Exclusion mutuelle distribuée

Plan

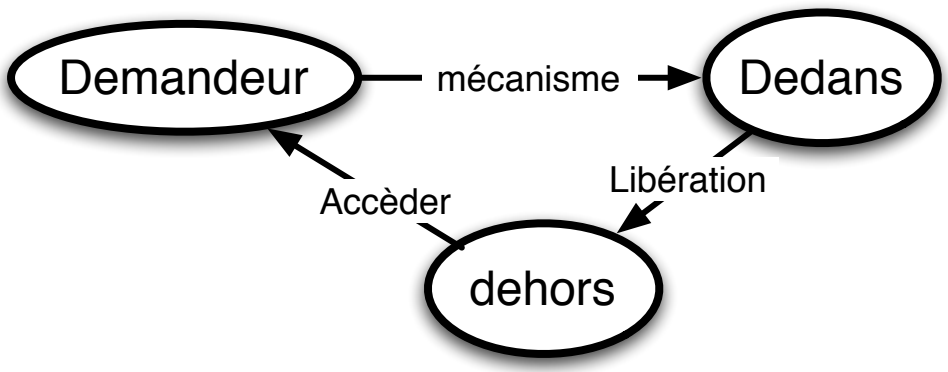
Exclusion mutuelle

- Problème
  - Contexte de plusieurs processus s'exécutant en même temps
  - Trouver un mécanisme distribué qui permet d'accéder à une ressource partagée par un seul processus à la fois

Exclusion mutuelle

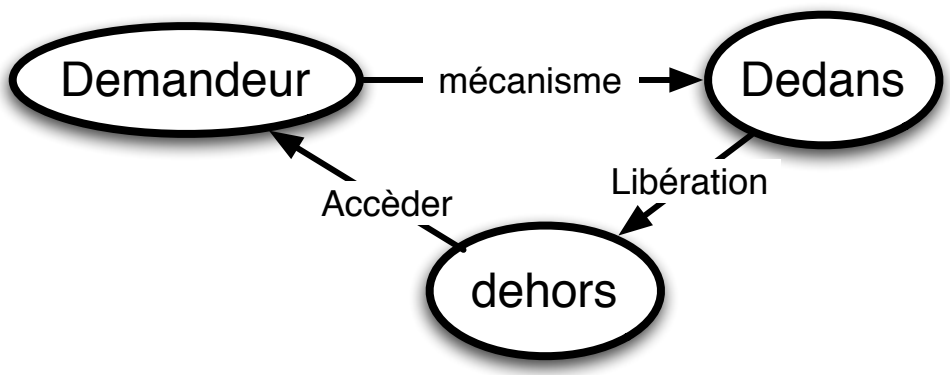
- Problème
  - Contexte de plusieurs processus s'exécutant en même temps
  - Trouver un mécanisme distribué qui permet d'accéder à une ressource partagée par un seul processus à la fois

  - Diagramme d'états pour les processus



Propriété de l'exclusion mutuelle

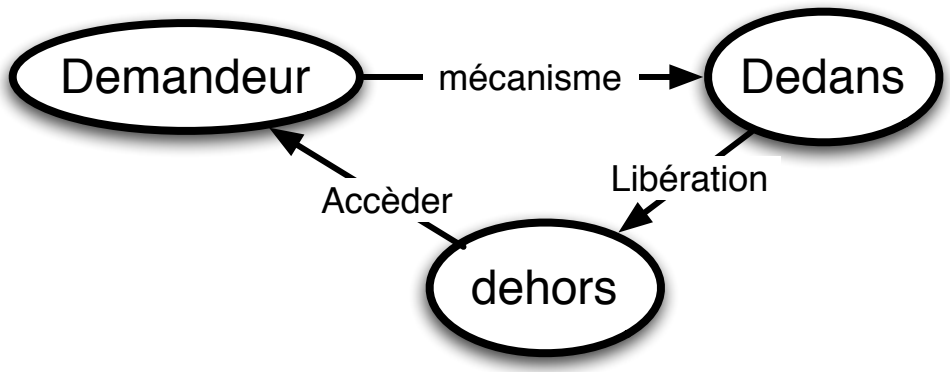
- Sûreté (safety) : au plus un processus est à la fois dans la section critique (dans l'état dedans)



Propriété de l'exclusion mutuelle

- Sûreté (safety) : au plus un processus est à la fois dans la section critique (dans l'état dedans)

- Vicacité (liveness) : tout processus demandant à entrer dans la section critique (à passer dans l'état *dedans*) y entre en un état fini.



