HAL
open science

De l'utilisation de la priorité en presence d'exclusion mutuelle

C. Kaiser

To cite this version:

C. Kaiser. De l'utilisation de la priorité en presence d'exclusion mutuelle. RR-0084, INRIA. 1981. inria-00076477

HAL Id: inria-00076477

https://inria.hal.science/inria-00076477

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

IRIA
CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél. 954 90 20

Rapports de Recherche

N° 84

DE L'UTILISATION DE LA PRIORITÉ EN PRÉSENCE D'EXCLUSION MUTUELLE

Claude KAISER

Juillet 1981

2070

DE L'UTILISATION DE LA PRIORITE EN PRESENCE D'EXCLUSION MUTUELLE

Claude KAISER

INRIA
Domaine de Voluceau
BP 105 - Rocquencourt
78153 LE CHESNAY

et

CNAM
292, rue Saint Martin
75141 PARIS CEDEX 03

Résumé

Nous montrons que la juxtaposition de deux techniques éprouvées, d'une part l'allocation d'unités centrales à l'aide de priorités fixes, d'autre part, la gestion de l'exclusion mutuelle, peut entraîner des délais de réponse qui ne sont pas les plus courts possibles. Ceci peut se révéler contraire aux spécifications des systèmes informatiques "temps réel". Nous proposons deux modifications pour améliorer le délai de réponse et montrons que cela peut entraîner des modifications dans la réalisation des primitives de synchronisation. Cette première partie de nature pragmatique est complétée par un essai de spécification plus précise de la notion d'urgence des processus d'un système informatique.

Abstract

It is shown that the use of fixed priority for processors scheduling and the constraint of mutual exclusion can conflict and result in response times which are not the shortest possible and which are therefore no longer acceptable for real time systems. Two modifications are proposed for the processors scheduling, one of which needs to modify also the implementation of synchronizing primitives. The first part of the paper is a pragmatic analysis ; the second part concerns a more formal presentation of the notion of urgency as applied to the processes of a computing system.

TABLE DES MATIERES

(TABLE OF CONTENTS)

1 - Introduction 1

2 - Présentation de la situation concrète 1

3 - Situation de conflit entre priorité et exclusion mutuelle 4

3.1 - Modélisation du système 5

3.2 - Le tournoi conflictuel 5

4 - Modification de l'algorithme d'allocation 6

4.1 - Priorité spéciale pour la section critique 6

4.2 - Réglage adaptatif de la priorité 7

5 - Un problème de réalisation de la méthode avec priorité spéciale 8

5.1 - Modification des opérations sur sémaphores 9

5.2 - Reexamen de la construction des opérations sur sémaphore 11

6 - Conclusion de l'approche pragmatique et extensions du problème 12

7 - Un essai de spécification 13

7.1 - Le formalisme utilisé 13

7.2 - Modèle pour des processus de même urgence 15

7.3 - Spécification de l'urgence 19

Remerciements 23

Bibliographie 23

DE L'UTILISATION DE LA PRIORITE EN PRESENCE D'EXCLUSION MUTUELLE

C. KAISER

Février 81

RAPPORT DE RECHERCHE N° 6

1 - Introduction

Les applications qualifiées de "temps réel" sont celles qui demandent aux systèmes informatiques de réagir dans un délai maximum fixé, en réponse à des événements extérieurs aléatoires. Comme à cause de leur coût, on ne peut pas toujours dédier un processeur à chaque événement, il faut savoir controler les choix qu'impliquent la simultanéité de plusieurs événements. Une opinion fréquente est qu'il suffit de donner une priorité fixe à chaque processus chargé de la réponse à un événement et d'attribuer les processeurs selon cette priorité ; en réalité, cette méthode n'est valable que si les processus ne font pas accès à des ressources communes.

Dans cet article, nous étudions une situation où le contrôle des processus par priorité fixe tombe en défaut parce qu'ils doivent respecter une contrainte d'exclusion mutuelle.

Nous exposons ensuite deux améliorations consistant à modifier la priorité d'un processus pendant qu'il s'exécute en exclusion mutuelle ; l'une d'elle lui associe une priorité fixe, l'autre adapte sa priorité en fonction des processus en attente de la fin de l'exclusion mutuelle.

Ce contrôle par priorité doit être réalisé par les primitives du système d'exploitation, et en particulier par celles qui interfèrent avec l'allocation des unités centrales. C'est le cas de la plupart des primitives de synchronisation. Nous montrons un exemple où la construction de primitives de synchronisation, ne permet pas de réaliser le contrôle voulu.

La priorité des processus ne suffit plus à caractériser les réactions du système. Nous sommes amenés à spécifier une notion voisine appelée urgence.

Notre approche reste pragmatique pour l'étude du fonctionnement sous le contrôle des priorités, mais pour spécifier l'urgence, nous nous appuyons sur le formalisme présenté par Abrial et Schuman [1].

2 - Présentation de la situation concrète

Dans les systèmes informatiques et plus particulièrement dans ceux qui sont utilisés pour la conduite d'applications industrielles, on se trouve souvent confronté à la situation suivante :

- En réponse à des événements extérieurs, (tels qu'un signal d'alerte, de dépassement de seuil ou de passage par une position,...), il faut commander certaines opérations (telles que la fermeture d'une sécurité, l'arrêt d'un moteur, ou la lecture d'un disque,...) en s'occupant d'abord des réactions les plus urgentes, toutes autres affaires cessantes si nécessaire.

Quand plusieurs opérations entrent en conflit pour l'utilisation d'une ressource du système (telle qu'une unité centrale, ou un enregistrement de données) ou de l'application (telle qu'un moteur), on règle ce conflit en se basant sur l'urgence relative de chaque réaction. Cette urgence fait partie des spécifications de l'application.

Les systèmes d'exploitation des constructeurs fournissent une solution traditionnelle, simple, à ce genre de problème. Pour chacune des réactions possibles, on prépare à l'avance un processus informatique qui sera déclenché par l'événement extérieur correspondant. A chaque processus est attachée une variable entière, appelée numéro de priorité du processus. En cas de conflit, c'est le processus qui a le plus petit numéro qui est choisi. Si plusieurs processus ont le même numéro, c'est qu'il est indifférent de choisir l'un plutôt que l'autre ; le choix peut être quelconque.

