

Exclusion mutuelle

Exemple 1: variable partagée

```

var compteur : integer := 0;

process p1
begin
  ...
  compteur := compteur + 1;
  ...
end p1;

process p2
begin
  ...
  compteur := compteur - 1;
  ...
end p2;

```

Arrows indicate that the two boxed statements are accessing a shared resource.

Code généré par le compilateur

```

mov    compteur,r1      mov    compteur,r1
add    r1,1             sub    r1,1
mov    r1,compteur      mov    r1,compteur

```

Résultat: **compteur** peut contenir **arbitrairement -1, 0 ou 1.**

Présentation du problème

programme concurrent
= (plusieurs) processus +
données/informations partagées;

ressource critique

= donnée/information partagée
qui ne doit pas être accédée par plusieurs
processus en même temps;

une ressource critique doit être accédée en *exclusion mutuelle*;

section critique

= partie du code d'un processus qui accède une
ressource critique.

Exemple 2: partage d'une console

```

module console;
defines écrire;
  procedure écrire (s: string);
  ...
end console;

process p1;
begin
  ...
  console.écrire(
    'erreur sur disque');
  ...
end p1;

process p2;
begin
  ...
  console.écrire(
    'imprimante pas prête');
  ...
end p2;

```

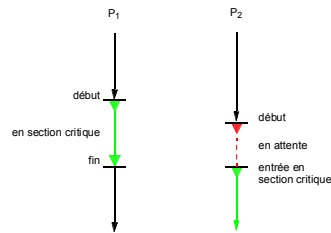
Affichage possible:

```
impreur rimansur tele pdisquas eprête
```

Démarche

On va essayer de résoudre le problème de l'exclusion mutuelle grâce à un module *section_critique*:

```
module section_critique;
  défines début, fin;
  ....
end section_critique;
```



```
process p1;                                process p2;
begin                                       begin
  ...                                     ...
  section_critique.début;                 section_critique.début;
  console.écrire('erreur sur disque');    console.écrire('imprimante pas prête');
  section_critique.fin;                   section_critique.fin;
  ...                                     ...
end p1;                                    end p2;
```

```
var compteur: integer := 0;
```

```
process p1;                                process p2;
begin                                       begin
  ...                                     ...
  section_critique.début;                 section_critique.début;
  compteur := compteur + 1;               compteur := compteur - 1;
  section_critique.fin;                   section_critique.fin;
  ...                                     ...
end p1;                                    end p2;
```

Section critique: spécification

Sûreté (safety):

- à tout instant, un seul processus peut s'exécuter entre *section_critique.début* et *section_critique.fin* ;

Vivacité (liveness):

- un processus doit pouvoir entrer en section critique si aucun processus ne l'occupe;
- un processus qui désire entrer en section critique doit pouvoir le faire au bout d'un temps fini.

Solutions

- attente active;
- masquage des interruptions;
- utilisation de verrous;
- utilisation de sémaphores.

Algorithme de Peterson (1981)

type **noprss** = 1..2;

```

module SC;
defines début, fin;
  var demande: array noprss of boolean;  // initialement false
      tour: noprss;

procedure début (no: noprss);
begin
  demande[no] := true; 1
  tour := 3 - no; 2
  repeat
    until (not demande[3-no] or (tour = no)); 3 4
  end début;

procedure fin(no: noprss);
begin
  demande[no] := false;
end fin;

```

Exclusion mutuelle par attente active

Tentative de solution:

```

module SC;
defines début, fin;
  var occupé: boolean;  // initialement false

```

```

procedure début;
begin
  while occupé do end while;
  occupé := true;
end début;

```

```

procedure fin;
begin
  occupé := false;
end fin;

```

```

begin
  occupé := false;
end SC;

```

Représentation de l'état du programme

(pos p_1 , pos p_2 , dem[1], dem[2], tour)