Exclusion mutuelle

- Spécifications
  - Imposer un ordre sur l'exécution des sections critiques
  - Algorithme symétrique, décentralisé
  - Algorithme équitable

- Solution
  - Imposer un ordre global (ex : Algorithme de Ricart-Agrawala)
  - Imposer une topologie
    - Anneau virtuel
    - Arbre

Plan

Plus précisément

- Différentes types d'algorithmes
  - Algorithme Bakery
  - Algorithmes basés sur les jetons
    - Algorithme basé sur la circulation d'un jeton
  - Algorithme de Suzuki-Kasami
  - Algorithme de Ricart-Agrawala

Algorithme Bakery

- Principe : Un serveur centralise et gère l'accès à la ressource

Algorithme Bakery

- Principe : Un serveur centralise et gère l'accès à la ressource
- Algorithme :
  - Pour le site i
    1. il envoie une requête au serveur quand il veut accéder à la ressource
    2. Il accède à la ressource quand il reçoît l'autorisation du serveur
    3. Il informe le serveur quand il libère la ressource
  - Pour le serveur :

    Le serveur reçit les demandes d'accès et envoie les autorisations d'accès aux processus en fonction d'un ordre.

Analyse de l'algorithme

- Les propriétés de vivacité et de sûreté sont assurées.

Analyse de l'algorithme

- Les propriétés de vivacité et de sûreté sont assurées.

- Avantages :
  - Très simple à mettre en oeuvre
  - Simple pour gérer la concurrence d'accès à la ressource

Analyse de l'algorithme

- Les propriétés de vivacité et de sûreté sont assurées.

- Avantages :
  - Très simple à mettre en oeuvre
  - Simple pour gérer la concurrence d'accès à la ressource

- Inconvénients :
  - Nécessite un site particulier pour gérer l'accès
  - Possibilité d'avoir goulot d'étranglement
  - Non robuste au panne du serveur.

Plus précisément

- Différentes types d'algorithmes
  - Algorithme Bakery
  - Algorithmes basés sur les jetons
    - Algorithme basé sur la circulation d'un jeton
  - Algorithme de Suzuki-Kasami
  - Algorithme de Ricart-Agrawala

Exclusion mutuelle sur un anneau

- Les sites émettent et recoivent des messages

  Ils ont un prédécesseur et un successeur
- Un algorithme simple : circulation d'un jeton

Exclusion mutuelle sur un anneau

- Les sites émettent et recoivent des messages

  Ils ont un prédécesseur et un successeur
- Un algorithme simple : circulation d'un jeton
  - Seul le site qui possède le jeton peut accèder à la ressource

Exclusion mutuelle sur un anneau

- Les sites émettent et recoivent des messages

  Ils ont un prédécesseur et un successeur
- Un algorithme simple : circulation d'un jeton
  - Seul le site qui possède le jeton peut accèder à la ressource
  - Si un site possède le jeton, et qu'il ne veut pas accéder à la ressource critique alors

    Il envoie son jeton à son successeur

Exclusion mutuelle sur un anneau

- Les sites émettent et recoivent des messages

  Ils ont un prédécesseur et un successeur
- Un algorithme simple : circulation d'un jeton
  - Seul le site qui possède le jeton peut accèder à la ressource
  - Si un site possède le jeton, et qu'il ne veut pas accéder à la ressource critique alors

    Il envoie son jeton à son successeur
- Si un site libère la ressource critique alors

  Il envoie son jeton à son successeur

Exclusion mutuelle sur un anneau

- Les sites émettent et recoivent des messages

  Ils ont un prédécesseur et un successeur
- Un algorithme simple : circulation d'un jeton
  - Seul le site qui possède le jeton peut accèder à la ressource
  - Si un site possède le jeton, et qu'il ne veut pas accéder à la ressource critique alors

    Il envoie son jeton à son successeur
- Si un site libère la ressource critique alors

  Il envoie son jeton à son successeur
- A l'initialisation : il faut avoir un unique jeton.

