جامعة هواري بومدين للعلوم و التكنولوجيا
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique
Département d'Informatique

Concours d'accès au Doctorat LMD Informatique 2019 – 2020

Le 26/10/2019

Matière 2 : Réseaux + au choix : Systèmes distribués/ Sécurité des systèmes,
Coefficient **3**, **Spécialité : RSI**

Partie : Systèmes distribués
Durée : 1h.

Exercice (10 pts=1+ 1+2+6)

On considère un système distribué composé de *N* processus *P(i), i = 1, N* où *i* est l'identité du processus *P(i)* connectés selon une topologie *physique* connexe. Ces processus sont organisés selon une arborescence logique *supposée optimale et les feuilles sont organisées selon un anneau unidirectionnel supposé optimal, établies au préalable.*

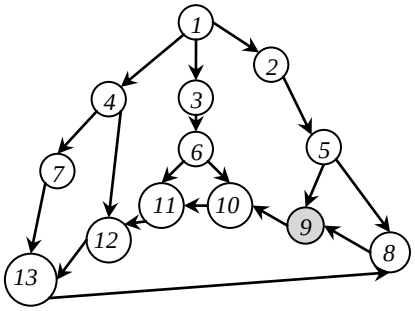
On désire implémenter le modèle du client/serveur sur cette structure de telle sorte qu'un processus feuille de l'arbre nommé S connu de tous est le serveur et les autres sont des clients. Un client qui désire un service envoie une requête qui indique le numéro du service dans l'intervalle *[0, M-1]* qui suit la structure établie vers le serveur et le message de réponse (qui contient le numéro de service comme réponse) suit le chemin permis par la structure vers le client.

- Donner un exemple typique de schéma qui représente cette structure.
- Donner les structures de données de base à utiliser et expliquer leurs rôles
- Donner les idées de base de votre solution.
- Ecrire une solution distribuée à l'aide du langage algorithmique.

Solution de l'épreuve Systèmes distribués

Exercice (10 pts=1+ 1+2+6)

- **<u>Un exemple type de la structure</u>**



- **<u>Structures de base et rôles</u>**

$Succ_i$= … ;//il s'agit du successeur de i dans l'anneau
$pere_i$: entier ; // il s'agit du père de i dans l'arborescence
$fils_i$: ensemble// il s'agit de l'ensemble des fils de i dans l'arborescence

Toutes ces variables sont supposées déjà initialisées.

Un paramètre important nommé *routeur* est utilisé dans les messages de requête et de réponse. Si le demandeur du service est un nœud non feuille de l'arborescence, *routeur* sera alors initialisée à *k* lorsque la requête arrive au premier nœud feuille *k*. *k* est donc le nœud par lequel la réponse remonte dans l'arborescence vers le destinataire.

- **<u>Idées de base</u>**

- L'algorithme utilise une structure d'arborescence dont les feuilles sont organisées en un anneau unidirectionnel, le tout est optimal. Tous les nœuds peuvent formuler une requête sauf le serveur.
- Si le demandeur de service est un nœud non feuille de l'arborescence, la requête descend jusqu'au premier nœud feuille de l'arborescence, celui-ci met son nom dans une variable *routeur* qui est un paramètre du message de requête. Ce paramètre est aussi véhiculé dans le message de réponse. Il sert à retrouver ce nœud feuille lorsque la réponse circule dans l'anneau, ce qui permet à ce dernier de remonter la réponse vers la hiérarchie à travers son père sur le chemin du destinataire.
- Si la requête est générée par un nœud feuille de l'arborescence, celle-ci circule dans l'anneau jusqu'à l'arrivée au serveur, la réponse continue à circuler sur l'anneau jusqu'au nœud destinataire.

- **<u>Une solution distribuée</u>** : La solution est symétrique

<u>Contexte local de Pi</u>
S=…. ;
M=…. ;
N=…. ; // c'est le nombre de nœuds de l'arborescence ou sinon une grande valeur.
$Succ_i$= … ;// déjà initialisé au niveau des nœuds de l'anneau.
$pere_i$: entier ; // déjà initialisé
$fils_i$: ensemble// déjà initialisé
$nums_i$: entier ;
<u>Messages</u>
requete (id : entier, nums : entier, routeur : entier) ;
reponse (id : entier, nums : entier, routeur : entier) ;

Lorsqu'un processus i demande un service ;
Début
Si (i<>S) Alors
$nums_i$: rand () Mod M ;
Si ($fils_i$<> ϕ) Alors soit k $\in$ $fils_i$:envoyer (requete(i, $nums_i$, 0) à k
Sinon envoyer (requete (i, $nums_i$, 0) $succ_i$
Fsi
Fin ;

A la réception d'un message requete (id, nums, routeur) de j
Début
Si ($fils_i$<> ϕ) Alors soit k $\in$ $fils_i$:envoyer (requete(id, nums, routeur)) à k
Sinon
Si (i<>S) Alors Si (j=$pere_i$) Alors routeur :=i ; envoyer (requete (id, nums, routeur)) à $succ_i$
// La requête vient d'atteindre la feuille
Sinon envoyer (requete (id, nums, routeur)) à $succ_i$
Fsi
Sinon $nums_i$:=traiter_requete(id, nums) ;
Si (routeur<>i) Alors envoyer (reponse(id, $nums_i$, routeur)) à $succ_i$
Sinon envoyer (reponse(id, $nums_i$, routeur)) à $pere_i$
Fsi
Fsi
Fin ;

A la réception d'un message reponse (id, nums, routeur) de j
Début
Si (i=id) Alors memoriser(nums)
Sinon
Si ($fils_i$=ϕ) Alors
Si (i<>routeur)Alors envoyer (reponse (id, nums, routeur)) à $succ_i$
// La réponse circule sur l'anneau
Sinon envoyer (reponse (id, nums, routeur)) à $pere_i$
// La réponse commence à monter dans l'arborescence
Fsi
Sinon envoyer (reponse(id, nums, routeur) à $pere_i$
// La réponse continue à remonter dans l'arborescence
Fsi
Fin ;

- Une autre solution

Contexte local de Pi
S=…. ;
M=…. ;
N=…. ; // c'est le nombre de nœuds de l'arborescence ou sinon une grande valeur.
$Succ_i$= … ;// déjà initialisé au niveau des nœuds de l'anneau.
$pere_i$: entier ; // déjà initialisé
$fils_i$: ensemble// déjà initialisé
R_i : Tableau [1..N] de entier ;// initialisé à faux ;
// R_i[k]= vrai signifie que le nœud k est un ancêtre du nœud feuille i.
$nums_i$: entier
Messages
requete (id : entier, nums : entier) et *reponse (id : entier, nums : entier) ;*

Lorsqu'un processus i demande un service ;
Début
Si (i<>S) Alors
$nums_i$: rand () Mod M ;
Si ($fils_i$<> ϕ) Alors soit k $\in$ $fils_i$:envoyer (requete(i, $nums_i$) à k
Sinon envoyer (requete (i, $nums_i$) $succ_i$
Fsi
Fin ;

A la réception d'un message requete (id, nums) de j
Début
Si ($fils_i$<> ϕ) Alors soit k $\in$ $fils_i$:envoyer (requete(id, nums)) à k
Sinon
Si (i<>S) Alors Si (j=$pere_i$) Alors R_i[id] :=vrai ; envoyer (requete (id, nums)) à $succ_i$
// La requête vient d'atteindre la feuille
Sinon envoyer (requete (id, nums)) à $succ_i$
Fsi
Sinon $nums_i$:=traiter_requete(id, nums) ;
Si (Ri[id]=faux Alors envoyer (reponse(id, nums)) à $succ_i$
Fsi Sinon envoyer (reponse(id, nums)) à $pere_i$
Fsi
Fin ;

A la réception d'un message reponse (id, nums) de j
Début
Si (i=id) Alors memoriser(nums)
Sinon
Si ($fils_i$=ϕ) Alors
Si (R_i[id]=faux)Alors envoyer (reponse (id, nums)) à $succ_i$
// La réponse circule sur l'anneau
Sinon envoyer (reponse (id, nums)) à $pere_i$
// La réponse commence à monter dans l'arborescence
Fsi
Sinon envoyer (reponse(id, nums) à $pere_i$
// La réponse continue à remonter dans l'arborescence
Fsi
Fin ;

