**Exercice 1**

L’objectif est de comparer deux évènements e1 et e2 qui se passent dans deux sites différents. Nous supposerons que l’évènement e1 (resp. e2) est un évènement local du site 1 (resp. 2).

1. Supposons que les deux sites ont une horloge de Lamport comme système de datation. Comparer les deux évènements e1 et e2 dans les deux situations suivantes :
2. la date de l’évènement e1 est 4 et celle de e2 est 3.
3. la date de l’évènement e1 est 4 et celle de e2 est 4.
4. Supposons que les deux sites ont une horloge vectorielle comme système de datation et que le système distribué est composé de trois sites. Comparer les deux évènements e1 et e2 dans les deux situations suivantes :
5. la date de l’évènement e1 est (4, 3, 2) et celle de e2 est (5, 2, 1).
6. la date de l’évènement e1 est (5, 3, 6) et celle de e2 est (4, 2,5).

**Correction**

1. –
2. On ne peut rien dire sur l’ordre causal des deux évènements e1 et e2.

Car H(e2)<H(e1) n’implique pas que e2$\rightarrow $e1.

1. On peut dire que les deux évènements e1 et e2, sont indépendants.
2. –
3. On peut dire que les deux évènements e1 et e2 sont indépendants, car aucune des deux relations suivantes n’est vraie : (4,3,2)< (5,2,1) & (5,2,1)<(4,3,2).
4. On peut dire que e2$\rightarrow $ e1, car (4,2,5) < (5,3,6).

**Exercice 2**

On considère un système distribué constitué de 4 sites P1, P2, P3, P4, s’envoyant des messages de façon asynchrone comme représenté par la figure suivante. Les événements d’un processus, représentés par des gros points noirs, sont soit des événements locaux (étapes d’un calcul), soit des envois ou des réceptions de messages.

Ces événements sont datés par un système d’horloges de Lamport, initialisées à 0 dans chaque état initial.

1. Indiquez au-dessus de chaque point de la figure, la valeur de l’horloge du processus où se produit l’événement correspondant.

Ces événements sont datés par un système d’horloges vectorielles, initialisées à [0, 0, 0, 0] dans chaque état initial.

2. Indiquez à côté de chaque point de la figure la valeur de l’horloge du processus où se produit l’événement correspondant.

3. On considère les événements a, b, c et d de la figure. Cochez toutes les formules justes :

$$∎ a\rightarrow b ∎ a\rightarrow c ∎ b\rightarrow c ∎a\rightarrow d$$

$$∎ b\rightarrow a ∎ c\rightarrow a ∎ c\rightarrow b ∎d\rightarrow a$$

$$∎ a\left|\left|b ∎ c\right|\right|a ∎ b|\left|c ∎ a\right||d$$

**Correction**

1. Les valeurs des horloges de Lamport scalaires pour cette exécution répartie sont :

**2**

**3**

**3**

**4**

**4**

**5**

**5**

**5**

**6**

**7**

**6**

**6**

**6**

**7**

**8**

**8**

**7**

**9**

**10**

**9**

**11**

**8**

1. Les valeurs des horloges vectorielles pour cette exécution répartie sont :

**0 0 0 0**

0 **0 0 0**

**0 0 0 0**

0 **0 0 0**

**1 0 0 0**

0 **1 0 0**

**0 0 1 0**

0 **0 0 1**

**2 0 0 0**

0 **2 2 0**

**0 0 2 0**

0 **0 0 2**

**3 0 3 4**

0 **3 4 0**

**0 0 3 0**

0 **0 3 3**

**4 0 3 4**

1 **4 4 0**

**0 0 4 0**

0 **0 3 4**

**5 5 4 0**

1 **5 4 0**

**0 1 5 0**

0 **1 8 0**

**0 1 6 0**

**0 1 7 6**

**0 0 3 5**

**0 1 7 0**

**0 1 9 0**

**0 1 10 0**

**0 6 7 7**

**0 1 7 7**

1. Pour répondre on va comparer les valeurs des horloges vectorielles des évènements.
2. $∎ a\rightarrow b ∎ a\rightarrow c √ b\rightarrow c √a\rightarrow d$
3. $∎ b\rightarrow a ∎ c\rightarrow a ∎ c\rightarrow b ∎d\rightarrow a$
4. $√ a\left|\left|b √ c\right|\right|a ∎ b|\left|c ∎ a\right||d$

**Exercice 3**

On considère la figure suivante décrivant une exécution d'un calcul réparti :

1. Quel est le passé causal de r4 ?

2. Quel est le futur causal de r3 ?

3. Le chemin $e1 \rightarrow r2 \rightarrow e5 \rightarrow r4 \rightarrow e7 \rightarrow r6$ est-il un chemin causal liant e1 à r6 ?

