
Contrôle centralisé	*Respecte la sémantique UNIX, mais n'est pas robuste et a une mauvaise scalabilité.*

Vous avez 2h30.
Bon travail.

Corrigé examen – NFP111
Systèmes et applications répartis

Session du 15 septembre 2006

Aucune notes personnelles ou documents ne sont autorisés.

1) Les contraintes pour la réalisation des systèmes répartis

1.1) Quelles sont les cinq contraintes majeures d'un système réparti.

1. La transparence (accès concurrent, migration, parallélisme).
2. La flexibilité.
3. La fiabilité et la disponibilité.
4. La performance.
5. L'évolutivité (scalabilité).

***La transparence**, c'est masquer la répartition au travers des spécifications suivantes :*

- *La localisation des ressources n'est pas perceptible.*
- *La migration des ressources d'un lieu à un autre n'implique aucune modification de l'environnement utilisateur.*
- *Le nombre de ressources dupliquées n'est pas connu (invisibilité).*
- *La concurrence d'accès aux ressources n'est pas perceptible.*
- *Invisibilité du parallélisme sous-jacent offert par l'ensemble de l'environnement d'exécution.*

***La flexibilité** ou modularité. C'est faire en sorte que le système soit le plus modulaire possible. Pour ce faire, les systèmes répartis sont construits par l'ajout de services reposant sur un micro-noyau. Si de nouveaux services doivent être implémentés, ils sont ajoutés au système.*

***La performance** d'un système réparti c'est sa faculté à gérer un grand nombre :*

- *D'utilisateurs.*
- *De machines.*
- *De services.*
- *De trafic.*
- *De processus.*

Les métriques de la performance sont :

- *Le temps de réponse.*
- *Le débit.*

- *L'utilisation du système.*
- *La capacité réseau utilisée.*

L'évolutivité (scalability) *c'est la faculté qu'a un système à rester efficace et performant malgré la croissance du nombre d'utilisateurs et du nombre de ressources.*

1.2) Qu'appelle-t-on Fiabilité. Quelles sont les différents aspects que doit prendre en compte un système réparti pour être fiable.

La fiabilité et la disponibilité *d'un système réparti repose essentiellement sur l'indépendance des composants du système et la redondance des composants logiciels et matériels. Pour qu'un système soit fiable et disponible, il faut qu'il soit :*

- *Tolérant aux pannes.*
- *D'un fonctionnement sûr.*
- *Sécurisé.*

1.3) Quels sont les trois modes de panne qui existent dans un système réparti.
Les trois modes de panne d'un système réparti sont :

1. ***Normal*** *: répond aux spécifications.*
2. ***Dégradé*** *: répond aux spécifications mais pertes de fonctionnalités.*
3. ***Panne*** *: ne répond plus aux spécifications.*

2) RPC (communications entre le client et le serveur)

2.1) Que se passe-t-il lorsqu'un serveur tombe en panne alors qu'un client a demandé un traitement ? Quels sont les deux cas à considérer côté serveur ? Quelles sont les trois solutions (écoles) que peut suivre un client ?

1. *Le client ne peut pas localiser le serveur*
   - *Renvoi d'un message « Cannot locate server »*
2. *L'appel du client n'arrive pas au serveur*
   - *Rajout d'un compteur de temps dans le noyau. Après un certain délai le message est renvoyé.*
   - *Après un certain nombre d'essais, renvoi d'un message « Cannot locate server »*
3. *La réponse du serveur n'arrive pas au client*
   - *Une information est rajoutée dans le message qui détermine quelle réponse a été perdue (N° de séquence ou heure de départ).*

Le serveur est en panne

- *Il faut dissocier la panne avant l'exécution de la requête ou après l'exécution.*
- *Trois écoles pour résoudre ce problème :*
  1. *Attendre que le serveur redémarre et relancer le traitement.*
  2. *Abandon du client et rapport de l'erreur.*
  3. *Ne rien dire au client (aucune garantie)*
- *En règle générale, crash du serveur $\Rightarrow$ crash du client.*

2.2) Que se passe-t-il lorsqu'un échange devient orphelin : c'est-à-dire lorsqu'un client tombe en panne alors qu'il a demandé un traitement à un serveur ? Quelles sont les quatre solutions qui peuvent être mises en place par le client pour pallier ce problème ?
Si le client envoie une requête et s'il tombe en panne avant la réponse, l'échange est dit ***orphelin.*** *Cela a pour effet de gaspiller du temps CPU, de bloquer des ressources, lorsqu'il redémarre il peut recevoir des messages antérieurs à la panne (problème de confusion).*

- *Quatre solutions peuvent être envisagées :*
  1. *L'extermination : Le client enregistre les appels dans une log. Au redémarrage l'orphelin est détruit.*
  2. *La réincarnation : Le client tient compte du temps. Lorsqu'il redémarre, il annule les échanges ente deux époques et reprend les échanges orphelins.*
  3. *La réincarnation à l'amiable : Comme précédemment, avec en plus la vérification avec le serveur concerné des échanges orphelins.*
  4. *Expiration du délai : Un compteur de temps détermine quand un échange devient orphelin. Les orphelins sont détruits.*

3) La synchronisation dans les sytèmes répartis

3.1) Qu'appelle-t-on synchronisation dans les systèmes répartis ?
On appelle synchronisation dans les systèmes répartis, tout mécanisme qui permet de pallier l'absence de mémoire partagée et qui permet aux différents processus distribués de se synchroniser.

3.2) Pourquoi peut-il être nécessaire de synchroniser les horloges des différents ordinateurs dans un système réparti ?
La synchronisation va permettre de pallier l'absence de mémoire partagée. En effet, au sein d'uns système réparti les processus voulant coopérer et avoir accès aux mêmes ressources ne peuvent pas utiliser les mécanismes de régions critiques, sémaphores, exclusion mutuelle, utilisés dans les systèmes centralisés.

3.3) Quelle est la différence entre horloge physique et horloge logique ?
Tous les ordinateurs ont une horloge physique qui est en fait un oscillateur au quartz. Un compteur logique et un registre particulier sont associés à cette horloge physique. Chaque oscillation diminue le compteur de 1 lorsqu'il est à 0 le compteur génère une interruption, récupère le contenu du registre et recommence. À chaque interruption, l'horloge logicielle est mise à jour. Aussi, aucune structure répartie ne pourra maintenir des horloges physiques synchronisées. En effet, les faibles variations des oscillateurs sont répercutées et très vite les machines sont déphasées. Il a donc été nécessaire de pallier cette problématique, c'est le but des ***horloges logiques****.*

3.4) Décrivez l'algorithme de Lamport pour la gestion des horloges logiques.
L'horloge logique se base sur la suite d'évènements où l'un arrive avant un autre avec une relation « est arrivé avant » symbolisée par « → ».
Ex : Soit deux événements A et B d'un processus. Si A se passe avant B ; A peut être l'envoi d'un message par un processus et B le message reçu, le message ne peut pas être reçu avant l'envoi donc $\boldsymbol{A \rightarrow B}$.

***Lamport** associe aux évènements une notion de temps C() qui implique que pour **A → B** nous avons C(A) < C(B). C représente le temps de l'horloge, C croît donc toujours.*
L'algorithme de Lamport permet ainsi de synchroniser les horloges des différentes machines du système réparti.

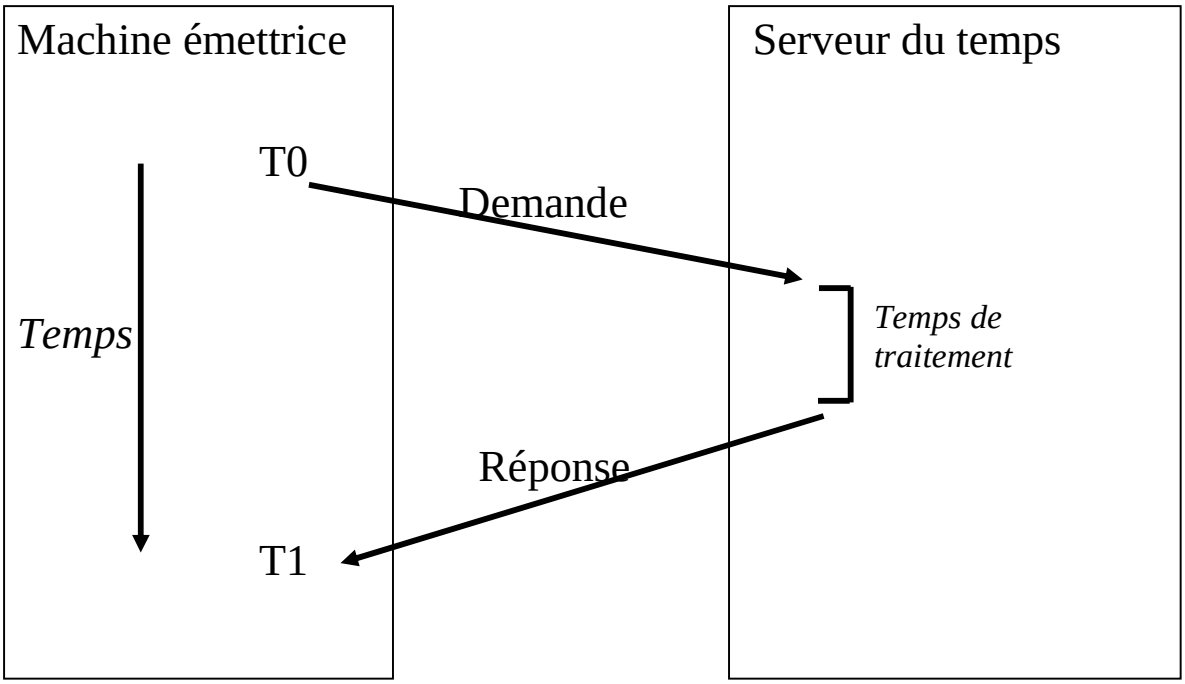
Les règles de l'algorithme sont les suivantes :
- *Si A est arrivé avant B alors C(A) < C(B)*
- *Si A et B représentent l'envoi d'un message et la réception d'un message alors C(A) < C(B), si ce n'est pas le cas réellement pour les horloges alors l'algorithme ajustera le temps par C(B) = C(A)+1 et modifiera aussi l'horloge.*
- *Pour tous les évènements A et B, C(A) est différent de C(B).*

Il est à noter que si deux processus n'interagissent pas entre eux, la synchronisation de leurs horloges n'est pas utile.
Ce qui importe dans le cas de l'algorithme de Lamport n'est pas la cohésion du temps mais l'ordre.

3.4) Décrivez l'algorithme de Christian pour la gestion des horloges physiques.
Dans les systèmes « à temps réel » le temps précis (absolu) est nécessaire. Il faut donc aller au-delà de l'algorithme de Lamport pour synchroniser le système.
Les mécanismes de synchronisation physiques reposent sur un modèle général.
Chaque machine a son horloge qui envoie régulièrement les interruptions enregistrées par l'horloge logicielle, avec une origine qui est régulièrement comparée à un temps universel. Si la différence reste dans une fenêtre admise,rien ne se passe, sinon les horloges logicielles sont ajustées.

*L'**algorithme de Christian** est basé sur ce principe.*

T0 et T1 sont mesurés avec la même horloge, plusieurs mesures et une approche statistique donneront une meilleure évaluation de T1.



- *Une machine reçoit le TUC (Temps Universel Coordonné), les autres devront rester synchronisées avec elle.*
- *Un temps de transaction de référence est établi : le temps de transaction, envoi-réponse, entre chaque machine et la machine référence. Il est préférable d'utiliser une approche statistique (valeur moyenne et sigma).*
- *Les machines émettrices gardent trace des temps de référence et font des moyennes (corrigent si problème de réseau).*
- *Régulièrement chaque machine envoie un message à la machine de référence et compare le temps de la transaction avec le temps de référence. Si le temps dépasse la fenêtre admise (moyenne ± sigma), la machine corrigera progressivement, de préférence, l'horloge logicielle.*

Vous avez 2h30.
Bon travail.

Corrigé examen – NFP111
Systèmes et applications répartis

Session du 7 juillet 2006

Aucune notes personnelles ou documents ne sont autorisés.

1) Les concepts des systèmes répartis

1.1) Qu'appelle-t-on système réparti ?

Un système réparti est un ensemble d'ordinateurs indépendants reliés en réseau et donnant l'impression à l'utilisateur d'un système monoprocesseur.

1.2) Quelles sont les caractéristiques techniques d'un système réparti ?

Les caractéristiques techniques d'un système réparti sont les suivantes :

- *Une gestion unique de la communication interprocessus.*
- *Un système de fichier unique.*
- *Un noyau ou un micro-noyau identique sur tous les postes (nœuds).*

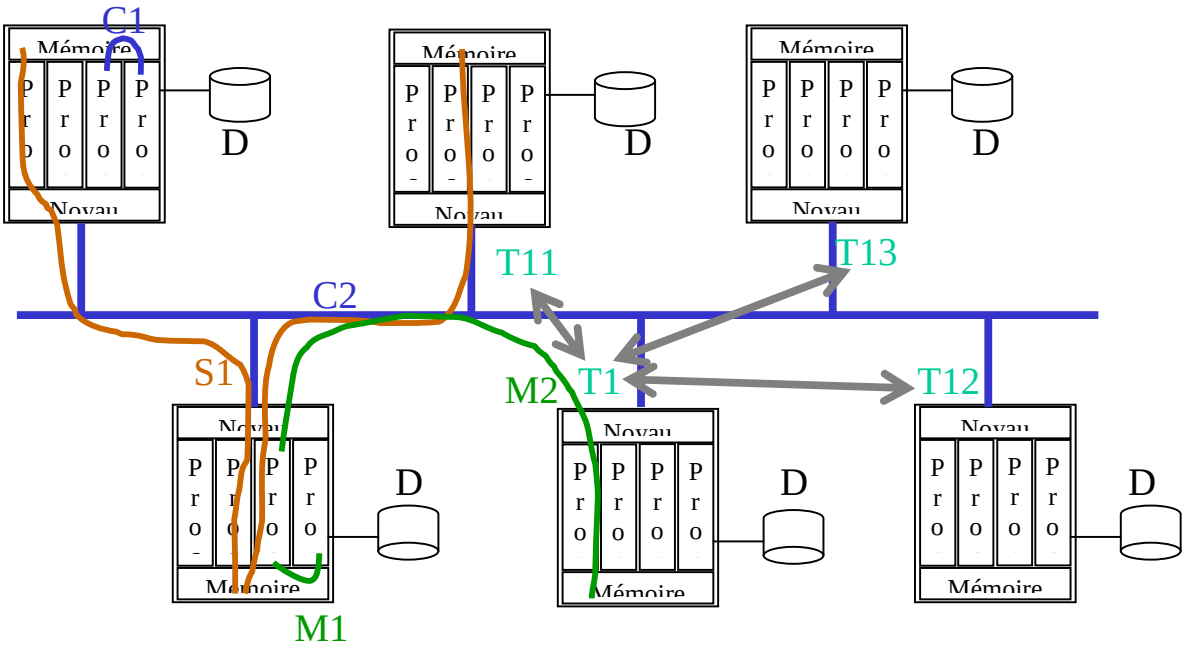
1.3) Décrivez le fonctionnement d'un système réparti (vous pouvez vous aider d'un graphique si nécessaire).