جامعة هواري بومدين للعلوم و التكنولوجيا
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique
Département d'Informatique

Concours d'accès au Doctorat 3 ième Cycle Informatique 2018 – 2019

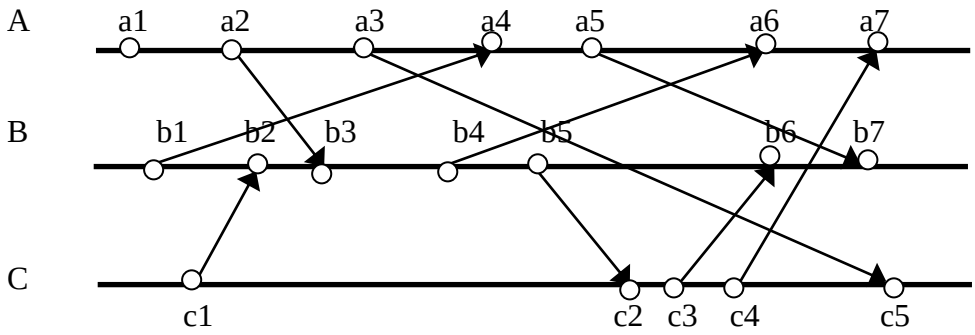
Le 24/10/2018

Matière 2 : Réseaux + au choix : Systèmes distribués/ Sécurité des systèmes,
Coefficient **3**, durée **2 Heures.**
(Spécialité : RSI)

Partie 2 : Systèmes distribués

Exercice 1: 4 pts= (1+0.75+0.75+1.5)

Soit la structure d'événements S= (E, <) définie par le diagramme de temps suivant :



1- Dater les évènements avec l'horloge vectorielle de Mattern.
2- En utilisant les horloges, donner la relation d'ordre pour les couples d'évènements: (a7, b6) (a4, c2).
3- Dans quels types d'algorithmes répartis a-t-on besoin des coupures? pourquoi est-ce important d'avoir une coupure consistante dans ce type d'algorithmes ?
4- Donner un exemple de coupure consistante et un exemple de coupure non consistante (il suffit de préciser le dernier évènement de la coupure au niveau de chaque processus), avec justification pour chaque réponse (vous avez le choix pour la méthode).

Exercice2: 6pts= (1.5+1.5+3)

On considère un système distribué composé de N processus, P_i, $i = 1, N$; où i est l'identité du processus P_i connectés selon une topologie *physique* connexe. On cherche à construire un index distribué d'un réseau p2p (pair à pair) sur un arbre construit avec l'**algorithme d'exploration en parallèle** (il faudra construire l'arborescence). Initialement, chaque processus P_i, possède une liste L_i de ressources R_i à partager lui-même. A la fin de l'algorithme, chaque processus Pi doit avoir comme résultat un index $IndxP_i$ indiquant les ressources détenues par les processus du sous-arbre dont il est la racine. Tandis que le processus racine possédera un index global.

NB. Exemple d'index : une liste de la forme (P_k, L_k), le processus P_k, possède la liste de ressource L_k.

Questions :
1-Donner les structures de données et les messages utilisés pour cet algorithme.
2-Donner le principe détaillé de l'algorithme.
3-Ecrire une solution distribuée à ce problème.

Bon courage.

Solution de l'épreuve Systèmes distribués

Exercice 1: 4 pts =(1+0.75+0.75+1.5)

1- Datation des évènements avec l'horloge vectorielle de Mattern.

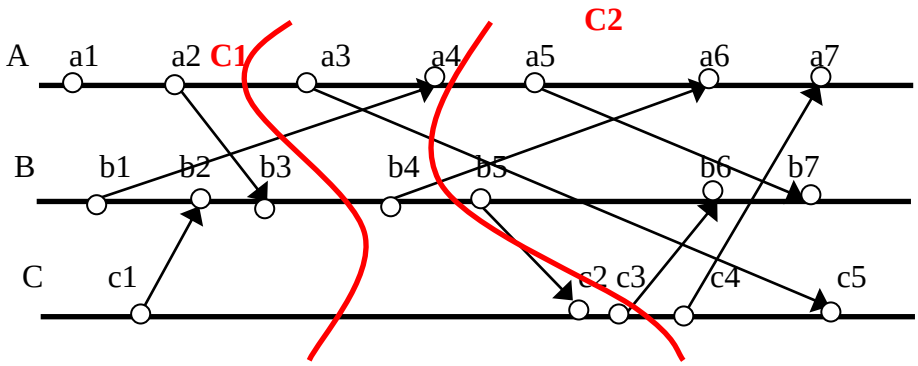
***Processus A**: a1(1,0,0), a2(2,0,0), a3(3,0,0), a4(4,1,0), a5(5,1,0), a6(6,4,1), a7(7,5,4).*
***Processus B**: b1(0, 1, 0), b2(0,2,1), b3(2,3,1), b4(2,4,1), b5(2,5,1), b6(2,6,3), b7(5,7,3).*
***Processus C:** c1(0,0,1), c2(2,5,2), c3(2,5,3), c4(2,5,4), c5(3,5,5).*

2- Relations d'ordre des couples d'évènements:

a7(7,5,4) et b6(2,6,3) sont deux évènements indépendants.
***a4(4,1,0) et c2(2,5,2) sont deux évènements indépendants**.*

*3- **Les coupures sont utilisées dans les algorithmes de calcul d'état global. Une coupure consistante permet de capturer un état global cohérent, et ce dernier permet de faire un redémarrage (après panne) cohérent et sans perte d'informations**.*

4- Un exemple de coupure consistante et un exemple de coupure non consistante :



Exemple de coupure consistante : C1 (Justification : pour tout évènement e' de réception de message inclue dans la coupure, l'évènement e d'envoi du même message est aussi dans la coupure)
Exemple de coupure inconsistante : C2 (Justification : l'évènement c2 réception d'un message est inclus dans l'état du processus C alors que l'évènement b5 d'envoi du même message sur le processus B, ne fait pas partie de l'état de ce dernier).

Exercice2: 6pts= (1.5+1.5+3)

1- Les structures de données et les messages

Les structures :

On va définir :

- Une structure struct ***proRes*** : <P_k, L_k> avec L_k : liste de Ressources R_m détenues par le processus Pk.
- Une liste de données ***IndxP$_i$***=Liste d'éléments de type struct ***proRes*(P$_k$, L$_k$)**. P_k est l'identité d'un processus donné et L_k, une liste de ressources détenues par le processus P_k.

Les messages :

- ***explore** () ;*
- ***reponse*** (***IndxP$_j$*** : Liste de structure proRes) ;

2- Le principe de l'algorithme

Initialement, chaque processus possède sa propre liste de ressources à partager lui-même, dans une structure de type ***proRes***. Tandis que la structure ***indexP$_i$*** est initialisée à ***proRes$_i$*** et va être mise à jour avec la construction de l'arbre. On va utiliser l'algorithme d'exploration en parallèle pour construire l'arbre et pour remonter les indexes locaux des feuilles vers la racine. Lors de la phase d'exploration, le processus P0 envoie un message explore à tous ses voisins. Tout processus P_i qui

reçoit le message ***explore ()*** pour la première fois, considère l'émetteur comme son père et envoie lui-même le même message à tous ses voisins. Si pas de voisins, le processus P_i répond à son père en envoyant son propre index ***indexP$_i$*** (le message ***reponse(indexP$_i$)).*** Pour tout message redondant ***explore()*** reçu, le processus P_i répond immédiatement à l'émetteur P_j par un message **Réponse (null).**

A la réception d'un message ***reponse (indexP$_j$)*** sur P_i, si ***indexP$_j$*** n'est pas null, l'index local ***indexP$_i$*** est mis à jour (***indexP$_i$=indexP$_i$*** union ***indexP$_j$***) et le processus P_j est ajouté dans une liste filsi. Sinon le message est simplement ignoré. L'algorithme se termine quand le processus racine de l'arbre reçoit les réponses de tous ses voisins.