4. La coupure indiquée est-elle cohérente ? Pourquoi ?

5. L'état global x = 2 & y = 4 n'est pas survenu dans l'exécution présentée (x passe à 3 après l'événement

 e3, et y passe à 4 après l'événement e6). Cet état est-il pourtant cohérent ?

6. Donner la valeur des horloges de Lamport pour tous les événements.

7. Déterminer l'histoire causale de m et en déduire si les délivrances r5; r3; r6; r7 respectent la causalité.

**Correction**

1. Passé causal de r4 : {e1, e2, r1, e4, e5, e3, r3}
2. Futur causal de r3 : {r4, e7, r6, r7}
3. Non, ce n’est pas un chemin valide.
4. La coupure n’est pas cohérente. Par exemple, on a e3$\rightarrow $ r1, avec r1$ϵ$ à la coupure et e3 non. Plus précisément, un état cohérent ne peut pas contenir la réception d’un message sans son émission
5. Bien que l’état (x=2 & y=4) ne survienne pas dans l’exécution présentée, il est cohérent, car on peut obtenir une coupure cohérente qui contient cet état.
6. –
7. L'histoire causale Hc (m) d'un message m est l'ensemble des messages qui ont leurs évènements d'émission précédent causalement l'émission de m donc:

Hc(m) ={me5, me4, me2, me1}.

Les délivrances r3 et r5 respectent la causalité car leurs évènements d’émissions ne sont pas en relation de causalité. Par contre, les délivrances r6 et r7 ne respectent la causalité ; en effet, on bien e4$\rightarrow $e7, mais non (r7$\rightarrow $r6).

**Exercice 4**

On considéré les échanges de messages entre 3 sites A; B; C représentés par le chronogramme

suivant :



1. Dans le chronogramme de la figure (1), quelle particularité implique que ce dernier ne représente pas un calcul diffusant ?

2. Pourquoi certaines actions des processus (à préciser) ne peuvent pas être considérées dans ce chronogramme comme atomiques ?

3. Les événements i2 et r5 sont-ils causalement liés ?

4. Donner une coupure cohérente incluant les événements i2 et e4.

5. Donner une coupure non cohérente incluant les événements i2 et e4.

6. Déterminer la valeur de l'horloge vectorielle de l'événement r4. Justifier cette valeur soit par raisonnement, soit en calculant les horloges des événements le précédant.

7. Déterminer l'histoire causale du message m6 et en déduire si les délivrances r6; r7 respectent la causalité.

**Correction**

1. Dans un calcul diffusant, un processus initial,P0 par exemple, débute le calcul en envoyant un ou plusieurs messages vers d’autres processus. Puis tous les processus, y compris ce processus initial, adoptent le comportement suivant :

Loop

 /\* un pas de calcul \*/

 recevoir(m) ;

 traiter m;

 envoyer 0 à N−1 messages ;

 end loop

 D’après ce qui précède, le calcul représenté n’est pas diffusant car il ne respecte pas le modèle :

réception, traitement (activité processus), émission (voir éclipses rouges).

1. Les actions (voir éclipses bleues) ne peuvent pas être considérées comme atomiques, à cause des évènements réception qui se produisent au milieu des activités des processus.

1. Les évènements i2 et r5 ne sont pas liés causalement car ils ne satisfont à aucune des conditions d’une relation de causalité à savoir :
* e1 et e2 appartiennent au même site et l’un d’eux se produit avant l’autre
* e1 ou e2 est évènement d’émission et l’autre l’évènement de réception correspondant
* il existe un évènement e3 tel que ; e1$\rightarrow $e3 et e3$\rightarrow $e2.
1. Coupure cohérente incluant i2 et e4 :

1. Coupure non cohérente incluant i2 et e4 :

1. L’histoire causale du message m6 est l’ensemble des messages dont les évènements d’émissions sont reliés causalement à e5 (émission de m6). Donc on a :

 H(m6)={m3,m1,m2}

**Exercice 5**

**1-**Définir la notion de coupure cohérente. Pour l’exécution d’un calcul réparti proposé dans la figure ci-dessous, identifier les coupures cohérentes. Justifier votre réponse.