Le numéro de priorité du processus peut, selon les systèmes, être fixé une fois pour toute ou calculé en fonction d'une date limite, ou échéance, avant laquelle la réaction à l'événement extérieur doit s'être produite.

Ainsi en est-il pour l'allocation des unités centrales. Comme ni la date d'occurence des événements, ni la durée d'exécution de chaque processus ne sont connues a priori (dans le dernier cas, cette durée dépend de l'état de l'application au moment de l'événement), on ne peut faire un ordonnancement déterministe ; on maintient alors en permanence la liste des processus candidats aux unités centrales et on alloue celles-ci, avec réquisition, aux processus les plus prioritaires de cette liste. L'allocation des unités centrales est réexaminée à chaque événement du système qui modifie la liste des processus candidats (occurrence d'un événement extérieur, blocage d'un processus, entrée-sortie,...).

Cette méthode d'allocation se révèle conforme aux spécifications quand les processus sont indépendants entre eux. Mais souvent pendant une phase de leur déroulement, les processus peuvent être amenés à lire et à modifier des données communes (variables d'état de l'application) ; pour garder la cohérence de ces données,

il suffit de n'autoriser leur accès qu'en exclusion mutuelle [6] . Dans ce cas, un seul processus à la fois peut exécuter sa phase d'accès aux données communes, phase qu'on appelle section critique du processus. Cette contrainte d'exclusion mutuelle peut alors entrer en conflit avec la priorité et entraîner le non respect des spécifications d'urgence.

Dans la pratique, cette contrainte d'exclusion mutuelle peut être cachée au concepteur d'une application ; celui-ci croit spécifier et programmer des processus indépendants alors qu'en fait le système d'exploitation qui est fourni par le constructeur introduit à son insu des sections critiques pour la gestion des ressources.

Nous donnerons un exemple révélateur de ce conflit et nous présenterons des améliorations à l'algorithme d'allocation par priorité.

Nous montrerons un autre cas de non respect de la spécification qui sera dû, cette fois, à la difficulté d'implémenter l'algorithme d'allocation.

Nous terminerons par quelques propositions pour une définition plus rationnelle de l'urgence.

Nota 1. Nous avions rencontré ce problème lors de la construction du système ESOPE (*) [2] et il a été rappelé récemment à notre attention par C. Carré et D. Slosberg à propos du Système GECOS sur CII-HB 64 [4] ; ces deux systèmes utilisent des sémaphores pour gérer les sections critiques.

2. Avec la baisse du coût du matériel, la tendance actuelle consiste à multiplier et à spécialiser les calculateurs utilisés dans les applications ; toutefois, quand il y a des données ou des fichiers communs, l'utilisation de monoprocesseurs ou de multiprocesseurs avec mémoire commune persiste.

3. L'allocation des unités centrales est surtout étudiée dans le cadre de l'ordonnancement déterministe quand tous les paramètres d'un processus (date de début, durée d'exécution échéance, prédécesseur, successeur,...) sont fixes et connus à l'avance [5,9] . L'ordonnancement déterministe avec priorité ou échéance [3, 13, 14, 15] conduit à des algorithmes itératifs qui se rapprochent de l'ordonnancement non déterministe que nous considérons ici. Des valeurs limites de performances dans le cas d'ordonnancements déterministes avec sections critiques ont été données dans [12].

(*) le Système ESOPE a été réalisé à l'INRIA de 1969 à 1972.

3 - Situation de conflit entre priorité et exclusion mutuelle

3.1. Modélisation du système

Pour montrer que le conflit entre priorité et exclusion mutuelle peut entraîner le non respect des spécifications d'urgence, considérons la "situation" suivante entre les n processus d'un système régi par des priorités fixes.

Soit un ensemble de n processus T_1, T_2,..., T_n dont les programmes suivent le schéma :

Programme pour T_i : <u>début</u> ; attendre (E_i) ; α_i ; β_i ; γ_i <u>fin</u>

- où α_i, β_i et γ_i sont des suites d'instruction qui ne contiennent ni branchement d'une suite vers une autre, ni instruction de blocage de processus (comme opération d'entrée-sortie, sémaphore,...) ; les durées d'exécution de α_i, β_i, γ_i par une unité centrale qui serait affectée à T_i tout seul ne sont pas connues a priori, mais sont bornées ; on les note a_i, b_i, c_i ;

- où E_i l'événement extérieur qui déclenche le processus ; on note e_i la date de l'occurrence de E_i et f_i la date de fin d'exécution de T_i ; e_i et f_i ne sont pas connues a priori ; on sait seulement que $f_i \geqslant e_i + a_i + b_i + c_i$;

- où β_i est la section critique de T_i relativement aux données communes ; c'est pendant l'exécution de β_i qu'il y fait accès ; dans notre système il n'y a qu'une ressource critique, et partant, il n'y a qu'une section critique par processus.

Tous les processus n'utilisent pas nécessairement les données communes ; dans certains cas, les suites β_i et γ_i sont vides ; cette remarque a comme conséquence d'écarter la solution où une unité centrale et la ressource critique seraient allouées globalement à un processus T_i dès l'occurrence de l'évènement E_i ; on ne sait pas faire de prédiction sur l'utilisation des ressources ; celles-ci sont donc allouées à la demande.

Pour simplifier l'écriture, nous supposons que les processus sont numérotés selon le rang de l'urgence des actions qu'ils réalisent : ce que réalise T_1 est le plus urgent, ce que réalise T_n est le moins urgent.

A chaque processus T_i, on attache un numéro de priorité fixe p_i : si T_i est plus urgent que T_j, on a $p_i < p_j$. On a donc $p_1 < p_2 < \dots < p_n$. Les priorités $p_1, p_2, \dots, p_n$ sont utilisées pour gérer la file d'attente aux unités centrales et la file d'attente d'exclusion mutuelle.