3- L'algorithme :

Contexte local du processus Pi :

Les variables

proRes$_i$:<structure déjà initialisée par la liste de ressources partagées par P_i>.
indexP$_i$: liste de structures de type prosRes.
voisins$_i$: ensemble des identités des processus voisins de P_i.
pere$_i$: identité du processus père de P_i.
fils$_i$: ensemble des identités des processus fils de P_i.
reçu$_i$: booleen.
nbsucc$_i$: entier.

Les messages :

- explore ()
- reponse (index$_j$: liste de structures proRes$_i$)

Initialisation :

Début
reçu$_i$:=false ; voisins$_i$:=<les voisins de pi> ; nbsucc$_i$:=|voisins$_i$|; indexP$_i$:=proRes$_i$;
Si (i=0) **Alors**
reçu$_i$:=true ; pere$_i$:=i ;
$\forall$ k $\in$ voisins$_i$: envoyer (explore ()) à P_k
Fsi
Fin ;

A la réception d'un message explore() de P_j :

Début
Si (reçu$_i$=true)
Alors envoyer (reponse(null)) à P_j
Sinon
pere$_i$:=j ; reçu$_i$:=vrai ;
Si (voisins$_i$ <>null) **Alors** $\forall$ k $\in$ voisins$_i$: envoyer (explore ()) à P_k
Sinon envoyer reponse (indexP$_i$) à P_{perei}.
Fsi
Fsi
Fin ;

A la réception d'un message reponse (indexP$_j$) de P_j :

Début
nbsucc$_i$:=nbsucc$_i$-1 ;
Si (index$_j$ <>null) **Alors**
fils$_i$:=fils$_i$ + j ;
indexP$_i$:= indexP$_i$ + indexP$_j$;
Fsi ;
Si (nbsucc$_i$=0) **Alors Si** (pere$_i$=i) **Alors « fin de l'algorithme »**
Sinon envoyer reponse(indexP$_i$) à P_{perei} ;
Fsi
Fsi
Fin ;

جامعة هواري بومدين للعلوم و التكنولوجيا
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique
Département d'Informatique

Concours d'accès au Doctorat 3 ième Cycle Informatique 2017 – 2018

Le 29/10/2017

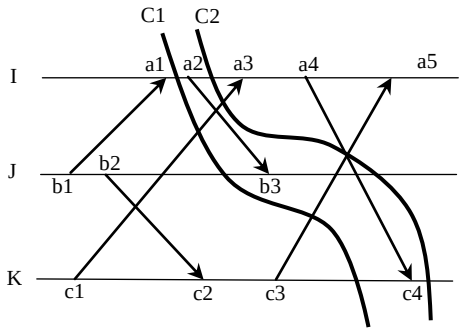
Matière 2 : Réseaux + au choix : Systèmes distribués/ Sécurité des systèmes,
Coefficient **3**, durée **2 Heures.**
(Spécialité : RSI)

Partie 2 : Systèmes distribués

Exercice 1:

Soit la structure d'événements S= (E, <) définie par le diagramme de temps suivant :

1- Dater les événements de la structure en utilisant les horloges vectorielles de Mattern.
2- Donner la relation entre les couples d'événements suivants en utilisant les horloges vectorielles : (b3, c4) ; (a1, c3).
3- Vérifier la nature de chacune des coupures C1 et C2 à l'aide du théorème connu dans ce contexte.
4- Pour les coupures consistantes, donc l'état global correspondant est consistant, donner les messages en transit pour chacune et pour chaque canal (émetteur--récepteur).



Exercice 2 :

1. Proposer un principe de construction d'une arborescence couvrante.
2. Adapter le principe précédent pour proposer un algorithme d'élection. Donner le principe et l'algorithme correspondant à votre proposition.
3. Discuter clairement les avantages et les inconvénients de votre proposition ainsi que sa tolérance aux défaillances.

Solution de l'épreuve Systèmes distribués

Exercice 1 : 8pts =(2 + 2 +3 +1)

1- Datation des événements

$$a1=\begin{matrix}1\\1\\0\end{matrix} \quad a2=\begin{matrix}2\\1\\0\end{matrix} \quad a3=\begin{matrix}3\\1\\1\end{matrix} \quad a4=\begin{matrix}4\\1\\1\end{matrix} \quad a5=\begin{matrix}5\\2\\3\end{matrix}$$

$$b1=\begin{matrix}0\\1\\0\end{matrix} \quad b2=\begin{matrix}0\\2\\0\end{matrix} \quad b3=\begin{matrix}3\\3\\0\end{matrix}$$

$$c1=\begin{matrix}0\\0\\1\end{matrix} \quad c2=\begin{matrix}0\\2\\2\end{matrix} \quad c3=\begin{matrix}0\\2\\3\end{matrix} \quad c4=\begin{matrix}4\\2\\4\end{matrix}$$

2-Relation de précédences entre événements en utilisant les horloges vectorielles

- $(b3<c4)$ car $\begin{matrix}3\\3\\0\end{matrix} = H(b3) < > H(c4) = \begin{matrix}4\\2\\4\end{matrix}$

- $(a1< c3)$ car $\begin{matrix}1\\1\\0\end{matrix} = H(a1) < > H(c3) = \begin{matrix}0\\2\\3\end{matrix}$

3- Vérification de la nature de chacune des coupures C1 et C2 à l'aide du théorème.

- $H(C1)= Max\ (H(a1), H(b2), H(c3)) = Max\begin{pmatrix}1&0&0\\1&2&2\\0&0&3\end{pmatrix} = \begin{matrix}1\\2\\3\end{matrix}$

Pour que C1 soit cohérente, $H(C1)$ doit être égale à $(H(a1[1], b2[2], c3[3]) = \begin{matrix}1\\2\\3\end{matrix}$

Elle est égale → C1 est cohérente.

- $H(C2)= Max\ (H(a2), H(b3), H(c4)) = Max\begin{pmatrix}2&3&4\\1&3&2\\0&0&4\end{pmatrix} = \begin{matrix}4\\3\\4\end{matrix}$

Pour que C2 soit cohérente, $H(C2)$ doit être égale à $(H(a2[1], b3[2], c4[3]) = \begin{matrix}2\\3\\4\end{matrix}$

Ce n'est pas le cas → C2 n'est pas cohérente.

4- Messages en transit

Pour les coupures consistantes, donc l'état global correspondant est consistant, les messages en transit pour la coupure C1 et pour chaque canal sont :
La topologie est complète, étant donné que les messages s'échangent entre tous les processus.
Au niveau de I :
Etat du canal qui le relie avec J est : ϕ
Etat du canal qui le relie avec K est : {(c1, a3) ; (c3,a5)}
Au niveau de J :
Etat du canal qui le relie avec I est : ϕ
Etat du canal qui le relie avec K est : ϕ
Au niveau de K :
Etat du canal qui le relie avec I est : ϕ
Etat du canal qui le relie avec J est : ϕ

Exercice 2 : Correction non disponible pour le moment.

جامعة هواري بومدين للعلوم و التكنولوجيا
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique
Département d'Informatique

Concours d'accès au Doctorat 3 ième Cycle Informatique 2016 – 2017

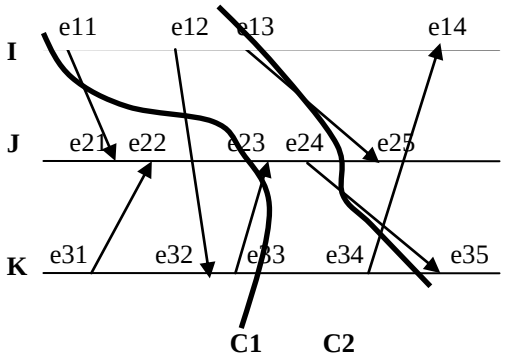
Le 26/10/2016

Matière 2 : Réseaux + au choix : Systèmes distribués/ Sécurité des systèmes,
Coefficient **1**, durée **2 Heures.**
(Spécialité : RSI)

Partie 2 : Systèmes distribués

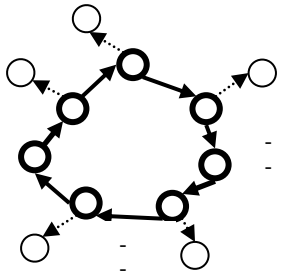
Exercice 1 : (5 pts= 1,5 + 1+ 1,5 + 1)

Soit la structure d'événements S= (E, <) définie par le diagramme de temps suivant :
1- Dater les événements de la structure en utilisant les horloges vectorielles de Mattern.
2- Donner la relation entre les couples d'événements suivants en utilisant les horloges vectorielles : (e13, e32) ; (e11, e35).
3- Vérifier la nature de chacune des coupures C1 et C2.
4- Pour les coupures consistantes, donc l'état global correspondant est consistant, donner les messages en transit pour chacune et pour chaque canal.