Remarque

La validité de l'algorithme repose sur l'intégrité de l'anneau et sur l'existence et l'unicité du jeton

- sureté : unicité du jeton
- vivacité : existence du jeton + intégrité de l'anneau (+ durée limitée de la section critique)

Maintien de l'intégrité de l'anneau

- Hypothèses : pas de perte de message, défaillance des sites (aucune déconnextion du réseau)

Maintien de l'intégrité de l'anneau

- Hypothèses : pas de perte de message, défaillance des sites (aucune déconnextion du réseau)
- Principe :
  - Auto-surveillance (chaque site surveille son prédécesseur et son successeur)
  - En cas de détection de défaillance : reconfiguration de la topologie

Maintien de l'intégrité de l'anneau

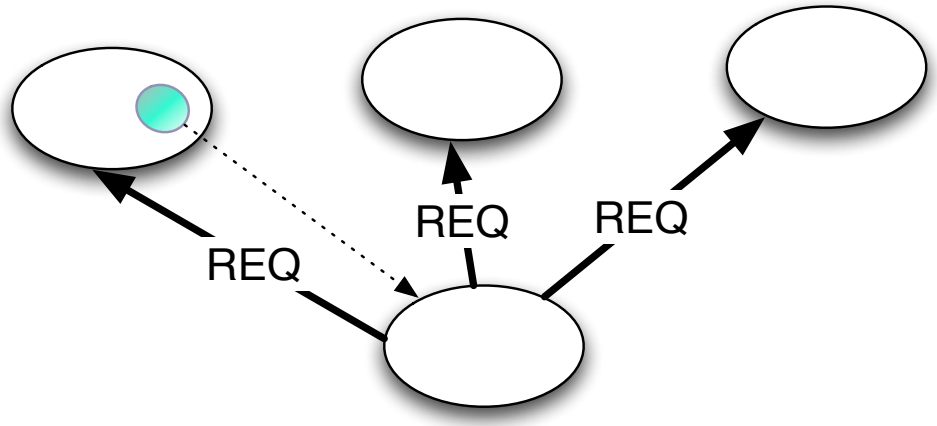
- Hypothèses : pas de perte de message, défaillance des sites (aucune déconnextion du réseau)
- Principe :
  - Auto-surveillance (chaque site surveille son prédécesseur et son successeur)
  - En cas de détection de défaillance : reconfiguration de la topologie
- Existence et unicité du jeton
  - détection de la perte du jeton
  - En cas de détection de perte : élection d'un site regénérer un jeton.

Plus précisément

- Différentes types d'algorithmes
  - Algorithme Bakery
  - Algorithmes basés sur les jetons
    - Algorithme basé sur la circulation d'un jeton
  - Algorithme de Suzuki-Kasami
  - Algorithme de Ricart-Agrawala

Algorithme à jeton :

- Principe : Un demandeur envoie sa requête à tous les processus et celui qui a le jeton lui répond



Algorithme à jeton :

- Principe : Un demandeur envoie sa requête à tous les processus et celui qui a le jeton lui répond

- Structure de données :
  - Pour chaque site i :

    un tableau req_i où $req_i[j]$ est le n^o de la dernière requête de j
  - Pour le jeton :

    un tableau $last$ où $last[j]$ est le n^o de la dernière visite de j dans la ressource critique.
    une queue Q correspondant aux sites qui attendent

Plus formellement : sur le site i (1/2)

Procédure d'acquisition

1. $etat_i \leftarrow demande$;
2. Si ($jeton\_present_i = faux$) alors
   - $req[i] \leftarrow req[i] + 1$; diffuser <Demande,i,req[i]>
3. Sinon
   3.1 $etat_i \leftarrow dedans$; ACCEDER A LA RESSOURCE

Réception de <Jeton, last,Q> venant de $\ell \rightarrow$

1. $jeton\_present_i = vrai$;
2. $etat_i \leftarrow dedans$; ACCEDER A LA RESSOURCE

Plus formellement : sur le site i (2/2)

Lors de la réception du message <Demande,ℓ,num> de ℓ

1. $req[\ell] \leftarrow max(req[\ell], num])$
2. Si ($jeton\_present_i = vrai$ et $etat_i \neq demande$) alors
   - Si $req[\ell] = last[\ell] + 1$, alors envoyer $< Jeton, last, Q \backslash \ell >$ à ℓ ; $jeton\_present_i = faux$

Procédure Libération

1. $etat_i \leftarrow sorti$; $last[i] \leftarrow req_i[i]$;
2. $\forall k \neq i :\neq (k \notin Q) \wedge req[k] = last[k] + 1 \rightarrow$ ajouter k à Q ;
3. Si $Q \neq \emptyset$ alors envoyer $< Jeton, last, Q \backslash \ell >$ au 1er site ℓ de Q

Plus précisément

- Différentes types d'algorithmes
  - Algorithme Bakery
  - Algorithmes basés sur les jetons
    - Algorithme basé sur la circulation d'un jeton
  - Algorithme de Suzuki-Kasami
  - Algorithme de Ricart-Agrawala

Algorithme basée sur les permissions

- Principe

  Gestion d'une file d'attente distribuée.