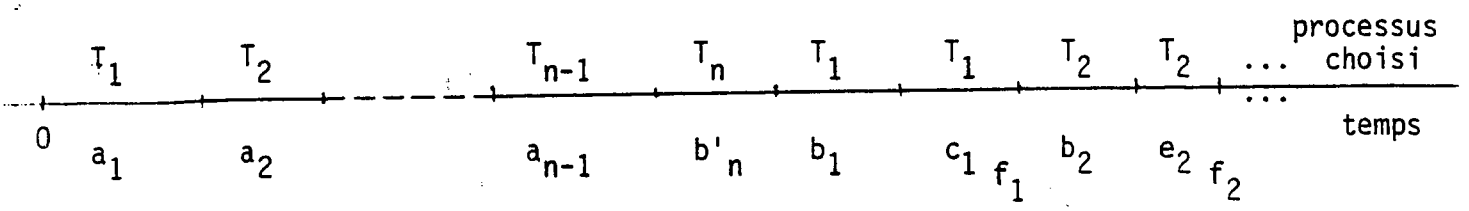
3.2. Le tournoi conflictuel

Considérons le tournoi suivant défini par l'état du système à l'instant $t = 0$:

- T_n exécute β_n : on dit qu'il occupe la ressource critique R
- $T_1, T_2, \dots T_{n-1}$ sont déclenchés par des évènements extérieurs ; c'est-à-dire que $e_1 = e_2 = \dots = e_{n-1} = 0$
  calculons les dates $f_1, f_2, \dots$ de fin d'exécution des processus $T_1, T_2, \dots$ et comparons les à leur date au plus près.

A $t = 0$, tous les processus deviennent candidats aux unités centrales. Supposons qu'il y en ait moins que de processus ; T_n se fait réquisitionner son unité centrale et est bloquée en section critique. Quand, à $t = a_1$, T_1 a fini d'exécuter α_1 et veut entrer en section critique β_1, il ne peut le faire car T_n est encore en section critique. T_1 est donc bloqué en attente d'exclusion mutuelle et son unité centrale est allouée à un autre processus. T_1 reste bloqué jusqu'à ce qu'il y ait une unité centrale pour T_n ; mais comme T_n est le moins prioritaire, il faut attendre que tous les autres processus plus prioritaires soient servis.

Pour un système avec une unité centrale, le tournoi se déroule comme suit :



et il vient

$$f_1 = b'_n + a_1 = b_1 + c_1 + \sum_{i=2}^{n-1} a_i \quad \text{où } b'_n \leqslant b_n$$

$$f_2 = f_1 + b_2 + c_2, \dots$$

On a bien $f_1 < f_2 \ldots < f_n$ mais f_1 peut être très nettement supérieure à sa valeur minimale f_1^m , où $f_1^m = b'_n + a_1 + b_1 + c_1$

L'allocation avec priorités fixes peut donc entraîner, à l'occasion des retards excessifs dans la réponse à des évènements extérieurs.

Nota 1. Avec plusieurs unités centrales, l'expression de f_1 est plus complexe car les valeurs relatives des a_i interviennent. Toutefois, on trouve toujours 3 termes :

- $a_1 + b_1 + c_1$ qui exprime la durée de la réponse du processus T_1,
- un terme qui correspond à la libération de la section critique par T_n,
- un terme qui correspond à l'exécution des instructions $\alpha_2, \alpha_3, \ldots, \alpha_{n-1}$

2. Dans certains systèmes [7] , on utilise les échéances pour exprimer l'urgence. A chaque processus T_i, on associe une échéance d_i qui exprime que l'on souhaite que $f_i - e_i \leqslant d_i$; l'allocation des unités centrales doit alors satisfaire un critère global tel que :

- minimiser le nombre de processus pour lesquels $f_i - e_i > d_i$,
- maximiser $\Sigma\, q_i\, (d_i + e_i - f_i)$, où q_i est une pondération.

Labetoulle [13] a montré que l'algorithme optimal pour des processus indépendants consiste à prendre l'échéance d_i comme valeur pour la priorité p_i. Si on généralise cette règle sans précaution, on obtient la même anomalie qu'avec les priorités fixes.

4 - Modifications de l'algorithme d'allocation

Plusieurs méthodes permettent d'éviter ce défaut : elles consistent à faire évoluer la priorité d'un processus au cours de son exécution.

4.1. : Priorité spéciale pour la section critique

M1 : Une priorité spéciale p_s est attribuée à la section critique et tout processus T_i en section critique reçoit la priorité $p'_i = \min(p_i, p_s)$; en sortie de section critique, T_i reprend la priorité p_i.

Quand $p_s \leqslant p_1$ cela revient à interdire la réquisition d'une unité centrale pendant que le processus qui l'utilise est en section critique (cela revient à fonctionner interruptions masquées) ; cette interdiction peut avoir un effet néfaste sur les processus qui n'utilisent pas la ressource critique, c'est-à-dire pour les processus T_j pour lesquels β_j et γ_j sont vides.

C'est pourquoi on prend un réglage intermédiaire pour p_s, ce qui permet de partager les processus en deux classes, les superprioritaires et les autres. Si p_s est tel que $p_{j-1} < p_s \leqslant p_j$ alors tous les processus T_k tels que $k < j$ sont superprioritaires. Un processus superprioritaire qui n'utilise pas la ressource critique n'est pas retardé. S'il l'utilise, le tournoi où apparait le conflit entre priorité et exclusion mutuelle ne peut se produire qu'avec des processus superprioritaires ; ceci limite l'importance du retard dû aux Σa_i.

Cette méthode est simple à mettre en oeuvre, aussi est-elle souvent utilisée [4] .