Exercice 2 : (5 pts= 2 + 1+ 2)
On suppose un ensemble de processus liés par une structure connexe définie par le réseau physique de la figure ci-dessous. Pour cela, chaque processus possède trois variables : $succ_i$, $pere_i$ et $fils_i$. Si $succ_i = -1$, cela veut dire que le processus est un nœud feuille de l'arbre. Si $pere_i = i$, cela veut dire que le processus fait partie de l'anneau. Chaque arbre contient seulement un nœud racine et un nœud feuille.

On désire réaliser le modèle client/serveur de telle sorte qu'un seul processus S (connu de tous) de l'anneau est serveur et que seulement les nœuds feuilles des différents arbres sont des clients. Donc, les autres nœuds de l'anneau servent d'intermédiaires pour les requêtes et les réponses. Les services demandés/fournis sont numérotés de 1 à M et le serveur retourne le numéro de service demandé comme réponse à ce service. Les différents messages empruntent les voies de la structure physique définie pour arriver à destination.

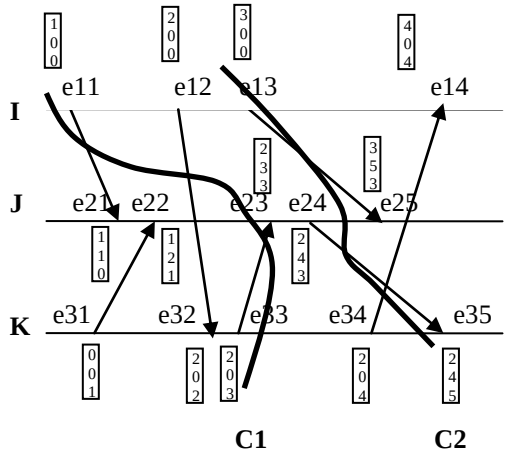


A- Donner le principe de cet algorithme.
B- Lister les différentes primitives de traitement des messages et expliquer leurs paramètres.
C- Ecrire cet algorithme.

Correction

Exercice 1 : (5 pts= 1,5 + 1+ 1,5 + 1)

1- Datation des événements suivants en utilisant les horloges de Mattern.



2- Relation entre les couples d'événements suivants en utilisant les horloges de Mattern :

- (e13// e32) car (3 0 0) = $H(e_{13})$ // $H(e_{32})$ = (2 0 2)
- (e11< e35) car (1 0 0) = $H(e_{11})$ < $H(e_{35})$ = (2 4 5)

3- Nature de chacune des coupures C1 et C2 en utilisant les horloges de Mattern.

Nature de la coupure C1:

$H(C1)$= Max ($H(e_{01})$, $H(e_{22})$, $H(e_{33})$)
= Max((0 0 0), (1 2 1), (2 0 3))= (1 2 3)
X= ($H(e_{01}[1])$, $H(e_{22}[2])$, $H(e_{33}[3])$)= (0 2 3)
C1 n'est pas consistante car H(C1) <> X.

Nature de la coupure C2:

$H(C2)$= Max ($H(e_{13})$, $H(e_{24})$, $H(e_{34})$)
= Max((3 0 0), (2 4 3), (2 0 4))= (3 4 4)
X= ($H(e_{13}[1])$, $H(e_{24}[2])$, $H(e_{34}[3])$)= (3 4 4)
C2 est consistante car H(C2) = X.

Ou bien

C1 n'est pas consistante car l'événement de réception e21 appartient à la coupure C1 et son événement d'émission e11 n'y appartient pas. De même, respectivement, que e32 avec e12.

C2 est constante car chaque événement appartenant à la coupure C2, tous les événements qui le précèdent causalement appartiennent aussi à C2.

4- Les messages en transit de la coupure C2 (seule coupure consistante):

au niveau du processus I :	(J->I : ϕ) ;	(K->I : (e_{34}, e_{14})).
au niveau du processus J :	(I->J : (e_{13}, e_{25})) ;	(K->J : ϕ).
au niveau du processus K :	(I->K : ϕ) ;	(J->K : (e_{24}, e_{35})).

Exercice 2 : (5 pts= 2 + 1+ 2)

1- <u>Le principe de fonctionnement de l'algorithme :</u>

Un processus client désirant un service, le choisie dans l'intervalle [0..M-1] et véhicule une requête, contenant entre autres le numéro du service et l'identité du père de ce nœud, à son père. Cette requête suit la structure de l'anneau jusqu'à l'arrivée au serveur s. Donc, chaque nœud la recevant la transmis à son successeur s'il n'est pas lui-même le serveur. Le serveur, quand il reçoit cette requête, la sert et renvoie un message de réponse à son fils si celui-ci est destinataire final sinon à son successeur avec comme destinataire le père du demandeur. La réponse suit la structure jusqu'au père du nœud demandeur. Celui-ci la transmit à son fils, qui à tour mémorise la réponse localement.

2- Les différentes primitives et leurs paramètres.

Le texte de l'algorithme est symétrique car tous les processus utilisent les mêmes messages et leurs comportements diffèrent selon leurs identités et les événements qui se produisent localement.

Trois primitives sont utilisées :

- *A la demande de service ;*
- *A la réception de requête (orig, s)* de Pj ; //*orig* représentent le père du nœud feuille demandeur de service de numéro s.
- A la réception de *reponse (dest, s)*, // *orig* représentent le père du nœud feuille demandeur de service de numéro s.

3- Une solution distribuée à ce problème :

Contexte de Pi

Const Serveur=… ;// c'est le nom du serveur.

M=… ;

Var $pere_i$ *: entier ; //* déjà initialisé à père de i s'il existe, sinon à i

$fils_i$ *: ensemble : /** déjà initialisé à fils de i s'il y a lieu sinon à vide.

$succ_i$ *: entier ; // déjà initialisé*

s_i *: entier ;*

Messages

- *requête (orig, s)*
- *reponse (dest, s)*

A la demande d'un service

entier s_i*;*

Début

Si ($succ_i$*=-1) // le nœud feuille de l'arbre.*

Alors s_i*:= choisir_service (M) ; envoyer (requête (*$pere_i$*,* s_i*) à* $pere_i$

Fsi

Fin ;

A la réception d'un message requête (orig, s) de j

Début

Si (i<>serveur)

Alors envoyer (requête (orig, s) à $succ_i$

Sinon s_i *:= servir_requete(orig, s) ; Si (orig = i) Alors envoyer (reponse (orig,* s_i*) à* $fils_i$

Sinon envoyer (reponse (orig, s_i*) à* $succ_i$

Fsi

Fsi

Fin ;

A la réception d'un message reponse (dest, s) de j

Début

Si (dest =i) Alors envoyer (reponse (dest, s) à $fils_i$

Sinon Si dest=$pere_i$ *Alors memoriser(s)*

Sinon envoyer (reponse (orig, s_i*) à* $succ_i$

Fsi

Fsi

Fin :

جامعة هواري بومدين للعلوم و التكنولوجيا
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique
Département d'Informatique

Concours d'accès au Doctorat LMD Informatique, 2014/2015

Epreuve de Systèmes et Réseaux

(Option : Systèmes Informatiques) **USTHB le 15/10/2014**

Partie : Systèmes répartis

On désire gérer deux types différents de ressources réutilisables *R1* et *R2* avec respectivement *n1* et *n2* instances. Chaque processus ne peut demander et acquérir qu'un type de ressource à la fois et au nombre désiré mais ne peut exprimer de nouvelle demande que s'il n'a pas de ressources acquises.

Pour cela, on utilise un serveur central *centr* dont les rôles sont de recevoir les requêtes et les libérations des clients et de les orienter vers le serveur secondaire approprié. Il permet aussi le contrôle des requêtes des clients. On utilise alors deux serveurs secondaires : *Serv1* gère la ressource R1 et *Serv2* gère la ressource R2. On suppose que les clients ont des liens physiques directs avec *centr* et avec les deux autres serveurs et que *centr* a des liens physiques directs avec les deux serveurs.