- Algorithme

  1. Chaque site *i* voulant accèder à la ressource date sa demande
  2. Il envoie à tout le monde

     un message de demande d'accès + sa date
  3. Tous les sites ne voulant pas y accèder ou voulant accéder mais ayant une date de demande postérieure à la sienne lui donnnent leur accord.
  4. Pour les autres, ils attendent qu'ils ont accèdé et libéré la ressource pour lui donner leur accord.
  5. Dès qu'il a reçu l'accord de tout le monde, il rentre dans la section critique.

Algorithme basée sur les permissions

- Principe

  Gestion d'une file d'attente distribuée.

- Algorithme

  1. Chaque site *i* voulant accèder à la ressource date sa demande
  2. Il envoie à tout le monde

     un message de demande d'accès + sa date
  3. Tous les sites ne voulant pas y accèder ou voulant accéder mais ayant une date de demande postérieure à la sienne lui donnnent leur accord.
  4. Pour les autres, ils attendent qu'ils ont accèdé et libéré la ressource pour lui donner leur accord.
  5. Dès qu'il a reçu l'accord de tout le monde, il rentre dans la section critique.

Algorithme basée sur les permissions

- Principe

  Gestion d'une file d'attente distribuée.

- Algorithme

  1. Chaque site *i* voulant accèder à la ressource date sa demande
  2. Il envoie à tout le monde

     un message de demande d'accès + sa date
  3. Tous les sites ne voulant pas y accèder ou voulant accéder mais ayant une date de demande postérieure à la sienne

     lui donnnent leur accord.
  4. Pour les autres, ils attendent qu'ils ont accèdé et libéré la ressource pour lui donner leur accord.
  5. Dès qu'il a reçu l'accord de tout le monde,

     il rentre dans la section critique.

Hypothèse : Tous les sites connaissent le nombre total de sites.

Plus formellement : sur le site i (2/2)

Procédure d'acquisition

1. $h_i \leftarrow h_i + 1$;
2. $D_i \leftarrow vrai$;$h\_d_i \leftarrow h_i$;
3. $rep\_attendues_i \leftarrow n - 1$;
4. pour tout ($j \in V \backslash \{i\}$) faire
   4.1 envoyer <Accès,h_d_i> à j

Procédure Libération

1. $D_i \leftarrow faux$; $h_i \leftarrow h_i + 1$;
2. pour tout $j \in V \backslash \{i\}$ faire
   Si $differe_i[j] = vrai$
   2.1 envoyer <ok> à j ;
   2.2 $differe_i[j] \leftarrow faux$;

Réception de <Accès,h_ℓ> venant de $\ell \rightarrow$

1. $h_i = max(h_i, h) + 1$;
2. si (non D_i) ou $(h\_d_i, i) > (h, \ell)$ alors envoyer <ok> à ℓ
3. sinon $differe_i[j] \leftarrow vrai$;

Réception de <ok> venant de $\ell \rightarrow$

1. $h_i = max(h_i, h) + 1$; $rep\_attendues_i \leftarrow rep\_attendues_i - 1$;
2. si ($rep\_attendues_i = 0$) alors ACCEDER A LA RESSOURCE

Propriété de sûreté

Propriété

L'algorithme vérifie l'algorithme de sûreté

Preuve :

- Pour en arriver là, il y a 3 cas possibles.

Propriété de sûreté

Propriété

L'algorithme vérifie l'algorithme de sûreté

Preuve :

- Par l'absurde, i et j rentrent en même temps dans la ressource critique.

- Pour en arriver là, il y a 3 cas possibles.

Propriété de sûreté

Propriété

L'algorithme vérifie l'algorithme de sûreté

Preuve :

- Par l'absurde, i et j rentrent en même temps dans la ressource critique.
- Donc, i a envoyé le message <okay> à j et j a envoyé le message <okay> à i.
- Pour en arriver là, il y a 3 cas possibles.

Propriété de sûreté

Propriété

L'algorithme vérifie l'algorithme de sûreté

Preuve :

- Par l'absurde, i et j rentrent en même temps dans la ressource critique.
- Donc, i a envoyé le message <okay> à j et j a envoyé le message <okay> à i.
- Pour en arriver là, il y a 3 cas possibles.
  1. i a fait sa requête à j avant que j puisse faire sa demande.

