Terminaison

Master 2 Informatique - UFR S.A.T

Pr. Ousmane THIARE

ousmane.thiare@ugb.edu.sn
[www.ousmanethiare.com]

10 mai 2024

Chapitre 6 : Terminaison

Chapitre 6 : Terminaison

Introduction

Soit un ensemble de n tâches interdépendantes au sein d'une application. Ces n tâches ou processus sont répartis sur un graphe de communication. Chaque processus en cours d'exécution, exécute un algorithme séquentiel et échange des messages avec les autres processus via le graphe de communication.
Le problème est le suivant : l'arrêt de tous les processus correspond-il à l'arrêt définitif de l'application ou n'est-ce qu'un état transitoire ? (c'est-à-dire un message peut être en transit et provoquer le redémarrage d'un, puis de tous les processus) ?

- Un processus est soit actif soit inactif : il est actif s'il exécute du code, il est inactif s'il n'a rien à faire ;

- Un processus est soit actif soit inactif : il est actif s'il exécute du code, il est inactif s'il n'a rien à faire ;
- Seuls les processus actifs peuvent envoyer des messages ;

Introduction
Définitions

- Un processus est soit actif soit inactif : il est actif s'il exécute du code, il est inactif s'il n'a rien à faire ;
- Seuls les processus actifs peuvent envoyer des messages ;
- Un processus ne peut passer d'un état inactif à un état actif que sur réception d'un message : à tout message correspond du code a exécuter. Par contre, il peut passer de l'état actif à inactif à tout moment (il a fini son code) ;

Introduction
Définitions

- Un processus est soit actif soit inactif : il est actif s'il exécute du code, il est inactif s'il n'a rien à faire ;
- Seuls les processus actifs peuvent envoyer des messages ;
- Un processus ne peut passer d'un état inactif à un état actif que sur réception d'un message : à tout message correspond du code a exécuter. Par contre, il peut passer de l'état actif à inactif à tout moment (il a fini son code) ;
- Tous les processus sont actifs au lancement de l'application.

Un deuxième problème auquel nous ne nous intéresserons pas, c'est de savoir si les processus ont "bien terminés" c'est-à-dire de savoir si le résultat est satisfaisant localement (chaque processus a bien fait le calcul prévu : cette estimation est en général faite par le processus lui-même) et globalement (le résultat global est bien le résultat recherché).

Introduction
Hypothèses

- Tout processus qui ne reçoit plus de message se termine dans un temps fini.

- Tout processus qui ne reçoit plus de message se termine dans un temps fini.
- Le réseau est FIFO

Cas d'un graphe en anneau uni-directionnel

Les processus sont "ordonnés" sur le réseau par ordre croissant de leur numéro.

Cas d'un graphe en anneau uni-directionnel

Principe de l'algorithme (Dijstra et Van Gasten - 1983)

On utilise un jeton à deux états Termine ou État_Transitoire. Le processus P_0 qui veut déterminer si l'application est terminée, émet le jeton dans l'état Terminé. Si celui-ci lui revient dans le même état, l'application est terminée. Sinon (donc le jeton revient dans l'état État_Transitoire), il faut relancer la détection de la terminaison.

Cas d'un graphe en anneau uni-directionnel
L'algorithme

Chaque processus Pi maintient les deux variables locales suivantes :

- $couleur_i$ = Blanc ou Noir initialisé à Blanc

Initialement, le jeton est à l'état Terminé en P_0 qui lance la détection dès que son état passe à Inactif en transmettant le jeton à P_1.

Chaque processus Pi maintient les deux variables locales suivantes :

- $couleur_i$ = Blanc ou Noir initialisé à Blanc
- $etat_i$ = Actif ou Inactif initialisé à Actif

Initialement, le jeton est à l'état Terminé en P_0 qui lance la détection dès que son état passe à Inactif en transmettant le jeton à P_1.

État du processus et transmission du jeton

- Lorsque P_i reçoit un message, il passe à l'état Actif

État du processus et transmission du jeton

- Lorsque P_i reçoit un message, il passe à l'état Actif
- Un processus Actif ne transmet pas le jeton, il le transmettra dès qu'il deviendra inactif.