1- Lister les différents messages au niveau de chaque type de processus et expliquer leurs paramètres.
2- Donner le principe de cet algorithme.
3- Ecrire l'algorithme

Correction de l'Epreuve de Systèmes Distribués
du Concours d'accès au Doctorat LMD Informatique, 2014/2015

Correction : (10 pts= 2+ 3+5)

1- Liste de messages :

Au niveau du client i :

demande (k, nb, source), où *k* est le numéro de la ressource demandée, *nb* le nombre d'instances demandées et *source* l'identité du processus demandeur.

autoris(k, ok), où k est le numéro de la ressource concernée par l'autorisation et *ok* est un booléen, sa valeur *vrai* indique l'autorisation d'accès aux ressources et sa valeur *faux* indique une interdiction faute d'une demande ou une acquisition en cours.

liberer (k, nb, source), où k est le numéro de la ressource concernée par la libération, *nb* le nombre d'instances libérées de la ressource et *source* l'identité du processus qui libère ces instances.

Au niveau de Serv[k]

demande (k, nb, source)

autoris(k, ok), la valeur de ok est toujours *vrai*

liberer (k, nb, source)

Au niveau de centr

demande (k, nb, source)

liberer (k, nb, source)

autoris(k, ok), la valeur de *ok* est toujours *faux* si ce message est envoyé.

2- Principe de l'algorithme

L'algorithme est composé de trois types de processus : les clients qui demandent l'une des deux ressources, le processus centr qui reçoit les requêtes des clients et les deux serveurs qui servent les clients.

- Un client qui désire nb instances d'une ressource *R[k]*, envoie sa demande contenant l'identité et le nombre d'instances de la ressource et sa propre identité au processus *centr* et attend la réception d'une autorisation d'accès à cette ressource. Une fois reçue cette autorisation, il accède à cette ressource et à la fin de son utilisation, il envoie un message de libération vers le serveur concerné.
- Le processus *centr*, en recevant une requête d'un client donné, il vérifie si se processus n'a pas de demande en instance, auquel cas il ne lui répond pas (ou il lui répond négativement, c'est une autre option) ; dans le cas contraire, il enregistre sa requête localement et l'aiguille vers le serveur approprié. Quand ce serveur reçoit une libération d'un client il note localement cette libération et l'aiguille vers le serveur concerné.
- Chaque serveur *k* entretient une variable qui indique à tout moment le nombre d'accès disponibles pour la ressource *k* (initialement *n1* pour *R[1]* et *n2* pour *R[2]*) et une file d'attente pour la ressource gérée. Quand le serveur *k* reçoit une requête sur la ressource qu'il gère, il retourne immédiatement une autorisation au client concerné si le nombre d'accès disponibles pour la ressource n'est pas suffisant auquel cas il débite le nombre à allouer du nombre d'instances libres. Sinon, il insère cette requête dans la file d'attente associée à *R[k]* de manière fifo. A la réception d'un message de libération de la ressource *R[k]*, le serveur k récupère le nombre d'instances libérés et examine s'il peut satisfaire des processus selon l'ordre fifo pour leurs envoyer des messages d'autorisations.

3- Texte de l'algorithme

Un processus peut ne pas libérer à la fois toutes les instances déjà acquises.

Au niveau d'un client i

Contexte du client i

Const Serv[1]=… ;Serv[2]=…. ;Centr=… ;
A la demande de nb instances de la ressource R[k] ;
Début
envoyer (demande (k, nb i) à centr ;
Fin ;
A la réception de autoris(k, ok) de Pj ;
Début
Si ok Alors
< Accéder aux instances de la ressource (k)>
Sinon < retarder demande à la libration de toutes les instances acquises>
Fsi
Fin ;
A la fin d'utilisation de nb instances de la ressource(k) ;
Début
envoyer (libérer (k, nb, i)) à Centr
Fin ;

Au niveau du centr

Contexte de centr

Const Serv[1]=… ;Serv[2]=…. ;
Var f_acquis :file de format (id, nb) + ses procedures ;
/ file de processus utilisant des ressources */*
f_wait : file de format (id, nb)+ses procédures ;/ file d 'attente de ressources */*
A la réception de demande (k, nb, source) de Pj;
Début
Si non exite(f-acquis, source) alors inserer (f-acquis, nb, source) ; envoyer (demande (k, nb, source) à Serv[k] ; Sinon envoyer (autoris(k, faux)
Fin ;
A la réception de libérer (k, nb, source) de Pj;
Début
supprimer (f-acquis, nb, source) ; envoyer (libérer (k, nb, source) à Serv[k] ;
Fin ;

Au niveau de Serv[k]

Contexte de Serv[k]

$NbAccesDispo_i$: entier ;nb1=…nb2=…. ;
f[k] : file d'attente de format (id, nb) + ses procédures d'accès;
Initialisation
Début
Si (k=1) Alors $NbAccesDispo_i$:=nb1 fsi;
Si (k=2) Alors $NbAccesDispo_i$:=nb2 fsi ;
Fin ;
A la réception de demande (k, nb, source) de Pj;
Début
Si non vide (f[k])et ($NbAccesDispo_i$ >=nb) Alors envoyer (autoris(k, vrai)) à source
$NbAccesDispo_i$:=$NbAccesDispo_i$ –nb ;
Sinon inserer(f[k],nb, source) // Ordre fifo
Fsi
Fin ;
A la réception de liberer (k, nb, source) de Pj;
Var m : entier ;
Début
$NbAccesDispo_i$:=$NbAccesDispo_i$ +nb ;
tantque (f[k]<>ϕ) et(premier (f[k]).nb<= $NbAccesDispo_i$)Alors
$NbAccesDispo_i$:=$NbAccesDispo_i$ – premier (f[k]).nb ; supprimer(premier(f[k]);
envoyer ((autoris(k, vrai)) à premier (f[k]).id
Fait
Fin ;

جامعة هواري بومدين للعلوم و التكنولوجيا
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique
D é p a r t e m e n t d ' I n f o r m a t i q u e

Concours d'accès au Doctorat LMD Informatique, 2013/2014

Epreuve de Systèmes distribués

(Option : Systèmes Informatiques) **USTHB le 20/10/2013**

Exercice 1 : (9 pts=1+1+2+1+1.5+1.5+1)

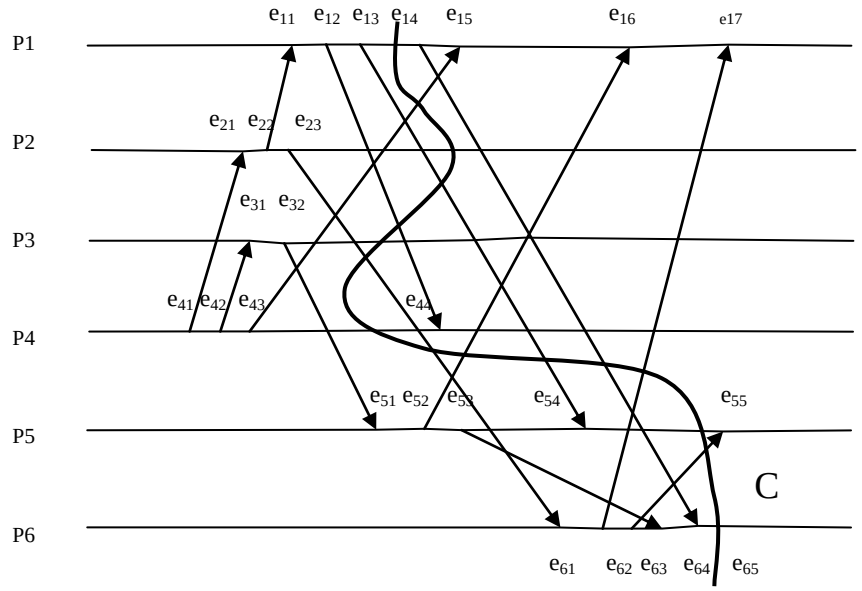
A/ Répondre aux questions suivantes :

a1- Quelle est l'utilité de l'approche micro - noyau dans la structuration des systèmes distribués ?

a2- Décrire comment utiliser le mécanisme d'appel de procédure à distance pour l'implémentation du modèle client/serveur.