     On a $h_j \geq h$ et i est plus prioritaire que j
     i n'a pas envoyé le message <ok> à j.

Propriété de sûreté

Propriété

L'algorithme vérifie l'algorithme de sûreté

Preuve :

- Par l'absurde, i et j rentrent en même temps dans la ressource critique.
- Donc, i a envoyé le message <okay> à j et j a envoyé le message <okay> à i.
- Pour en arriver là, il y a 3 cas possibles.
  1. i a fait sa requête à j avant que j puisse faire sa demande.
     Donc ce cas est possible.
     On a $h_j \geq h$ et i est plus prioritaire que j
     i n'a pas envoyé le message <ok> à j.

Propriété de sûreté

Propriété

L'algorithme vérifie l'algorithme de sûreté

Preuve :

- Par l'absurde, i et j rentrent en même temps dans la ressource critique.
- Donc, i a envoyé le message <okay> à j et j a envoyé le message <okay> à i.
- Pour en arriver là, il y a 3 cas possibles.
  1. i a fait sa requête à j avant que j puisse faire sa demande. Donc ce cas est possible.
  2. j a fait sa requête à i avant que i puisse faire sa demande. Donc ce cas est possible.

Propriété de sûreté

Propriété

L'algorithme vérifie l'algorithme de sûreté

Preuve :

- Par l'absurde, i et j rentrent en même temps dans la ressource critique.
- Donc, i a envoyé le message <okay> à j et j a envoyé le message <okay> à i.
- Pour en arriver là, il y a 3 cas possibles.
  1. i a fait sa requête à j avant que j puisse faire sa demande. Donc ce cas est possible.
  2. j a fait sa requête à i avant que i puisse faire sa demande. Donc ce cas est possible.
- Les deux sites ont envoyé leur requête avant qu'ils en recoivent.
  - Les deux sont dans l'état demandeur.
  - 0n a $(h_i, i) > (h_j, j)$ ou $(h_i, i) < (h_j, j)$.
  - Donc un seul aura la priorité

Propriété de sûreté

Propriété

L'algorithme vérifie l'algorithme de sûreté

Preuve :

- Par l'absurde, i et j rentrent en même temps dans la ressource critique.
- Donc, i a envoyé le message <okay> à j et j a envoyé le message <okay> à i.
- Pour en arriver là, il y a 3 cas possibles.
  1. i a fait sa requête à j avant que j puisse faire sa demande.
     Donc ce cas est possible.
  2. j a fait sa requête à i avant que i puisse faire sa demande.
     Donc ce cas est possible.
- Les deux sites ont envoyé leur requête avant qu'ils en recoivent.
  - Les deux sont dans l'état demandeur.
  - 0n a $(h_i, i) > (h_j, j)$ ou $(h_i, i) < (h_j, j)$.
  - Donc un seul aura la priorité

Donc ce cas est possible.

Propriété de vivacité

Propriété

L'algorithme vérifie l'algorithme de vivacité

Preuve :

Propriété de vivacité

Propriété

L'algorithme vérifie l'algorithme de vivacité

Preuve :

- Les demandes de section critique sont totalement ordonnées

Propriété de vivacité

Propriété

L'algorithme vérifie l'algorithme de vivacité

Preuve :

- Les demandes de section critique sont totalement ordonnées
- Supposons qu'à un instant donné (h_i, i) soit la plus petite

Propriété de vivacité

Propriété

L'algorithme vérifie l'algorithme de vivacité

Preuve :

- Les demandes de section critique sont totalement ordonnées
- Supposons qu'à un instant donné (h_i, i) soit la plus petite
  - Tous les autres sites donnent la priorité au site i
  - le site i est dans la section critique pendant un temps fini
  - Le site i finira par en sortir.

Propriété de vivacité

Propriété

L'algorithme vérifie l'algorithme de vivacité

Preuve :

- Les demandes de section critique sont totalement ordonnées
- Supposons qu'à un instant donné (h_i, i) soit la plus petite
  - Tous les autres sites donnent la priorité au site i
  - le site i est dans la section critique pendant un temps fini
  - Le site i finira par en sortir.
- Toute demande finira par avoir la plus petite estampille.

Résumé

- Introduction aux algorithmes de base (hors défaillances)
  - Exclusion mutuelle
  - Election
  - Terminaison
- Introduction à quelques approches générales
  - Ordonnancement des évènements (horloges).
  - u) Contraintes sur la topologie (anneaux virtuels, arbres)