État du processus et transmission du jeton

- Lorsque P_i reçoit un message, il passe à l'état Actif
- Un processus Actif ne transmet pas le jeton, il le transmettra dès qu'il deviendra inactif.
- Lorsqu'il a fini de calculer, il passe à l'état Inactif.

État du processus et transmission du jeton

- Lorsque P_i reçoit un message, il passe à l'état Actif
- Un processus Actif ne transmet pas le jeton, il le transmettra dès qu'il deviendra inactif.
- Lorsqu'il a fini de calculer, il passe à l'état Inactif.
- Lorsqu'il envoie un message vers un processus P_j tel que j < i, il passe à l'état Noir.

Cas d'un graphe en anneau uni-directionnel
L'algorithme

Pourquoi ? Parce qu'il se peut que le processus P_j soit ré-activé par ce message APRÈS que P_j ait transmis le jeton à l'état État_Transitoire. Donc, si P_i transmet un jeton Termine et que tous les processus P_k>i sont inactifs, alors le jeton arrivera en P_0 dans l'état Termine. Donc si P_i se contentait de transmettre le jeton en signalant qu'il est inactif, rien ne préviendrait P_0 du réveil de P_j. D'où lorsque P_i recevra le jeton, dès qu'il sera inactif, il transmettra le jeton dans l'état Etat_Transitoire (en fait, cela revient à ce que P_i demande un nouveau tour). Puis il pourra repasser à Blanc.
D'où, un processus Inactif transmet le jeton d'état :

- identique à l'état du jeton reçu si il est dans l'état Blanc

Cas d'un graphe en anneau uni-directionnel
L'algorithme

Pourquoi ? Parce qu'il se peut que le processus P_j soit ré-activé par ce message APRÈS que P_j ait transmis le jeton à l'état État_Transitoire. Donc, si P_i transmet un jeton Termine et que tous les processus P_k>i sont inactifs, alors le jeton arrivera en P_0 dans l'état Termine. Donc si P_i se contentait de transmettre le jeton en signalant qu'il est inactif, rien ne préviendrait P_0 du réveil de P_j. D'où lorsque P_i recevra le jeton, dès qu'il sera inactif, il transmettra le jeton dans l'état Etat_Transitoire (en fait, cela revient à ce que P_i demande un nouveau tour). Puis il pourra repasser à Blanc.
D'où, un processus Inactif transmet le jeton d'état :

- identique à l'état du jeton reçu si il est dans l'état Blanc
- Etat_Transitoire si il est dans l'état Noir, puis passe à l'état Blanc.

Détection de la terminaison Si le jeton revient en P_0 avec l'état

- Terminé alors FIN : Terminaison Détectée.

Cas d'un graphe en anneau uni-directionnel
L'algorithme

Détection de la terminaison Si le jeton revient en P_0 avec l'état

- Terminé alors FIN : Terminaison Détectée.
- Etat_Transitoire ou si P_0 est actif, alors dès que P_0 redevient inactif, il remet le jeton à l'état Terminé et devient Blanc. Puis il relance la détection en transmettant le jeton à P_1.

Cas d'un arbre couvrant
Rappel

Lorsque l'on est en phase d'initialisation des calculs, seuls les pères peuvent transmettre des demandes vers les fils directs.
Pour un sommet Pi de l'arborescence, posons

$$etat_{i,local}(P_i) = \begin{cases} 1 \; si \; P_i \; a \; fini \; son \; code \\ 0 \; sinon \end{cases} \quad (1)$$

et D_{P_i} ensemble des descendants de P_i à qui P_i a transmis une demande.
Soit $etat_i(P_j)$ la vision qu'a le processus P_i de l'état du processus P_j alors

$$etat_i(P_i) = \begin{cases} etat_{i,local}(P_i) \; si \; D_i = \emptyset \\ etat_{i,local}(P_i) \times \prod_{P_j \in D_i} etat_i(P_j) \; sinon \end{cases} \quad (2)$$