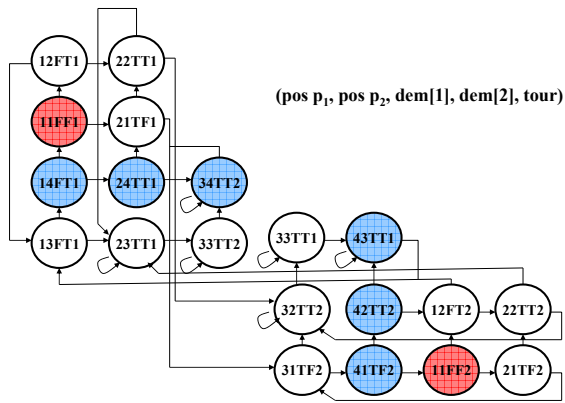
- p_1 en section critique: (4,-,-,-);
- p_2 en section critique: (-,4,-,-);

Sûreté

- pas d'état (4,4,-,-);

Vivacité

- depuis l'état (3,-,-,-) on finit par arriver dans (4,-,-,-);
- depuis l'état (-,3,-,-) on finit par arriver dans (-,4,-,-).



Exclusion mutuelle par masquage des interruptions

Pas d'interruption du processeur

=> pas de commutation de processus;

=> pas d'accès concurrent à la ressource critique.

Exemple:

process p1;
begin

...

masquer les interruptions;

console.écrire(

'erreur sur disque');

autoriser les interruptions;

...

end p1;

process p2;
begin

...

masquer les interruptions;

console.écrire(

'imprimante pas prête');

autoriser les interruptions;

...

end p2;

```

process p1;
begin
  ...
  SC.début(1);
  console.écrire(
    'erreur sur disque');
  SC.fin(1);
  ...
end p1;

process p2;
begin
  ...
  SC.début(2);
  console.écrire(
    'imprimante pas prête');
  SC.fin(2);
  ...
end p2;
```

Inconvénients:

- empêche les e/s en mode interruption dans la section critique;
- risque de perte d'interruptions si la SC est longue.

Solution à retenir dans le cas suivant:

- la SC est courte;
- pas d'e/s en mode interruption dans la SC.

Exclusion mutuelle à l'aide de verrous

Idée: introduire un nouveau mécanisme:

un verrou *v* peut être vu comme un objet sur lequel sont définies les deux méthodes *verrouiller* et *déverrouiller*.

Attributs de l'objet: une variable *état* et une liste *en_attente* de processus:

- *état* peut être ouvert ou fermé;
- *en_attente* mémorise les processus en attente devant le verrou fermé;
- *verrouiller* bloque le processus appelant en queue de liste si *état=fermé*;
- *déverrouiller* débloque le processus en tête de la liste.

```
class verrou;
  état: (ouvert, fermé);   en_attente: queue de processus;

  procedure verrouiller( );
  begin
    if état = ouvert then état := fermé;
    else bloquer le processus en queue de en_attente;
    end if;
  end verrouiller;

  procedure déverrouiller();
  begin
    if en_attente non vide then débloquent le processus en tête de en_attente;
    else état := ouvert;
    end if;
  end déverrouiller;

end verrou;
```

v: verrou;

process p1;	process p2;
begin	begin
...	...
v.verrouiller;	v.verrouiller;
console.écrire(console.écrire(
'erreur sur disque');	'imprimante pas prête');
v.déverrouiller;	v.déverrouiller;
...	...
end p1;	end p2;

- les variables *état* et *en_attente* sont des ressources critiques ;
- *verrouiller* et *déverrouiller* constituent des sections critiques;
- ces sections critiques sont courtes et ne font pas d'e/s: elles peuvent être implémentées par masquage des interruptions;
- les verrous sont mis en œuvre par le noyau du système (tout comme les processus entre lesquels le noyau partage le CPU);
- le noyau est exécuté avec les interruptions masquées.

Multiprocesseur à mémoire partagée

le masquage des interruptions ne résout pas le problème d'exclusion mutuelle;

solution utilisée: instruction permettant de *manière indivisible de lire et d'écrire* un même emplacement en mémoire;

exemples de telles instructions:

- `test&set`,
- `readModifyWrite`,
- `compare&swap`.

Init: $x \leftarrow 0$;

repeat
until `test&set(x) = 0` ;

SECTION CRITIQUE

$x \leftarrow 0$;

test&set

type T = 0..1;

function `test&set` (a: adresse): T;

begin

 return (*memoire* [a]) ;

memoire [a] := 1 ;

end `test&set` ;