B/ On considère la trace d'exécution donnée par la structure d'événements de la Figure suivante. Il s'agit du résultat d'une exploration en parallèle des processus P1, P2,…P5 (reliés par une topologie de communication connexe donnée) sans transport d'information de contrôle.

b1- Déduire la topologie qui relie ces processus.



b2- Déduire le nombre de messages redondants et l'arborescence construite.

b3- Si on considère le transport d'informations de contrôle, donner les messages redondants qui seront éliminés ?

b4- Dater uniquement les événements du passé de la coupure à l'aide des horloges de Mattern.

b5- Donner la nature de la coupure C en se basant sur les horloges de Mattern.

Exercice 2 : (11 pts= 2.5 + 1 + 4.5 + 1 + 2)

On considère un système distribué composé de *N* processus *P(i), i = 1, N* où *i* est l'identité du processus *P(i)* connectés selon une topologie *physique* connexe. Ces processus sont organisés selon un anneau logique bidirectionnel (i.e. chaque nœud ne peut communiquer qu'avec son prédécesseur et son successeur dans l'anneau) *supposée optimale* (i.e. chaque voisin dans l'anneau est aussi un voisin dans le réseau).

On désire implémenter un service d'exclusion mutuelle pour trois ressources différentes sur cette structure en supposant qu'un processus dans l'anneau est le serveur de *tous les autres* processus. Chaque processus désirant utiliser une ressource donnée, la demande au serveur en envoyant sa requête, qui contient le numéro de la ressource et une estampille locale (selon les horloges de Lamport), à travers la structure selon un sens choisi de manière aléatoire. Le serveur répond dans le sens inverse du sens d'arrivée de la demande. Tous les autres messages liés au service d'exclusion mutuelle doivent circuler à travers la structure logique établie.

a- Donner le principe de fonctionnement de l'algorithme
b- Lister les différents messages à utiliser.
c- Ecrire l'algorithme.
d- Donner la complexité moyenne en nombre de messages pour réaliser *une section critique.*
e- Donner les modifications nécessaires pour inclure le serveur comme client ?

Bon courage

Correction de l'Epreuve de Systèmes Distribués
du Concours d'accès au Doctorat LMD Informatique, 2013/2014

Exercice 1 : (9 pts=1+1+2+1+1.5+1.5+1)

A/

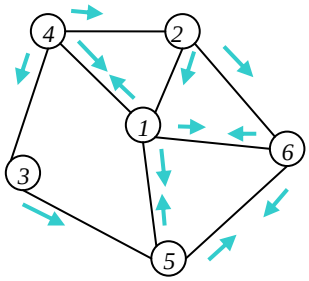
a1-

- Simplification dans la conception et extension des systèmes distribués (SDs)
- Séparation entre les services de bases communs des SDs et des services implémentés par des serveurs spécifiques au besoin

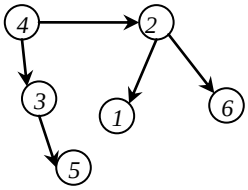
a2- Dans le modèle client/serveur, le serveur implémente un certain nombre de services qu'il fournit aux clients. On peut imaginer l'implémentation de chaque service par une procédure que le serveur invoque quand il reçoit la requête du client. Ce qui veut dire que le serveur dispose d'une table de correspondance entre le nom du service demandé par le client et le nom de la procédure qui le réalise. Le résultat du service est retourné par la procédure au serveur, ce dernier le fait passer au client.
On peut utiliser le mécanisme d'appel de procédure à distance pour l'implémentation du modèle client/serveur comme suit : Le client fait appel directement à la procédure qui réalise le service par son nom en lui fournissant les paramètres nécessaires. La procédure lui retourne directement le résultat.

B/

b1- La topologie



b2- Nombres messages redondants : 8.
-L'arborescence



b3- Nombre messages redondants éliminés :

(1,4) ; (2,1) ;
Si 6 reçoit en premier de 1 Alors
(6,5) ; (6,2) ; Si 5 reçoit en premier de 1 Alors (5, 6) ; (5,3) Sinon /* reçoit de 3 */(1,5) Fsi
Sinon
Si 6 reçoit en premier de 2 Alors
(5,6) ; (6,1) ; (6,5) ; Si 5 reçoit en premier de 1 Alors (5, 6); (5,3) Sinon /* reçoit de 3*/(1,5) Fsi
Sinon
Si 5 reçoit en premier de 3 Alors (5, 1) ; (6,5) ; (6,1) ; (6,2)
Sinon
Si 5 reçoit en premier de 1 Alors (5, 1) ; (6,5) ; (6,1) ; (6,2) ; (5,3) Fsi
Fsi
Fsi
Fsi ;

(6, 5) si le nœud 6 reçoit le message d'exploration en premier de 5 ou de 1 au lieu de 2.

b4- Datation des événements à l'aide des horloges de Mattern:

H(e11)=(1 2 0 1 0 0) ;H(e12)=(2 2 0 1 0 0) ; H(e13)=(3 2 0 1 0 0) ;H(e14)=(4 2 0 1 0 0) ;H(e15)=(5 2 0 3 0 0) ;
H(e16)=(6 2 2 3 2 0) ; H(e17)=(7 2 3 3 2 2) ;
H(e21)=(0 1 0 1 0 0) ;H(e22)=(0 2 0 1 0 0) ; H(e23)=(0 3 0 1 0 0) ;
H(e31)=(0 0 1 2 0 0) ;H(e32)=(0 0 2 2 0 0) ;
H(e41)=(0 0 0 1 0 0) ;H(e42)=(0 0 0 2 0 0) ; H(e43)=(0 0 0 3 0 0) ; H(e44)=(2 2 0 4 1 0) ;
H(e51)=(0 0 2 2 1 0) ;H(e52)=(0 0 2 2 2 0) ; H(e53)=(0 0 2 2 3 0) ; H(e54)=(3 2 2 2 4 0) ; H(e55)=(3 3 2 2 5 3) ;
H(e61)=(0 3 0 1 0 1) ;H(e62)=(0 3 0 1 0 2) ; H(e63)=(0 3 0 1 0 3) ; H(e64)=(0 3 2 2 3 4) ; H(e65)=(4 3 2 2 3 5) ;

b5- Vérification de la nature de C à l'aide des horloges de Mattern :
H(C)= Max (H(e13), H(e23), H(e32), H(e43), H(e(54), H(e(65))
= Max ((3 2 0 1 0 0) , (0 3 0 1 0 0), (0 0 2 2 0 0), (0 0 0 3 0 0) , (3 2 2 2 4 0), (4 3 2 2 3 5))
=(4 3 2 3 4 5)
Pour que C soit cohérente, H(C) doit être égale à :
(H(e13)[1], H(e23[2]), H(e32)[3], H(e43)[4], H(e(54)[5], H(e(65)[6])= (3 3 2 3 4 5) ;
Elle est différente → C n'est pas cohérente.

Exercice 2 : (11 pts= 2.5 + 1 + 4.5 + 1 + 2)

a-Le principe de fonctionnement de l'algorithme :
Le serveur entretient trois files d'attente fifos, une par ressource une entrée par ressource qui indique si celle-ci est libre ou occupée.