Cas d'un arbre couvrant
Rappel

Lorsque l'on est en phase de calcul : tous les calculs ont été initiés, il suffit alors que chaque processus transmette la vision de son état dès que celui-ci est égal à 1, pour qu'un noeud puisse calculer son état. D'où :

En phase d'initialisation

- $etat_0(P_0) = 0$

1. $etat_i(P_i)$ a été remis à 0 à la réception P_i du message demandant un calcul

Cas d'un arbre couvrant
Rappel

Lorsque l'on est en phase de calcul : tous les calculs ont été initiés, il suffit alors que chaque processus transmette la vision de son état dès que celui-ci est égal à 1, pour qu'un noeud puisse calculer son état. D'où :

En phase d'initialisation

- $etat_0(P_0) = 0$
- chaque fois qu'un message part d'un processus P_i vers un fils P_j :

1. $etat_i(P_i)$ a été remis à 0 à la réception P_i du message demandant un calcul

Cas d'un arbre couvrant
Rappel

Lorsque l'on est en phase de calcul : tous les calculs ont été initiés, il suffit alors que chaque processus transmette la vision de son état dès que celui-ci est égal à 1, pour qu'un noeud puisse calculer son état. D'où :

En phase d'initialisation

- $etat_0(P_0) = 0$
- chaque fois qu'un message part d'un processus P_i vers un fils P_j :
  - P remet $etat_i(P_i)$ à 0 [1], $D_i = D_i \cap \{j\}$ puis P_i "attend" l'état de P_j pour recalculer son propre état ;

1. $etat_i(P_i)$ a été remis à 0 à la réception P_i du message demandant un calcul

Cas d'un arbre couvrant
Rappel

Lorsque l'on est en phase de calcul : tous les calculs ont été initiés, il suffit alors que chaque processus transmette la vision de son état dès que celui-ci est égal à 1, pour qu'un noeud puisse calculer son état. D'où :

En phase d'initialisation

- $etat_0(P_0) = 0$
- chaque fois qu'un message part d'un processus P_i vers un fils P_j :
  - P remet $etat_i(P_i)$ à 0 [1], $D_i = D_i \cap \{j\}$ puis P_i "attend" l'état de P_j pour recalculer son propre état ;
  - P_j remet $etat_j$, $local(P_j)$ et $etat_j(P_j)$ à 0, passe à l'état Actif et exécute son code.

1. $etat_i(P_i)$ a été remis à 0 à la réception P_i du message demandant un calcul

En phase de détection

- dès que $etat_{i\neq 0}(P_{i\neq 0}) = 1$, P_i transmet 1 à son père

En phase de détection

- dès que $etat_{i\neq 0}(P_{i\neq 0}) = 1$, P_i transmet 1 à son père
- dès que $etat_0(P_0) = 1$, l'application est terminée.

Cas général

Algorithme de Misra (1983) On suppose que le le graphe de communication est fortement connexe (c'est-à-dire que tout processus Pi peut envoyer un message à tout $P_{0<j\neq i\leq n}$: pas nécessairement directement mais en passant éventuellement en passant d'autres processus). Dans ce cas, la théorie des graphes nous assure qu'il existe un circuit [2] C qui comprend chaque arc (au moins une fois) du graphe du réseau.

2. ce circuit n'est pas toujours évident à trouver avec un algorithme réparti

Cas général

Algorithme de Misra (1983) On va utiliser un jeton pour trouver le nombre nb de processus visités suivant le circuit C qui soient restés inactifs entre deux passages de ce jeton.
Il est donc évident que l'application sera terminée lorsque nb = taille(C), où taille(C) est la taille du circuit en nombre de visites de processus (peut être supérieur à n car on peut passer plusieurs fois par un même processus).