Nota : une variante de cette méthode consiste à modifier la priorité d'une valeur fixe s pendant la section critique ; T_i reçoit alors la priorité $p'_i = p_i - s$

4.2. : Réglage adaptatif de la priorité

Une autre méthode peut être proposée en remarquant que le processus en section critique ne devient gênant que lorsqu'un processus plus urgent que lui veut à son tour entrer en section critique. Cette remarque conduit à :

M2 : Le processus T_i qui se trouve en section critique tandis que les processus $T_j, T_k, \ldots, T_n$ attendent pour respecter l'exclusion mutuelle reçoit la priorité $p'_i = \min(p_i, p_j, p_k, \ldots, p_n)$; tout passage en attente d'un nouveau processus T_h conduit à réévaluer p'_i ; quand le processus T_i sort de la section critique, il reprend la priorité p_i.

Compte-tenu de l'hypothèse qu'on ne sait pas, quand un processus T_k est déclenché par l'évènement E_k, s'il doit ou non passer en section critique, cette méthode respecte bien l'urgence.

Montrons en effet que, si un processus T_i attend, c'est au profit de processus plus urgents ou bien d'un processus qui est en section critique.

a) Si T_i n'est pas en attente de section critique, les unités centrales sont utilisées pour des processus T_g tels que $p_g < p_i$, donc plus urgents que T_i ou pour des processus T_h tels que $p'_h < p_i$; donc, comme $p'_h = \min(p_h, p_j, p_k, \ldots)$, si T_h n'est pas plus urgent que T_i, l'un au moins des processus en attente de section critique l'est et c'est pour lui donner l'accès à la section critique qur T_i est arrêté;

b) Si T_i est en attente de section critique, soit les unités centrales sont utilisées pour des processus T_g tels que $p_g < p_i$, donc tous plus urgents qur T_i, soit certaines unités centrales sont utilisées pour des processus T_h moins urgents que T_i mais alors le processus T_k en section critique est parmi eux car, par la règle M2, il a reçu momentanément la priorité $p'_k = \min(p_i, p_h, \ldots) \leqslant p_i$ pour libérer la section critique au profit de T_i ou de processus plus urgents que lui.

<u>Corollaire</u> : Le système ne peut tomber en interblocage tant qu'il y a au moins une unité centrale en bon fonctionnement.
S'il y a interblocage, tous les processus sont bloqués ; le plus prioritaire est également bloqué or s'il est bloqué, on vient de voir que c'est au profit d'un processus plus prioritaire que lui (il n'y en a pas) ou d'un processus en section critique, il y a donc un processus qui n'est pas bloqué ; on aboutit à une contradiction ; il n'y a donc pas d'interblocage possible.

5 - Un problème de réalisation de la méthode avec priorité spéciale

La mise en oeuvre de la méthode M1 où une priorité spéciale p_s est attribuée à la section critique pose un problème de compatibilité entre spécifications partielles. Dans le cas présent, il y a d'une part la spécification d'urgence qui fournit un ordre pour allouer les unités centrales, d'autre part la spécification d'exclusion mutuelle. Or certains mécanismes qui réalisent l'exclusion mutuelle garantissent aussi l'absence d'interblocage ; pour ce faire, on s'arrange pour que l'état dans lequel un, ou plusieurs, processus serait en attente devant une section critique vide soit un état impossible. Or c'est de cet état dont on a besoin pour lever le conflit entre priorité et exclusion mutuelle.

5.1. : Modification des opérations sur sémaphores

Pour mettre cette situation en évidence, supposons qu'on utilise un système ou une machine à sémaphores [2, 4] . Les processus T_i se programment alors :

Programme pour T_i : début $P(S_i)$; α_i ; P (mutex) ; β_i ; V (mutex) ; γ_i fin
avec : mutex, un sémaphore commun d'exclusion mutuelle, initialisé à 1,

- Si, un sémaphore privé à T_i, initialisé à 0.

Le processus T_i est déclenché par l'événement extérieur E_i dont l'effet s'exprime par $V(S_i)$.

Nota : Habermann a montré [10] l'absence d'interblocage de cette programmation de l'exclusion mutuelle.

Pour appliquer la méthode M1, et attribuer une priorité spéciale à la section critique, il faut changer la priorité de chaque processus. Ce changement peut se faire avant ou après la primitive P(mutex) par une instruction spéciale, ou bien être intégré dans une primitive P modifiée.

Faire ce changement avant P (mutex) revient à donner la nouvelle priorité p_s à tous les processus qui veulent entrer dans leur section critique, qu'ils y entrent réellement ou qu'ils soient mis dans la file d'attente du sémaphore mutex ; cela a deux inconvénients : d'une part la priorité ne joue plus dans cette file d'attente, d'autre part tous les processus qui attendent d'entrer en section critique se coalisent pour passer l'un derrière l'autre, quelle que soit leur priorité originelle.

Faire ce changement après P (mutex) conserve le risque de réquisition de l'unité centrale entre le P (mutex) et ce changement, donc pendant que le processus est en section critique.

Le changement, pour être correct, doit donc se faire dans les primitives P et V utilisées pour l'exclusion mutuelle.

Cela ne suffit pas. Considérons en effet comment pourraient être modifiés les sémaphores tels que Dijkstra les a définis [8]. Un tel sémaphore S est constitué d'une variable entière E(S) et d'une file d'attente F(S).

<u>Primitive</u> P'(S)

<u>début</u> <u>commentaire</u> soit T_i le processus de priorité p_i qui exécute cette primitive ;
E(S) : = E(S) - 1 ;
<u>si</u> E(S) $\leqslant$ 0 <u>alors</u>

<u>début</u> arrêter la progression du processus T_i et pour cela, ranger le vecteur d'état de T_i dans la file d'attente F(S), en tenant compte de sa priorité p_i, y noter l'adresse de l'instruction suivante de T_i et libérer l'unité centrale qu'utilise T_i
<u>fin</u>

<u>sinon</u> <u>commentaire</u> le passage est libre
<u>début</u> donner à T_i la priorité p_s et noter l'ancienne priorité p_i <u>fin</u>
<u>fin</u>

<u>Primitive</u> V'(S)

<u>début</u> <u>commentaire</u> soit T_i le processus qui exécute cette primitive.;
redonner à T_i son ancienne priorité p_i ;
E(S) : = E(S) + 1 ;
<u>si</u> E(S) $\leqslant$ 0 <u>alors</u>
<u>début</u>
sortir le vecteur d'état du processus le plus prioritaire de la file F(S)
<u>commentaire</u> : soit T_j ce processus ;
donner à T_j la priorité p_s et noter l'ancienne priorité p_j ;
placer T_j parmi les candidats aux unités centrales
<u>fin</u>
<u>fin</u>

La primitive V'(S) remplit deux rôles :

- elle sort de section critique le processus T_i avec la priorité p_s,
- elle élit et fait entrer en section critique le processus T_j et lui attribue la priorité p_s.