Un système réparti s'appuie et est basé sur les contraintes suivantes :

- *La flexibilité.*
- *La fiabilité.*
- *La performance.*
- *L'évolutivité.*

Le graphique suivant illustre une modélisation d'un système réparti et ses principaux composants. Le fonctionnement peut être décrit au travers des sujets suivants :

- ***C1 & C2** : Communication interprocessus.*
- ***D** : Stockage des données.*
- ***M1 & M2** : mémoire partagée et répartie.*
- ***T1** : Transactions et transactions imbriquées.*
- ***S1** : Synchronisation..*



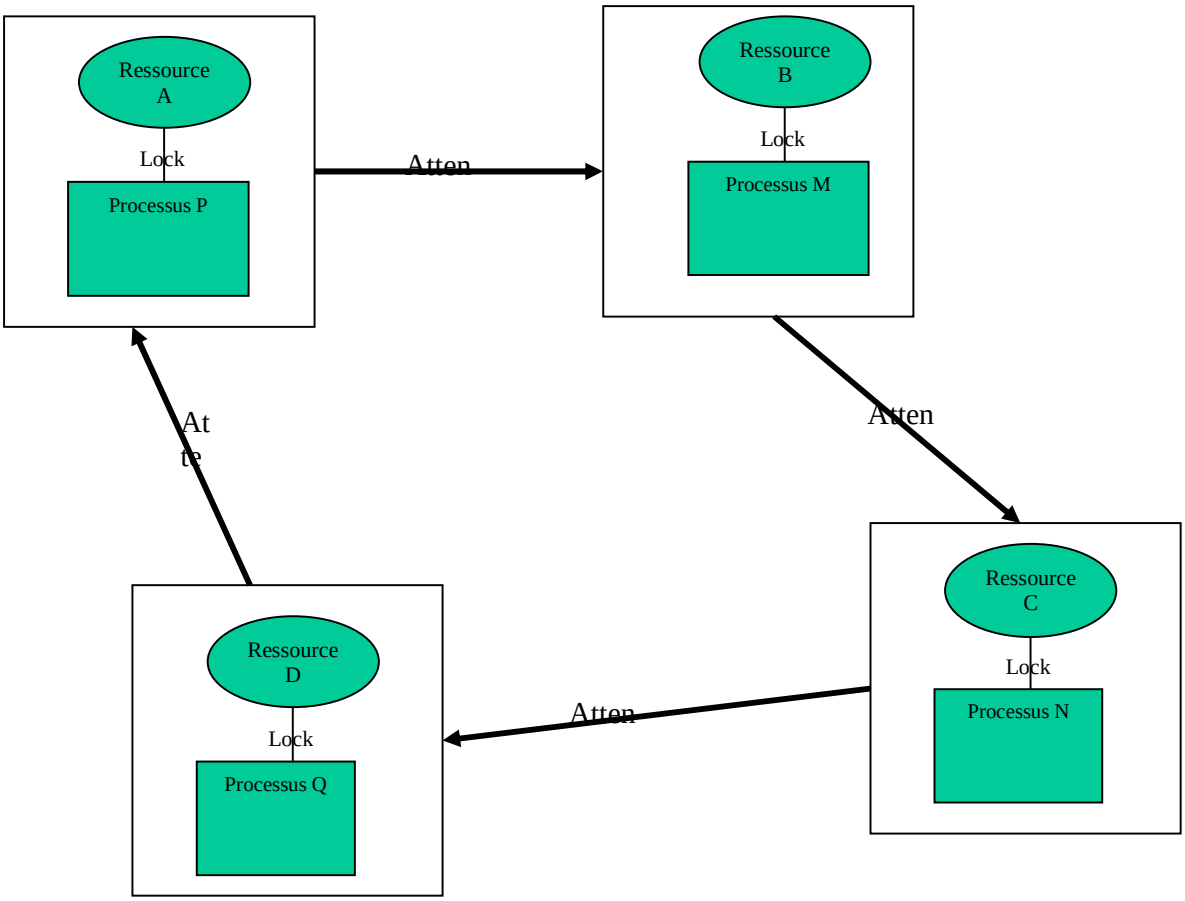
2) Les transactions réparties

2.1) Décrivez un deadlock (étreinte fatale) de transactions dans un système réparti.

Repérer un deadlock ou etreinte fatale dans un système réparti est assez délicat. Un deadlock survient lorsque plusieurs processus font appel à des ressources partagées et que chacun ayant posé des locks sur les ressources est en attente de la libération de ressources lockées par un autre processus.

Dans l'exemple ci-dessous,

- *Le processus P attend la ressource B détenue par le processus M.*
- *Le processus M attend la ressource C détenue par le processus N.*
- *Le processus N attend la ressource D détenue par le processus Q.*
- *Le processus Q attend la ressource A détenue par le processus P.*



2.2) Décrivez le mécanisme de sonde pour la résolution des deadlocks dans un système réparti.

La résolution des deadlocks dans un système réparti implique que l'on a accès aux informations réparties sur l'utilisation des ressources. Ces informations peuvent être gérées de manière centralisée ou répartie. L'algorithme du mécanisme de sonde est un algorithme réparti puisque chaque intervenant gère le deadlock.

Description du mécanisme de sonde :

Ce mécanisme se découpe en trois étapes :

1. ***L'initialisation (initiation) :*** *Un serveur s'aperçoit qu'un processus P attend un processus M et M attend pour accéder à un objet situé sur un autre serveur. Le serveur envoie une* ***sonde <P→M>*** *au serveur où M est bloqué.*
2. ***Détection :*** *Le serveur qui reçoit une sonde doit vérifier s'il y a un deadlock (cycle dans la sonde).*

- *Si tel n'est pas le cas, il rajoute son processus N à la sonde et envoie la sonde sur le serveur qui bloque son processus N :* ***<P→M→N>.***
- *Sinon, si un cycle est détecté :* ***<P→M→N→P>*** *alors passage à l'étape 3.*

3. ***Résolution :*** *Lorsqu'un deadlock est détecté, un processus notifié dans la sonde est abandonné (les règles d'abandon de processus dépendent de l'algorithme ou approche mis en place)..*

2.3) Quelles sont les six règles de COMMIT dans les transactions imbriquées ?
Les six règles sont les suivantes :

1. *Une transaction (TXN) peut effectuer un COMMIT ou un ABORT uniquement après que l'exécution de ses TXN-Filles soit terminée.*
2. *Lorsqu'une TXN-Fille est terminée, elle peut indépendamment prendre la décision d'effectuer un COMMIT (provisionnel) ou un ABORT (définitif).*
3. *Si une TXN-Parent effectue un ABORT toutes les TXN-Filles effectuent un ABORT.*
4. *Si une TXN-Fille effectue un ABORT, la TXN-Parent a le choix d'effectuer un COMMIT ou un ABORT.*
5. *Si la TXN de plus haut niveau (root) effectue un COMMIT toutes les TXN-Filles qui avaient effectué un COMMIT provisionnel effectue un COMMIT définitif.*
6. *Si la TXN de plus haut niveau (root) effectue un ABORT toutes les TXN-Filles effectuent un ABORT.*

3) Les mémoires partagées et réparties

3.1) Qu'appelle-t-on mémoire partagée ? Dans quels systèmes peut-elle être rencontrée ?
La mémoire partagée est un espace situé dans la mémoire centrale d'un ordinateur et qui peut être partagée par plusieurs processus.
La mémoire partagée se trouve dans les systèmes centralisés et les systèmes répartis.

3.2) Qu'appelle-t-on modèle de cohérence dans un système à mémoire répartie et partagée ?
Un modèle de cohérence est essentiellement un contrat entre le logiciel et la mémoire. Le contrat impose au logiciel certaines règles d'utilisation et d'accès à la mémoire et garantit en retour, dans l'hypothèse où les règles ne sont pas violées par le logiciel, que la mémoire se comportera bien.

La meilleure utilisation de la mémoire se fait avec les modèles les moins permissifs. Au plus les modèles sont permissifs et au plus les risques sont grands.

3.3) Qu'appelle-t-on modèle de cohérence stricte ?
*La règle du modèle de cohérence stricte est : « **Toute lecture d'une position mémoire x, renvoie la valeur stockée par l'opération d'écriture x la plus récente**. »*
Cette définition est naturelle et évidente, par contre elle implique l'existence dans le système d'un temps global absolu. Cela afin que la notion de plus récente ne puisse pas être ambiguë.

4) Les processus

4.1) Donnez les endroits où peut être implémentée la librairie de gestion des threads ?
La librairie de gestion des threads peut être implémentée :

1. *Dans l'**espace Utilisateur** : le noyau n'a aucune connaissance de cette gestion.*
2. *Dans le **Noyau** : aucun « runtime » n'est nécessaire. Le noyau gère une table des processus et threads associées.*
3. *De manière **hybride** : Une implémentation hybride peut-être envisagée, elle tirera avantage des bonnes performances de la première solution (espace utilisateur) et de la facilité d'utilisation de la seconde (noyau). On peut se référer l'approche décrite par Anderson & Al (1991).*

4.2) Décrivez l'implémentation de la librairie de gestion des threads dans l'espace utilisateur.
*Dans l'implémentation de la librairie de threads dans le noyau**, aucun « runtime n'est nécessaire.***
*Quand **une thread veut créer une nouvelle thread ou en détruire une** existante, un **appel au noyau** est effectué et **le noyau fera la création ou la destruction**.*
*Pour gérer toutes les **threads le noyau a une table par processus. Cette table contient une entrée par thread**. Cette entrée permet de stocker les informations relatives à la thread (registres, état, priorité, etc.).*
*Lorsqu'une **thread est bloquée** (synchro ou autres apples bloquants), **le noyau peut lancer une autre thread du processus**.*
***La gestion des threads par le noyau à un coût important (impact des performances)**. Pour minimiser ce coût il est possible pour le noyau de conserver l'environnement d'une thread après sa destruction.*

Vous avez 2h30.
Bon travail.

Corrigé Examen - UV19302
Systèmes et applications répartis – B4

Session du 16 septembre 2005

Aucune notes personnelles ou documents ne sont autorisés.

1) Les concepts des systèmes répartis

1.1) Qu'appelle-t-on système réparti ?

Un système réparti est un ensemble d'ordinateurs indépendants reliés à un réseau et donnant l'impression à l'utilisateur d'un système monoprocesseur.

1.2) Quels sont les avantages et les désavantages d'un système réparti par rapport à un système centralisé ?

Les avantages d'un système réparti sont les suivants :

- *Il est plus économique d'utiliser un ensemble de petites machines (PCs ou stations) que de travailler sur un grand système.*
- *Un ensemble de petites machines reliées en réseau peut avoir une puissance supérieure que celle d'un système centralisé.*
- *La tolérance aux pannes est plus importante dans un système distribué, une machine tombant en panne peut facilement être remplacée et toutes les autres continuent à fonctionner.*
- *L'évolutivité du système (scalability) se fait de manière simple par l'ajout de nouvelles machines sur le réseau.*
- *Certaines applications nécessitent de part l'un architecture d'être hébergées par des machines physiquement différentes.*

Les désavantages d'un système réparti sont les suivants :

- *Peu de modèles de fonctionnement. Quelles sont les applications susceptibles d'être hébergées par des systèmes répartis.*
- *Quels sont les systèmes d'exploitation, compilateurs, applications, les plus adéquats.*
- *Comment présenter la distribution à l'utilisateur.*
- *Le réseau susceptible de perdre des messages oblige à la mise en place de logiciels de contrôle supplémentaires.*

1.3) Quels sont les désavantages d'un système réparti ?

Logiciel	*Peu de logiciels et d'applications existent aujourd'hui.*
Réseau	*Le réseau peut être facilement saturé ou dû à sa latence conduisent à des problèmes de temps de réponse. ?*
Sécurité	*Les accès à l'information sont multiples donc difficiles à contrôler.*

1.4) Quelles sont les différences entre serveurs fortement couplés et serveurs faiblement couplés ?

Un matériel fortement couplé est un matériel où les composants le constituant sont physiquement proches et reliés par des moyens de communication très rapides (par exemple deux processeurs sur un même circuit). À l'inverse les composants d'un matériel faiblement couplé ne sont pas rapprochés et les moyens de communication sont lents (par exemple deux machines dans un réseau).

2) Communication dans les systèmes répartis

2.1) Qu'appelle-t-on un groupe dans les systèmes répartis ?

Le mécanisme de RPC met en jeu deux parties (le client et le serveur). Parfois les circonstances font que la communication nécessite l'implication de plusieurs processus (pas uniquement deux). Par exemple un ensemble de serveurs qui offrent des services de redondance. Dans ce cas, le client va devoir envoyer un message à plusieurs serveurs à la fois ce que ne sait pas faire le RPC. Un des moyens d'y parvenir est de mettre en place un mécanisme de communication de groupe.
Un groupe est un ensemble de processus formant une entité unique. Un message envoyé à ce groupe sera reçu par tous les membres du groupe au même moment.

2.2) Quelle est la différence entre groupe ouvert et groupe fermé ?

***Un groupe fermé** est un ensemble de processus formant une entité unique qui ne peut recevoir de messages de l'extérieur. Les échanges de message ne s'effectuent qu'à l'intérieur du groupe.*
*Un **groupe ouvert** est un ensemble de processus formant une entité unique qui peut recevoir des messages de l'extérieur.*

2.3) Quelle est la différence entre groupe hiérarchique et groupe de pairs ?

***Un groupe hiérarchique** est composé d'un processus coordinateur qui reçoit l'ensemble des messages et les distribue aux processus travailleurs. Si le coordinateur disparaît, les travailleurs éliront l'un d'entre eux comme coordinateur.*
***Un groupe de pairs** est composé de processus reliés entre eux. Tous reçoivent les messages et des votes sont mis en place pour décider qui fait quoi. Le vote rajoute un temps de latence qui diminue les performances.*

3) Synchronisation dans les systèmes répartis

3.1) À quoi sert la synchronisation ?

La synchronisation va permettre de pallier l'absence de mémoire partagée. En effet, au sein d'uns système réparti les processus voulant coopérer et avoir accès aux mêmes ressources ne peuvent pas utiliser les mécanismes de régions critiques, sémaphores, exclusion mutuelle, utilisés dans les systèmes centralisés.

3.2) En quoi les algorithmes distribués complexifient la synchronisation dans un système réparti ?

La complexité est essentiellement liée à l'utilisation d'algorithmes répartis. Il n'est pas en l'état actuel des technologies d'avoir toutes les informations d'un système réparti stocké dans un endroit unique. Les prises de décision sont donc de fait très difficiles.
L'information étant répartie sur un ensemble de machines, les processus vont devoir prendre des décisions en s'appuyant sur des informations locales. De plus, il n'existe aucune horloge commune.

3.3) Décrivez l'algorithme de Lamport qui permet la synchronisation d'horloges logiques.
Tous les ordinateurs ont une horloge physique qui est en fait un oscillateur au quartz. Un compteur logique et un registre particulier sont associés à cette horloge physique. Chaque oscillation diminue le compteur de 1 lorsqu'il est à 0 le compteur génère une interruption, récupère le contenu du registre et recommence. À chaque interruption, l'horloge logicielle est mise à jour. Aussi, aucune structure répartie ne pourra maintenir des horloges physiques synchronisées. En effet, les faibles variations des oscillateurs sont répercutées et très vite les machines sont déphasées. Il a donc été nécessaire de pallier cette problématique, c'est le but des ***horloges logiques****.*

L'algorithme de Lamport
L'horloge logique se base sur la suite d'évènements où l'un arrive avant un autre avec une relation « est arrivé avant » symbolisée par « → ».
Ex : Soit deux événements A et B d'un processus. Si A se passe avant B ; A peut être l'envoi d'un message par un processus et B le message reçu, le message ne peut pas être reçu avant l'envoi donc $A \rightarrow B$.
Lamport *associe aux évènements une notion de temps C() qui implique que pour* $A \rightarrow B$ *nous avons C(A) < C(B). C représente le temps de l'horloge, C croît donc toujours.*
L'algorithme de Lamport permet ainsi de synchroniser les horloges des différentes machines du système réparti.

Les règles de l'algorithme sont les suivantes :
- *Si A est arrivé avant B alors C(A) < C(B)*
- *Si A et B représentent l'envoi d'un message et la réception d'un message alors C(A) < C(B), si ce n'est pas le cas réellement pour les horloges alors l'algorithme ajustera le temps par C(B) = C(A)+1 et modifiera aussi l'horloge.*
- *Pour tous les évènements A et B, C(A) est différent de C(B).*

Il est à noter que si deux processus n'interagissent pas entre eux, la synchronisation de leurs horloges n'est pas utile.
Ce qui importe dans le cas de l'algorithme de Lamport n'est pas la cohésion du temps mais l'ordre.

Vous avez 2h30.

