

Chapitre 4

Parallélisme et interaction des processus

Sommaire

① Terminologies

② Les sémaphores

③ Les moniteurs

Définition (1)

- ▶ **Système multi-tâches:** plusieurs processus concurrents s'exécutent sur une machine mono-processeur
- ▶ **Système parallèle:** plusieurs processus concurrents s'exécutent sur plusieurs processeurs avec une **mémoire commune**
- ▶ **Système réparti:** plusieurs processus concurrentes s'exécutent sur des machines distinctes reliées en réseau et ne partageant pas de mémoire commune

▶

Définition (1)

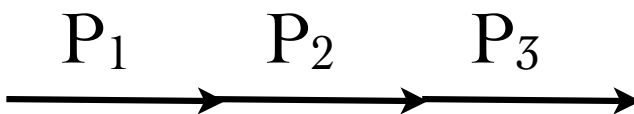
- ▶ **Simultanéité totale ou parallélisme réel:** le nombre de processeurs est au moins égal au nombre de processus
- ▶ **pseudo-simultanéité ou pseudo-parallélisme:** l'exécution imbriquée de plusieurs processus sur un seul processeur

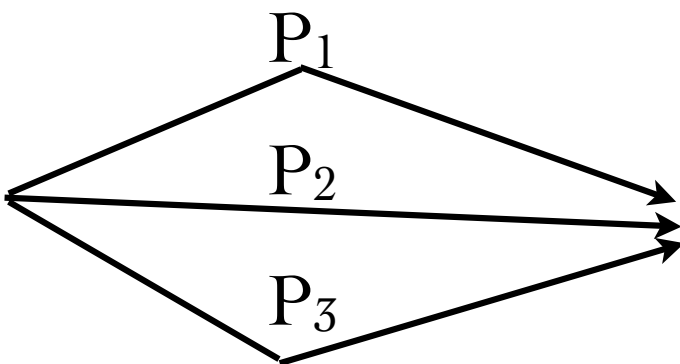
Définition (2)

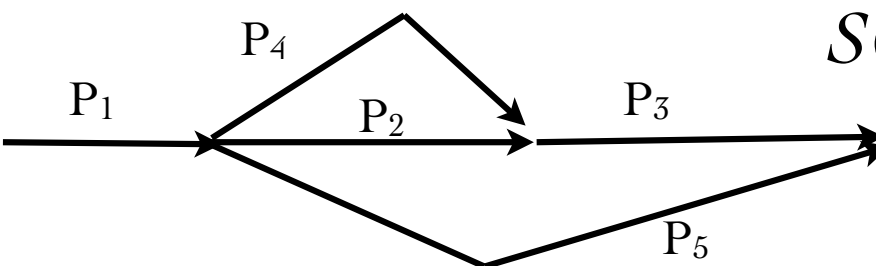
- ▶ **Processus indépendants:** l'évolution de l'un n'influe pas et ne dépend pas de l'autre
- ▶ **Processus en coopération:** chacun mène un travail distinct de l'autre sans compétition sur l'accès à certaine ressources mais avec un objectif commun demandant la mise en place de synchronisation et de communication
- ▶ **Processus en compétition:** lorsqu'ils sont en coopération mais lorsque, de plus, ils sont en compétition sur l'accès à certaines ressources

Les coroutines

- **Coroutines:** processus qui s'exécutent de manière concurrentes et pouvant manipuler des données communes