Ce second rôle est néfaste car il peut y avoir des processus de plus grande urgence que T_i, en attente d'unité centrale et cette situation n'est pas envisagée ; il y a alors coalition des processus en attente de la section critique au détriment des autres processus.

5.2. : Réexamen de la construction des opérations sur sémaphores

Cela nous conduit à préférer une définition moins classique des sémaphores [16, 17] , où le processus qui est extrait de la file F(S) par la primitive V(S) réexécute la primitive P(S), ce qui permet d'allouer les unités centrales avant d'allouer la section critique.

Il vient alors :

```
Primitive P"(S)
   début commentaire soit Ti le processus de priorité pi qui exécute cette primitive ;
   si E(S) ≥ 1 alors
          début commentaire  le passage est libre ;
          E(S) : = E(S) - 1 ;
          donner à Ti la priorité ps et noter l'ancienne priorité pi
          fin
   sinon début
             arrêter la progression de Ti et pour cela ranger le vecteur d'état
             de Ti dans la file d'attente F(S), en tenant compte de sa priorité pi,
             y noter l'adresse de l'instruction en cours dans Ti, c'est-à-dire P"(S) ;
             commentaire : ceci est la différence importante avec la primitive P
                           classique ;
             libérer l'unité centrale qu'utilise Ti
             fin
   fin

Primitive V"(S)
   début commentaire soit Ti le processus qui exécute cette primitive ;
   redonner à Ti son ancienne priorité pi ;
   E(S) : = E(S) + 1 ;
   si F(S) n'est pas vide alors
          début
          sortir le vecteur d'état du processus le plus prioritaire de la file F(S)
          commentaire : soit Tj ce processus ;
          placer Tj parmi les candidats aux unités centrales
          fin
   fin
```

Le processus T_j ne pourra entrer en section critique que s'il a une unité centrale pour exécuter la primitive P"(S), donc ici l'allocation d'unité centrale vient avant l'allocation de la section critique.

Nota 1. La méthode M2 ne pose pas de problème de réalisation avec les sémaphores à la Dijkstra. En effet la primitive V(mutex) de sortie de section critique est exécutée par le processus T_i :

- quand toutes les unités centrales ne sont pas requises pour des processus de plus grande urgence que T_i, et dans ce cas il n'y a pas de processus plus urgent que T_i qui soit en attente d'unité centrale,
- ou bien quand un processus T_j en attente de section critique est d'une urgence telle qu'il devrait avoir une unité centrale pour s'exécuter et dans ce cas c'est pour pouvoir sortir le vecteur d'état de T_j le plus tôt possible de la file F(mutex) que l'on favorise l'exécution de V(mutex) par T_i.

2. Le problème de réalisation qui vient d'être montré n'est pas spécifique aux sémaphores. Il se produit aussi avec le mécanisme des moniteurs [11] .

6 - Conclusion de l'approche pragmatique et extensions du problème

Nous avons montré que la juxtaposition de deux techniques éprouvées, d'une part l'allocation d'unité centrale à l'aide de priorités fixes, d'autre part la gestion de l'exclusion mutuelle, pouvait conduire à ne pas respecter la spécification du problème ; nous avons obtenu une solution correcte en considérant le problème globalement. Dans la pratique,cette démarche n'est pas facile dans la mesure où le concepteur d'une application ne peut modifier le système d'exploitation ; ici il faut pouvoir changer l'algorithme d'allocation des unités centrales et l'implantation, quelquefois cablée ou microprogrammée, des sémaphores.

Le problème présenté pour une seule section critique peut être étendu au cas de plusieurs sections critiques. Quand les sections critiques sont parcourues l'une après l'autre, l'extension est triviale car chaque section critique peut être examinée à tour de rôle. Les sections critiques peuvent aussi être emboitées de la façon suivante : la section critique pour la ressource R1, la section critique pour la ressource R2, ... la section critique pour la ressource Rm contiennent chacune une section critique pour la ressource R0. Dans ce cas, on peut avoir un tournoi pour chaque ressource R1, R2, ... Rm, et un tournoi entre m processus pour la ressource R0.

Dans l'allocation avec priorités fixes, des retards supplémentaires peuvent être introduits par chaque tournoi.

Dans la méthode M1 d'allocation avec priorité spéciale, les processus se retrouvent tous avec la même priorité pour l'accès à R0. Si on veut raffiner, on peut donner une priorité spéciale différente pour chaque ressource R1, R2,... Rm ; mais d'une part le choix de ces priorités n'est pas évident, d'autre part un processus peut très bien avoir plusieurs sections critiques ce qui changera sa priorité au cours du temps alors qu'il conserve la même urgence.

La méthode M2 avec le réglage adaptatif de la priorité s'étend simplement en attribuant au processus T_i en section critique pour R0 la priorité $p''_i = \min(p'_i, p'_j, p'_k, \ldots, p'_m)$ si $T_j, T_k, \ldots, T_m$ attendent l'accès à R0. Les résultats obtenus se conservent en considérant les priorités $p'_i \ldots$ au lieu des priorités $p_i, \ldots$

7. Un essai de spécification

L'approche pragmatique précédente a permis de montrer la complexité d'un problème apparemment simple. L'étape suivante nous paraît être d'introduire une spécification rigoureuse et formelle du problème pour en déduire d'une part des propriétés globales du système et d'autre part les propriétés que doivent nécessairement posséder les mécanismes élémentaires d'implantation.