Bon travail.

Corrigé Examen - UV19302
Systèmes et applications répartis – B4

Session du 1 juillet 2005

Aucune notes personnelles ou documents ne sont autorisés.

1) L'exclusion mutuelle

Lorsqu'un processus doit modifier des structures de données partagées, il entre d'abord en zone critique en s'assurant qu'aucun processus n'est en train d'utiliser la structure de données à laquelle il veut accéder. Ces zones critiques sont protégées à l'aide de mécanismes particuliers. Trois mécanismes d'exclusion mutuelle sont fréquemment utilisés dans les systèmes répartis :

1. L'algorithme centralisé (un coordinateur assure le contrôle).
2. L'algorithme réparti (le contrôle est réparti sur les différents processus).
3. L'algorithme anneau à jeton (le contrôle est réparti à l'aide d'un jeton sur les différents processus).

Complétez le tableau (redessinez-le sur votre copie) en précisant les problèmes éventuels liés au type d'algorithme et commentez les résultats fournis dans les colonnes **Messages par entrée/sortie** et **Délai avant d'entrer dans la zone**.

Mécanisme	**Messages par entrée/sortie**	**Délai avant entrée dans la zone** (en temps messages)	**Problèmes**
Centralisé	3	2	
Distribué	2(n-1)	2(n-1)	
En anneau	1 à l'infini	0 à n-1	

L'algorithme centralisé :

On simule ce qui est fait dans un système centralisé. Un coordinateur central gère les ressources. C'est une application du principe utilisé sur un système avec un seul processeur. J'ai deux processus P1 et P2 qui veulent accéder à une même ressource. P1 demande au coordinateur si la ressource est libre, si elle est libre, le coordinateur accepte, le processus entre alors en zone critique. Si P2 veut utiliser la ressource, il demande au coordinateur si la ressource est libre, le coordinateur sait que P1 est en zone critique, il met à jour la queue de gestion de la ressource en rajoutant P2 en liste d'attente et refuse l'accès. Lorsque P1 sort de la zone critique, elle est libérée, le coordinateur sait que P2 peut rentrer en zone critique, il accepte la demande et met à jour la file d'attente.

Message en entrée/sortie :

- *Trois messages au maximum pour demander une ressource.*

Délai avant entrée dans la zone :

- *Deux messages au maximum avant d'entrer en zone critique.*

Les problèmes sont :

- *Un seul point de panne (SPF).*
- *Le coordinateur est dans le cas d'un système important un goulet d'étranglement.*

L'algorithme distribué :

L'exclusion mutuelle répartie résout le problème de l'unique point de panne. L'algorithme de Lamport sera utile pour la cohérence des temps. En effet pour la mise en place d'un MUTEX réparti il faut être certain de l'ordre. Quand un processus veut entrer dans la zone critique, il construit un message contenant le nom de la zone, son N° et l'heure courante. Il envoie le message à tous les processus y compris à lui-même. Un accusé de réception sera renvoyé par l'ensemble des processus (maintien de la fiabilité du système). Les processus répondront en fonction de leur état lié à la zone considérée (3 cas) :

1. *Le récepteur n'est pas dans la zone et ne veut pas y rentrer, il répond OK.*
2. *Le récepteur est dans la zone, il ne répond pas. Il charge la demande dans sa file d'attente.*
3. *Le récepteur veut entrer dans la zone, il compare l'heure du message avec celle du message qu'il a déjà envoyé. Si l'heure est inférieure, il répond OK sinon il place la demande dans sa file d'attente et il ne répond pas.*

Quand sa demande est partie, le processus attend son acceptation par tous les autres processus, il entre ensuite en zone critique. Quand il en sort, il répond OK aux processus qui sont dans sa file d'attente et les enlève de la file.

Message en entrée/sortie :

- *Avec n processus, 2(n-1) messages au maximum pour demander une ressource (n-1 requêtes aux processus et n-1 autorisation).*

Délai avant entrée dans la zone :

- *Avec n processus, 2(n-1) messages maximum pour demander une ressource (n-1 requêtes aux processus et n-1 autorisation).*

Les problèmes sont :

- *Un processus en panne peut-être interprété comme un refus .*
- *Il n'y a plus un seul point de panne mais n (algo n fois pire que le précédent).*

L'algorithme anneau à jeton :

Il utilise le principe de l'anneau à jeton. Un anneau logique est construit. Le processus rentrera en zone critique si et seulement si il a le jeton. À un instant t, seul un processus est détenteur du jeton. Chaque processus constituant l'anneau passe le jeton à son successeur. Le successeur renvoie un accusé de réception.

Message en entrée/sortie :

- *1 message si le jeton est récupéré par le processus dès qu'il veut rentrer en zone critique, l'infini si personne n'est intéressé par le jeton.*

Délai avant entrée dans la zone :

- *1 message si le jeton est récupéré par le processus dès qu'il veut rentrer en zone critique, n-1 messages si le jeton vient juste de passer devant le processus (Il doit faire le tour des n-1 processus avant de revenir).*

Les problèmes sont :

- *La gestion de la perte du jeton dans l'hypothèse où le processus qui a le jeton tombe en panne.*

2) Les transaction réparties

Décrivez le rôle du coordinateur dans le protocole de commit à deux phases pour les transactions réparties.

Le protocole de commit à deux phases est conçu pour permettre à tous les participants d'abandonner une transaction.

Le coordinateur gère la participation des différents acteurs, assure le vote pour le commit général, et décide du commit ou de l'abort de la transaction en fonction des retours obtenus des participants.

3) Processus et processeurs

3-1) Quelles sont les différences essentielles entre les processus et les « threads ».

- *Un processus léger (Thread) est un sous-processus.*
- *Il dépend d'un processus.*
- *Il permet le parallélisme au sein d'un processus.*
- *Il ne nécessite pas un nouvel espace d'adressage. Il utilise et partage celui créé par son processus père.*
- *Il n'effectue aucune réservation de cache.*

Les différences qui peuvent être faites entre processus et processus légers sont les suivantes:

- *Les processus légers ne sont pas aussi indépendants que les processus.*
- *Ils n'ont pas d'espace d'adressage propre.*
- *Tous les processus légers partagent les mêmes variables globales.*
- *Les processus légers au sein d'un même processus ne sont pas sécurisés.*
- *L'exécution d'un processus léger est séquentielle.*

3-2) Décrivez le fonctionnement de l'algorithme de Lamport dans le cadre des pannes byzantines survenant sur des processeurs. Pour la description, vous pourrez choisir un nombre de processeurs n=4 dont un processeur défaillant.

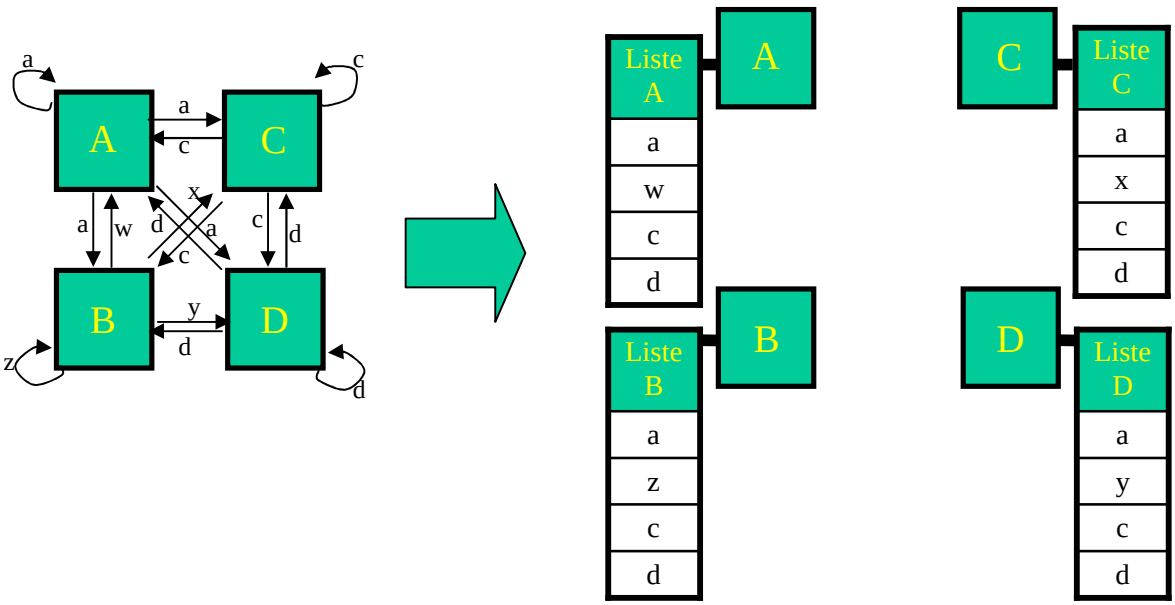
Nombre de processeurs n=4.

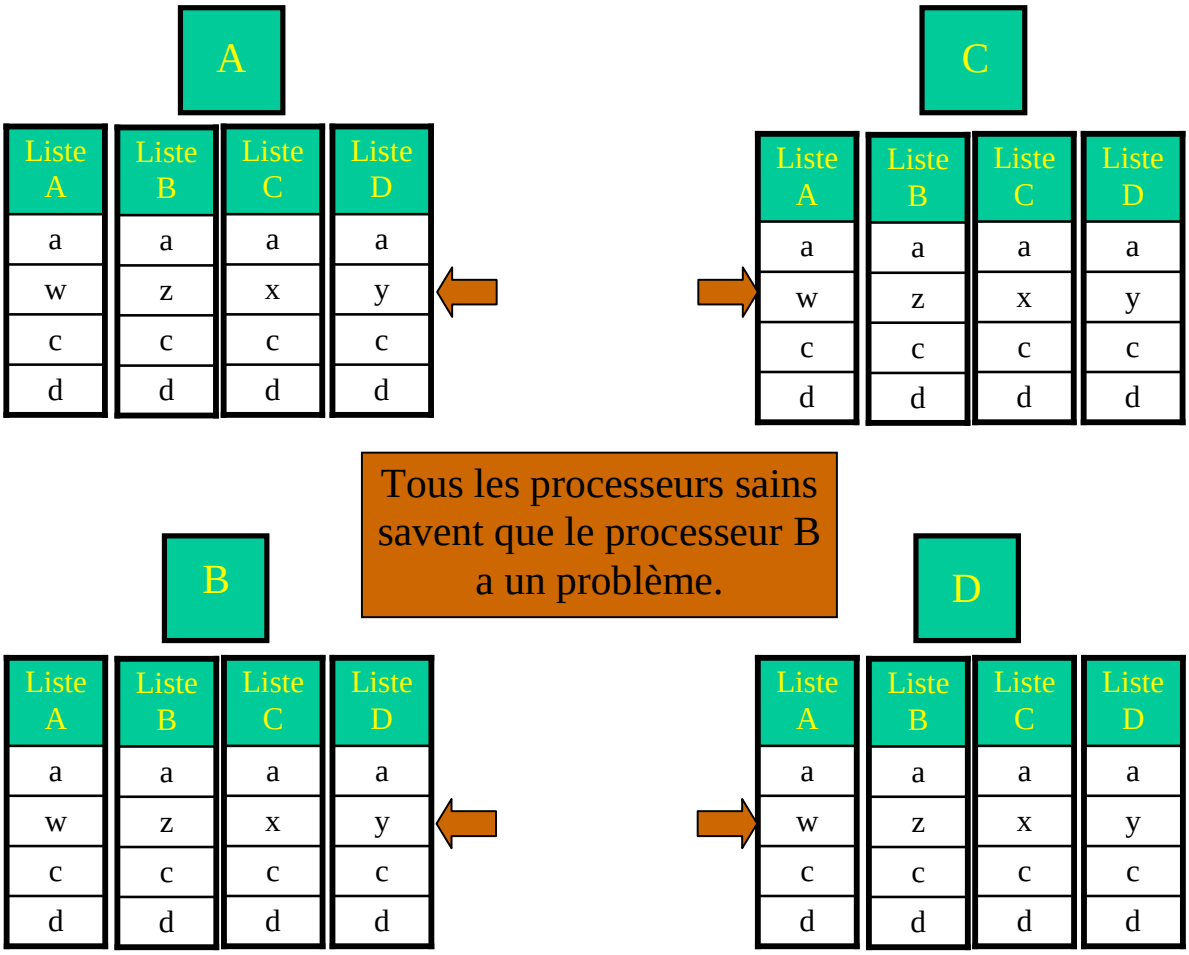
Nombre de processeurs défaillants nd=1.

L'algorithme effectue les opérations suivantes :

- *Chacun des n processeurs envoie son identifiant à tous les processeurs et à lui-même. Quatre messages sont ainsi envoyés.*
- *Chacun des n processeurs réceptionne les quatre messages.*
- *Chacun des n processeurs construit la liste des identifiants qu'il a reçu..*
- *Chacun des n processeurs envoie la liste composée des quatre identifiants à tous les autres processeurs.*

- *Chacun des n processeurs vérifie les listes qu'il a reçues et identifie le processeur qui a un problème en comparant les identifiants dans toutes les listes.*





Pour que cet algorithme fonctionne, si m processeurs sont défaillants alors 2m + 1 processeurs valides sont nécessaires.

- *Soit **n** processeurs dont **m** défaillants.*
- *Nous avons donc **n-m** processeurs sains.*
- *Si l'on veut pouvoir obtenir un agrément le nombre de processeurs sains doit être strictement supérieur au nombre de processeurs défaillants **n-m > m.***
- *D'où **n > 2m** donc **n doit être au moins égal à 2m+1.***

Sachant que dans le cadre de pannes byzantines il est nécessaire d'avoir 2n+1 processeurs pour pallier n processeurs défectueux, si je prends l'hypothèse où huit processeurs sont présents dans une machine. Quel est le nombre maximum de processeurs subissant des pannes byzantines que l'on peut détecter avec l'algorithme de Lamport.

En théorie si 8 processeurs il faut 2k+1 pour détecter les pannes byzantines soit : 2k+1 = 8 ; k = 7/2 = 3,5 soit k=3.

*Mais l'algorithme de Lamport fonctionne si et seulement si 2/3 des processeurs sont sains donc si (2 * 8) / 3 de processeurs sont sains soit : 5,33 processeurs sains. Un*

processeur ne pouvant être découpé le résultat est donc : ***6 processeurs sains et 2 défectueux.***

4) Les systèmes de fichiers distribués

4-1) Qu'appelle-t-on transparence de nommage. Quels en sont les désavantages.
La ***transparence de nommage*** *implique que les fichiers ne soient pas explicitement nommés. Cela implique :*

1. ***La transparence de localisation*** *: le nom du chemin ne doit pas donner d'indication sur le lieu où se trouve le fichier.*
2. ***L'indépendance du lieu*** *:Pas de référence à une machine ou a un serveur.*

4-2) Expliquez en quoi consiste la sémantique de session, ses avantages et ses limites.
La sémantique de session impose la règle suivante : « Les modifications apportées à un fichier sont renvoyées au serveur à la fermeture du fichier. Seuls les processus locaux voient les modifications ».
Les machines sont averties d'une modification à la fermeture du fichier. C'est plus performant que la sémantique UNIX, mais cela pose la problématique de la cohérence de l'information si deux fichiers sont modifiés en même temps.

5) Mémoire partagée et répartie

5-1) Qu'appelle-t-on modèles de cohérence.
Les modèles de cohérence permettent de gérer plusieurs copies de pages mémoire dans des mémoires dans systèmes répartis. L'objectif est d'éviter les goulets d'étranglement associés à l'utilisation concurrente d'une page mémoire unique. Il existe plusieurs modèles de cohérence.
Modèles de cohérence:

- *Cohérence stricte.*
- *Cohérence séquentielle.*
- *Cohérence causale.*
- *PRAM (Pipelined RAM).*
- *Cohérence processeurs (faible, relâchée, par entrée).*

5-2) Qu'est-ce que la cohérence séquentielle. Quel type de problème permet-elle d'éviter.
La règle de la cohérence séquentielle est la suivante : « Le résultat de toute exécution est identique à une exécution séquentielle et ordonnée de tous les processus, et les opérations de chaque processus apparaissent dans l'ordre d'exécution de son programme. »
Cela évite les problèmes d'incohérence ou d'indéterminisme dus à des résultats dépendants de l'ordre d'exécution des instructions.

