Synchronisation de haut niveau

Maîtrise d'informatique

Philippe MARQUET

Philippe.Marquet@lifl.fr

Laboratoire d'informatique fondamentale de Lille

Université des sciences et technologies de Lille

- Création janvier 1998
-
- La dernière version de ce cours est accessible à partir de `http://www.lifl.fr/~marquet/ens/pp/`
- `$Id: synchro.tex,v 1.2 2002/02/25 07:15:35 marquet Exp $`

Références et remerciements

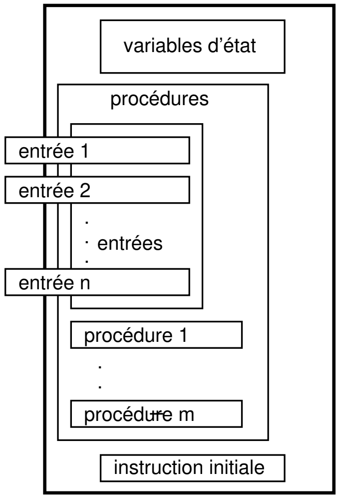
- *Principes des Systèmes d'Exploitation des Ordinateurs*
  Sacha KRAKOWIAK
  Dunod Informatique, 1987
- *Operating Systems: Design and Implementation*
  Andrew S. TANENBAUM
  Prentice-Hall, 1987 (traduction française chez InterÉditions, 1994)
- *Systèmes d'Exploitation des Ordinateurs*
  Éric DELATTRE
  Cours de Maîtrise et DEA informatiques, USTL, 1987–88
- *Programmation Parallèle*
  Jean-François MÉHAUT
  Cours de Maîtrise d'informatique, USTL, 1996

Synchronisation avec sémaphores

- Les sémaphores = outil général pour tout type de problème de synchronisation
  (Voir cours de système)
  - sémaphore = variable globale
  - pas de relation entre le sémaphore et les données à protéger
  - sémaphores accessibles n'importe où dans le programme
  - pas de contrôle possible du bon usage des sémaphores
- ⇒ Souvent difficile à utiliser ; erreurs difficiles à détecter
- Solution : support langage pour la synchronisation

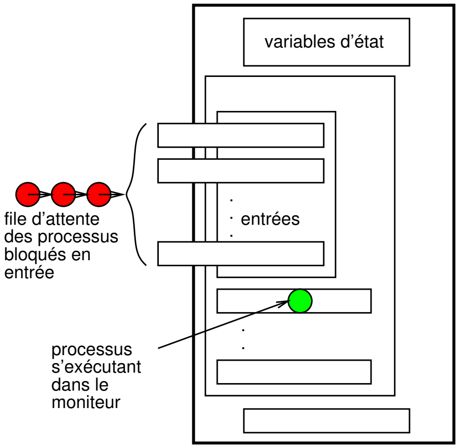
Moniteurs de Hoare

- *Moniteur de Hoare*
  - structure de programmation
  - un module de programme
  - contrôle les accès à des données partagées
- Variables d'état
  - partagées entre les procédures du module
  - encapsulées dans le module
- Procédures
  - locales
  - *entrées* : seules visibles depuis l'extérieur
- *Synchronisation* entre les processus parallèles qui appellent les entrées



Mécanisme d'exécution du moniteur

- Exclusion mutuelle entre les entrées du moniteur
  - ⇒ à un instant donné : un seul processus dans le moniteur
  - ⇒ protection des variables d'état
- Si un second processus cherche à entrer dans le moniteur
  - ⇒ bloqué jusqu'à libération du moniteur par le processus précédent
  - ⇒ *file d'attente* des processus bloqués en entrée du moniteur



Synchronisation dans le moniteur

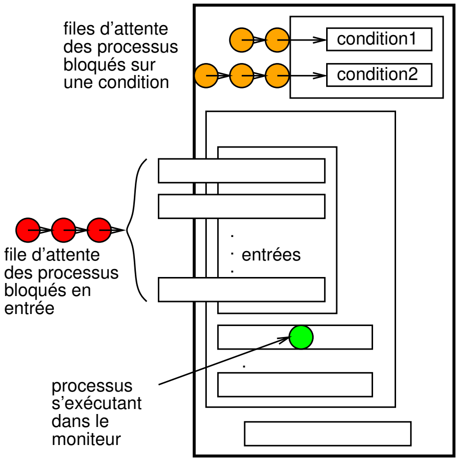
- Le processus s'exécutant dans le moniteur
  - peut s'apercevoir qu'il ne peut pas continuer et vouloir s'endormir
  - peut aussi s'apercevoir qu'un processus en attente doit reprendre son activité
- Une variable de type *condition*
  - variable d'état du moniteur (⇒ accessible uniquement de l'intérieur du moniteur)
  - réalise une *synchronisation*
  - manipulée par deux opérations :
    - `attendre` "wait"
    - `signaler` "signal"

Synchronisation dans le moniteur (cont'd)

- Pour une condition `c` :