Etant donné que le processus serveur est situé sur l'anneau, la requête portant sur la ressource critique désirée (soit r) estampillée à l'aide des horloges de Lamport (H(r)) et accompagnée de l'identité du nœud demandeur (soit k) est envoyée le long de l'anneau dans un sens choisi aléatoirement. Cette requête est transmise de proche en proche jusqu'à l'arrivée au serveur. Celui-ci retourne immédiatement une autorisation à travers le nœud qui lui a transmis cette requête (soit j) si la file *f*(r) est vide (i.e. la ressource est libre). Puis enfile la requête avec les informations (k, r, j, H(r)) dans la file *f(r*). Le processus destinataire d'une autorisation, quand il la reçoit, accède à sa SC.
Un processus k qui sort de sa section critique pour une ressource *r*, véhicule un message de libération accompagné de (jk *r)* dans un sens choisi aléatoirement. Quand le serveur reçoit ce message, il purge le premier élément de la file *f(r)* et si la file n'est toujours pas vide, il véhicule un message d'autorisation sur le chemin indiqué par la requête.

a- Les différents messages à utiliser.
Trois messages sont utilisés :
Requête (k, r, NS) et *Autorisation (k, r), Liberation (r)* où *k* est l'identité du processus demandeur, *r* est le numéro du service et NS est l'estampille de la requête.

b- Une solution distribuée à ce problème :
On suppose que le coordinateur est nommé *serveur.*

Contexte de Pi
Const serveur$_i$= .... ; N=... ;// N : Nombre de processus du réseau
succ$_i$: entier ; / déjà initialisé à successeur de i dans l'anneau*
pred$_i$: entier : / déjà initialisé à prédécesseur de i dans l'anneau*
NS$_i$: entier :=0 ;
Messages
Requête (k, r, NS) et *Autorisation(k, r, NS), Liberation (r)* où *k* est l'identité du processus demandeur, *r* est le numéro du service et *NS* est l'estampille de la requête.

A la demande de la ressource ® :
Début
Si (i<>serveur$_i$) Alors NS$_i$++ ; l :=choisir (succ$_i$, pred$_i$) ; envoyer (requête (i, r, NS$_i$) à l
Fsi
Fin :
A la réception d'un message requête (k, r, NS) de j
Var l : entier ;
Début
NS$_i$:=Max (NS$_i$, NS) ;
Si (i<>serveur$_i$) Alors si j= pred$_i$ alors l := succ$_i$ sinon l := pred$_i$ Fsi ; envoyer (requête (k, r, NS) à l
Sinon Si (f(r)=Φ) Alors inserer (k, r, j, NS); envoyer(autorisation (k, r, NS$_i$) à j

Sinon inserer (k, r, j, NS)
// insère le triplet (k, j, NS) dans f(r) en respectant l'ordre fifo selon NS
Fsi
Fsi
Fin :
A la réception d'un message autorisation (k, r, NS) de j
Var l : entier ;
Début
Si (i<>k) Alors si(j= $pred_i$) alors l := $succ_i$ sinon l := $pred_i$ Fsi ; envoyer(autorisation (k, r) à l
Sinon NS_i :=Max (NS_i, NS) ;<entrer en SC pour r>
Fsi
Fin :
A la sortie de la section critique de r
Var l : entier ;
Début l :=choisir ($succ_i$, $pred_i$) ; envoyer (liberation (r)) à l Fin :
A la réception de liberation (r) de j
Var l : entier ;
Début
Si (i<>$serveur_i$) Alors si(j= $pred_i$) alors l := $succ_i$ sinon l := $pred_i$ Fsi ; envoyer (liberation (r)) à l
Sinon supprimer (f(r)) ;// Supprime l'élément en tête de f(r)
Si(f(r)<> ϕ) Alors envoyer (autorisation (premier (f(r)).k, r, NS_i)
à premier (f(r).j)
Fin ;// la file f(r) est de format (k, j, NS) où k est le nom du demandeur de r, j est le nœud qui a transmis la requête au serveur et NS est l'estampille.

c- *La complexité moyenne par requête :*
3*N/2 messages en supposant que chaque message (demande, autorisation et libération) parcoure la moitié de l'anneau.

Pour inclure le serveur comme client
La primitive de demande devient :

A la demande de la ressource (r) :
Début
Si (i<>$serveur_i$) Alors NS_i++ ; l :=choisir ($succ_i$, $pred_i$) ; envoyer (requête (i, r, NS_i) à l
Sinon Si (f(r)<> ϕ) Alors inserer (k, r, j, NS); <Entrer en SC pour r>
Sinon inserer (k, r, j, NS)
Fsi
Fsi
Fin ;

A la sortie de la section critique de r
Var l : entier ;
Début Si (i<>$serveur_i$) Alors l :=choisir ($succ_i$, $pred_i$) ; envoyer (liberation (r)) à l
Sinon
supprimer (f(r)) ;// Supprime l'élément en tête de f(r)
Si(f(r)<> ϕ) Alors envoyer (autorisation (premier (f(r)).k, r, NS_i)
à premier (f(r).j)
Fsi
Fsi
Fin ;

جامعة هواري بومدين للعلوم و التكنولوجيا
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique
Département d'Informatique

Concours d'accès au Doctorat LMD Informatique, 2012/2013

Epreuve de Systèmes distribués

(Option : Systèmes Informatiques)

USTHB le 26/11/2012

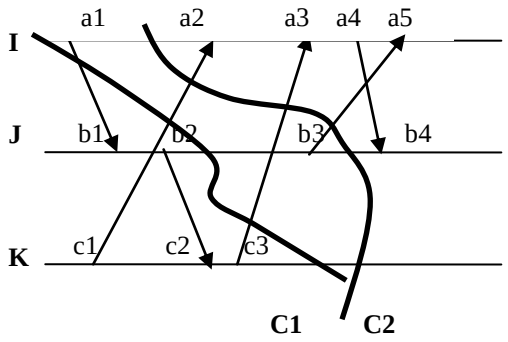
Exercice 1 : (9 pts=1+1.5+2+1+1.5+2)

A/ Répondre aux questions suivantes :

- Pourquoi sont structurés les algorithmes distribués de manière non déterministe ?
- Quel est l'handicap des horloges logiques de Lamport? Qu'apportent de nouveau les horloges vectorielles de Mattern?

B/ Soit la structure d'événements S= (E, <) définie par le diagramme de temps suivant :

1- Dater les événements de la structure en utilisant les horloges vectorielles de Mattern.
2- Donner la relation entre les couples d'événements suivants en utilisant les horloges vectorielles : (c3, b4) ; (a1, c3).
3- Vérifier la nature de chacune des coupures C1 et C2 à l'aide du théorème connu dans ce contexte.
4- Pour les coupures consistantes, donc l'état global correspondant est consistant, donner les messages en transit pour chacune et pour chaque canal.



Exercice 2 : (11 pts= 2.5 + 1 + 4.5 + 1 + 2)

On considère un système distribué composé de *N* processus *P(i), i = 1, N* où *i* est l'identité du processus *P(i)* connectés selon une topologie *physique* connexe. Ces processus sont organisés selon une arborescence logique (i.e. chaque nœud ne peut communiquer dans les deux sens qu'avec son père et ses fils, s'il y a lieu, dans l'arborescence) *supposée optimale* (i.e. chaque voisin dans l'arborescence est aussi un voisin dans le réseau).

On désire implémenter un service d'exclusion mutuelle pour deux ressources différentes sur cette structure en supposant que le processus racine de l'arborescence est le serveur de *tous les autres* processus. Chaque processus désirant utiliser une ressource donnée, la demande au serveur en envoyant sa requête, qui contient le numéro de la ressource et une estampille locale (selon les horloges de Lamport), à travers la structure. Tous les autres messages liés au service d'exclusion mutuelle doivent circuler à travers la structure logique établie.

a- Donner le principe de fonctionnement de l'algorithme
b- Lister les différents messages à utiliser.
c- Ecrire l'algorithme.
d- Donner la complexité moyenne en nombre de messages pour réaliser *une section critique.*
e- Que faut-il modifier pour inclure le serveur comme client ?

Bon courage

Correction de l'Epreuve de Systèmes Distribués
du Concours d'accès au Doctorat LMD Informatique, 2012/2013

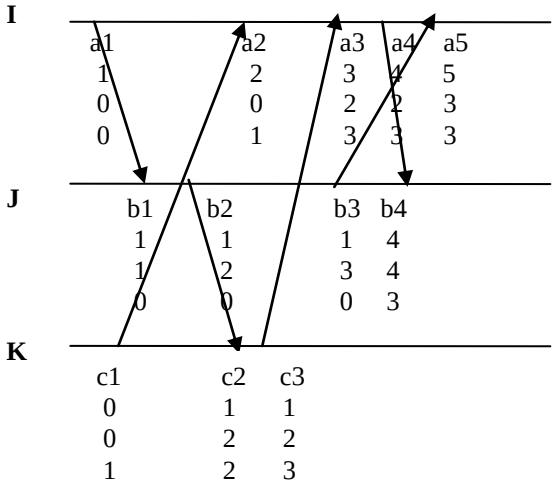
Exercice 1 : (9 pts=1+1.5+2+1+1.5+2)

A/ - Les algorithmes distribués sont structurés de manière non déterministe pour attendre et répondre aux événements qui peuvent se produire de manière asynchrone.