Vous avez 2h30.

Bon travail.

Corrigé Examen - UV19302
Systèmes et applications répartis – B4

Session du 10 septembre 2004

Aucune notes personnelles ou documents ne sont autorisés.

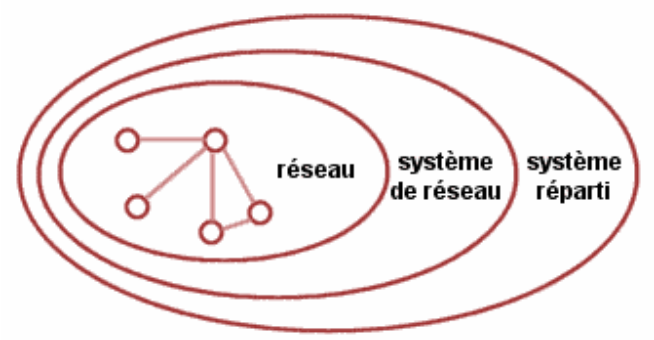
1) Systèmes répartis – Les concepts

1-1) Quels sont les principaux désavantages des systèmes répartis ?

Logiciel	*Peu de logiciels et d'applications existent aujourd'hui.*
Réseau	*Le réseau peut être facilement saturé ou dû à sa latence conduisent à des problèmes de temps de réponse. ?*
Sécurité	*Les accès à l'information sont multiples donc difficiles à contrôler.*

1-2) Quelle est la différence essentielle entre un système d'exploitation réparti et un système d'exploitation réseau ?

Dans le cas d'un réseau, tout est fait explicitement (on se connecte d'abord explicitement à une machine donnée, on soumet explicitement une commande à exécuter sur cette machine donnée, on doit explicitement déplacer les fichiers...), alors que dans le cas d'un système réparti, tout est réalisé automatiquement par le système d'exploitation. La différence est donc une différence au niveau du logiciel système, et non pas au niveau du matériel (type d'ordinateur, type de connexion...). On peut résumer ceci par le schéma suivant :



1-3) Quelles sont les tâches primaires d'un micro-noyau ?

Quatre tâches primaires sont présentes dans les micro-noyaux des systèmes répartis :

1. *Mécanismes de communication inter-processus.*
2. *Gestion basique de la mémoire.*
3. *Gestion des processus et ordonnancement.*
4. *Gestion bas niveau des entrées – sorties.*

1-4) Qu'appelle-t-on transparence de concurrence ? Cette propriété existe-t-elle en natif sur les systèmes centralisés
Les systèmes répartis sont accédés en principe par de nombreux utilisateurs utilisant des machines indépendantes. On parle de ***concurrence*** *lorsque plusieurs utilisateurs cherchent à accéder à une même ressource en même temps.* ***La transparence de concurrence*** *d'un système fait que les utilisateurs ne se rendent pas comptent de ces accès concurrents. Un des* ***mécanismes*** *utilisé pour cette problématique est le verrouillage (lock) de la ressource lors de l'accès et le déverrouillage (unlock) lors de la libération de la ressource.*

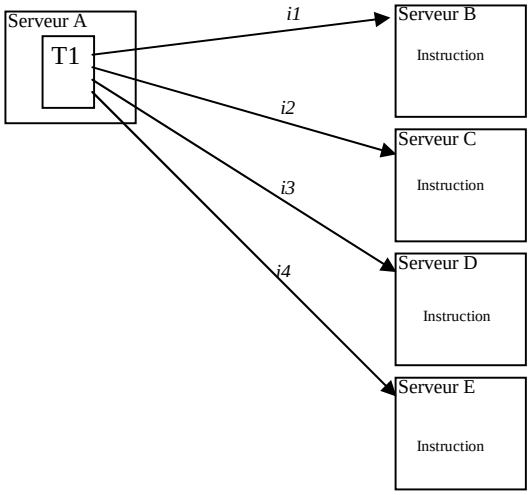
2) Réseau

2-1) Dans la plupart des protocoles réseau s'appuyant sur des couches, chaque couche a son propre entête (*« header »*) alors qu'il serait certainement plus facile d'avoir un seul entête pour le message. Quelle est la raison de cette multiplication des entêtes ?
Pour les réseaux et afin de les rendre homogènes, un modèle a été défini et implémenté : ***le modèle ISO****. Chaque couche du modèle ISO est indépendante de la couche précédente et suivante. La raison de la multiplication des entêtes est de rendre la communication inter-couches plus facile, mais aussi évolutive. Une modification dans une couche (rajout ou changement de norme) n'a pas d'impact sur les autres couches.*

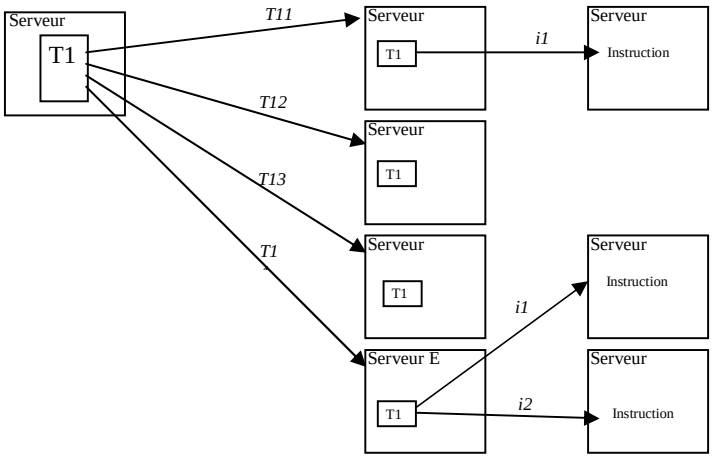
3) Les transactions

3-1) Qu'appelle-t-on transaction répartie ?
Une transaction répartie est une transaction dont des instructions élémentaires la constituant peuvent être exécutées sur des serveurs différents.

3-2) Au travers de deux graphiques montrez l'exécution possible d'une transaction répartie simple (graphique 1) et celle d'une transaction répartie imbriquée (graphique 2).



Graphique 1



Graphique 2

4) Les systèmes de fichiers distribués

4-1) Expliquez en quoi consiste la sémantique UNIX, ses avantages et ses limites.
Le système assure que l'on obtient dans tous les cas la dernière version du fichier demandé. C'est-à-dire que la dernière mise à jour a été prise en compte. Le principal avantage est que l'on est certain d'avoir la dernière mise à jour du fichier. Le principal désavantage est que son implémentation nécessite la centralisation du serveur de fichier et rend les opérations de lecture-écriture séquentielles.

4-2) Dans un système de fichiers distribué comment peut-on être certain d'obtenir une sémantique UNIX ? Dans ce cas il existe tout de même un problème mineur dû au réseau. Voyez-vous lequel ?
Pour être certain qu'un système de fichier assure la sémantique UNIX, il faut que toutes les opérations passent par un serveur de fichiers central et qu'aucun mécanisme de cache ne soit implémenté. Dans le cas précédent, un problème mineur persiste : une lecture peut être demandée après une écriture mais à cause du réseau arriver sur le serveur central avant l'écriture. Dans ce cas, la valeur retournée par le serveur est l'ancienne.

5) La mémoire répartie

5-1) Dans un système à mémoire répartie, les concepts de pointeurs et de variables globales sont différents de ceux qui sont mis en place dans un « système traditionnel ». Pourquoi ?
Les pointeurs et variables globales font référence à des adresses-mémoire définies et gérées localement par le système. Tous les processus s'exécutant sur un même système physique peuvent ainsi partager et accéder à ces zones. Les processus d'un système réparti s'exécutent sur des machines physiques différentes, les adresses-mémoire sont dès lors différentes et ne peuvent plus être accédées de la même manière ni partagées.

5-2) Pouvez-vous citer un exemple mettant en évidence la différence citée à la question 5-1 ?
Les exemples sont :

- *La variable globale « errorno ».*
- *La gestion des tableaux.*
- *Etc.*

Corrigé Examen - UV19302
Systèmes et applications répartis – B4

Session du 2 juillet 2004

Aucune notes personnelles ou documents ne sont autorisés.

1) Systèmes répartis – Les concepts

1-1) Qu'appelle-t-on matériel fortement couplé et faiblement couplé ?
Un matériel fortement couplé est un matériel où les composants le constituant sont physiquement proches et reliés par des moyens de communication très rapides (par exemple deux processeurs sur un même circuit). A l'inverse les composants d'un matériel faiblement couplé ne sont pas rapprochés et les moyens de communication sont lents (par exemple deux machines dans un réseau).

1-2) Quelles sont les caractéristiques, avantages et désavantages du matériel fortement couplé.
La principale caractéristique d'un matériel fortement couplé est le délai très court entre envoi et réception de messages entre deux composants. Les systèmes dont le matériel est fortement couplé sont utilisés pour les calculs parallèles : résolution d'un même problème.

2) Remote Procedure Call

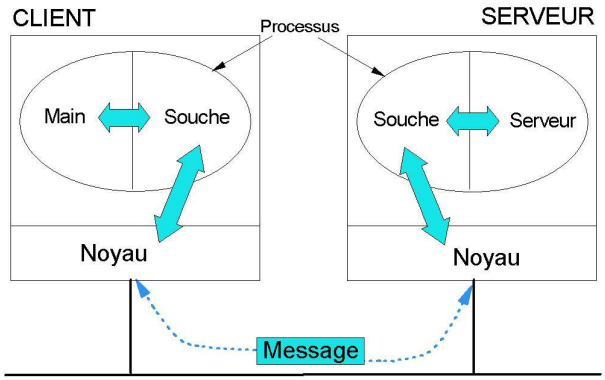
2-1) Comment fonctionne un appel RPC : notions de souches (*stub*), gestion de la pile, etc.
Afin d'assurer le principe de transparence, le mécanisme RPC met en place des versions différentes d'une procédure selon qu'elle est locale ou distante.

Pour effectuer un appel RPC :

La version cliente de la procédure *est constituée d'une souche cliente (« client stub ») qui est placée dans la librairie locale. L'appel se fait ensuite comme avec une procédure locale (la pile est gérée localement) et l'exécution est prise en charge par le noyau. Par contre, les paramètres ne sont pas stockés dans les registres et aucune donnée n'est demandée au noyau. Les paramètres sont stockés dans un message construit par le noyau et envoyé au serveur distant qui contient la partie serveur de la procédure. Le client attend ensuite que le serveur lui réponde.*

La version serveur de la procédure *est constituée d'une souche serveur et de la procédure. Lorsque le message parvient au serveur, le noyau passe le message à la souche correspondante (qui est en attente de messages). La souche extrait les paramètres du message, renseigne la pile et appelle la procédure qui pense être appelée par un client local.*

Lorsque la procédure a terminé son traitement, elle renseigne la pile et rend la main à la souche (voir schéma suivant).



3) Les transactions

3-1) Les deux problèmes majeurs liés à l'exécution de transactions concurrentes sont :

- La perte de mise à jour.
- Les récupérations inconsistantes.

Expliquez en quoi consistent ces deux problèmes. Vous pouvez vous aider d'exemples.

La perte de mise à jour :

Lors d'exécution concurrente de deux transactions U et T, des mises à jour peuvent être perdues parce que n'ayant pas eu connaissance des modifications de U elle viendra effectuer ses propres mises à jour et ainsi détruire celles de U (voir illustration ci-dessous)).

Exemple

Compte banquaire : A=100€ ;B=200€ ;C=300€ ;

Transaction T : *Balance=b.getBalance() ;* *b.setBalance(balance*1.1) ;* *a.debit(Balance/10) ;*	***Transaction U :*** *Balance=b.getBalance() ;* *b.setBalance(balance*1.1) ;* *c.debit(Balance/10) ;*
Balance=b.getBalance() ; [200€] *b.setBalance(balance*1.1) ; [220€]* *a.debit(Balance/10) ; [80€]*	*Balance=b.getBalance() ; [200€]* *b.setBalance(balance*1.1) ; [220€]* *c.debit(Balance/10) ; [280€]*

Les récupérations inconsistantes :

Lors d'exécution concurrente de deux transactions, des récupérations inconsistantes peuvent être effectuées parce que les informations récupérées et utilisées par une transaction peuvent être des informations incomplètement (ou en cours de) mises à jour par l'autre.
(voir illustration ci-dessous)).

Exemple
Compte banquaire : A=200€ ;B=200€ ;

Transaction V :	***Transaction W :***
a.debit(100) ; *b.credit(100);*	*Totaldescomptes() ;*
a.debit(100) ; [A=100€] *b.credit(100); [B=300€]* ***Total des comptes A et B = 400€***	*Total=a.getBalance() ; [100€]* *Total=Total+b.getBalance() ; [100€+200€]* *Total=Total+c.getBalance() ;* *…* ***Total des comptes A et B = 300€***

3-2) Montrez comment les règles de gestion de conflits associées au lock permettent de pallier ces deux problèmes.
Les règles de gestion de conflit sont les suivantes :

1. *Si une transaction T effectue une opération de lecture sur un objet, alors une transaction concurrente ne pourra pas effectuer d'opération d'écriture sur ce même objet.*
2. *Si une transaction T effectue une opération d'écriture sur un objet, alors une transaction concurrente ne pourra pas effectuer d'opération de lecture ou d'écriture* sur ce même objet.

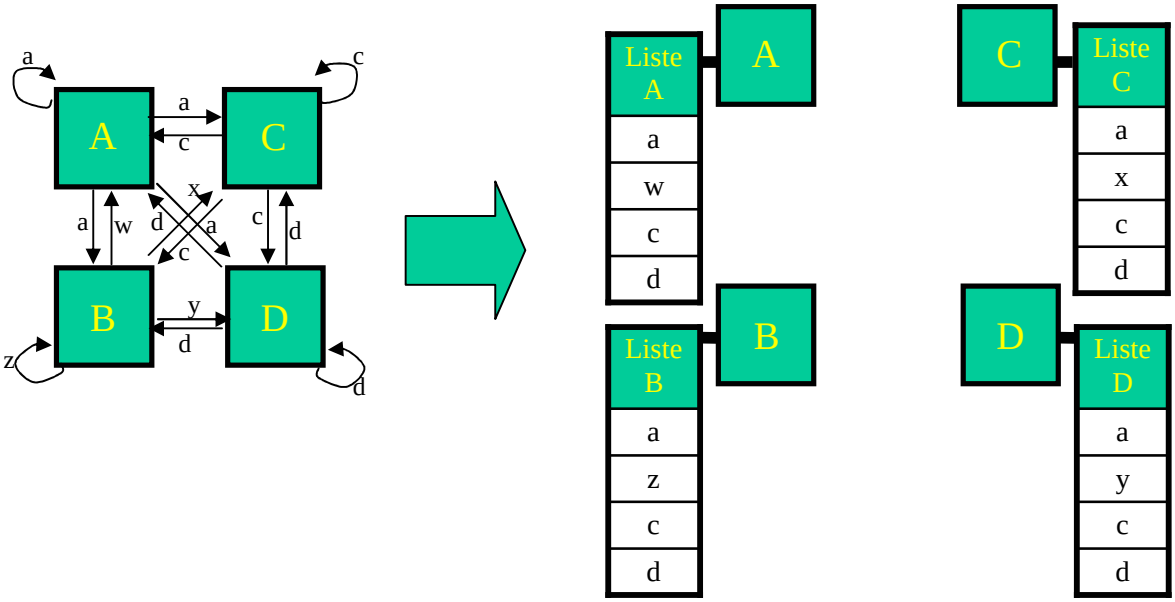
Lock posé	**Demande de lock en lecture**	**Demande de lock en écriture**
Aucun	OK	OK
Lecture	OK	Wait
Ecriture	Wait	Wait