La suite de cette note est une modeste tentative dans cette voie ; le formalisme utilisé est celui qu'Abrial et Schuman ont présenté dans [1] ; il a pour nous le mérite de rester proche de la connaissance intuitive et pragmatique des systèmes ; nous commençons par la présentation de ce formalisme puis nous l'utiliserons pour en déduire quelques propriétés du système et spécifier la notion d'urgence.

7.1. Le formalisme utilisé

Dans le formalisme présenté par Abrial et Schuman [1] , un système est caractérisé par son état qui évolue sous l'influence d'événements. Partant d'un état initial, l'histoire du système est défini comme la suite des états et des événements qui représentent cette évolution. On spécifie toutes les histoires admissibles en précisant l'ensemble fini des événements, l'ensemble des états dans lequel peut se trouver l'état initial et l'ensemble des états que l'on admet comme états finaux. Chaque événement est spécifié en lui associant son domaine, c'est-à-dire

le sous ensemble d'états dans lesquels il peut se produire. Il vient alors :

Soit S un ensemble (l'ensemble des états du système),
I un ensemble fini (l'ensemble des noms d'événements),
S_i (pour $i \in I$) une famille de sous-ensembles de S,
e_i une famille de fonctions de S_i dans S (les événements)

on a :

$$e_i : S_i \rightarrow S$$

tel que :

$$\forall_i \in I \quad . \forall_s \in S. \quad e_i(s) \neq s$$

Soit encore S_b (l'ensemble des états initiaux) et S_f (l'ensemble des états finaux), deux sous ensembles de S tels que

$$S_b \subset (S_f \cup (\bigcup_{i \in I} S_i))$$

$$(S_f \cap (\bigcup_{i \in I} S_i)) = \emptyset$$

Alors le 6-uplet $(S, S_b, S_f, I, S_i, e_i)$ est appelé un système.

Un système une fois initialisé n'atteint pas toujours un état final ; il peut :

- tomber en interblocage s'il atteint son état où aucun événement n'est autorisé,
- entrer dans une suite périodique d'états,
- durer indéfiniment.

On dit qu'un système admet une terminaison correcte si toutes ses histoires sont finies et se terminent dans un état final.

Hypothèses sous jacentes

Pour pouvoir observer les histoires du système, ce modèle comprend deux hypothèses :

a) c'est un modèle algébrique ; le temps n'est pas pris en considération. Donc, si un événement est possible, parce que le système se trouve dans un état appartenant au domaine de cet événement, alors l'événement peut être considéré et la transition d'état correspondante, si elle se produit, est instantanée. Il n'y a pas d'arrêt du système dans un état autre qu'un état final.

b) Deux évènements, e_1 et e_2, ne peuvent se produire "en même temps" et causer en parallèle une modification de l'état du système. Chaque transition d'état ne peut être la conséquence que d'un seul événement. On doit supposer l'existence d'un mécanisme élémentaire qui permet de garantir cette hypothèse. Par contre, l'indéterminisme existe entre e_1 et e_2 : e_1 peut succéder à e_2, tout comme e_2 peut succéder à e_1.

7.2. : Modèle pour des processus de même urgence

Pour commencer, appliquons le formalisme ci-dessus à une situation simplifiée où tous les processus ont la même urgence. Pour introduire la possibilité de réquisition des unités centrales, on suppose que les programmes suivent le schéma :

Programme pour Ti : <u>début</u> ; attendre (Ei) ; α_{i1} ; α_{i2} ; β_{i1} ; β_{i2} ; γ_{i1} ; γ_{i2} <u>fin</u>

où la séquence β_{i1} ; β_{i2} doit être exécutée en section critique. La réquisition peut se faire en fin de chaque phase.

Un état s du système précise :

x : le nombre des processus non déclenchés par un événement "extérieur"

y_1 : le nombre des processus en phase α_{i1} de leur exécution,

y_2 : le nombre des processus en phase α_{i2} de leur exécution,

z_1 : le nombre des processus en phase β_{i1} ,

z_2 : le nombre des processus en phase β_{i2} ,

w_1 : le nombre des processus en phase γ_{i1} ,

w_2 : le nombre des processus en phase γ_{i2} ,

t : le nombre des processus ayant terminé.

Les événements du système sont :

e_1 : arrivée d'un événement "extérieur", déclenchant un processus,

e_2 : passage d'un processus de la phase α_{i1} à la phase α_{i2} ,

c_1 : entrée d'un processus en section critique ,

c_2 : passage d'un processus dans la 2e phase β_{i2} de sa section critique ,

d_1 : sortie d'un processus de section critique ,

d_2 : passage d'un processus de la phase γ_{i1} à la phase γ_{i2} ,

g : fin d'un processus.

L'ensemble des états du système est :

$$S = \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \{0,1\} \times \{0,1\} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$$

L'ensemble des noms d'événements est :

$$I = \{e_1 , e_2 , c_1 , c_2 , d_1 , d_2 , g \}$$

L'ensemble des états initiaux est :

$$S_b = \mathbb{N} \times \{0\} \times \{0\} \times \{0\} \times \{0\} \times \{0\} \times \{0\} \times \{0\}$$

L'ensemble des états finaux est :

$$S_f = \{0\} \times \{0\} \times \{0\} \times \{0\} \times \{0\} \times \{0\} \times \{0\} \times \mathbb{N}$$

L'évènement e_1 est :

$$e_1 (x, y_1, y_2, z_1, z_2, w_1, w_2, t) = (x - 1, y_1 + 1, y_2, z_1, z_2, w_1, w_2, t)$$

et son domaine est :

$$S_{e_1} = \mathbb{N}^+ \times \mathbb{N} \times \mathbb{N} \times \{0,1\} \times \{0,1\} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$$

de même on a :

$$e_2 (x, y_1, y_2, z_1, z_2, w_1, w_2, t) = (x, y_1 - 1, y_2 + 1, z_1 , z_2 , w_1, w_2, t)$$

$$S_{e_2} = \mathbb{N} \times \mathbb{N}^+ \times \mathbb{N} \times \{0,1\} \times \{0,1\} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$$