- Ils ne peuvent pas déduire la précédence entre les événements par simple comparaison de leurs estampilles. Les horloges de Matern remédient à ce problème, ils permettent de déduire si les événements sont concurrents ou séquentiels par simple comparaison des estampilles.

B/ Soit la structure d'événements S= (E, <) définie par le diagramme de temps suivant :

1- Datation des événements à l'aide des horloges de Mattern:



2- Relation de précédences entre événements en utilisant les horloges vectorielles :

- $(c3 < b4)$ car $\begin{matrix}1\\2\\3\end{matrix} = H(c3) <= H(b4) = \begin{matrix}4\\4\\3\end{matrix}$

- $(a1 < c3)$ car $\begin{matrix}1\\0\\0\end{matrix} = H(a1) <= H(b3) = \begin{matrix}1\\2\\3\end{matrix}$

4- Vérification de la nature de chacune des coupures C1 et C2 à l'aide du théorème.

- $H(C1) = \text{Max}(H(a0), H(b2), H(c3)) = \text{Max}\left(\begin{matrix}0\\0\\0\end{matrix}\ \begin{matrix}1\\2\\0\end{matrix}\ \begin{matrix}1\\2\\3\end{matrix}\right) = \begin{matrix}1\\2\\3\end{matrix}$

Pour que C1 soit cohérente, $H(C1)$ doit être égale à $(H(a0[1], b2[2], c3[3]) = \begin{matrix}0\\2\\3\end{matrix}$

Ce n'est le cas → C1 n'est pas cohérente.

- $H(C2) = \text{Max}(H(a1), H(b3), H(c3)) = \text{Max}\left(\begin{matrix}1\\0\\0\end{matrix}\ \begin{matrix}1\\3\\0\end{matrix}\ \begin{matrix}1\\2\\3\end{matrix}\right) = \begin{matrix}1\\3\\3\end{matrix}$

Pour que C2 soit cohérente, $H(C2)$ doit être égale à $(H(a1[1], b3[2], c3[3]) = \begin{matrix}1\\3\\3\end{matrix}$

Elle est égale → C2 est cohérente.

5- Pour les coupures consistantes, donc l'état global correspondant est consistant, les messages en transit pour la coupure C2 et pour chaque canal sont :
La topologie est complète, étant donné que les messages s'échangent entre tous les processus.
Au niveau de I :
Etat du canal qui le relie avec J est : {(b3, a5)}
Etat du canal qui le relie avec K est : {(c1, a2) ; (c3,a3)}
Au niveau de J :
Etat du canal qui le relie avec I est : ϕ
Etat du canal qui le relie avec K est : ϕ
Au niveau de K :
Etat du canal qui le relie avec I est : ϕ
Etat du canal qui le relie avec J est : ϕ

Exercice 2 : (11 pts= 2.5 + 1 + 4.5 + 1 + 2)

a- *Le principe de fonctionnement de l'algorithme :*
Etant donné que la racine de l'arborescence est le processus serveur, un processus client est alors un nœud qui possède un père sur la structure. La requête, qui porte sur un numéro de la ressource critique désirée, est envoyée le long de la branche qui mène vers la racine. Pour cela, le processus demandeur envoie cette requête à son père, celui-ci la transmis à son tour à son père et ainsi de suite jusqu'à ce qu'elle arrive à la racine. Pour tracer le chemin de retour d'une requête, chaque processus *i* (y compris le serveur) qui reçoit une requête d'un client *k* de l'un de ses fils *j*, met *T[k, r]=j*, où *T* est un tableau local de taille *N*. Ceci veut dire : '*'quand je reçois l'autorisation sur la requête du client k de mon père pour la ressource r, je la lui envoie à travers j*''. En retour, l'autorisation commence le parcours à partir de la racine. En général, quand un processus reçoit une autorisation de son père pour une destination *x*, s'il est le destinataire il rentre en section critique pour la ressource demandée, sinon il la transmit à son fils *T[, r]*.
Remarque : On peut transporter dans la requête l'ensemble Z des nœuds traversés par la requête. Au retour, pour envoyer la réponse sur le chemin du destinataire, celle-ci est envoyée à chaque fois au nœud fils qui appartient à Z.
Quand le serveur reçoit une requête d'un nœud fils *k* sur une ressource *r*, si la file locale est vide (i.e. la ressource est libre), il enfile la requête avec son estampille sur la file *f(r*) et envoie une autorisation accompagnée de l'identité du nœud originaire de la requête et *r* au fils *T[k, r]*, sinon il se contente d'enfiler la requête en respectant l'ordre de Lamport.
Un processus qui sort de sa section critique pour une ressource *r*, véhicule un message de libération accompagné du numéro *r* sur la branche qui le lie à la racine. Quand le serveur reçoit ce message, il purge le premier élément de la file *f(r)* et si la file n'est toujours pas vide, il véhicule un message d'autorisation sur la branche qui le lie au processus en tête de file.
b- *Les différents messages à utiliser.*
Trois messages sont utilisés :
Requête (k, r, NS) et *Autorisation(k, r), Liberation (r)* où *k* est l'identité du processus demandeur, *r* est le numéro du service et NS est l'estampille de la requête.

c- *Une solution distribuée à ce problème :*
On suppose que la racine est de numéro *racine.*

Contexte de Pi
Const racine$_i$= …. ; N=… ;// N : Nombre de processus du réseau
pere$_i$: entier ; / déjà initialisé à père de i s'il existe, sinon à i*
fils$_i$: ensemble : / déjà initialisé*
T$_i$: tableau[1..N, 1..2] de entier ;
NS$_i$: entier :=0 ;
Messages
Requête (k, r, NS) et *Autorisation(k, r), Liberation (r)* où *k* est l'identité du processus demandeur, *r* est le numéro du service et *NS* est l'estampille de la requête.

A la demande de la ressource r :
Début
Si (i<>racine$_i$) Alors NS_i++ ; envoyer (requête (i, r, NS_i) à pere$_i$
Fsi
Fin :
A la réception d'un message requête (k, r, NS) de j
Début
T[k, r] :=j ; NS_i :=Max (NS_i, NS) ;
Si (i<>racine$_i$) Alors envoyer (requête (k, r, NS) à pere$_i$
Sinon Si (f(r)=Φ) Alors inserer (f, k, r, NS);envoyer(autorisation (k, r, NS_i) à T_i[k, r]
Sinon inserer (f, k, r, NS)
// insère k, NS dans f(r) en respectant l'ordre fifo selon NS
Fsi
Fsi
Fin :
A la réception d'un message autorisation (k, r, NS) de j
Début
Si (i<>k) Alors envoyer(autorisation (k, r, NS_i) à T_i[k, r]
Sinon NS_i :=Max (NS_i, NS) ;<entrer en SC pour r>
Fsi
Fin :
A la sortie de la section critique de r
Début
envoyer (liberation (r)) à pere$_i$
Fin :
A la réception de liberation (r) de j
Début
Si (i<>racine$_i$) Alors envoyer (liberation (r)) à pere$_i$
Sinon supprimer (f(r)) ;// Supprime l'élément en tête de f(r)
Si(f(r)<>Φ) Alors envoyer (autorisation (premier (f).client, premier (f(r).ress, NS_i)
à T_i[premier (f).client]
Fin ;// la file f(r) est de format (client, ress, NS) où client est le nom du demandeur de ress
// d'estampille NS.

d- *La complexité moyenne par requête :*
Supposons que la profondeur maximale de l'arborescence est *d*. Sachant que la requête, son autorisation et sa libération ensemble font *3d* pas au maximum et au minimum *3* pas, la complexité moyenne *(3d+3)/2 messages*.

e- *Pour inclure le serveur comme client*
La primitive de demande devient :
A la demande d'une ressource r
Début
NS_i++ ;
envoyer (requête (i, r, NS_i) à pere$_i$
// Si i=racine$_i$, le messgae est envoyé au même processus car pere$_i$=i.
Fin ;