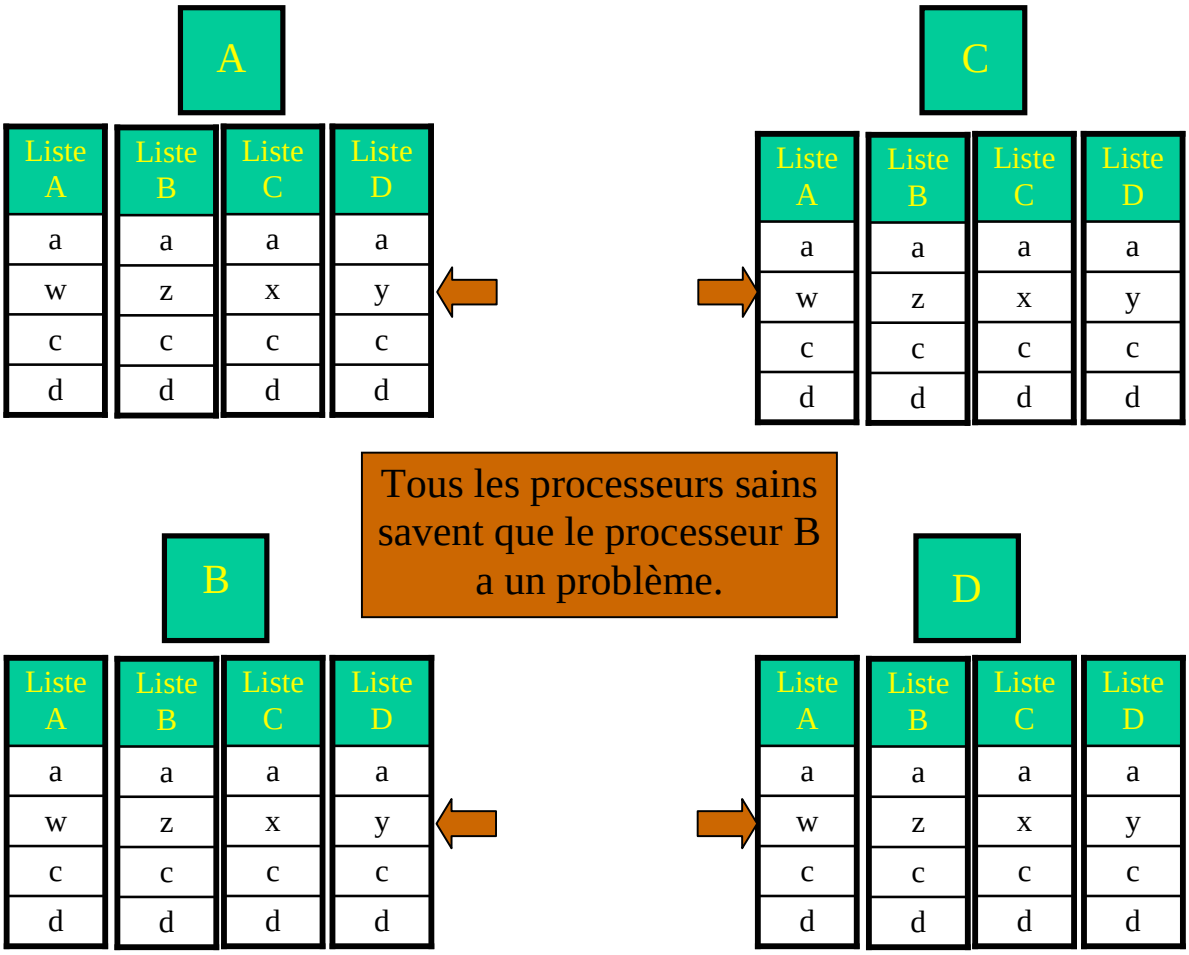
4) Processus et processeurs

4-1) Décrivez le fonctionnement de l'algorithme de Lamport dans le cadre des pannes byzantines survenant sur des processeurs. Pour la description vous pourrez choisir un nombre de processeurs n=4 dont un processeur défaillant.
Nombre de processeurs n=4.
Nombre de processeurs défaillants nd=1.

L'algorithme effectue les opérations suivantes :

- *Chacun des n processeurs envoie son identifiant à tous les processeurs et à lui même. Quatre messages sont ainsi envoyés.*
- *Chacun des n processeurs réceptionne les quatre messages.*
- *Chacun des n processeurs construit la liste des identifiants qu'il a reçu..*
- *Chacun des n processeurs envoie la liste composée des quatre identifiants à tous les autres processeurs.*
- *Chacun des n processeurs vérifie les listes qu'il a reçu et identifie le processeur qui a un problème en comparant les identifiants dans toutes les listes.*





4-2) Prouver que pour que cet algorithme fonctionne, si m processeurs sont défaillants alors 2m + 1 processeurs valides sont nécessaires.

- *Soit **n** processeurs dont **m** défaillants.*
- *Nous avons donc **n-m** processeurs sains.*
- *Si l'on veut pouvoir obtenir un agrément le nombre de processeurs sains doit être strictement supérieur au nombre de processeurs défaillants **n-m > m.***
- *D'où **n > 2m** donc **n doit être au moins égal à 2m+1.***

5) Les systèmes de fichiers distribués

5-1) Un système de fichier distribué est construit, en règle générale, autour de deux composants distincts. Quels sont-ils ? Faites une description **d'un** de ces composants.

Un système de fichiers a généralement deux composants distincts:
1. *Le **service de fichiers** qui gère les opérations sur les fichiers (attribut, structure).*
2. *Le **service des répertoires** qui gère les opérations sur les répertoires (nom, longueur, taille, extension, arborescence).*

***Le service de fichiers** (File Service) spécifie ce qui est offert aux utilisateurs pour gérer les fichiers. Il définit les primitives disponibles, comment les utiliser, les actions qui peuvent être entreprises. C'est l'interface que va utiliser le client.*

***Le service des répertoires** gère les opérations sur les répertoires (nom, longueur, taille, extension, arborescence).*

5-2) Expliquez en quoi consiste la sémantique de session, ses avantages et ses limites.
Les modifications apportées à un fichier sont renvoyées au serveur à la fermeture du fichier. Seuls les processus locaux voient les modifications en cours.
Avantages :
- *Moins de trafic réseau.*
- *Facile à mettre en place.*

Limites
- *C'est le dernier qui met à jour le fichier qui l'emporte. Les modifications effectuées précédemment par d'autres sont perdues.*

6) La mémoire partagée et répartie

6-1) Quelle est la différence majeure entre mémoire partagée et mémoire répartie.
*La mémoire partagée est une zone de mémoire située sur une **même machine physique** et accessible par différents processeurs. Ce type d'architecture se retrouve sur les systèmes multi-processeurs.*
Plusieurs technologies existent:
- *Multiprocesseurs avec bus.*
- *Multiprocesseurs à anneau.*
- *Multiprocesseurs commutés.*
- *Architecture NUMA.*

*La mémoire répartie est une zone de mémoire découpée et située sur des **machines physiques différentes**. Cette mémoire peut être accessible par des processus/processeurs différents et répartis. Ce type d'architecture se retrouve sur les systèmes distribués.*
Des problèmes liés à cette technologie doivent être résolus notamment:
- *Les goulets d'étranglement.*
- *La cohérence.*

6-2) Donnez la règle associée au modèle de cohérence appelé « **Cohérence stricte** ». Expliquez au travers d'un exemple simple la problématique résolue par ce modèle.

« Toute lecture d'une position mémoire x retourne la valeur stockée par l'écriture la plus récente. »
Cette règle tient compte d'un temps global absolu, ainsi la détermination la plus récente ne sera pas ambiguë.

Corrigé Examen - UV19302
Systèmes et applications répartis – B4

Session du 12 septembre 2003

Aucune notes personnelles ou documents ne sont autorisés.

1) Remote Procedure Call

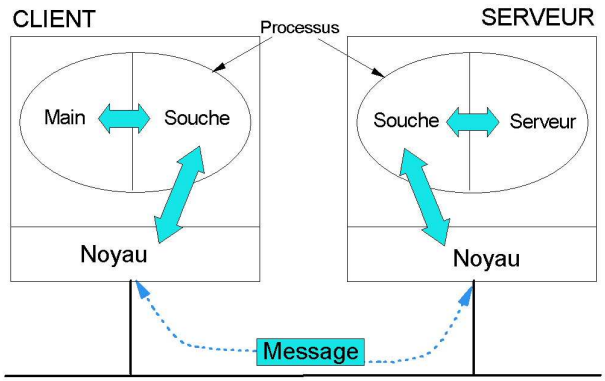
1-1) Qu'appelle-t-on RPC ? À quoi servent les RPCs ?
*Les **RPC**s (**Remote Procedure Call**) sont des procédures qui peuvent être invoquées par des programmes s'exécutant sur des postes distants. On évite ainsi aux programmeurs de programmer des entrées /sorties ou des envois de messages. L'appel à ces procédures distantes (remote) se fait de manière transparente pour l'utilisateur.*

1-2) Décrivez le fonctionnement client-serveur d'un appel RPC ? Vous considèrerez les comportements du client et du serveur.
Afin d'assurer le principe de transparence, le mécanisme RPC met en place des versions différentes d'une procédure selon qu'elle est locale ou distante.

Pour effectuer un appel RPC :

***La version cliente de la procédure** est constituée d'une souche cliente (« client stub ») qui est placée dans la librairie locale. L'appel se fait ensuite comme avec une procédure locale (la pile est gérée localement) et l'exécution est prise en charge par le noyau. Par contre, les paramètres ne sont pas stockés dans les registres et aucune donnée n'est demandée au noyau. Les paramètres sont stockés dans un message construit par le noyau et envoyé au serveur distant qui contient la partie serveur de la procédure. Le client attend ensuite que le serveur lui réponde.*

***La version serveur de la procédure** est constituée d'une souche serveur et de la procédure. Lorsque le message parvient au serveur, le noyau passe le message à la souche correspondante (qui est en attente de messages). La souche extrait les paramètres du message, renseigne la pile et appelle la procédure qui pense être appelée par un client local. Lorsque la procédure a terminé son traitement, elle renseigne la pile et rend la main à la souche (voir schéma suivant).*



1-3) Quelle peut être une solution envisageable pour la résolution du problème du passage des paramètres de taille variable avec des RPCs.
Le problème est fréquent avec des tableaux dont les tailles ne sont pas connues au préalable. ***Une des solutions*** *est de coder en dur les longueurs de paramètres. Dans l'hypothèse où la longueur du paramètre est inférieure à l'espace réservé, on perd de l'espace. Si la longueur est supérieure, on a une erreur.*

2) Les transactions

2-1) Qu'appelle-t-on:
- Transaction.
- Transaction atomique.
- Transaction imbriquée.

Une transaction est un ensemble d'opérations élémentaires exécutées lors de l'exécution de la transaction.

Une ***transaction atomique*** *est une séquence d'opérations qui s'exécute sur un serveur et qui* ***garantit*** *que l'****ensemble des opérations*** *qui composent la transaction* ***sera ou ne sera pas*** *exécuté. Elle permet ainsi de préserver la cohérence des informations manipulées par des activités concurrentes.*

Une ***transaction imbriquée*** *est structurée à partie d'un ensemble d'autres transactions (simples ou imbriquées).*

2-2) En vous référant à l'acronyme « A.C.I.D » ? Donnez pour chacune des lettres la propriété correspondante et une brève description de celle-ci.

A=Atomicité ;-)
- ***Atomicité :*** *Vu de l'extérieur la transaction apparaît comme indivisible. « Tout ou rien ».*
- ***Coherence:*** *La transaction ne viole pas les invariants du système.*
- ***Isolation:*** *Les transactions concurrentes n'interfèrent pas entre elles.*
- ***Durabilité:*** *Lorsqu'une transaction réussie, son résultat est permanent.*

2-3) Qu'appelle-t-on sérialisabilité?
La sérialisabilité est le fait que, malgré que les transactions puissent s'exécuter concurremment, elles ont le même résultat que si elles étaient sérialisées. Ceci est essentiel si l'on veut obtenir un système déterministe.

3) Processus et processeurs

3-1) Qu'appelle-t-on processus léger (*thread*) ? Quel est son rôle? Quelles sont les différences majeures qui peuvent être faites entre processus léger et processus?
- *Un processus léger (Thread) est un sous-processus.*
- *Il dépend d'un processus.*
- *Il permet le parallélisme au sein d'un processus.*
- *Il ne nécessite pas un nouvel espace d'adressage. Il utilise et partage celui créé par son processus père.*
- *Il n'effectue aucune réservation de cache.*

Les différences qui peuvent être faites entre processus et processus légers sont les suivantes:
- *Les processus légers ne sont pas aussi indépendants que les processus.*
- *Ils n'ont pas d'espace d'adressage propre.*
- *Tous les processus légers partagent les mêmes variables globales.*
- *Les processus légers au sein d'un même processus ne sont pas sécurisés.*
- *L'exécution d'un processus léger est séquentielle.*

3-2) Dans le cadre des processus légers, à quoi servent les techniques de gestion des zones critiques?
Les techniques de gestion des zones critiques dans le cadre des processus légers servent à protéger et partager les données de chaque processus léger. En effet, ceux-ci partageant le même espace d'adressage, le partage ou l'accès peut s'avérer problématique.
Les techniques classiques peuvent être utilisées:
- *Primitives de gestion de zones critiques.*
- *Sémaphores.*
- *MUTEX.*
- *Variables de condition.*

4) Les systèmes de fichiers distribués

4-1) Quel est le rôle d'un système de fichier?
Un système de fichier permet le stockage et l'organisation de l'information. Il est, en règle générale, associé à un système d'exploitation.
Ces principales fonctionnalités sont les suivantes:
- *Il permet la désignation des fichiers (souvent une arborescence).*
- *Il fournit une interface d'accès aux programmes.*
- *Il gère la correspondance entre nom symbolique et adresse physique sur le support.*
- *Il assure l'intégrité des données par rapport aux pannes.*
- *Il autorise les accès concurrents.*

- *Fournit des attributs associés aux fichiers (lecture, écriture, exécution).*
- *Fournit des mécanismes d'autorisation.*

4-2) Un système de fichier distribué est construit, en règle générale, autour de deux composants distincts. Quels sont-ils? Faites une description **d'un** de ces composants.

Un système de fichiers a généralement deux composants distincts:

3. *Le **service de fichiers** qui gère les opérations sur les fichiers (attribut, structure).*
4. *Le **service des répertoires** qui gère les opérations sur les répertoires (nom, longueur, taille, extension, arborescence).*

5) La mémoire partagée et répartie

5-1) Qu'appelle-t-on mémoire partagée? Dans quel type d'architecture de système la trouve-t-on?

La mémoire partagée est une zone de mémoire située sur une même machine physique et accessible par différents processeurs. Ce type d'architecture se retrouve sur les systèmes multi-processeurs.

Plusieurs technologies existent:

- *Multiprocesseurs avec bus.*
- *Multiprocesseurs à anneau.*
- *Multiprocesseurs commutés.*
- *Architecture NUMA.*

5-2) Qu'appelle-t-on mémoire répartie? Dans quel type d'architecture de système la trouve-t-on?

La mémoire répartie est une zone de mémoire découpée et située sur des machines physiques différentes. Cette mémoire peut être accessible par des processus/processeurs différents et répartis. Ce type d'architecture se retrouve sur les systèmes distribués.

Des problèmes liés à cette technologie doivent être résolus notamment:

- *Les goulets d'étranglement.*
- *La cohérence.*

5-3) Il existe plusieurs modèles de cohérence qui sont basés sur des règles précises et incontournables.

- Donnez les noms des modèles que vous connaissez.
- Décrivez l'un de ces modèles et donnez la règle qui le régit.