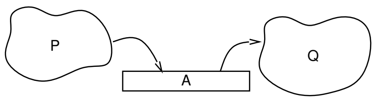
  `c.attendre` bloque le processus qui exécute `attendre` et le place en queue de file d'attente

  `c.signaler` retire un processus en tête de la file d'attente (si $\neq \emptyset$) et l'active
- Les conditions sont implantées comme des files d'attente de processus bloqués



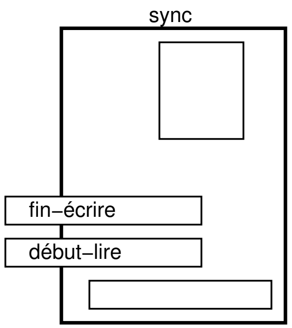
Exemple : séquencement de deux processus

- Un processus `P` transmet de l'information à un processus `Q` via un segment de donnée partagé `A`
- On suppose que la transmission n'a lieu qu'une seule fois



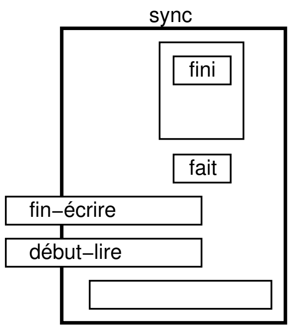
- Un moniteur `sync` à deux entrées : `fin-ecrire` et `debut-lire`
- Utilisation d'un moniteur par les deux processus :

```
# Processus P          # Processus Q
ecrire (A)             <debut de Q>
sync.fin-ecrire        sync.debut-lire
<suite de P>           lire (a)
```



Exemple : séquencement de deux processus *(cont'd)*

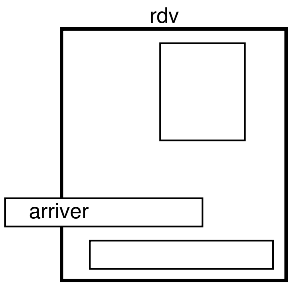
```
sync : moniteur ;
  var fait : booleen ;
      fini : condition ;

  entree fin-ecrire ;
  debut
    fait := vrai ;
    fini. signaler
  fin

  entree debut-lire ;
  debut
    si non fait alors
      fini. attendre
    fsi
  fin
```




```
  debut -- initialisation
    fait := faux
  fin
fin sync
```


Exemple : rendez-vous

- Soit N processus $\mathtt{P}_1 \ldots \mathtt{P}_N$
- Un *point de rendez-vous* dans le programme de chaque processus
  - point que le processus ne peut franchir avant que *tous* les processus aient atteint leur propre point de rendez-vous

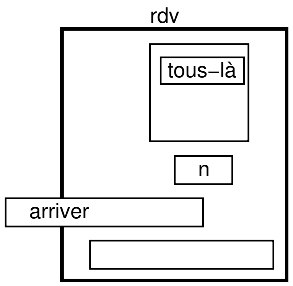


- Un moniteur `rdv` à une entrée : `arriver`
- Utilisation d'un moniteur par le processus $\mathtt{P}_i$

```
<debut de Pi>
rdv.arriver
<suite de Pi>
```


Exemple : rendez-vous (cont'd)

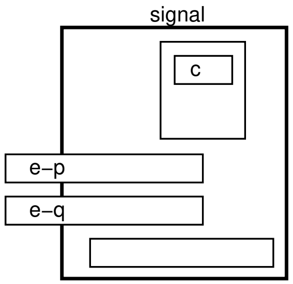
```
rdv : moniteur ;
  var n        : entier ;
      tous-la : condition ;

  entree arriver ;
  debut
    n := n+1 ;
    si n < N alors
      tous-la. attendre
    fsi ;
    tous-la. signaler
  fin

  debut -- initialisation
    n := 0
  fin
fin rdv
```



- Le dernier arrivé réveille un des processus qui attendent
- Celui-ci va réveiller le suivant
- Etc.

Problème du signal

- Soit un moniteur à deux entrées
- Deux processus `P` et `Q` exécutent respectivement les entrées `e-p` et `e-q`




```
signal : moniteur ;
  var c : condition ;

  entree e-p ;          entree e-q
  debut                 debut
    <debut de e-p>        <debut de e-q>
    c. attendre ;         c.signaler
    <reste de e-p>        <reste de e-q>
  fin                   fin
```

- Quand `Q` exécute `signaler`, les deux processus `P` et `Q` sont potentiellement actifs :
  - `P` exécute `<reste de e-p>`
  - `Q` exécute `<reste de e-q>`
- *Mais l'exclusion mutuelle doit être respectée*

Problème du signal (cont'd)

- Solution des moniteurs de Hoare :
  - file d'attente des signaleurs bloqués
    `c.signaler` retire un processus en tête de la file d'attente (si $\neq \emptyset$) et l'active
    et bloque le processus qui exécute le `signaler`
- Alternative : obliger le `signaler` à être la dernière instruction d'une entrée
- Alternative : variante de Kessels
  - Supprimer `signaler`
  - Blocage par `attendre (<condition>)`
  - `<condition>` est une expression booléenne
  - Réveil quand la condition prend la valeur vraie