Cas général

Algorithme de Misra (1983)

Posons :

- succ(P_i) successeur de P_i donné par le circuit [3] C

3. que l'on suppose construit

Cas général

Algorithme de Misra (1983)

Posons :

- succ(P_i) successeur de P_i donné par le circuit [3] C
- Pour chaque processus P_i les quatre variables locales suivantes :

3. que l'on suppose construit

Cas général

Algorithme de Misra (1983)

Posons :

- succ(P_i) successeur de P_i donné par le circuit [3] C
- Pour chaque processus P_i les quatre variables locales suivantes :
  - $couleur_i$ = Blanc ou Noir initialisé à Blanc

3. que l'on suppose construit

Cas général

Algorithme de Misra (1983)

Posons :

- succ(P_i) successeur de P_i donné par le circuit [3] C
- Pour chaque processus P_i les quatre variables locales suivantes :
  - $couleur_i$ = Blanc ou Noir initialisé à Blanc
  - $etat_i$ = Actif ou Inactif initialisé à Actif

3. que l'on suppose construit

Cas général

Algorithme de Misra (1983)

Posons :

- succ(P_i) successeur de P_i donné par le circuit [3] C
- Pour chaque processus P_i les quatre variables locales suivantes :
  - $couleur_i$ = Blanc ou Noir initialisé à Blanc
  - $etat_i$ = Actif ou Inactif initialisé à Actif
  - jeton_$present_i$ = Vrai ou Faux initialisé à Faux sauf pour P_k, k choisi aléatoirement

3. que l'on suppose construit

Cas général

Algorithme de Misra (1983)

Posons :

- succ(P_i) successeur de P_i donné par le circuit [3] C
- Pour chaque processus P_i les quatre variables locales suivantes :
  - $couleur_i$ = Blanc ou Noir initialisé à Blanc
  - $etat_i$ = Actif ou Inactif initialisé à Actif
  - jeton_$present_i$ = Vrai ou Faux initialisé à Faux sauf pour P_k, k choisi aléatoirement
  - nb_i = entier initialisé à 0

3. que l'on suppose construit

Cas général

Etat d'un processus

Lorsque P_i reçoit un message, il passe à l'état actif par : $etat_i$ = Actif

Chaque fois qu'il envoie un message, il mémorise ce fait par : $couleur_i$ = Noir. Ainsi, indirectement, il mémorise qu'il est possible qu'il ait réveillé un processus.

Lorsqu'il a fini de calculer, il passe à l'état inactif par : $etat_i$ = Inactif

Lorsqu'il reçoit le jeton, il effectue : nb_i = nb et jeton_$present_i$ = Vrai puis

s'il est inactif :

- soit il détecte la terminaison : FIN

s'il est Actif : il ne le transmettra que lorsqu'il redeviendra inactif.

Cas général

Etat d'un processus

Lorsque P_i reçoit un message, il passe à l'état actif par : $etat_i$ = Actif

Chaque fois qu'il envoie un message, il mémorise ce fait par : $couleur_i$ = Noir. Ainsi, indirectement, il mémorise qu'il est possible qu'il ait réveillé un processus.

Lorsqu'il a fini de calculer, il passe à l'état inactif par : $etat_i$ = Inactif

Lorsqu'il reçoit le jeton, il effectue : nb_i = nb et jeton_$present_i$ = Vrai puis

s'il est inactif :

- soit il détecte la terminaison : FIN
- soit il doit retransmettre le jeton

s'il est Actif : il ne le transmettra que lorsqu'il redeviendra inactif.

Cas général

Transmission du jeton

Si P_i est Inactif (ou si P_i passe à Inactif) avec jeton_$present_i$ = Vrai alors /* Pi transmet le jeton */ : si $couleur_i$= Noir [4]
alors /*on recommence la détection */
nb_i = 1
sinon /* on incrémente le nombre de processus inactifs déjà visité */
$nb_i = nb_i + 1$
$couleur_i$ = Blanc jeton_$present_i$ = Faux
envoyer(jeton, nb = nb_i) à succ(P_i)

Détection de la terminaison

si nb = taille(C) et $couleur_i$ = Blanc alors Terminaison Détectée

4. il a donc envoyé un message pendant sa phase d'activité