Modèles de cohérence:

- *Cohérence stricte.*
- *Cohérence séquentielle.*
- *Cohérence causale.*
- *PRAM (Pipelined RAM).*
- *Cohérence processeurs (faible, relâchée, par entrée).*

Cohérence stricte
« Toute lecture d'une position mémoire x retourne la valeur stockée par l'écriture la plus récente. »

Cohérence séquentielle
« Le résultat de toute exécution est identique à une exécution séquentielle et ordonnée de tous les processus, et les opérations de chaque processus apparaissent dans l'ordre d'exécution de son programme. »

Cohérence causale
« Les écritures qui ont potentiellement une relation causale doivent être vues par tous les processus dans le même ordre. Des écritures parallèles peuvent être vus dans un ordre différent sur différentes machines. »

PRAM
« Les écritures causales faites par un processus, sont vues par tous les processus dans le même ordre. Les écritures parallèles (concurrentes) peuvent être vues dans un ordre différent par des processus différents. »

Cohérence processeurs
« Les écritures causales faites par un processus, sont vues par tous les processus dans le même ordre. Les écritures parallèles (concurrentes) peuvent être vues dans un ordre différent par des processus différents. »

Examen - UV19302
Systèmes et applications répartis – B4

Session du 04 juillet 2003

Aucune notes personnelles ou documents ne sont autorisés.

1) Les systèmes répartis

1.1) Quelles seraient les raisons qui vous feraient concevoir un système réparti ?

Un système réparti est un ensemble d'ordinateurs indépendants reliés à un réseau et donnant l'impression à l'utilisateur d'un système monoprocesseur.

Les justifications d'un tel système sont multiples, pour ne citer que les plus importantes :

- *Il est plus économique d'utiliser un ensemble de petites machines (PCs ou stations) que de travailler sur un grand système.*
- *Les développeurs ne sont plus groupés sur le même site, mais peuvent être répartis à travers le monde. Ils ont la possibilité de partager puissance de calcul et données en même temps.*
- *Un ensemble de petites machines reliées en réseau ont une puissance supérieure comparé à un système centralisé.*
- *La tolérance aux pannes est plus importante dans un système distribué, une machine tombant en panne peut facilement être remplacée et toutes les autres continuent à fonctionner.*
- *L'extension du système (scalability) se fait de manière simple par l'ajout de nouvelles machines sur le réseau.*

1.2) Les objectifs majeurs de tout système réparti sont les suivants :

- La transparence (accès concurrent, migration, parallélisme).
- La flexibilité.
- La fiabilité et la disponibilité.
- La performance.
- L'évolutivité (scalabilité).

Vous expliquerez en quelques lignes ce que représentent ces termes.

La transparence, c*'est masquer la répartition au travers des spécifications suivantes :*

- *La localisation des ressources n'est pas perceptible.*
- *La migration des ressources d'un lieu à un autre n'implique aucune modification de l'environnement utilisateur.*
- *Le nombre de ressources dupliquées n'est pas connu (invisibilité).*
- *La concurrence d'accès aux ressources n'est pas perceptible.*
- *Invisibilité du parallélisme sous-jacent offert par l'ensemble de l'environnement d'exécution.*

La flexibilité *ou modularité. C'est faire en sorte que le système soit le plus modulaire possible. Pour ce faire, les systèmes répartis sont construits par l'ajout de services reposant sur un micro-noyau. Si de nouveaux services doivent être implémentés, ils sont ajoutés au système.*

La fiabilité et la disponibilité *d'un système réparti repose essentiellement sur l'indépendance des composants du système et la redondance des composants logiciels et matériels. Pour qu'un système soit fiable et disponible, il faut qu'il soit :*

- *Tolérant aux pannes.*
- *D'un fonctionnement sûr.*
- *Sécurisé.*

La performance *d'un système réparti c'est sa faculté à gérer un grand nombre :*

- *D'utilisateurs.*
- *De machines.*
- *De services.*
- *De trafic.*
- *De processus.*

Les métriques de la performance sont :

- *Le temps de réponse.*
- *Le débit.*
- *L'utilisation du système.*
- *La capacité réseau utilisée.*

L'évolutivité (scalability) *c'est la faculté qu'a un système à rester efficace et performant malgré la croissance du nombre d'utilisateurs et du nombre de ressources.*

2) Les ordonnanceurs (*schedulers*)

Quelles sont les différentes approches que l'on peut concevoir pour des ordonnanceurs.
Vous expliquerez comment fonctionne dans certains ordonnanceurs la prioritisation en modes coopératif **et** préemptif.

Un ordonnanceur gère et régule l'exécution des tâches. Il va déterminer qui doit s'exécuter, quand et où. Il se base sur une prioritisation des tâches à exécuter. Le noyau recense les tâches qui sont à l'état « ACTIVE » et exécute celle qui a la priorité la plus haute. Les ordonnanceurs sont basés sur des algorithmes complexes, plusieurs approches peuvent être envisagées et combinées :

- ***Temps réel fourni par le matériel ou le logiciel ?***

Il y a beaucoup d'avantages à ce que le ***matériel*** *fournisse les mécanismes d'ordonnancement. Les plus importants sont la fiabilité, la sécurité et de moindres efforts de développement. Sinon, l'ordonnateur devra être programmé (****logiciel****).*

- ***Préemptif ou coopératif.***

Le mode ***préemptif*** *est un mode interruptible. C'est-à-dire que l'ordonnanceur gère le temps d'exécution d'une tâche. Le mode* ***coopératif*** *est un mode non interruptible, la tâche s'exécute*

jusqu'à ce qu'elle décide de s'arrêter et de rendre la main. Les deux modes peuvent être codés dans un même ordonnanceur (ex : coopératif pour les tâches systèmes et préemptif pour les tâches utilisateurs).

- ***Dynamique ou statique.***

Dynamique *signifie que les décisions de l'ordonnanceur sont prises lors de l'exécution.* ***Statique*** *signifie que les décisions sont planifiées avant l'exécution.*

- ***Centralisé ou décentralisé.***

On parle de ***centralisé*** *lorsque le système gère l'ordonnancement. En* ***décentralisé****, chaque processeur du système gère l'ordonnancement.*

- ***Notion de priorité.***

On prioritirise les tâches à exécuter. De ce fait, l'ordonnanceur pourra établir ses décisions en fonction de la ***priorité*** *d'une tâche. En règle générale, les tâches les plus prioritaires s'exécuteront avant les moins prioritaires.*

3) Synchronisation des horloges

Décrivez l'algorithme de Lamport permettant de synchroniser les horloges logiques dans un système réparti.

Tous les ordinateurs ont une horloge physique qui est en fait un oscillateur au quartz. Un compteur logique et un registre particulier sont associés à cette horloge physique. Chaque oscillation diminue le compteur de 1 lorsqu'il est à 0 le compteur génère une interruption, récupère le contenu du registre et recommence. À chaque interruption, l'horloge logicielle est mise à jour. Aussi, aucune structure répartie ne pourra maintenir des horloges physiques synchronisées. En effet, les faibles variations des oscillateurs sont répercutées et très vite les machines sont déphasées. Il a donc été nécessaire de pallier cette problématique, c'est le but des ***horloges logiques****.*

L'algorithme de Lamport

L'horloge logique se base sur la suite d'évènements où l'un arrive avant un autre avec une relation « est arrivé avant » symbolisée par « → ».
Ex : Soit deux événements A et B d'un processus. Si A se passe avant B ; A peut être l'envoi d'un message par un processus et B le message reçu, le message ne peut pas être reçu avant l'envoi donc $A \rightarrow B$.
Lamport *associe aux évènements une notion de temps C() qui implique que pour* $A \rightarrow B$ *nous avons C(A) < C(B). C représente le temps de l'horloge, C croît donc toujours.*
L'algorithme de Lamport permet ainsi de synchroniser les horloges des différentes machines du système réparti.

Les règles de l'algorithme sont les suivantes :

- *Si A est arrivé avant B alors C(A) < C(B)*
- *Si A et B représentent l'envoi d'un message et la reception d'un message alors C(A) < C(B), si ce n'est pas le cas réellement pour les horloges alors l'algorithme ajustera le temps par C(B) = C(A)+1 et modifiera aussi l'horloge.*
- *Pour tous les évènements A et B, C(A) est différent de C(B).*

Il est à noter que si deux processus n'interagissent pas entre eux, la synchronisation de leurs horloges n'est pas utile.
Ce qui importe dans le cas de l'algorithme de Lamport n'est pas la cohésion du temps mais l'ordre.

4) Remote Procedure Call

Lors d'un appel RPC, le client ou le serveur peut tomber en panne.

- Décrivez la problématique et une solution dans le cas d'un problème sur le serveur.
- Décrivez la problématique et une solution dans le cas d'un problème sur le client.

4. *Le client ne peut pas localiser le serveur*
   - *Renvoi d'un message « Cannot locate server »*
5. *L'appel du client n'arrive pas au serveur*
   - *Rajout d'un compteur de temps dans le noyau. Après un certain délai le message est renvoyé.*
   - *Après un certain nombre d'essais, renvoi d'un message « Cannot locate server »*
6. *La réponse du serveur n'arrive pas au client*
   - *Une information est rajoutée dans le message qui détermine quelle réponse a été perdue (N° de séquence ou heure de départ).*

Le serveur est en panne

- *Il faut dissocier la panne avant l'exécution de la requête ou après l'exécution.*
- *Trois écoles pour solutionner ce problème :*
  4. *Attendre que le serveur redémarre et relancer le traitement.*
  5. *Abandon du client et rapport de l'erreur.*
  6. *Ne rien dire au client (aucune garantie)*
- *En règle générale, crash du serveur $\Rightarrow$ crash du client.*

Le client est en panne

Si le client envoie une requête et s'il tombe en panne avant la réponse, l'échange est dit ***orphelin.*** *Cela a pour effet de gaspiller du temps CPU, de bloquer des ressources, lorsqu'il redémarre il peut recevoir des messages antérieurs à la panne (problème de confusion).*

- *Quatre solutions peuvent être envisagées :*
  5. *L'extermination : Le client enregistre les appels dans une log. Au redémarrage l'orphelin est détruit.*
  6. *La réincarnation : Le client tient compte du temps. Lorsqu'il redémarre, il annule les échanges ente deux époques et reprend les échanges orphelins.*
  7. *La réincarnation à l'amiable : Comme précédemment, avec en plus la vérification avec le serveur concerné des échanges orphelins.*
  8. *Expiration du délai : Un compteur de temps détermine quand un échange devient orphelin. Les orphelins sont détruits.*

5) L'élection

- À quoi servent les algorithmes d'élection.
- Décrivez l'algorithme d'élection de Bully.

Beaucoup d'algorithmes distribués ont besoin qu'un processus agisse en coordinateur, initiateur, séquenceur ou tout autre rôle spécifique. En règle générale, il n'est pas important de connaître qui va prendre ce rôle, il faut simplement être certain qu'un au moins le fera.
Afin que ce désignation soit effectuée (ce que l'on appelle élection) il est nécessaire que les processus puissent être différenciés, soit par un identifiant unique, soit par leur adresse IP. Les algorithmes d'élection utiliseront cet identifiant unique pour localiser le processus candidat.

L'algorithme d'élection de Bully

Le principe est que lorsqu'un processus s'aperçoit que le coordinateur ne répond plus, il initie une élection.
Soit P_1 le processus qui s'aperçoit que le coordinateur ne répond plus.
Le lancement de l'élection s'effectue comme suit :

1. *P_1 envoie un message d'election à tous les processus qui ont des identifiants supérieurs au sien.*
2. *Si personne ne répond, P_1 gagne l'election et devient coordinateur.*
3. *Si un processus P_2 répond, P_2 devient responsable de continuer l'election (retour au point 1), le travail de P_1 est terminé.*

6) Le protocole de commit à deux phases

Dans le protocole de commit à deux phases, la deuxième phase va permettre de voter pour le « commit » ou « l'Abort » d'une transaction.

- Expliquez dans le contexte d'une transaction imbriquée et répartie comment s'effectue ce vote dans un modèle « hiérarchique » et dans un modèle « à plat ».
- Donnez les avantages et inconvénients des deux modèles.

Le protocole de commit à deux phases dans les transactions imbriquées se déroule comme suit :
Phase 1

- *Les sous-transactions effectuent le traitement et envoient un **Abort** ou un **Commit prévisionnel** à la transaction parente.*
- *La transaction de niveau le plus haut, (transaction racine à la liste de toutes les sous-transactions, statuts et coordinateurs intermédiaires. Elle joue le rôle de coordinateur central.*
- *Elle demande aux participants à la transaction de voter*

Phase 2

- *C'est la phase du vote*

*Le vote peut-être **hiérarchique** soit **à plat**.*

Vote hiérarchique

C'est un protocole à multi-niveaux imbriqués.

- *Le coordinateur racine communique avec les coordinateurs de ses sous-txn filles et leur envoie un message de « CanCommit ?» aux transactions ayant effectuées un « Commit provisionnels ».*
- *Les coordinateurs intermédiaires font la même chose pour leur sous- txn filles.*

Ceci est fait pour l'ensemble de l'arbre.
Le coordinateur racine prendra la décision finale de Commit ou d'Abort.

Vote à plat

- *Le coordinateur racine communique avec tous les coordinateurs intermédiaires et leur envoie un message de « CanCommit ? » aux « Commit provisionnels ».*
- *Le coordinateur racine récupère les retours et prendra la décision finale de « Commit » ou d' « Abort ».*

Il gère donc la liste des participants à la TXN.

Avantages inconvénients

Hiérarchique :

☺ *Ne gère que les coordinateurs de ses sous-txn filles.*
☹ *Algorithme plus lent (beaucoup de messages entre les nœuds de l'arbre).*

A plat :

☺ *Algorithme très performant*
☹ *La racine gère tous les coordinateurs.*

7) Processus & Processeurs

Lors de la conception d'algorithmes d'allocation de processus à un processeur, plusieurs choix s'offrent aux développeurs :

1. Algorithme déterministe versus algorithme heuristique.
2. Algorithme centralisé versus algorithme décentralisé.
3. Algorithme optimal versus algorithme sous-optimal.

Pour **un des trois cas** précédents, décrivez la différence entre les deux choix.

Algorithme déterministe versus algorithme heuristique.

- *Déterministe fournit la solution, l'heuristique fournit une solution possible.*

Algorithme centralisé versus algorithme décentralisé.

- *Toutes les informations nécessaires sont centralisées dans le premier cas alors qu'elles sont réparties dans le second cas.*

Algorithme optimal versus algorithme sous-optimal.

- *L'optimal est le meilleur algorithme possible, l'optimal étant le plus acceptable.*

Dans le contexte de la migration de processus vers une machine moins chargée, donnez les trois critères qui permettent d'évaluer la charge d'une machine.

1. *Utilisation CPU.*
2. *Charge mémoire.*
3. *Bande passante réseau utilisée.*

Comment peut-on résoudre les pannes byzantines d'un processeur dans un environnement ou quatre processeurs sont en place et communiquent ?

- *Chaque processeur envoie son identifiant à tous les processeurs et à lui même.*
- *Il construit la liste des identifiants.*
- *Chaque processeur envoie la liste à tous les autres processeurs.*
- *Chaque processeur vérifie les listes qu'il a reçu et identifie le processeur qui a un problème en comparant les identifiants dans toutes les listes.*

Est-ce que cet algo fonctionne dans tous les cas ?
Non.
Si non, quel est le ratio, de processeurs sains vs processeurs en panne, nécessaire au bon fonctionnement de la méthode.
Si nous avons n processeurs, il faut que le nombre de processeurs sains soit strictement supérieur à 2/3 de n.

8) Système de fichiers distribués

Quelle est la différence essentielle entre **sémantique UNIX** et **sémantique de session?** Dans un système de fichiers distribué comment peut-on être certain d'obtenir une sémantique Unix ? Dans ce cas, il existe tout de même un problème mineur dû au réseau. Voyez-vous lequel ?

La sémantique UNIX
Le système assure que l'on obtient dans tous les cas la dernière version du fichier demandé. C'est-à-dire que la dernière mise à jour a été prise en compte.
La sémantique de session
Les modifications apportées à un fichier sont renvoyées au serveur à la fermeture du fichier. Seuls les processus locaux voient les modifications.

Comment obtenir une sémantique UNIX dans un système de fichiers distribué
Pour être certain qu'un système de fichier assure la sémantique UNIX, il faut que toutes les opérations passent par un serveur de fichiers central et qu'aucun mécanisme de cache ne soit implémenté.

