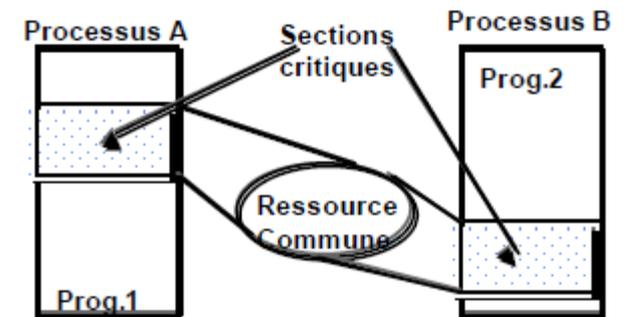


***Introduction à l'Algorithmique
Distribuée
Exclusion mutuelle***

Babahenini Mohamed Chaouki

Exclusion mutuelle distribuée

- Exclusion mutuelle
 - Contexte de plusieurs processus s'exécutant en parallèle
 - Accès à une ressource partagée par un seul processus à la fois.
- Exclusion mutuelle en distribué
 - Accès à une ressource partagée distante par un seul processus à la fois
 - Processus distribués
 - Requêtes et gestion d'accès via des messages échangés entre les processus
 - Nécessité de mettre en œuvre des algorithmes gérant ces échanges de messages pour assurer l'exclusion mutuelle

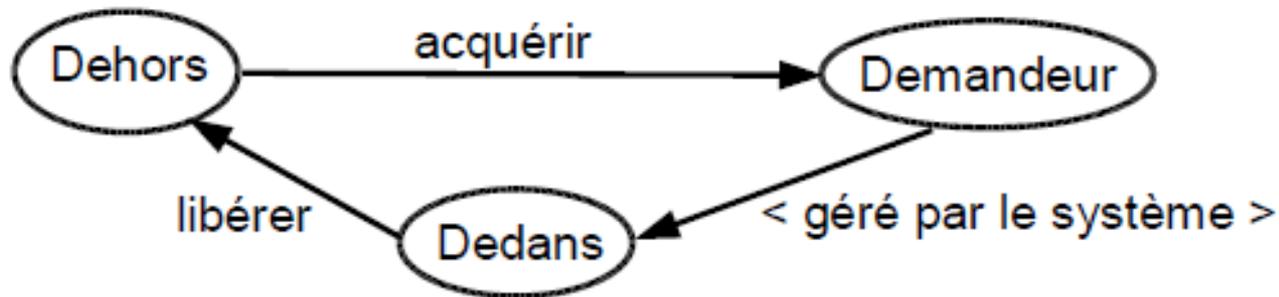


Rappel exclusion mutuelle

- Exclusion mutuelle
 - Une ressource partagée ou une section critique n'est accédée que par un processus à la fois
 - Un processus est dans 3 états possibles, par rapport à l'accès à la ressource
 - Demandeur : demande à utiliser la ressource, à entrer dans la section
 - Dedans : dans la section critique, utilise la ressource partagée
 - Dehors : en dehors de la section et non demandeur d'y entrer
 - Changement d'état par un processus
 - De *dehors* à *demandeur* pour demander à accéder à la ressource
 - De *dedans* à *dehors* pour préciser qu'il libère la ressource
 - Le passage de l'état *demandeur* à l'état *dedans* est géré par le système et/ou l'algorithme de gestion d'accès à la ressource