**2-** Quelles sont les modifications qu’il faut apporter à l’algorithme de Chandy-Lamport pour enregistré un état global fortement cohérent (c'est-à-dire, tous les états de canaux sont enregistrés vide) ?

**Correction**

1. Pour la définition, voir le cours. Pour ce qui est des coupures on a Cut 1 qui est une coupure cohérente, alors Cut 2 est une coupure incohérente. Dans Cut 2, on l’évènement g qui correspond à une réception dont l’évènement émission est h ; mais g appartient à Cut 2 mais h non (condition C1 non satisfaite). Cut1, satisfait les deux conditions C1 et C2.
2. Rappelons d’abord que : la superposition signifie qu’à chaque processus Pi est associé un processus observateur CTLi; ce processus peut lire l’état de Pi. La réception d’un message par Pi est réalisée par CTLi, qui le délivre ensuite à Pi; de même, Pi confie à CTLi les messages qu’il veut transmettre vers tout autre processus Pj; CTLi se charge alors de son émission vers CTLj (qui le délivre ensuite à Pj). Les processus CTLi sont chargés de capter un état global cohérent de l’application. Ils coopèrent entre eux à l’aide de messages de contrôle. Un tel schéma d’observation des processus Pi par des contrôleurs CTLi est appelé superposition. Par abus de langage, on confond parfois Pi et son observateur CTLi, lorsque ceci n’altère pas la compréhension.

Pour avoir un état global fortement cohérent (c'est-à-dire, tous les états de canaux sont enregistrés vide), on pourrait concevoir les CLTi de manière à les doter de mémoires tampons dans lesquelles ils stockeraient, dans l’ordre, les messages reçus, avant leurs délivrances. De cette façon, les $SC\_{ij}$ seraient toujours vides, à l’enregistrement. Mais, l’état local de chaque processus va contenir l’état de la mémoire tampon, ainsi ajoutée, avec les messages en attente de délivrance. L’algorithme modifié de Chandy-Lamport deviendrait :

**Règle d'envoi du marqueur pour le processus i**

(i) Le processus i enregistre son état (état de la mémoire tampon compris).

(ii) Initialise ses canaux à "vide"

(iii) Pour chaque canal C sortant sur lequel un marqueur n'a pas été envoyé :

* Le processus i envoie un marqueur sur C avant que i n'envoie d'autres messages sur C.

**Règle de réception des marqueurs pour le processus j**

À la réception d'un marqueur sur le canal C :

* **si** j n'a pas enregistré son état, **alors**

suivez la "Règle d'envoi des marqueurs".

 **Règle de réception des messages pour le processus j**

À la réception d'un message sur le canal C :

* Déposer le message dans la mémoire tampon.

**Exercice 6**

Illustrer le déroulement de l’algorithme de Chandy-Lamport sur l’exemple de la figure ci-dessous, dans lequel le processus P0 déclenche la prise d’instantané.

**Exercice 7**

L'algorithme de Chandy-Lamport suppose que les canaux de communication utilisés par les sites sont FIFOs. Pour lever cette restriction, on propose :

* de numéroter les clichés (on suppose pour cela que l'on dispose d'un mécanisme permettant d'établir un ordre global sur les différentes prises d’instantané).
* de superposer à chaque message applicatif le numéro du dernier cliché auquel le site émetteur du message a participé.

Questions

1. Sur quel mécanisme pourrait-on s'appuyer pour implanter une numérotation globale des différents clichés ?
2. Proposer une adaptation (simple) du protocole de Chandy-Lamport utilisant cette numérotation pour construire des coupures cohérentes sans supposer que les canaux de communication sont FIFO.
3. Montrer que cette adaptation simple permet de construire une coupure cohérente mais pas de capturer correctement les messages en transit.

**Correction**

1. En supposant que le lancement de la prise d’instantané puisse être activé par n’importe quel site (un seul à chaque prise d’instantané), et pour établir un ordre global sur les différentes prises d’instantanés, on pourrait s’appuyer sur des horloges vectorielles. De la sorte chaque prise d’instantané serait marquée par l’horloge vectorielle du site déclencheur ($Marker\\_Ho$).

Lorsque les canaux ne sont pas FIFO, la valeur de l’horloge permet de séparer les messages envoyés avant cette valeur de ceux envoyés après (comme un marqueur sert à séparer les messages envoyés avant le marqueur de ceux envoyés après, pour les canaux FIFOs). Les valeurs des horloges vectorielles sont utilisées pour marquer prises d’instantané. Un message est dit ancien s’il est envoyé horodaté avant $Marker\\_Ho$. Sinon, le message est dit nouveau. Le protocole adapté se déroule selon les règles suivantes :

**R1** :

* Le processus initiateur de la prise d’instantané, incrémente son $Marker\\_Ho$ locale puis capte son état selon la règle **R2**.
* Après réception des messages marqueurs de la part de tous les processus, l’initiateur diffuse un message "Terminate".