Et puis :

$$c_1 (x, y_1, y_2, 0, 0, w_1, w_2, t) = (x, y_1, y_2 - 1, 1, 0, w_1, w_2, t)$$

$$S_{c_1} = \mathbb{N} \times \mathbb{N} \times \mathbb{N}^+ \times \{0\} \times \{0\} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$$

Et puis :

$$c_2 (x, y_1, y_2, 1, 0, w_1, w_2, t) = (x, y_1, y_2, 0,1, w_1, w_2, t)$$

$$S_{c_2} = \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \{1\} \times \{0\} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$$

Et puis :

$$d_1 (x, y_1, y_2, 0, 1, w_1, w_2, t) = (x, y_1, y_2, 0, 0, w_1 + 1, w_2, t)$$

$$S_{d_1} = \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \{0\} \times \{1\} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$$

Et puis :

$d_2\ (x, y_1, y_2, z_1, z_2, w_1, w_2, t) = (x_1, y_1, y_2, z_1, z_2, w_1 - 1, w_2 + 1, t)$

$S_{d_2} = \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \{0,1\} \times \{0,1\} \times \mathbb{N}^+ \times \mathbb{N} \times \mathbb{N}$

Et enfin :

$g\ (x, y_1, y_2, z_1, z_2, w_1, w_2, t) = (x_1, y_1, y_2, z_1, z_2, w_1, w_2 - 1, t+ 1)$

$S_g = \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \{0,1\} \times \{0,1\} \times \mathbb{N} \times \mathbb{N}^+ \times \mathbb{N}$

Le nombre constant, n, des processus du système est un invariant. C'est

$x + y_1 + y_2 + z_1 + z_2 + w_1 + w_2 + t = n$

On peut noter quelques propriétés de ce système

a) $S = S_f \cup Se_1 \cup Se_2 \cup Sc_1 \cup Sc_2 \cup Sd_1 \cup Sd_2 \cup Sg$

Tout état $s \in S$ du système est soit un état final, soit un état appartenant au domaine d'un des événements du système ; l'interblocage est impossible.

b) Il existe une fonction V(s), qui est non négative, qui est strictement décroissante sur toute histoire admissible du système, et qui devient nulle quand le système est dans un état final.

Cette condition est suffisante pour que le système ne puisse ni entrer dans une suite périodique d'états, ni durer indéfiniment.

On peut prendre $V(s) = 7\,x + 6\,y_1 + 5\,y_2 + 4\,z_1 + 3\,z_2 + 2\,w_1 + w_2$

En effet

1) $\forall s \in (\bigcup_{i \in I} S_i)$, $V(s) > 0$

2) $\forall s \in S_f$, $V(s) = 0$

3) $\forall i \in I$, $\forall s \in Si$, $V(i(s)) < V(s)$

c) Par suite des propriétés a) et b) ci-dessus, le système admet une terminaison correcte. On se reportera à [1] pour les démonstrations de ces résultats.

7.3. : Spécification de l'urgence

Dans le modèle précédent lorsque plusieurs événements peuvent se produire parce que l'état s du système appartient à l'intersection des domaines de ces événements, on ne précise pas lequel de ces événements est choisi, car ce choix est indifférent puisque toutes les histoires sont admissibles. Seul importe de ne choisir qu'un seul événement à la fois.

Nous allons maintenant intervenir sur ce choix pour satisfaire des contraintes supplémentaires d'urgence et favoriser les événements qui font évoluer les processus les plus urgents.

On étiquette chaque processus en lui donnant une urgence fixe ; on crée ainsi une relation de précédence entre processus. Au lieu de représenter l'état du système par 8 variables qui sont des nombres de processus dans chaque phase, on utilise 8 variables qui repèrent les ensembles de processus dans chaque phase, Soit E l'ensemble fini des processus du système ; ces 8 variables définissent alors une partition de E.

La même démarche qu'en 7.2. peut être suivie avec ces nouvelles variables.

L'invariant du système devient :

$$x \cup y_1 \cup y_2 \cup z_1 \cup z_2 \cup w_1 \cup w_2 \cup t = E$$

ou encore, si $|u|$ représente le cardinal de u, puisqu'on a une partition de E :

$$|x| + |y_1| + |y_2| + |z_1| + |z_2| + |w_1| + |w_2| + |t| = n$$

La fonction V(s) est ici :

$$V(s) = 7\,|x| + 6\,|y_1| + 5|y_2| + 4|z_1| + 3|z_2| + 2|w_1| + |w_2|$$

Et les résultats du 7.2. se conservent.

Dans ce nouveau modèle, chaque domaine Si, pour $i \in I$, est un ensemble produit de familles de parties de E.

Notons S_{im} , le sous ensemble de Si qui valide l'événement i pour le processus d'étiquette m (c'est-à-dire que le processus m change de phase d'exécution). Notons i_m cette variété de i. Cette notation revient à considérer que l'événement i est formé de l'ensemble $\{i_1, i_2, \ldots, i_n\}$.

Pour un étiquetage donné des processus, en terme d'urgences, nous devons définir une règle de fonctionnement idéal. Pour ce faire, on privilégie un sous-ensemble des histoires admissibles, qu'on appelle le sous ensemble des histoires urgentes. Les algorithmes de controle de l'allocation et de la réquisition des processus sont "optimaux" lorsqu'ils arrivent à conduire le système selon une histoire urgente. Ces algorithmes sont appliqués chaque fois qu'une unité centrale est disponible, c'est-à-dire chaque fois qu'elle a fini le traitement d'une phase d'un processus.

Soit donc une sous suite h, appartenant au début d'une histoire urgente, et se terminant par un état s, non final. Nous définissons les prolongements de h qui appartiennent encore à des histoires urgentes ; pour cela nous considérons 2 cas de figure qui recouvrent tous les prolongements possibles de h, mais qui cependant ne s'excluent pas l'un l'autre.

a) Si $s \in Se_1$, c'est-à-dire si tous les processus ne sont pas encore déclenchés par les événements extérieurs, alors tout événement extérieur, qui, par nature, est incontrolable doit être accepté dans une histoire urgente ; donc toutes les histoires admissibles qui prolongent h par un événement e_1 sont des histoires urgentes.

b) Si $s \in Se_2 \cup Sc_1 \cup Sc_2 \cup Sd_1 \cup Sd_2 \cup Sg$, c'est-à-dire si certains processus ont été déclenchés, mais ne sont pas tous terminés, alors on peut restreindre les histoires admissibles.