Un problème mineur dû au réseau?
Dans le cas précédent, un problème mineur persiste: une lecture peut-être demandée après une écriture mais à cause du réseau arriver sur le serveur central avant l'écriture. Dans ce cas, la valeur retournée par le serveur est l'ancienne.

Examen - UV19302
Systèmes et applications répartis – B4

Vendredi 13 septembre 2002

Aucune notes personnelles ou documents ne sont autorisés.

1) Les systèmes répartis

Expliquez ce que sont des serveurs « avec état » et « sans état ». Quels sont les avantages et les inconvénients de ces deux types de serveurs.

Un serveur avec état conserve des informations liées à la requête d'un client jusqu'à que cette requête soit totalement satisfaite. Un serveur sans état (stateless) ne conserve aucune information.

Les avantages du serveur avec état:

- *Exécution plus rapide (les informations nécessaires sont connues par le serveur).*
- *Moins d'informations à transférer sur le réseau.*
- *Gestion de la concurrence.*

Les désavantages du serveur avec état:

- *Pas de tolérance aux pannes.*
- *Dans l'hypothèse où un client a un problème, comment le serveur est-il averti?*
- *Dans l'hypothèse où le serveur a un problème, comment les clients sont-ils avertis?*

Les avantages du serveur sans état:

- *Tolérant aux pannes.*

Les désavantages du serveur sans état:

- *Plus lent à l'exécution. Les paramètres doivent être interprétés à chaque fois, aucune anticipation n'est possible.*

2) L'ordonnancement

2.1) Quel est le rôle d'un ordonnanceur ou *« scheduler »* ? Dans ce cadre, vous expliquerez les notions suivantes:

- Temps réel fourni par le matériel ou le logiciel?
- Préemptif ou coopératif.
- Dynamique ou statique.
- Centralisé ou décentralisé.
- Notion de priorité.

Un ordonnanceur gère et régule l'exécution des tâches. Il va déterminer qui doit s'exécuter, quand et où. Il se base sur une prioritisation des tâches à exécuter. Le noyau recense les tâches qui sont à l'état « ACTIVE » et exécute celle qui a la priorité la plus haute. Les ordonnanceurs sont basés sur des algorithmes complexes, plusieurs approches peuvent être envisagées et combinées:

- ***Temps réel fourni par le matériel ou le logiciel ?***

*Il y a beaucoup d'avantages à ce que le **matériel** fournisse les mécanismes d'ordonnancement. Les plus importants sont la fiabilité, la sécurité et de moindres efforts de développement. Sinon, l'ordonnateur devra être programmé (**logiciel**).*

- ***Préemptif ou coopératif.***

*Le mode **préemptif** est un mode interruptible. C'est-à-dire que l'ordonnanceur gère le temps d'exécution d'une tâche. Le mode **coopératif** est un mode non interruptible, la tâche s'exécute jusqu'à ce qu'elle décide de s'arrêter et de rendre la main. Les deux modes peuvent être codés dans un même ordonnanceur (ex : coopératif pour les tâches systèmes et préemptif pour les tâches utilisateurs).*

- ***Dynamique ou statique.***

***Dynamique** signifie que les décisions de l'ordonnanceur sont prises lors de l'exécution. **Statique** signifie que les décisions sont planifiées avant l'exécution.*

- ***Centralisé ou décentralisé.***

*On parle de **centralisé** lorsque le système gère l'ordonnancement. En **décentralisé**, chaque processeur du système gère l'ordonnancement.*

- ***Notion de priorité.***

*On prioritirise les tâches à exécuter. De ce fait l'ordonnanceur pourra établir ses décisions en fonction de la **priorité** d'une tâche. En règle générale, les tâches les plus prioritaires s'exécuteront avant les moins prioritaires.*

2.2) Décrivez un algorithme qui permet de gérer les priorités et privilégie les appels systèmes (priorités les plus hautes).

Ce type d'algorithme distingue deux types de tâches : les tâches systèmes (noyau) et les tâches utilisateurs. Afin de privilégier les appels systèmes, les tâches systèmes sont gérées en mode coopératif (la tâche conserve l'usage du processeur le temps qu'elle veut) alors que les tâches utilisateurs sont gérées en mode préemptif (une tranche de temps est allouée par l'ordonnaceur).

3) La mémoire dans Chorus

3.1) Qu'appelle-t-on:

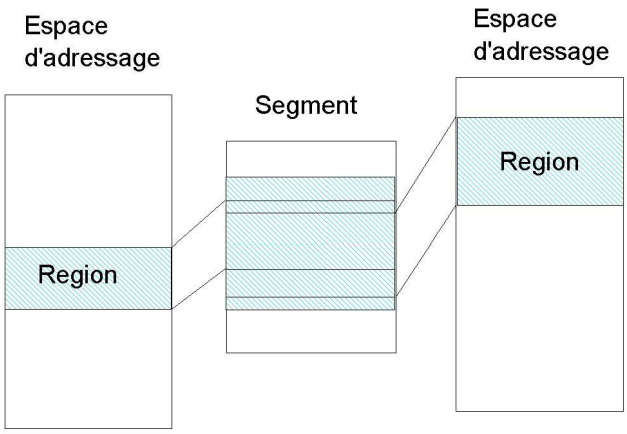
- Espace d'adressage.
- Segment.
- Région.

***Espace d'adressage**: C'est l'espace mémoire disponible pour un utilisateur (acteur).*

***Segment** : C'est l'unité de représentation des données. Les données peuvent être temporaires ou permanentes. Dans Chorus, le segment est identifié par une capacité.*

***Région** : C'est l'unité d'accès à la mémoire. Ce sont des zones qui se situent dans l'espace d'adressage d'un acteur. C'est la partie visible en mémoire d'un segment, qui permet de mettre en correspondance un segment dans l'espace d'adressage d'un acteur.*

3.2) Au travers d'un graphique, montrez comment deux acteurs Chorus peuvent se partager un segment pour leur région.



4) Système de fichiers distribué

4.1) Quelles sont les différences entre service de fichiers et serveur de fichiers?

Un service de fichiers *(File Service) spécifie ce qui est offert aux utilisateurs pour gérer les fichiers. Il définit les primitives disponibles, comment les utiliser, les actions qui peuvent être entreprises. C'est l'interface que va utiliser le client.*

Un serveur de fichiers *(File Server) est un process qui aide à l'implémentation du système de fichiers. Un système peut avoir un ou plusieurs serveurs de fichiers. Il est transparent pour l'utilisateur et pour le système.*

4.2) Un système de fichiers distribué est composé généralement de deux composants distincts, décrivez ces deux composants.

Un système de fichiers a généralement deux composants distincts:

5. *Le **service de fichiers** qui gère les opérations sur les fichiers (attribut, structure).*
6. *Le **service des répertoires** qui gère les opérations sur les répertoires (nom, longueur, taille, extension, arborescence).*

5) La validation (*Commit*)

Expliquer comment un protocole de validation à deux phases pour les transactions imbriquées assure que si une transaction de plus haut niveau (*top-level*) valide la transaction alors ses descendants ont obligatoirement validé ou abandonné leur traitement.

Le protocole de commit à deux phases dans les transactions imbriquées se déroule comme suit :

Phase 1

- *Les sous-transactions effectuent le traitement et envoie un **Abort** ou un **Commit prévisionnel** à la transaction parente.*

- *La transaction de niveau le plus haut, (transaction racine à la liste de toutes les sous-transactions, statuts et coordinateurs intermédiaires. Elle joue le rôle de coordinateur central.*
- *Elle demande aux participants à la transaction de voter*

Phase 2

- *C'est la phase du vote*

*Le vote peut-être **hiérarchique** soit **à plat**.*

Vote hiérarchique

C'est un protocole à multi-niveaux imbriqués.

- *Le coordinateur racine communique avec les coordinateurs de ses sous-txn filles et leur envoie un message de « CanCommit ?» aux transactions ayant effectuées un « Commit provisionnels ».*
- *Les coordinateurs intermédiaires font la même chose pour leur sous- txn filles.*

Ceci est fait pour l'ensemble de l'arbre.
Le coordinateur racine prendra la décision finale de Commit ou d'Abort.

Vote à plat

- *Le coordinateur racine communique avec tous les coordinateurs intermédiaires et leur envoie un message de « CanCommit ? » aux « Commit provisionnels ».*
- *Le coordinateur racine récupère les retours et prendra la décision finale de « Commit » ou d' « Abort ».*

Il gère donc la liste des participants à la TXN.

Avantages inconvénients

Hierarchique :

☺ *Ne gère que les coordinateurs de ses sous-txn filles.*
☹ *Algorithme plus lent (beaucoup de messages entre les nœuds de l'arbre).*

A plat :

☺ *Algorithme très performant*
☹ *La racine gère tous les coordinateurs.*

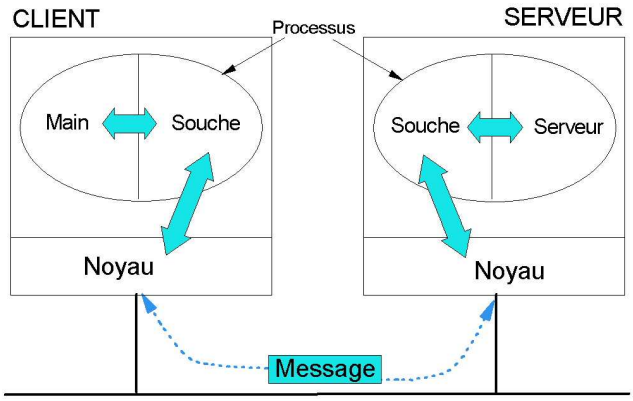
6) Remote Procedure Call

6.1) Quelles sont les différences entre le modèle Client-Serveur et RPC?

*Le modèle **client-serveur** est un modèle qui s'appuie sur des communications de type entrée/sortie (SEND/RECEIVE) dont le but est de rendre le comportement des systèmes répartis identique à celui des systèmes centralisés. Avec le **RPC**, aucun envoi de messages ou requête d'entrée /sortie n'est visible par le programmeur. Les appels aux procédures distantes se font de manière transparente, comme si les procédures étaient exécutées localement.*

6.2) Comment fonctionne un appel RPC (souche client, souche serveur, gestion de la pile).

Afin d'assurer le principe de transparence, le mécanisme RPC met en place des versions différentes d'une procédure selon qu'elle est locale ou distante.
Pour effectuer un appel RPC :
La version cliente de la procédure *est constituée d'une souche cliente (« client stub ») qui est placée dans la librairie locale. L'appel se fait ensuite comme avec une procédure locale (la pile est gérée localement) et l'exécution est prise en charge par le noyau. Par contre, les paramètres ne sont pas stockés dans les registres et aucune donnée n'est demandée au noyau. Les paramètres sont stockés dans un message construit par le noyau et envoyé au serveur distant qui contient la partie serveur de la procédure. Le client attend ensuite que le serveur lui réponde.*
La version serveur de la procédure *est constituée d'une souche serveur et de la procédure. Lorsque le message parvient au serveur, le noyau passe le message à la souche correspondante (qui est en attente de messages). La souche extrait les paramètres du message, renseigne la pile et appelle la procédure qui pense être appelée par un client local. Lorsque la procédure a terminé son traitement, elle renseigne la pile et rend la main à la souche (voir schéma suivant).*



7) Les transactions

7.1) Qu'est-ce qu'une transaction simple? Qu'est-ce qu'une transaction imbriquée?
*Une **transaction simple** est une séquence d'opérations qui s'exécute sur un serveur et qui est garantie comme atomique. Elle permet ainsi de préserver la cohérence des informations manipulées par des activités concurrentes.*

*Une **transaction imbriquée** est structurée à partie d'un ensemble d'autres transactions (simples ou imbriquées).*

7.2) Qu'appelle-t-on concurrence optimiste?
La concurrence optimiste permet aux transactions d'effectuer leur traitement jusqu'à qu'elles soient capables d'entériner leur traitement (COMMIT). Ensuite un contrôle de cohérence est effectué pour voir si des opérations conflictuelles ont été effectuées.

Corrigé Examen - UV19302
Systèmes et applications répartis – B4

Session du 28 juin 2002

Aucune notes personnelles ou documents ne sont autorisés.

1) Les systèmes répartis

On peut employer le terme de système réparti ou distribué lorsque le système dispose d'un ensemble de processus communicants, installés sur une architecture de processeurs, dans le but de résoudre en coopération un problème commun.

1.1) Quelles seraient les raisons qui vous feraient concevoir un système réparti?

Un système réparti est un ensemble d'ordinateurs indépendants reliés à un réseau et donnant l'impression à l'utilisateur d'un système monoprocesseur.

Les justifications d'un tel système sont multiples, pour ne citer que les plus importantes:

- *Il est plus économique d'utiliser un ensemble de petites machines (PCs ou stations) que de travailler sur un grand système.*
- *Les développeurs ne sont plus groupés sur le même site, mais peuvent être répartis à travers le monde. Ils ont la possibilité de partager puissance de calcul et données en même temps;*
- *Un ensemble de petites machines reliées en réseau ont une puissance supérieure comparé à un système centralisé.*
- *La tolérance aux pannes est plus importante dans un système distribué, une machine tombant en panne peut facilement être remplacée et toutes les autres continuent à fonctionner.*
- *L'extension du système (scalability) se fait de manière simple par l'ajout de nouvelles machines sur le réseau.*

1.2) Les objectifs majeurs de tout système réparti sont les suivants:

- La transparence (accès concurrent, migration, parallélisme).
- La flexibilité.
- La fiabilité et la disponibilité.
- La performance.
- L'évolutivité (scalabilité).

Vous expliquerez en quelques lignes ce que représentent ces termes.

La transparence, c*'est masquer la répartition au travers des spécifications suivantes:*

- *La localisation des ressources n'est pas perceptible.*
- *La migration des ressources d'un lieu à un autre n'implique aucune modification de l'environnement utilisateur.*
- *Le nombre de ressources dupliquées n'est pas connu (invisibilité).*
- *La concurrence d'accès aux ressources n'est pas perceptible.*
- *Invisibilité du parallélisme sous-jacent offert par l'ensemble de l'environnement d'exécution.*

La flexibilité *ou modularité. C'est faire en sorte que le système soit le plus modulaire possible. Pour ce faire, les systèmes répartis sont construits par l'ajout de services reposant sur un micro-noyau. Si de nouveaux services doivent être implémentés, ils sont ajoutés au système.*

La fiabilité et la disponibilité *d'un système réparti repose essentiellement sur l'indépendance des composants du système et la redondance des composants logiciels et matériels. Pour qu'un système soit fiable et disponible, il faut qu'il soit:*

- *Tolérant aux pannes.*
- *D'un fonctionnement sûr.*
- *Sécurisé.*

La performance *d'un système réparti c'est sa faculté à gérer un grand nombre:*

- *D'utilisateurs.*
- *De machines.*
- *De services.*
- *De trafic.*
- *De processus.*

Les métriques de la performance sont:

- *Le temps de réponse.*
- *Le débit.*
- *L'utilisation du système.*
- *La capacité réseau utilisée.*

L'évolutivité (scalability) *c'est la faculté qu'a un système à rester efficace et performant malgré la croissance du nombre d'utilisateurs et du nombre de ressources.*

2) Chorus

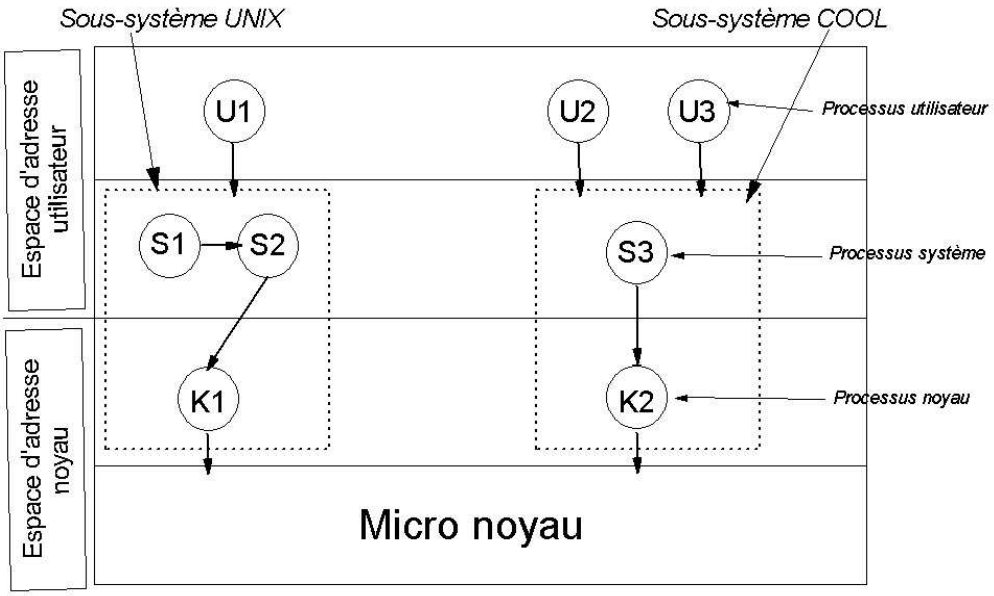
Une des particularités de Chorus est l'implémentation d'un sous-système UNIX. Que signifie dans le cas de Chorus le terme sous-système? Quelle est la raison majeure de ce type d'implémentation? Aidez-vous d'un schéma pour représenter le système Chorus et ses différents composants que sont les sous-systèmes, les applications, le noyau.

Le sous-système

Un sous-système est un ensemble de processus système et processus noyau qui collaborent à la réalisation d'un objectif commun.
La particularité d'un sous-système est de fournir aux utilisateurs une interface bien identifiée et définie. Un processus particulier du sous-sytème gère et contrôle les opérations qui se déroulent dans le sous-système.

Les raisons majeures *de ce type d'implémentation sont la réutilisation d'application et la communication entre applications s'exécutant sur des plateformes logicielles différentes.*

Schéma de Chorus



CHORUS

3) Système de fichiers distribués

Quelle est la différence essentielle entre **sémantique UNIX** et **sémantique de session?** Dans un système de fichiers distribué comment peut-on être certain d'obtenir une sémantique Unix? Dans ce cas, il existe tout de même un problème mineur dû au réseau. Voyez-vous lequel?

La sémantique UNIX

Le système assure que l'on obtient dans tous les cas la dernière version du fichier demandé. C'est-à-dire que la dernière mise à jour a été prise en compte.

La sémantique de session

Les modifications apportées à un fichier sont renvoyées au serveur à la fermeture du fichier. Seuls les processus locaux voient les modifications.

Comment obtenir une sémantique UNIX dans un système de fichiers distribué

Pour être certain qu'un système de fichier assure la sémantique UNIX, il faut que toutes les opérations passent par un serveur de fichiers central et qu'aucun mécanisme de cache ne soit implémenté.

Un problème mineur dû au réseau?

Dans le cas précédent, un problème mineur persiste: une lecture peut-être demandée après une écriture mais à cause du réseau arriver sur le serveur central avant l'écriture. Dans ce cas, la valeur retournée par le serveur est l'ancienne.

4) Transactionnel réparti

Présentez la problématique liée aux étreintes fatales (*dead-lock*) dans les systèmes répartis. Décrivez une méthode permettant de résoudre problématique.

Les dead-locks dans les systèmes répartis sont au moins identiques sinon pires que ceux détectés dans des systèmes non répartis. Deux types de dead-lock peuvent être envisagés: les dead-locks dus à la communication et les dead-locks dus aux ressources.

Un dead-lock de communication peut survenir lorsque:

Un processus A tente d'envoyer un message au processus B qui tente d'envoyer un message au processus C qui tente d'envoyer un message au processus A (A->B->C->A).

Un dead-lock dû aux ressources peut survenir lorsque des processus tentent d'accéder aux mêmes ressources d'E/S, fichiers, verrous, ou toutes autres ressources.

Dans tous les cas un dead-lock est dû à un cycle lors des demandes d'accès.

Différentes stratégies existent:

1. *L'algorithme de l'autruche: ne rien faire.*
2. *Détection : on laisse les dead-locks arriver puis on les détecte et on tente de les éliminer.*
3. *Prévention : Faire en sorte que statistiquement des dead-locks ne puissent pas arriver.*
4. *Elimination : L'allocation des ressources est telle qu'aucun dead-lock ne peut arriver.*

Une méthode (parmi d'autres)

C'est la méthode du détecteur global. On crée des graphes d'attente locaux qui sont les reflets des différentes actions et demandes des processus se déroulant localement. Ces graphes d'attente locaux sont régulièrement envoyés à un serveur central (le détecteur

global). Ce dernier crée un graphe d'attente global et surveille les cycles qui pourraient se produire dans ce graphe. Dès qu'un cycle est repéré, il se charge de stopper le processus en cause. Bien entendu cette méthode n'est pas la plus efficace puisque comme tout mécanisme centralisé nous sommes en présence d'un point de panne unique (Single Point of Failure).

5) Il n'y a pas de question 5)

6) Remote Procedure Call

6.1) Quelles sont les différences entre le modèle Client-Serveur et RPC?

*Le modèle **client-serveur** est un modèle qui s'appuie sur des communications de type entrée/sortie (SEND/RECEIVE) dont le but est de rendre le comportement des systèmes répartis identique à celui des systèmes centralisés. Avec le **RPC**, aucun envoi de messages ou requête d'entrée /sortie n'est visible par le programmeur. Les appels aux procédures distantes se font de manière transparente, comme si les procédures étaient exécutées localement.*

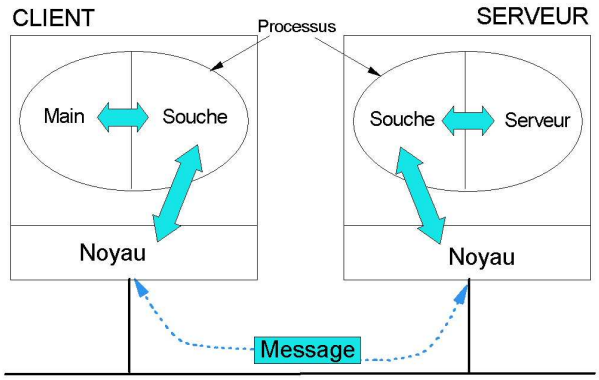
6.2) Comment fonctionne un appel RPC (souche client, souche serveur, gestion de la pile).

Afin d'assurer le principe de transparence, le mécanisme RPC met en place des versions différentes d'une procédure selon qu'elle est locale ou distante.

Pour effectuer un appel RPC :

***La version cliente de la procédure** est constituée d'une souche cliente (« client stub ») qui est placée dans la librairie locale. L'appel se fait ensuite comme avec une procédure locale (la pile est gérée localement) et l'exécution est prise en charge par le noyau. Par contre, les paramètres ne sont pas stockés dans les registres et aucune donnée n'est demandée au noyau. Les paramètres sont stockés dans un message construit par le noyau et envoyé au serveur distant qui contient la partie serveur de la procédure. Le client attend ensuite que le serveur lui réponde.*

***La version serveur de la procédure** est constituée d'une souche serveur et de la procédure. Lorsque le message parvient au serveur, le noyau passe le message à la souche correspondante (qui est en attente de messages).La souche extrait les paramètres du message, renseigne la pile et appelle la procédure qui pense être appelée par un client local. Lorsque la procédure a terminé son traitement, elle renseigne la pile et rend la main à la souche (voir schéma suivant).*



7) L'exclusion mutuelle

Lorsqu'un processus doit modifier des structures de données partagées, il entre d'abord en zone critique en s'assurant qu'aucun processus n'est en train d'utiliser la structure de données à laquelle il veut accéder. Ces zones critiques sont protégées à l'aide de mécanismes particuliers. Trois mécanismes d'exclusion mutuelle sont fréquemment utilisés dans les systèmes répartis:

4. L'algorithme centralisé (un coordinateur assure le contrôle).
5. L'algorithme réparti (le contrôle est réparti sur les différents processus).
6. L'algorithme anneau à jeton (le contrôle est réparti à l'aide d'un jeton sur les différents processus).

Complétez le tableau en précisant les problèmes éventuels liés au type d'algorithme et commentez les résultats fournis dans les colonnes **Messages par entrée/sortie** et **Délai avant d'entrer dans la zone**.

Mécanisme	**Messages par entrée/sortie**	**Délai avant entrée dans la zone** (en temps messages)	**Problèmes**
Centralisé	3	2	
Distribué	2(n-1)	2(n-1)	
En anneau	1 à l'infini	0 à n-1	

L'algorithme centralisé :

On simule ce qui est fait dans un système centralisé. Un coordinateur central gère les ressources. C'est une application du principe utilisé sur un système avec un seul processeur. J'ai deux processus P1 et P2 qui veulent accéder à une même ressource. P1 demande au coordinateur si la ressource est libre, si elle est libre, le coordinateur accepte, le processus entre alors en zone critique. Si P2 veut utiliser la ressource, il demande au coordinateur si la ressource est libre, le coordinateur sait que P1 est en zone critique, il met à jour la queue de gestion de la ressource en rajoutant P2 en liste d'attente et refuse l'accès. Lorsque P1 sort de la zone critique, elle est libérée, le coordinateur sait que P2 peut rentrer en zone critique, il accepte la demande et met à jour la file d'attente.

Message en entrée/sortie :

- *Trois messages au maximum pour demander une ressource.*

Délai avant entrée dans la zone :

- *Deux messages au maximum avant d'entrer en zone critique.*

Les problèmes sont :

- *Un seul point de panne (SPF).*
- *Le coordinateur est dans le cas d'un système important un goulet d'étranglement.*

L'algorithme distribué :

L'exclusion mutuelle répartie résout le problème de l'unique point de panne. L'algorithme de Lamport sera utile pour la cohérence des temps. En effet pour la mise en place d'un MUTEX réparti il faut être certain de l'ordre. Quand un processus veut entrer dans la zone critique, il construit un message contenant le nom de la zone, son N° et l'heure courante. Il envoie le message à tous les processus y compris à lui-même. Un accusé de réception sera renvoyé par

l'ensemble des processus (maintien de la fiabilité du système). Les processus répondront en fonction de leur état lié à la zone considérée (3 cas) :

4. *Le récepteur n'est pas dans la zone et ne veut pas y rentrer, il répond OK.*
5. *Le récepteur est dans la zone, il ne répond pas. Il charge la demande dans sa file d'attente.*
6. *Le récepteur veut entrer dans la zone, il compare l'heure du message avec celle du message qu'il a déjà envoyé. Si l'heure est inférieure, il répond OK sinon il place la demande dans sa file d'attente et il ne répond pas.*

Quand sa demande est partie, le processus attend son acceptation par tous les autres processus, il entre ensuite en zone critique. Quand il en sort, il répond OK aux processus qui sont dans sa file d'attente et les enlève de la file.

Message en entrée/sortie :

- *Avec n processus, 2(n-1) messages au maximum pour demander une ressource (n-1 requêtes aux processus et n-1 autorisation).*

Délai avant entrée dans la zone :

- *Avec n processus, 2(n-1) messages maximum pour demander une ressource (n-1 requêtes aux processus et n-1 autorisation).*

Les problèmes sont :

- *Un processus en panne peut-être interprété comme un refus .*
- *Il n'y a plus un seul point de panne mais n (algo n fois pire que le précédent).*

L'algorithme anneau à jeton :

Il utilise le principe de l'anneau à jeton. Un anneau logique est construit. Le processus rentrera en zone critique si et seulement si il a le jeton. À un instant t, seul un processus est détenteur du jeton. Chaque processus constituant l'anneau passe le jeton à son successeur. Le successeur renvoie un accusé de réception.

Message en entrée/sortie :

- *1 message si le jeton est récupéré par le processus dès qu'il veut rentrer en zone critique, l'infini si personne n'est intéressé par le jeton.*

Délai avant entrée dans la zone :

- *1 message si le jeton est récupéré par le processus dès qu'il veut rentrer en zone critique, n-1 messages si le jeton vient juste de passer devant le processus (Il doit faire le tour des n-1 processus avant de revenir).*

Les problèmes sont :

- *La gestion de la perte du jeton dans l'hypothèse où le processus qui a le jeton tombe en panne.*

8) Corba

L'OMG a défini une architecture standard pour les environnements à objets distribués: OMA (*Object Management Architecture*).

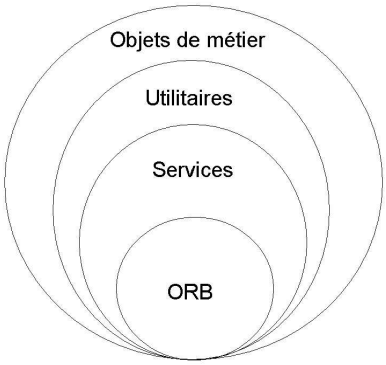
L'OMA comprend:

- ORB.
- Services.
- Utilitaires.
- Objets métier (relatifs à un domaine particulier).

Représentez au moyen d'un schéma cette architecture.

L'OMG définie une architecture standard pour les environnements objets distribués. Cette architecture est nommée OMA pour Object Management Architecture.
Elle comprend :

- ***ORB** : Des mécanismes protocolaires et le langage qui permettent aux objets distribués d'interopérer.*
- ***Services** : Un ensemble de fonctionnalités système pour les applications objets (ensemble de services pour des objets).*
- ***Utilitaires** : Un ensemble de fonctionnalités communes pour la manipulation de composants à un niveau applicatif (ensemble de services pour des applications).*
- ***Objets de métier** : Un ensemble d'objets dérivant de comportements standard que l'on retrouve dans les applications (ex : un client, une commande, un concurrent, un patient, un règlement, etc.).*



OMA Object Management Architecture

La couche service est composée de 16 services. Présentez en 5 lignes l'un d'entre eux.

Les 16 services sont les suivants :

1. *Nommage.*
2. *Courtier.*
3. *Cycle de vie.*
4. *Evènement.*
5. *Transaction.*
6. *Concurrence.*
7. *Persistance.*
8. *Base de données objet.*
9. *Requêtes.*
10. *Relation.*
11. *Externalisation.*
12. *Licence.*
13. *Propriétés.*
14. *Temps.*
15. *Sécurité.*
16. *Changement.*

Le service transaction *est l'un des services les plus importants de Corba et un des plus difficiles à mettre en œuvre. Il assure la gestion de la cohérence des objets multithreadés avec un protocole de validation à deux phases.*

9) L'invocation asynchrone avec futur dans les objets répartis

Expliquez en quelques lignes le fonctionnement du mécanisme d'invocation asynchrone avec futur (implicite et explicite).

Le concept d'objet futur

Après avoir lancé l'exécution concurrente d'une méthode, le contrôle doit être rendu à l'appelant. Ce dernier s'attend à recevoir un objet comme réponse. Or la réponse peut très bien ne pas être immédiatement disponible. Pour contourner ce problème, un objet spécifique est créé c'est l'objet ***futur****.*

Un objet futur est un objet particulier, destiné à contenir la réponse lorsque celle-ci sera disponible quelque part dans le futur. Tant que l'appelant n'a pas besoin du contenu de la réponse il peut manipuler l'objet futur comme tel (stockage par référence dans une structure de données ou passage en paramètre par exemple). Ce n'est que lorsque le contenu de la réponse est vraiment nécessaire qu'il risque d'y avoir attente.

Lorsque le client fournit explicitement l'objet futur où il s'attend à trouver la réponse, on parle de ***futur explicite****. Le client doit alors disposer d'un moyen de se mettre en attente de la réponse, lorsque le contenu de celle-ci est requis.*

Si au contraire le client reçoit et manipule un objet futur sans en être conscient, on parle de ***futur implicite****. Dans ce dernier cas, le run-time du système doit se charger de bloquer tout client essayant d'accéder à un objet futur dont le contenu n'est pas encore disponible.*

Dans le cas de l'invocation asynchrone avec futur explicite, précisez qui construit l'objet résultat.

Le client (voir au-dessus).