**R2** : un processus $P\_{j}$ qui capte son état local :

* initialise l’état de tous les canaux à "vide" ;
* envoie sur tous ses canaux de sortie dans un message de contrôle (appelé mH), contenant une copie de son $Marker\\_Ho$, pour signifier à ses voisins qu’il a enregistré son état local à la date marquée par $Marker\\_Ho $;
* Un processus arrête l’algorithme de prise d’instantané après réception du message "Terminate" ;

**R3** : Un processus inclut un message reçu dans l’état du canal seulement si c’est un ancien message;

1. Cette adaptation simple permet de construire une coupure cohérente puisque les deux conditions de la cohérence d’une coupure (C1 et C2 vues en cours) sont satisfaites. En effet, pour la condition C1 on a $send\left(m\_{ij}\right)ϵLS\_{i}⇒m\_{ij}ϵ SC\_{ij}⊕ m\_{ij}ϵ LS\_{j}$. Le message dans $LS\_{i}$, est un message envoyé horodaté avec la prise d’instantané précédente. S’il arrive avant mH, il sera enregistré dans $LS\_{j}$ lorsque Pj aura enregistré son état locale ; mais s’il arrive après mH, alors il ne sera pas enregistré dans $LS\_{j}$ mais il sera considéré dans $CS\_{ij}$, puisque la date de $m\_{ij}$ sera celle de l’ancienne prise d’instantané.

Pour le condition C2, $send\left(m\_{ij}\right)\nni LS\_{i}$ cela veut dire que $m\_{ij}$ est horodaté avec la nouvelle prise d’instantané et par conséquent ne sera pris en considération ni dans $SC\_{ij}$ ni dan $LS\_{j}$.

Cependant, cette adaptation ne permet pas de capturer correctement les messages en transit. En effet, puisque les canaux ne sont pas FIFOs, il se peut que le message mH arrive avant un message $m\_{ij}$ qui appartienne à $LS\_{i} $; et on ne sait pas quand ce message parvienne à $P\_{j}$.

**Exercice 8 (Travail à rendre)**

Un parking possède p portes par lesquelles des voitures peuvent entrer ou sortir. Le parking ne doit jamais contenir plus de N voitures. À chaque porte se trouve un gardien ; les gardiens communiquent en s’envoyant des messages (de façon asynchrone, par exemple à l’aide de pigeons voyageurs supposés fiables) ; ils doivent faire en sorte que la capacité du parking ne soit jamais dépassée. Il faut par ailleurs que, s’il y a régulièrement des sorties, toute voiture qui attend pour entrer par la porte de son choix y parvienne en un temps fini. Décrire un algorithme réparti, qui sera appliqué par chacun des gardiens, afin résoudre le problème. Cet algorithme utilisera des principes présentés en cours. Le nombre de messages échangés entre deux entrées de voitures devra être borné.

Mises en gardes :

* Une idée (incorrecte) serait de partager initialement les N places du parking en p lots de N/p places, à la responsabilité de chaque gardien. Dans ce cas le gardien peut laisser rentrer N/p voitures, plus autant de voitures qu’il a laissé sortir. Cependant, avec ce schéma si toutes les voiture sortent parla même porte, cette dernière porte devient la seule entrée possible et les voitures qui cherchent à rentrer par une autre porte y restent bloquées (ce qu’on ne veut pas)
* Une autre idée (tout aussi incorrecte) serait d’imaginer que les gardiens tentent de garder le compte des places libres en diffusant (aux autres gardiens) des messages “entrée” ou “sortie” à chaque fois qu’une voiture entre ou sort par leur porte. Le problème est que ces diffusions ne sont pas délivrées instantanément et qu’elles peuvent aussi se croiser. Cela devient critique lorsqu’il n’y a plus beaucoup de places libres. Par exemple s’il ne reste plus qu’une place libre, deux gardiens pourraient simultanément laisser rentrer chacun une voiture.

**Exercice 9** – Exclusion Mutuelle

Nous considérons un système distribué composé d’un ensemble de N processus fiables$\prod\_{}^{}=\{p\_{1},p\_{2},…,P\_{N}\}$. Il y a un seul processus par site ou nœud.

Les processus ne partagent pas de mémoire et communiquent uniquement par passage de messages. Les nœuds sont supposés être connectés par des liens de communications point à point fiables (ni perte ni duplication) et FIFO. Le graphe de communication est considéré complet, i.e., tout processus peut communiquer avec n’importe quel autre processus.

Le système contient un ensemble $R=\{r\_{1},r\_{2},…,r\_{m}\}$ de m ressources différentes. Aucun ordre n’est défini sur R.

Les algorithmes d’exclusion mutuelle généralisée permettent de gérer les accès concurrents des processus sur un ensemble de ressources partagées. Autrement dit, un processus peut demander un sous-ensemble de k ressources $(k < m)$. Par exemple, le processus $p\_{1}$ peut demander les ressources $r\_{1}$ et $r\_{3}$ tandis que $p\_{2}$ les ressources $r\_{2}$, $r\_{3}$ et $r\_{5}$. Pour demander l’accès à $k$ ressources, l’application appelle la fonction $Request\\_CS(Dk)$ où $Dk$ est un ensemble de $k$ ressources et pour les libérer la fonction $Release\\_CS()$. La section critique concerne, par conséquent, l’accès à toutes les ressources du sous-ensemble demandé. De plus, un processus peut initier une nouvelle requête si et seulement si la requête qu’il a précédemment demandée a été satisfaite, c’est-à-dire que la section critique se termine par l’exécution de $Release\\_CS()$. Par conséquent, il y a au plus $N$ requêtes pendantes dans le système.

1. Supposons que chaque ressource $r\_{i}$ est gérée par une instance d’un algorithme d’exclusion mutuelle classique (exemple : Ricart-Agrawala, Suzuki-Kasami, Naimi-Tréhel, etc.). Une instance qui gère la ressource $r\_{i}$ offre donc les fonctions $Request\\_CS\_{i}()$ et $Release\\_CS\_{i}()$. La fonction $Request\\_CS(D\_{k})$ et $Release\\_CS()$ consisteraient alors à l’appel à un ensemble de fonctions $Request\\_CS\_{i}()$ et $Release\\_CS\_{i}()$ respectivement pour $r\_{i}$ quelconque $ϵ D\_{k}$.

Quel problème peut-il arriver avec une telle solution ?

 Le problème qui peut arriver avec une telle solution est celui de l’interblocage. En effet, supposons que deux processus $p\_{i}$ et $p\_{j}$ veulent accéder, respectivement, aux sous-ensembles de ressources $\{r\_{1},r\_{2},r\_{3}\}$ et $\{r\_{2},r\_{3}\}$, selon les ordres suivants :

* $p\_{i}$ demande l’accès à $r\_{1} $; il l’obtient
* $p\_{j}$ demande l’accès à $r\_{2} $; il l’obtient
* $p\_{i}$ demande l’accès à $r\_{3} $; il l’obtient
* $p\_{j}$ demande l’accès à $r\_{3} $; il ne l’obtient pas immédiatement, sa requête est en attente dans la file de $p\_{i}$
* $p\_{i}$ demande l’accès à $r\_{2} $; il ne l’obtient pas immédiatement, sa requête est en attente dans la file de $p\_{j}$

1. On considère les mêmes hypothèses qu’à la question précédente, mais on définit maintenant un ordre total $≺$ sur l’ensemble des ressources R où $r\_{i} ≺ r\_{j}$ si $i < j$. Désormais, les processus doivent demander les ressources requises de $D\_{k}$ en respectant l’ordre $≺$. Ceci résout-il le problème de la question A ? Pourquoi ?

Le problème reste posé puisqu’on peut avoir une situation dans laquelle deux processus $p\_{i}$ et $p\_{j}$ veulent accéder, respectivement, aux sous-ensembles de ressources $\{r\_{1},r\_{2},r\_{4}\}$ et $\{r\_{2},r\_{3},r\_{4}\}$, selon les ordres suivants :

* $p\_{i}$ demande l’accès à $r\_{1} $; il l’obtient
* $p\_{j}$ demande l’accès à $r\_{2} $; il l’obtient
* $p\_{i}$ demande l’accès à $r\_{2} $; il ne l’obtient pas immédiatement, sa requête est en attente dans la file de $p\_{j}$
* $p\_{j}$ demande l’accès à $r\_{3} $; il l’obtient
* $p\_{i}$ demande l’accès à $r\_{4} $; il l’obtient
* $p\_{j}$ demande l’accès à $r\_{4} $; il ne l’obtient pas immédiatement, sa requête est en attente dans la file $p\_{i}$
1. Nous reprenons à partir de maintenant les hypothèses initiales décrites au début de l’exercice et que par conséquent, il n’y a plus d’ordre total défini sur R.

Nous proposons un algorithme qui se caractérise par les propriétés suivantes :

* Chaque ressource $r\_{i}$ est :
* représentée par un unique jeton $t\_{i}$
* associée à une file d’attente distribuée dont le premier élément est le possesseur du jeton
* Avant de demander un quelconque jeton de ressource $t\_{i}$, un processus doit demander l’acquisition d’un jeton unique de contrôle noté $T\_{c}$. Ce jeton peut être vu comme une ressource partagée auxiliaire et son accès exclusif est géré avec l’algorithme de Naimi-Tréhél (voir PowerPoint joint).
* Le jeton de contrôle contient un vecteur $VT$ de m entrées ($m$ est égal au nombre de ressources du système). A chaque ressource $r\_{i}$, l’entrée correspondante indique :
* soit l’identifiant du dernier site demandeur de $r\_{i}$
* soit la valeur nil qui indique que le jeton $t\_{i}$, est libre et est inclus dans $T\_{c}$ (aucun processus ne possède ou n’a demandé $t\_{i}$)
* A la réception du jeton de contrôle, un processus prend toute ressource requise incluse dans $T\_{c}$ et envoie pour chaque jeton manquant, un message INQUIRE au dernier demandeur. Chaque entrée correspondante à une ressource requise dans $VT$ est alors mise à jour en y affectant l’identifiant du processus courant qui est devenu le dernier demandeur. Le jeton de contrôle peut désormais être libéré et renvoyé à un prochain demandeur éventuel. Toutefois avant la libération, le processus met dans $VT$ toutes les ressources non utilisées qu’il possède.
* A la réception d’un message INQUIRE pour un jeton $t\_{i}$ :
* si le site receveur possède $t\_{i}$ et ne l’utilise pas, il envoie immédiatement $t\_{i}$ à l’émetteur par un message ACK
* sinon, si le jeton est utilisé ou pas encore possédé, l’envoi de ACK est ajourné jusqu’ à ce que le receveur libère $t\_{i}$ lors de l’appel à $Release\\_CS()$.
* Initialement toute ressource est incluse dans le jeton de contrôle. Ce dernier est détenu par le processus $p\_{1}$ qui est par conséquent la racine $(father = nil)$ de l’arbre dynamique gérant $T\_{c}$. La variable $father$ de tous les autres processus est donc égale à $p\_{1}$.

Les types de messages de cet algorithme sont donc :

* $REQ CONTROL$ : un message pour demander le jeton de contrôle
* $TOKEN CONTROL$ : un message transmettant le jeton de contrôle
* $INQUIRE$ : un message demandant un jeton ressource
* $ACK$ : un message transmettant un jeton de ressource

Quel est le rôle du jeton de contrôle dans l’algorithme ?

Le rôle du jeton de contrôle est résoudre le problème de la question A ; en rendant atomiques $Request\\_CS(Dk)$ et ($Release\\_CS()$).

1. Supposons un système réparti avec 4 processus ($p\_{1,}….,p\_{4})$ et 5 ressources $(r\_{1,}….,r\_{5})$. Nous considérons que juste après l’initialisation du système, les processus exécutent les primitives $Request\\_CS$ dans l’ordre suivant :
2. $p\_{2,} : Request\\_CS(\{r\_{1,}, r\_{3,}\})$ ;
3. $p\_{3,} : Request\\_CS(\{ r\_{4,}\})$ ;
4. $p\_{4,} : Request\\_CS(\{r\_{1,},r\_{2,}, r\_{4,}\})$ ;

Donnez à l’état initial et après chaque exécution des primitives l’état du système à savoir :

* l’arbre dynamique gérant $T\_{c}$
* le contenu du vecteur du jeton de contrôle
* le possesseur de $T\_{c}$
* les possesseurs des jetons de ressource ainsi que les files distribuées associées.

De plus, donnez les différents transferts de messages (type, expéditeur, destinataire) entre chaque état. En tout, 4 états vous sont donc demandés. Il vous est recommandé de faire un schéma pour chaque état.

**Etat1 :** *Démarrage*

$$P\_{1}$$

$$P\_{3}$$

$$P\_{4}$$

$$P\_{2}$$

$$P\_{4}$$

$$father:P\_{1}$$

$$file1:\{P\_{1}\}$$

$$file2:\{P\_{1}\}$$

$$file3:\{P\_{1}\}$$

$$file4:\{P\_{1}\}$$

$$file5:\{P\_{1}\}$$

$$P\_{2}$$

$$father:P\_{1}$$

$$file1:\{P\_{1}\}$$

$$file2:\{P\_{1}\}$$

$$file3:\{P\_{1}\}$$

$$file4:\{P\_{1}\}$$

$$file5:\{P\_{1}\}$$

$$r\_{1}$$

$$r\_{2}$$

$$r\_{3}$$

$$r\_{4}$$

$$r\_{5}$$

$$t\_{1}\rightarrow nil $$

$$t\_{2}\rightarrow nil $$

$$t\_{3}\rightarrow nil $$

$$t\_{4}\rightarrow nil $$

$$t\_{5}\rightarrow nil $$

$$nil$$

$$nil$$

$$nil$$

$$nil$$

$$nil$$

$$T\_{c}$$

$$P\_{3}$$

$$father:P\_{1}$$

$$file1:\{P\_{1}\}$$

$$file2:\{P\_{1}\}$$

$$file3:\{P\_{1}\}$$

$$file4:\{P\_{1}\}$$

$$file5:\{P\_{1}\}$$

$$P\_{1}$$

$$father:nil$$

$$file1:\{P\_{1}\}$$

$$file2:\{P\_{1}\}$$

$$file3:\{P\_{1}\}$$

$$file4:\{P\_{1}\}$$

$$file5:\{P\_{1}\}$$

**Etat2 :** $p\_{2} : Request\\_CS(\{r\_{1,}, r\_{3,}\})$

$$P\_{2}$$

$$nil$$

$$P\_{2}$$

$$nil$$

$$nil$$

$$T\_{c}$$

 Messages : $P\_{2} \rightarrow P\_{1} :REQ CONTROL$

$$P\_{1}$$

$$father:P\_{2}$$

$$file1:\{P\_{2}\}$$

$$file2:\{P\_{2}\}$$

$$file3:\{P\_{2}\}$$

$$file4:\{P\_{2}\}$$

$$file5:\{P\_{2}\}$$

$$P\_{3}$$

$$father:P\_{1}$$

$$file1:\{P\_{1}\}$$

$$file2:\{P\_{1}\}$$

$$file3:\{P\_{1}\}$$

$$file4:\{P\_{1}\}$$

$$file5:\{P\_{1}\}$$

$$P\_{4}$$

$$father:P\_{1}$$

$$file1:\{P\_{1}\}$$

$$file2:\{P\_{1}\}$$

$$file3:\{P\_{1}\}$$

$$file4:\{P\_{1}\}$$

$$file5:\{P\_{1}\}$$

$$P\_{2}$$

$$father:nil$$

$$file1:\{P\_{2}\}$$

$$file2:\{nil\}$$

$$file3:\{P\_{2}\}$$

$$file4:\{nil\}$$

$$file5:\{nil\}$$

$$r\_{1}$$

$$r\_{2}$$

$$r\_{3}$$

$$r\_{4}$$

$$r\_{5}$$

$$t\_{1}\rightarrow P\_{2}$$

$$ $$

$$t\_{2}\rightarrow nil $$

$$t\_{3}\rightarrow P\_{2}$$

$$ $$

$$t\_{4}\rightarrow nil $$

$$t\_{5}\rightarrow nil $$

$$P\_{1}$$

$$P\_{2}$$

$$P\_{3}$$

$$P\_{4}$$

 $P\_{1} \rightarrow P\_{2} :TOKEN CONTROL$

**Etat3 :** $p\_{3} : Request\\_CS(\{r\_{4}\})$

 Messages : $P\_{3} \rightarrow P\_{1} :REQ CONTROL$

$$P\_{2}$$

$$nil$$

$$P\_{2}$$

$$P\_{3}$$

$$nil$$

$$T\_{c}$$

 $P\_{1} \rightarrow P\_{2}: REQ CONTROL$

$$P\_{4}$$

$$P\_{3}$$

$$P\_{2}$$

$$P\_{1}$$

$$r\_{1}$$

$$r\_{2}$$

$$r\_{3}$$

$$r\_{4}$$

$$r\_{5}$$

$$t\_{1}\rightarrow P\_{2}$$

$$ $$

$$t\_{2}\rightarrow nil $$

$$t\_{3}\rightarrow P\_{2}$$

$$ $$

$$t\_{4}\rightarrow P\_{3} $$

$$t\_{5}\rightarrow nil $$

$$P\_{2}$$

$$father:P\_{3}$$

$$file1:\{P\_{2}\}$$

$$file2:\{P\_{3}\}$$

$$file3:\{P\_{2}\}$$

$$file4:\{P\_{3}\}$$

$$file5:\{P\_{3}\}$$

$$P\_{4}$$

$$father:P\_{1}$$

$$file1:\{P\_{1}\}$$

$$file2:\{P\_{1}\}$$

$$file3:\{P\_{1}\}$$

$$file4:\{P\_{1}\}$$

$$file5:\{P\_{1}\}$$

$$P\_{3}$$

$$father:nil$$

$$file1:\{P\_{2}\}$$

$$file2:\{nil\}$$

$$file3:\{P\_{2}\}$$

$$file4:\{P\_{3}\}$$

$$file5:\{nil\}$$

$$P\_{1}$$

$$father:P\_{3}$$

$$file1:\{P\_{2}\}$$

$$file2:\{P\_{2}\}$$

$$file3:\{P\_{2}\}$$

$$file4:\{P\_{2}\}$$

$$file5:\{P\_{2}\}$$

 $P\_{2} \rightarrow P\_{3} :TOKEN CONTROL$

**Etat3 :** $p\_{4} : Request\\_CS(\{r\_{1},r\_{2},r\_{4}\})$

$$P\_{2}$$

$$P\_{4}$$

$$P\_{2}$$

$$P\_{3}$$

$$nil$$

$$T\_{c}$$

 Messages : $P\_{4} \rightarrow P\_{1} :REQ CONTROL$ $P\_{1} \rightarrow P\_{3}: REQ CONTROL$ $P\_{3} \rightarrow P\_{4} :TOKEN CONTROL$ $P\_{4} \rightarrow P\_{3} :INQUIRE$ $P\_{4} \rightarrow P\_{3} :INQUIRE$

$$P\_{4}$$

$$P\_{3}$$

$$P\_{2}$$

$$P\_{1}$$

$$r\_{1}$$

$$r\_{2}$$

$$r\_{3}$$

$$r\_{4}$$

$$r\_{5}$$

$$t\_{1}\rightarrow P\_{2}$$

$$ $$

$$t\_{2}\rightarrow P\_{4} $$

$$t\_{3}\rightarrow P\_{2}$$

$$ $$

$$t\_{4}\rightarrow P\_{3} $$

$$t\_{5}\rightarrow nil $$

$$P\_{2}$$

$$father:P\_{3}$$

$$file1:\{P\_{2}\}$$

$$file2:\{P\_{3}\}$$

$$file3:\{P\_{2}\}$$

$$file4:\{P\_{3}\}$$

$$file5:\{P\_{3}\}$$

$$P\_{4}$$

$$father:P\_{1}$$

$$file1:\{P\_{2},P\_{4}\}$$

$$file2:\{P\_{4}\}$$

$$file3:\{P\_{1}\}$$

$$file4:\{P\_{3},P\_{4}\}$$

$$file5:\{P\_{1}\}$$

$$P\_{3}$$

$$father:P\_{4}$$

$$file1:\{P\_{2}\}$$

$$file2:\{P\_{4}\}$$

$$file3:\{P\_{2}\}$$

$$file4:\{P\_{3}\}$$

$$file5:\{P\_{4}\}$$

$$P\_{1}$$

$$father:P\_{4}$$

$$file1:\{P\_{1}\}$$

$$file2:\{P\_{1}\}$$

$$file3:\{P\_{1}\}$$

$$file4:\{P\_{1}\}$$

$$file5:\{P\_{1}\}$$

1. Expliquez informellement comment la propriété de sûreté de l’algorithme est assurée.

**Réponse**

Le premier élément de la sureté est l’accès exclusif à $T\_{c}$. Ceci permet au processus détenteur de réserver les ressources libres sans risque d’interférences. Pour les ressources occupées, le processus attend leurs libérations tout en gardant $T\_{c}$ ce qui lui permet d’être sûr que c’est lui qui les utilisera dès qu’elles seront libérées .

1. Nous souhaitons maintenant avoir le pseudocode de cet algorithme. Pour cela nous vous demandons de :
* Donner les variables locales à chaque processus ainsi que la procédure d’initialisation.
* Spécifier les contenus de chaque type de message.
* Donner le pseudocode des fonctions $Request\\_CS(D\_{k})$ et $Release\\_CS()$.
* Donner pour chaque type de message la primitive gérant la réception du message en question.