Remarquons d'abord que s'il n'y a qu'une seule unité centrale, c'est celle-ci qui engendre la suite des changements d'état et on peut ignorer le temps écoulé entre deux changements. Ceci n'est plus vrai s'il y a plusieurs unités centrales, car une fois une unité centrale allouée à une phase d'un processus, elle devient inutilisable pour les autres phases pendant un certain temps ; négliger ce fait viendrait à considérer la présence de n unités centrales comme équivalente à celle d'une unité centrale n fois plus rapide, ce qui n'est pas le cas ici.

Donc quand il y a n unités centrales, la définition d'une histoire urgente doit prendre en considération l'invalidation des événements i_m, et, partant des domaines S_{im}, associés aux processus m auxquels une des n - 1 unités centrales est déjà allouée au moment de l'allocation de la n^e unité centrale. L'état s doit être étendu pour prendre en compte toutes les allocations possibles de ces n - 1 unités centrales ; on ne le fait pas ici pour ne pas compliquer l'écriture.

Soit donc k l'étiquette du processus le plus urgent tel que :

i) $k \in y_1 \cup y_2 \cup z_1 \cup z_2 \cup w_1 \cup w_2$.

ii) et aucune unité centrale ne lui soit allouée.

Alors ce cas b) se subdivise en deux :

b1) S'il existe un i_k tel que $s \in Si_k$, c'est-à-dire si le processus k peut être exécuté, alors l'histoire qui prolonge h par i_k est une histoire urgente.

b2) S'il n'existe pas un i_k tel que $s \in Si_k$, alors on considère toutes les histoires admissibles qui prolongent h et qui conduisent le système dans un état s' pour lequel il existe un i_k tel que s' Si_k. Les sous suites s...s' sont de longueur variable. Comptons dans chaque sous suite le nombre de changements d'états qui ne sont pas dus à des événements extérieurs et repérons les sous suites qui ont le plus petit nombre de tels changements. Soit j le premier événement "non extérieur" d'une telle sous suite. Alors l'histoire qui prolonge h par j est une histoire urgente.

On peut faire quelques remarques sur cette définition :

- les histoires urgentes sont aussi des histoires admissibles, donc les propriétés vues en 7.2 sont conservées ;
- les sous suites s...s' ne sont pas vides et il existe toujours un événement j, car on a vu en 7.2. que le système admettait une terminaison correcte ;
- de même les sous suites s...s' ne sont pas infinies, à cause de l'existence de la fonction V(s) ;

- enfin, la définition récurrente d'une histoire urgente converge bien, car toujours à cause de l'existence de la fonction V(s), il n'y a pas de boucle infinie.

On complète cette définition récurrente des histoires urgentes en notant que :

c) si $s \in Sb$, alors s est une histoire urgente.

Nota :

1. L'algorithme M2 de règlage adaptatif de la priorité, présenté en 4.2., est une réalisation de cette spécification.

2. L'algorithme d'allocation avec priorités fixes n'en est qu'une approximation car on n'applique pas correctement le point b2).

Remerciements :
Gérard Césaroni, Philippe Darondeau, Xavier Rousset de Pina, Gérard Florin et Stéphane Natkin ont bien voulu me faire profiter de leurs critiques sur des versions préliminaires de ce papier. Je les en remercie. Je conserve cependant la responsabilité des inexactitudes et des obscurités qui subsistent dans la version actuelle.

Bibliographie :

[1] ABRIAL J.R., SCHUMAN S.A. Non deterministic system specification. International symposium on semantics of concurrent computation, Evian 1979. Lecture notes in computer science n° 70, Springer Verlag

[2] BETOURNE C., FERRIE J., KAISER C., KRAKOWIAK S., MOSSIERE J. System design and implementation using parallel processes. IFIP. Congress 1971

[3] BLAZEWICZ J. Scheduling dependant tasks with different arrival times. Modelling and performance Evaluation of Computer Systems. (Gelenbe, Beilner, ed)

[4] CII-HB, série 60, GECOS niveau 64

[5] COFFMAN E.G., JR (Ed). Computer and jobshop scheduling theory. John Wiley and Sons, N.Y. 1976

[6] CROCUS. Systèmes d'exploitation des ordinateurs. Dunod. 1975

[7] DERVILLE D., KAISER C., PEIROTES Y., TELLIER P. Le système Haliotis. AFCET, RAIRO 1, nov. 1967

[8] DIJKSTRA E.W. The structure of the T.H.E. multiprogramming system Communications of the ACM. 11, 5. May 1968.

[9] GONZALES M.J., JR Deterministic Processor Scheduling. ACM Computing Surveys 9, 3, sept 1977

[10] HABERMANN A.N., Synchronization of Communicating processes Communications of the ACM 15,3. Mars 1972

[11] HOARE C.A.R. Monitors : An operating system structuring concept. Communications of the ACM 17.10, oct. 1974

[12] KAFURA D. Task scheduling with critical constraints. IFIP Congress 1977

[13] LABETOULLE J. Un algorithme optimal pour la gestion des processus en temps réel. AFCET, RAIRO. 8, février 1974

[14] LIU C.L., LAYLOND J.W. Scheduling algorithms for multiprogramming in a hard real time environment. Journal of the ACM. 10, 1. janvier. 1973

[15] MANACHER G.K. Production and stabilization of real time task schedules. Journal of the ACM. 14, 3 july 1967

[16] OWICKI S, GRIES D. An axiomatic proof technique for parallel programs. Acta Informatica 6, 1976

[17] PRESSER L., Multiprogramming coordination. ACM Computing Survey 7,1 mars 1975

++++++++++++++++++++

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique