Synchronisation des processus

Principes et concepts

Gérard Padiou

Département Informatique et Mathématiques appliquées
ENSEEIHT

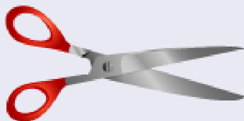
Septembre 2007

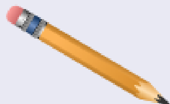
Plan

1. Introduction
   - Hypothèses
   - Problème sémantique
   - Concurrence
   - Coopération
2. Formalisme de description des systèmes parallèles
   - Les systèmes de transitions
   - Interprétation opérationnelle
   - L'expression des propriétés
   - L'expression des propriétés
3. Le problème de l'exclusion mutuelle
   - Modélisation et spécification
   - Solutions : en théorie...
   - Solutions : en pratique
4. Schémas génériques de synchronisation

Activités parallèles et partage de ressources

Compétition, concurrence



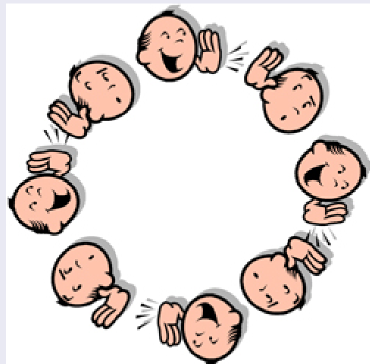


Coopération

Modèle centralisé ou réparti

Modèle centralisé

Modèle réparti



Les processus et les ressources

- les clients, les consommateurs, les utilisateurs : processus
- les ressources (critiques) :
  - mémoire (octet, mot, bloc,. . .),
  - le temps processeur,
  - des événements,
  - un fichier,
  - . . .

Modèle centralisé

- Partage du processeur : ordonnanceur (scheduler) ;
- Partage de variables : mécanismes de synchronisation.

Exécution parallèle « correcte » ?

Comment juger si une exécution parallèle est correcte ?

Sémantique d'entrelacement

Soit deux activités séquentielles $A = a_1; \ldots; a_n$ et $A' = a'_1; \ldots; a'_m$
Leur exécution en parallèle $A \| A'$ est correcte ssi le résultat est le même que celui obtenu par l'exécution séquentielle $A; A'$ ou $A'; A$.

Exemple

$$A = (x = x + 1; y = x) \text{ et } A' = (y = y - 1; x = y)$$

$$\{x = 0 \wedge y = 0\} \quad A; A' \quad \{x = 0 \wedge y = 0\}$$

$$\{x = 0 \wedge y = 0\} \quad A'; A \quad \{x = 0 \wedge y = 0\}$$

MAIS $\{x = 0 \wedge y = 0\} \quad A \| A' \quad \{x = 0 \wedge y = 0\}$ est faux

Problème : des processus ne se terminent pas ...

Comment encore juger si une exécution parallèle est correcte ?

Approche : Observation des états des processus tout au long de leur exécution

$\Rightarrow$ utilisation de la notion d'invariant

Problème pratique

Le parallélisme conduit à une explosion des séquences d'exécution possibles.

Analyse difficile dû au nombre d'états à vérifier.

Exemple de conflit

```
bool reserver ( int p ) {
  (1) if ( p > nDispo) return false ; /* test */
  (2)   nDispo = nDispo - p ; /* décrémentation */
  (3) return true ;
}
```

Exemple

$$\{nDispo = 12\}$$

$$\text{reserver}(10) \parallel \text{reserver}(5)$$

$$\{(nDispo = 7) \vee (nDispo = 2)\}$$

Exemple

Entrelacement possible pour reserver(10) ∥ reserver(5)

$$\{nDispo = 12\}$$

```
(1) if ( 10 > nDispo) return false ;
(1) if ( 5 > nDispo) return false ;
(2)   nDispo = nDispo - 5 ;
(3) return true ;
(2)   nDispo = nDispo - 10 ;
(3) return true ;
```

$\{nDispo = -3\}$ (overbooking. . .)

Exemple de coopération

int cpt = 0 ; // variable partagée

Capteur

```
process Capteur() {
  while (true) {
    attendre(passage) ;
    cpt++ ;
  }
}
```

Consignateur

```
process Consignateur() {
  while (true) {
    attendre(delai) ;
    printf("passés= "+cpt) ;
    cpt = 0 ;
  }
}
```

Problème de validité des exécutions

- Système réactif (pas de terminaison) ;
- Maintien d'un invariant vérifié lorsque les processus sont passifs ;
- Introduction de variables auxiliaires.

Exemple

```
process Capteur() {
  while (true) {
    attendre(passage) ;
    cpt++ ; Tot++ ;
  }
}
```

invariant $Tot = \sum_i v_i + cpt$

La notion de section critique

Section de code qui ne peut être exécutée que par un SEUL processus à la fois

- La section critique représente une ressource critique ≡ ressource utilisable par un seul client à la fois ;
- Si un processus P a commencé à exécuter une section critique, aucun autre processus ne peut commencer à exécuter cette section TANT QUE le processus P n'a pas terminé.

Exemple

```
bool reserver ( int p ) {
    (1) if ( p > nDispo) return false ;
    (2) nDispo = nDispo - p ;
    (3) return true ;
}
```

La notion d'exclusion mutuelle

L'exclusion mutuelle consiste à garantir qu'un seul processus au plus exécute une section critique parmi un ensemble E fixé de sections critiques. L'exécution d'une de ces sections par un processus exclut tous les autres processus qui tente d'exécuter une des sections critiques de E (y compris celle effectivement « utilisée »).

- L'ensemble des sections constitue en fait UNE ressource critique : un processus qui exécute une des sections a donc acquis le droit d'accès global à l'ensemble.
- Si un processus P a commencé à exécuter une section critique, aucun autre processus ne peut commencer à exécuter une des sections TANT QUE le processus P n'a pas terminé.

Cas du capteur-consignateur

Ensemble composé de deux sections critiques :

- L'incrémentation du compteur ;
- La lecture et la remise à zéro du compteur.

Exemple

```
process Capteur() {
  while (true) {
    attendre(passage) ;
    cpt++ ;
  }
}
```

Exemple

```
process Consignateur() {
 while (true) {
  attendre(delai) ;
   printf("passés= "+cpt) ;
   cpt = 0 ;
 }
}
```

Comment spécifier un problème de synchronisation ?

Comportements et propriétés des processus

- Abstraire le comportement des processus : utilisation des automates d'états finis et/ou systèmes de transitions :
  - Trouver les ressources partagées : points de conflits ou coopération (ici, variables globales partagées entre processus) ;
  - Déterminer les transitions à ordonnancer, synchroniser ;
  - Préciser les états des processus.
- Exprimer les propriétés : utilisation d'une logique temporelle :
  - Exprimer les états acceptables, permis sous forme de propriétés de sûreté ;
  - Exprimer les transitions qui doivent s'exécuter sous forme de propriétés de vivacité ;
  - Préciser des règles d'ordonnancement des transitions (propriétés d'équité).

Plan

1. Introduction
   - Hypothèses
   - Problème sémantique
   - Concurrence
   - Coopération
2. Formalisme de description des systèmes parallèles
   - Les systèmes de transitions
   - Interprétation opérationnelle
   - L'expression des propriétés
   - L'expression des propriétés
3. Le problème de l'exclusion mutuelle
   - Modélisation et spécification
   - Solutions : en théorie. . .
   - Solutions : en pratique
4. Schémas génériques de synchronisation

Un exemple : les systèmes de transitions

Définition

Quadruplet $S =< \mathcal{V}, Init, \Sigma, \mathcal{T} >$

- $\mathcal{V}$: ensemble fini de variables d'état ;
- $\Sigma = \{\sigma_i\}$: ensemble des états possibles ;
- *Init* : prédicat vérifié initialement par les variables ;
- $\mathcal{T} = \{\tau : \Sigma \rightarrow P(\Sigma)\}$: ensemble fini des transitions ;

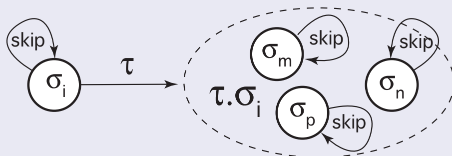
Plus précisément :

$\forall \sigma_i \in \Sigma : \tau.\sigma_i$ est l'ensemble **fini** des états successeurs accessibles de l'état σ_i par la transition τ.

Les systèmes de transitions
Sémantique opérationnelle

Les transitions

Une transition : $\tau.\sigma_i = \{\sigma_m, \sigma_n, \sigma_p\}$
La transition *skip* : $\forall \sigma_i \in \Sigma :: skip.\sigma_i = \sigma_i$



Les exécutions

Une exécution $\sigma = \sigma_0 \xrightarrow{\tau_0} \sigma_1 \xrightarrow{\tau_1} \sigma_2 \xrightarrow{\tau_2} \ldots$
Plus formellement : une suite (infinie) $\sigma = (\sigma_0, \sigma_1, \ldots)$ telle que :

$$\sigma_0.Init \wedge \forall i \geq 0 :: \langle \exists \tau \in \mathcal{T} :: \sigma_{i+1} \in \tau.\sigma_i \rangle$$

Les systèmes de transitions

Equité

- Faible : une transition toujours activable est finalement activée :

$$\forall \tau \in \mathcal{T} :: \neg \langle \exists i_0 : i_0 \geq 0 :: \langle \forall i \geq i_0 :: \tau.\sigma_i \neq \emptyset \wedge \sigma_{i+1} \notin \tau.\sigma_i \rangle \rangle$$

- Forte : une transition infiniment souvent activable est finalement activée :

$$\forall \tau \in \mathcal{T} :: \neg \langle \exists i_0 : i_0 \geq 0 :: \langle \forall i \geq i_0 :: \langle \exists j \geq i :: \tau.\sigma_j \neq \emptyset \rangle \wedge \sigma_{i+1} \notin \tau.\sigma_i \rangle \rangle$$

Les systèmes de transitions

Sûreté et Vivacité

Propriétés sur un ensemble d'exécutions infinies $\sigma \in \Sigma^\omega$

Notation : préfixe fini de σ : $\sigma_{0..i} = \sigma_0 \ldots \sigma_i$

- Sûreté : une exécution σ ne vérifie pas une propriété de sûreté P s'il existe un préfixe dont aucune extension ne vérifie P.

$$\sigma \notin P \Rightarrow \langle \exists i : i \geq 0 :: \langle \forall \alpha \in \Sigma^\omega :: \sigma_{0..i}\alpha \notin P \rangle \rangle$$

- Vivacité : une exécution σ vérifie une propriété de vivacité P si tout préfixe de σ possède un suffixe qui vérifie P.

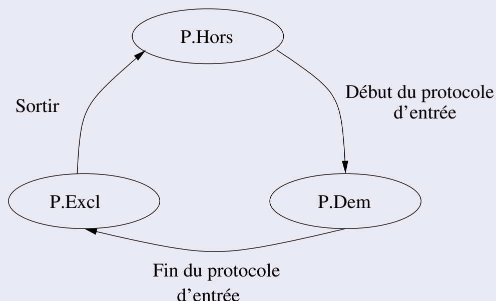
$$\sigma \in P \Rightarrow \langle \forall i : i \geq 0 :: \langle \exists \alpha : \alpha \in \Sigma^\omega :: \sigma = \sigma_{0..i}\alpha \in P \rangle \rangle$$

Plan

1. Introduction
   - Hypothèses
   - Problème sémantique
   - Concurrence
   - Coopération
2. Formalisme de description des systèmes parallèles
   - Les systèmes de transitions
   - Interprétation opérationnelle
   - L'expression des propriétés
   - L'expression des propriétés
3. Le problème de l'exclusion mutuelle
   - Modélisation et spécification
   - Solutions : en théorie...
   - Solutions : en pratique
4. Schémas génériques de synchronisation

L'exclusion mutuelle : étape de modélisation

Graphe de transitions d'un processus



L'exclusion mutuelle : spécification du problème

- Exclusion mutuelle (Sûreté) :

$$\textbf{invariant} \ \forall\, i,j \ : \ 0..N-1 \ :: \ P_i.Excl \wedge P_j.Excl \Rightarrow i = j$$

- Absence d'interblocage (Sûreté) : **Pas de :**

$$\forall I \subseteq \{0..N-1\} \ : \ \textbf{stable}\ (\forall i \in I \ : \ P_i.Dem)$$

- Absence de famine (Vivacité) :

$$\forall i \ : \ 0..N-1 \ :: \ P_i.Dem \ \mapsto \ P_i.Excl$$

- Indépendance (Vivacité) :

$$(\forall i \neq i_0 \ : \ P_i.Hors) \ \wedge \ P_{i_0}.Dem \ \mapsto \ P_{i_0}.Excl$$

Tentative de solution : un test TROP simple ...

```
boolean Libre = true ; /* variable partagée */

Entrer( pid i) {
  /* Pi.Hors → Pi.Dem */
  while( ! Libre ) {} ;
  Libre = false ;
  /* Pi.Dem → Pi.Excl */
}

Sortir(pid i) { Libre = true ; }
```

Tentative de solution : exclusion mais ...

```
boolean EnAccès [N] = false,... ;

Entrer(pid i) {
  EnAccès[i] = true ; boolean seul ;
  do {
    seul = true ;
    for (int k=0 ; k<N ; k++)
      if ( k != i ) seul = (seul && !EnAccès[k]) ;
  } until seul ;
/* Pas d'autre processus en accès */
}

Sortir(pid i) { EnAccès[i] = false ; }
```

Tentative de solution : exclusion et pas d'interblocage, mais ...

```
pid Tour = 0 ;

Entrer( pid i) {
   while ( i != Tour ) {}
}

Sortir(pid i) {
   Tour = (Tour + 1) mod N ;
}
```

Une solution pour 2 processus : algorithme de Peterson

```
type pid = 0..1 ;
pid Tour = 0 ; boolean EnAccès [N] = false,... ;

Entrer( pid i) {
  EnAccès[i] = true ;
  Tour = 1-i ;
  /* attente si l'autre processus est demandeur */
  /* et c'est son tour */
  while(EnAccès[1-i] && ( 1-i = Tour ) {}
}
Sortir(pid i) {
  EnAccès[i] = false ;
}
```

Une solution pour N ≥ 2 : algorithme de Lamport (Bakery)

```
type pid = 0..N-1 ;
boolean EnAcq[N] = false,... ; int num[N] = 0,... ;
Entrer( pid i) {
  /* Phase d'acquisition d'un numéro */
  EnAcq[i] = true ;
  int max=1 ;for (k=0 ;k<N ;k++) max=max+num[k] ;
  num[i]=max ; EnAcq[i]=false ;
  /* Phase d'attente éventuelle de son tour */
  for (k=0 ; k<N ; k++) {
   while (EnAcq[k]) do{} ;
   while (num[k] != 0 && (num[k],k) ≺ (num[i],i)) {} ;
  }
}
Sortir(pid i) { num[i] = 0 ; }
```

En pratique : Cas monoprocesseur

```
boolean Libre = true ;
Entrer( pid i) {
  IT_Masquer() ;
  while ( !Libre) {
  /* Fenêtre interruptible */
    IT_Démasquer() ; commuter() ; IT_Masquer() ;
  }
  Libre = false ;
  IT_Démasquer() ;
}

Sortir(pid i) { Libre = true ; }
```

En pratique : Cas multiprocesseur

- Rappel de l'instruction atomique TAS :

TAS(x) lit la valeur de x et affecte -1 à x

```
int Libre = 0 ;

Entrer( pid i) {
  while (TAS(Libre)<0) {}
}

Sortir(pid i) { Libre = 0 ; }
```

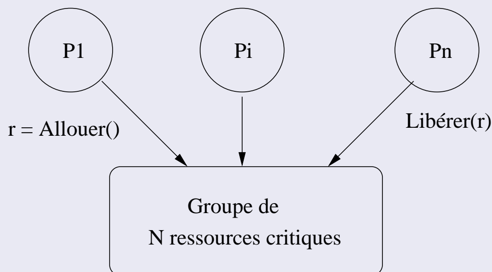
Synchronisation par attente passive

- Solutions avec attente active (boucle while) sur une condition :
  - consomme du temps processeur ;
  - acceptable si les processus sont en exclusion mutuelle pendant des périodes courtes.
- Idée : Attente passive
  - if <condition fausse> alors suspendre le processus ;
  - Réveil automatique d'un processus bloqué lorsque la condition est devenue vraie ;
  - Nécessite la gestion de files d'attente de processus bloqués.
- Mécanismes de synchronisation masquant la gestion des files de processus bloqués.

Plan

1. Introduction
   - Hypothèses
   - Problème sémantique
   - Concurrence
   - Coopération
2. Formalisme de description des systèmes parallèles
   - Les systèmes de transitions
   - Interprétation opérationnelle
   - L'expression des propriétés
   - L'expression des propriétés
3. Le problème de l'exclusion mutuelle
   - Modélisation et spécification
   - Solutions : en théorie. . .
   - Solutions : en pratique
4. Schémas génériques de synchronisation

Schéma d'allocateur de ressources critiques

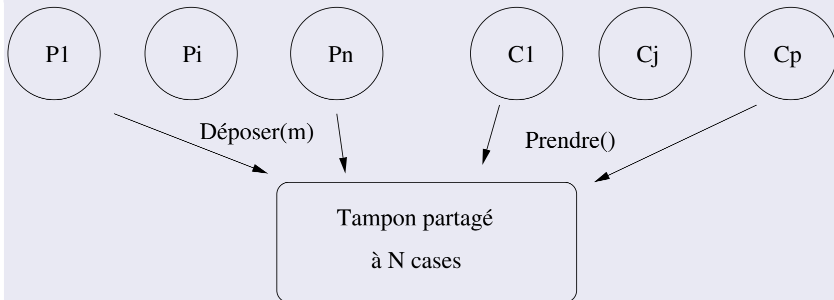


Hypothèse : toute ressource allouée est finalement libérée.

- Sûreté : $0 \leq$ #Allouer - #Libérer $\leq$ N
- Vivacité : Toute requête finit par être servie.

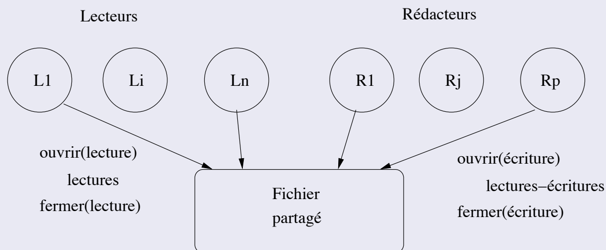
Notation : $\#op$ = nombre d'exécutions de op.

Schéma producteurs-consommateurs



- Sûreté : $0 \leq$ #Déposer - #Prendre $\leq$ M
- Vivacité : consommation dans l'ordre chronologique de production

Schéma des lecteurs-rédacteurs



- int *nl* : nombre de lecteurs en cours
- int *nr* : nombre de rédacteurs en cours

- Sûreté : $(0 \leq nr \leq 1) \land (nl \geq 0) \land (nr == 0 \lor nl == 0)$
- Vivacité : toute requête d'ouverture finit par être satisfaite.