Rappel exclusion mutuelle

- Diagramme d'états de l'accès en exclusion mutuelle



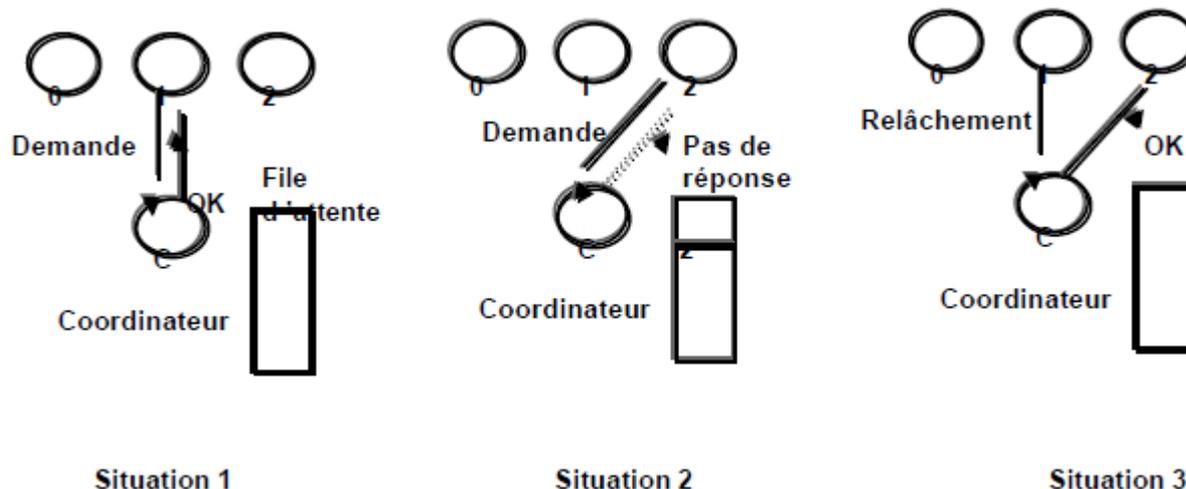
- L'accès en exclusion mutuelle doit respecter deux propriétés
 - Sûreté (safety) : au plus un processus est à la fois dans la section critique (dans l'état *dedans*)
 - Vivacité (liveness) : tout processus demandant à entrer dans la section critique (à passer dans l'état *dedans*) y entre en un temps fini

Techniques de l'exclusion mutuelle distribuée

- Plusieurs grandes familles de méthodes
 - Contrôle par un serveur qui centralise les demandes d'accès à la ressource partagée (Algorithme centralisé)
 - Contrôle par permission (Algorithme distribué – Ricart et Agrawala).
 - Les processus s'autorisent mutuellement à accéder à la ressource
 - Contrôle par jeton
 - Un jeton circule entre les processus et donne l'accès à la ressource
 - La gestion et l'affectation du jeton – et donc l'accès à la ressource – est faite par les processus entre eux
 - Deux approches : jeton circulant en permanence ou affecté à la demande des processus

Algorithme centralisé - Principe général

- Un serveur centralise et gère l'accès à la ressource
- Satisfaire les demandes des différents sites dans l'ordre où elles sont formulées => l'existence d'un ordre dans les requêtes => utilisation des horloges logiques scalaires.
- suppose que les canaux de communication entre les différents sites respectent la propriété FIFO.



Algorithme centralisé

- Un processus P_c est élu comme coordinateur
- Un processus voulant accéder à la ressource (quand il passe dans l'état *demandeur*) envoie une requête au coordinateur. Il se bloque jusqu'à ce qu'il reçoit un message OK.
- Quand le coordinateur lui envoie l'autorisation, il accède à la ressource (passe dans l'état *dedans*)
 - Si aucun processus n'est dans la SC S alors le coordinateur envoie un message OK à P_i
 - Si un autre processus P_j est dans la SC S alors :
 - Mettre la requête de P_i dans la file d'attente
 - Attendre que P_j libère la SC (passe dans l'état *dehors*) en envoyant un message LIBERE au coordinateur.
 - *Le coordinateur* retire le premier processus dans la file d'attente et lui envoie un message OK.
- Le coordinateur reçoit les demandes d'accès et envoie les autorisations d'accès aux processus demandeurs
 - Avec par exemple une gestion FIFO : premier processus demandeur, premier autorisé à accéder à la ressource

Algorithme centralisé - Bilan

- Avantages
 - Très simple à mettre en œuvre
 - Simple pour gérer la concurrence d'accès à la ressource (Utilise 3 messages (Demande, Libre, OK) pour chaque entrée/sortie d'une section critique.
- Inconvénients
 - Nécessite un élément particulier pour gérer l'accès
 - Si un processus ne reçoit pas de OK : Ressource utilisé ou coordinateur en panne ?
 - Potentiel point faible, goulot d'étranglement

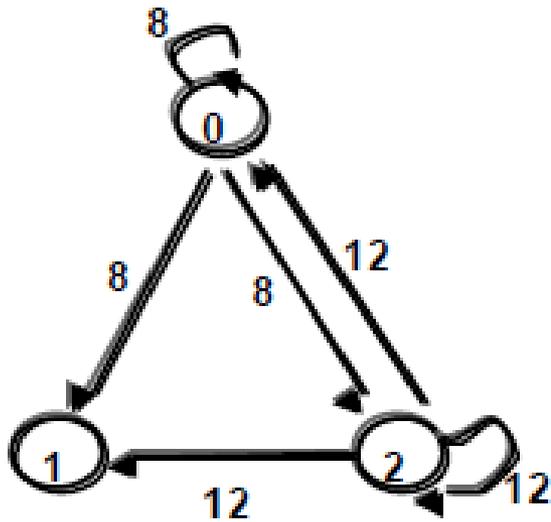
EM: Algorithme distribué

Ricart & Agrawala, 81

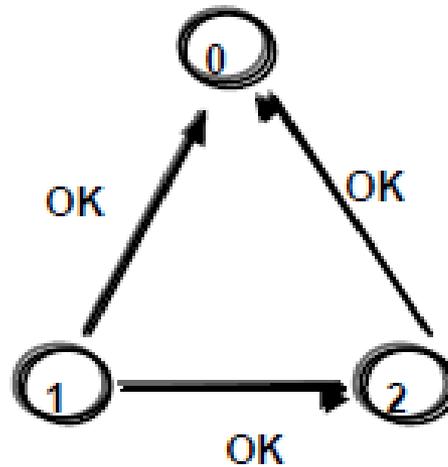
- Hypothèse : il existe un ordre total sur les événements (ex. utilisation de l'algorithme Lamport)
- Principe: duplication de la file d'attente des demandes de SC sur tous les sites
- Chaque processus demande l'autorisation à tous les autres (sauf lui par principe)
 - Liste des processus à interroger par le processus P_i pour accéder à la ressource : $R_i = \{ 1, \dots, N \} - \{ i \}$
- Se base sur une horloge logique (Lamport) pour garantir le bon fonctionnement de l'algorithme
 - Ordonnancement des demandes d'accès à la ressource
 - Si un processus ayant fait une demande d'accès reçoit une demande d'un autre processus avec une date antérieure à la sienne, il donnera son autorisation à l'autre processus
 - Et passera donc après lui puisque l'autre processus fera le contraire

EM: Algorithme distribué

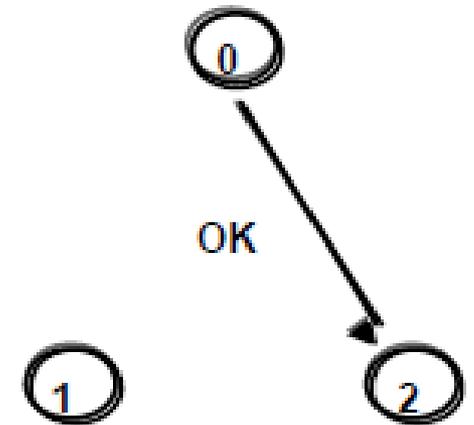
Ricart & Agrawala, 81



P0 et P2 veulent entrer dans une même section critique



P0 est plus ancien, il gagne et entre en section critique



P2 entre en section critique

EM: Algorithme distribué

- Fonctionnement

- Chaque processus gère les variables locales suivantes
 - Une horloge *H_i*
 - Une variable *dernier* qui contient la date de la dernière demande d'accès la ressource
 - L'ensemble *R_i*
 - Un ensemble d'identificateurs de processus dont on attend une réponse : *attendu*
 - Un ensemble d'identificateurs de processus dont on diffère le renvoi de permission si on est plus prioritaire qu'eux : *différé*
- Initialisation
 - $H_i = \text{dernier} = 0$
 - $\text{différé} = \emptyset, \text{attendu} = R_i$

EM: Algorithme distribué

- Fonctionnement

- Si un processus veut accéder à la ressource, il exécute
 - $H_i = H_i + 1$
 - $\text{dernier} = H_i$
 - $\text{attendu} = R_i$
 - Envoie une demande de permission à tous les processus de R_i avec estampille (H_i, i)
- Se met alors en attente de réception de permission de la part de tous les processus dont l'identificateur est contenu dans *attendu*
- Quand l'ensemble *attendu* est vide, le processus a reçu la permission de tous les autres processus
 - Accède alors à la ressource partagée
 - Quand accès terminé
 - Envoie une permission à tous les processus dont l'id est dans *différé*
 - *différé* est ensuite réinitialisé ($\text{différé} = \emptyset$)

EM: Algorithme distribué

- Fonctionnement

- Quand un processus P_i reçoit une demande de permission de la part du processus P_j contenant l'estampille (H, j)
 - Met à jour H_i : $H_i = \max(H_i, H)$
 - Si P_i pas en attente d'accès à la ressource : envoie permission à P_j
 - Sinon, si P_i est en attente d'accès à la ressource
 - Si P_i est prioritaire : place j dans l'ensemble *différé*
 - On lui enverra la permission quand on aura accédé à la ressource
 - Si P_j est prioritaire : envoi permission à P_j
 - P_j doit passer avant moi, je lui envoie ma permission
 - La priorité est définie selon la datation des demandes d'accès à la ressource de chaque processus
 - Le processus prioritaire est celui qui a fait sa demande en premier
 - Ordre des dates : l'ordre \ll de l'horloge de Lamport :
(dernier, i) \ll (H, j) si ((dernier < H) ou (dernier = H et i < j))
- Quand processus P_i reçoit une permission de la part du processus P_j
 - Supprime l'identificateur de P_j de l'ensemble attendu : $\text{attendu} = \text{attendu} - \{j\}$

EM: Algorithme distribué

- Bilan

- Robuste (mais, savoir exactement combien de processus sont vivants (N doit constamment être exacte))
- Sûreté : vérifiée (et prouvable...) et vivacité : assurée grâce aux datations et aux priorités associées
- Utilise $2(N-1)$ messages pour chaque entrée/sortie d'une section critique
- Si un parmi le N processus tombe en panne => Blocages des processus voulant entrer en SC
 - Solution : Utilisation d'un message acquittement +Timeout
- En général : L'algorithme est lent, complexe est plus couteux que l'algorithme centralisé néanmoins il prouve l'existence d'un algorithme distribué d'exclusion mutuelle

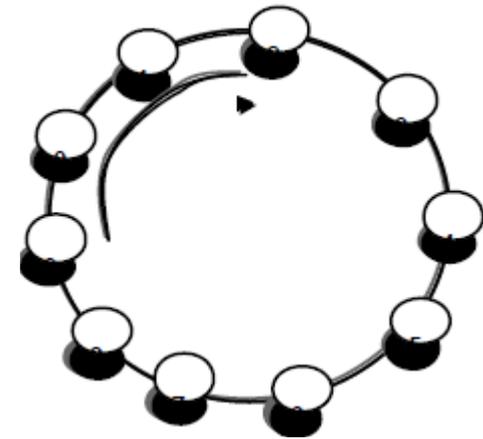
EM: Algorithme distribué

- Améliorations

- [Carvalho & Roucairol, 83]
 - Si P_i veut accéder plusieurs fois de rang à la ressource partagée et si P_j entre 2 accès (ou demandes d'accès) de P_i n'a pas demandé à accéder à la ressource
 - Pas la peine de demander l'autorisation à P_j car on sait alors qu'il donnera par principe son autorisation à P_i
 - Limite alors le nombre de messages échangés
- [Chandy & Misra, 84], améliorations tel que
 - Les processus ne voulant pas accéder à la ressource et qui ont déjà donné leur permission ne reçoivent pas de demande de permission
 - Horloges avec des datations bornées (modulo m)
 - Pas d'identification des processus

Exclusion mutuelle : méthode par jeton

- établir un anneau logique entre les processus
- introduire un jeton dans l'anneau
- le processus possédant le jeton peut entrer en section critique. Il garde le jeton jusqu'à sa sortie de la section critique. Il passe ensuite le jeton au processus suivant dans l'anneau logique
- sinon, il passe le jeton au processus suivant dans l'anneau logique
- Respect des propriétés
 - Sûreté : grâce au jeton unique
 - Vivacité : l'algorithme doit assurer que le jeton circule bien entre tous les processus voulant accéder à la ressource



En cas de panne:

- la panne d'un processus -> reformer l'anneau
- la perte de jeton -> algorithmes de détection et de régénération du jeton une fois l'anneau reformé

Exclusion mutuelle : méthode par jeton

- Inconvénients
 - Nécessite des échanges de messages (pour faire circuler le jeton) même si aucun site ne veut accéder à la ressource
 - Si le jeton est perdu, il faut le redémarrer. La détection de la perte est difficile puisqu'il n'y a pas de temps alloué
 - Temps d'accès à la ressource peut être potentiellement relativement long
 - Si le processus $i+1$ a le jeton et que le processus i veut accéder à la ressource et est le seul à vouloir y accéder, il faut quand même attendre que le jeton fasse tout le tour de l'anneau
- Avantages
 - Très simple à mettre en œuvre
 - Intéressant si nombreux processus demandeurs de la ressource
 - Jeton arrivera rapidement à un processus demandeur
 - Equitable en terme de nombre d'accès et de temps d'attente
 - Aucun processus n'est privilégié

Comparaison entre les 3 algorithmes

Type d'algorithme	Nombre de messages par entrée/sortie	Temps avant entrée	Problèmes
Centralisé	3	2	Défaillance du coordinateur
Distribué	$2(n-1)$	$2(n-1)$	Défaillance d'un processus
Anneau à jeton	1 à ∞	0 à $n-1$	Perte de jeton, défaillance d'un processus

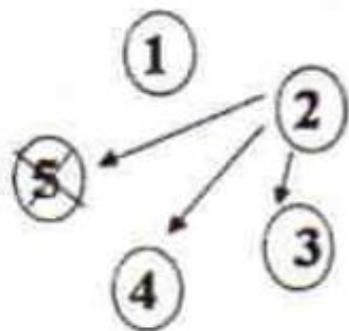
Algorithme d'élection

- Dans certaines configurations, il est important d'avoir un processus de coordination, appelé le coordinateur de l'exclusion mutuelle
- Election d'un coordinateur en prenant les hypothèses suivantes :
 - Chaque processus à un numéro unique, ce numéro définit une priorité, le plus grand numéro a la plus grande priorité.
 - Il y a un processus par machine (pour simplifier l'exemple).
 - Chaque processus connaît les numéros des autres processus.
 - Le processus ne sait pas quels sont les processus actifs ou inactifs.
 - Lorsqu'une élection démarre, le coordinateur est élu et tous les processus sont d'accord.
- L'objectif étant d'organiser une élection qui garantit que :
 - le processus élu est actif et unique
 - l'élection est terminée avec l'accord de tous les processus sur l'identité du processus élu

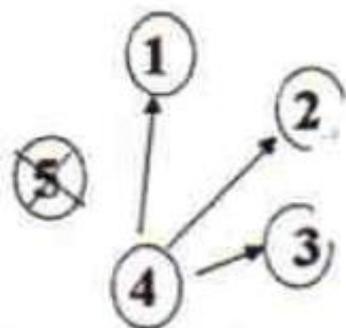
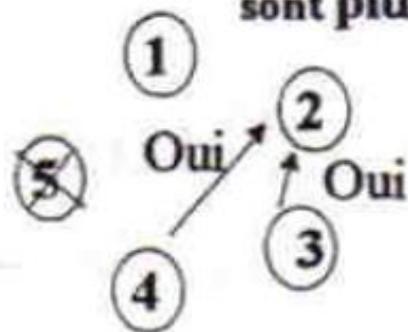
L'algorithme Bully

- Lorsqu'un processus s'aperçoit que le coordinateur ne répond plus, il initie une élection.
- **Principe :**
 - Un processus P détecte qu'une élection doit être initiée.
 - P envoie un message ELECTION à tous les processus ayant des n° plus élevés que lui.
 - Si aucun ne répond, P gagne l'élection et devient coordinateur.
 - Si un processus avec un numéro plus élevé répond, il prend la main et P a terminé.

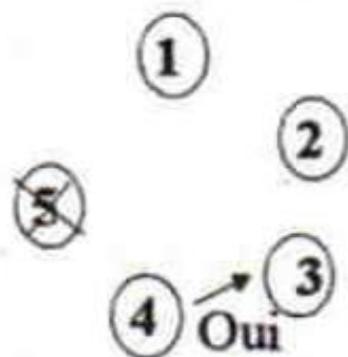
5 est le coordinateur, tombe en panne, 2 démarre l'élection



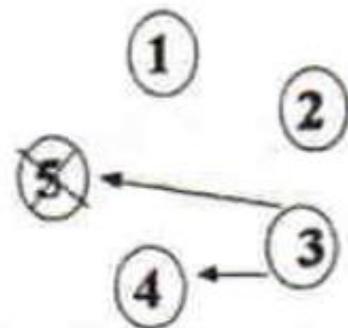
2 constate que 3 et 4 sont plus élevés, il s'arrête



4 signale qu'il est le coordinateur



3 constate que 4 est plus grand, il s'arrête



3 démarre une élection

L'algorithme anneau

- C'est une élection en réseau à anneau mais sans le jeton. C'est un processus ordonné car les membres connaissent leurs successeurs. Les processus sont placés dans un anneau virtuel (groupe).
- **Principe :**
 - Lorsqu'un processus s'aperçoit que le coordinateur est en panne, il initie une élection.
 - Il construit un message (qui remplace le jeton) et y place son numéro.
 - Le message ELECTION circule successivement d'un processus au suivant et chaque membre rajoute son numéro dans la liste (triée par ordre décroissant).
 - Si le successeur est en panne, le message est envoyé au suivant (jusqu'à ce qu'un processus valide soit trouvé).
 - Lorsque le message revient à l'initiateur (il trouve son numéro dans la liste), il change l'intitulé du message : ELECTION devient COORDINATOR.
 - Le message refait un tour et tous les membres prennent connaissance du processus qui est le nouveau coordinateur et quels sont les membres qui constituent l'anneau.
 - Le message est ensuite détruit et tous les membres continuent leur travail.