Série  $S(P_1, P_2, P_3)$

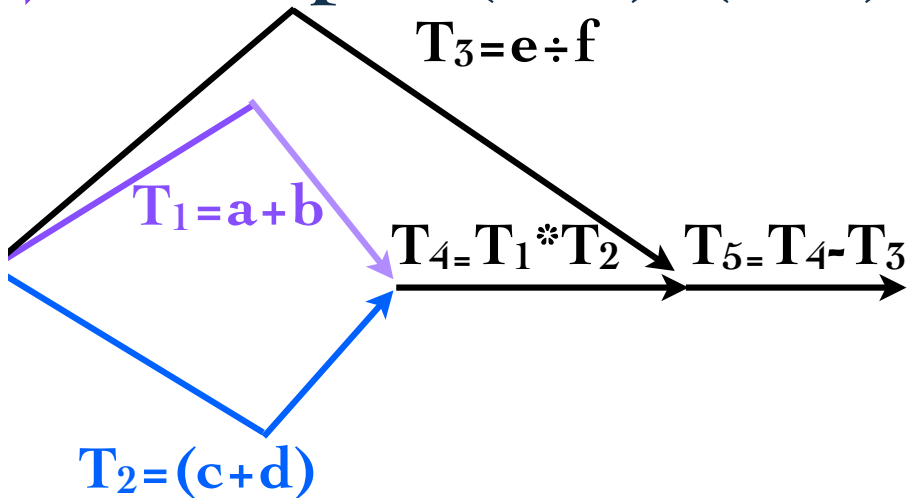
Parallèle  $P(P_1, P_2, P_3)$

Série/parallèle  $S(P_1, P(P_4, S(P_2, P_3), P_5))$

Tâches

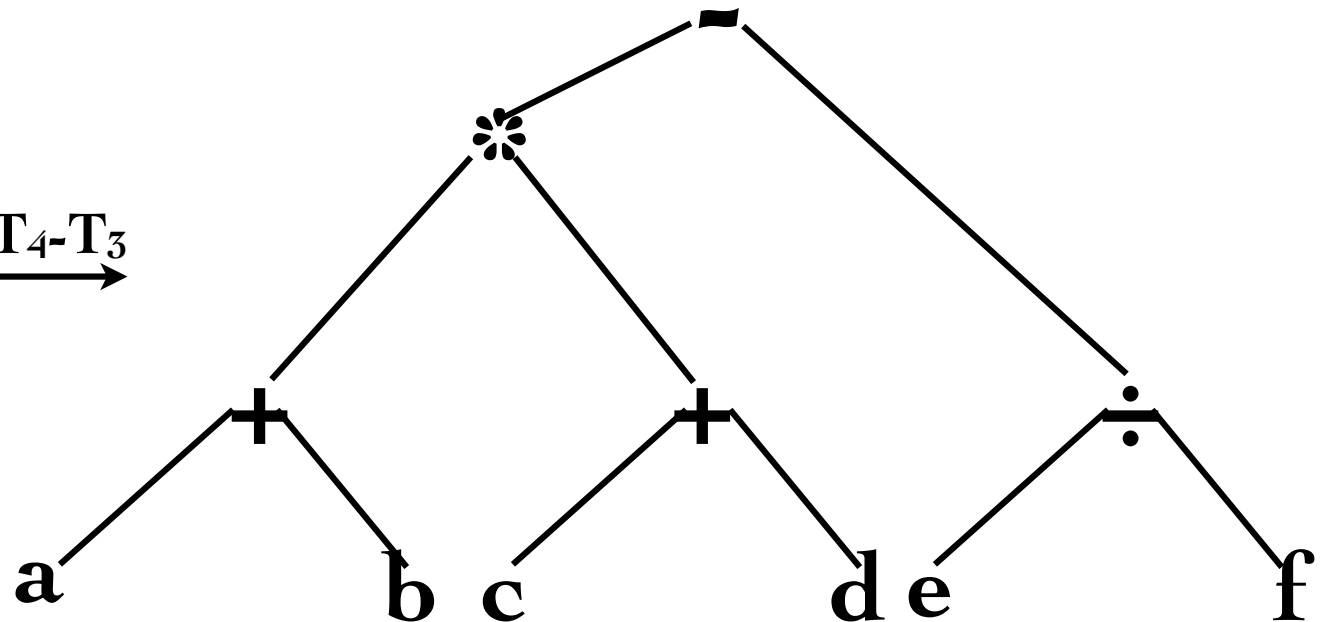
- Une tâche est une unité élémentaire de traitement ayant une cohérence logique
- Un processus P peut être constitué de tâches $T_1T_2T_3T_4T_5$ ou uniquement de $T_1T_3T_5$ par exemple

► Exemple: $(a+b)*(c+d)-(e \div f)$



Schéma

Arbre d'exécution



Décomposition d'un processus en tâches (1)

- ▶ Un processus P peut être constitué de plusieurs tâches T_1, T_2, \dots, T_N
- ▶ **Relation de précédence (notée $<$):** $T_i < T_j$ si la date de fin (f_i) de la tâche T_i est inférieure celle du début (d_j) de la tâche T_j .
- ▶ Un système de tâches peut se représenter sous forme de graphe orienté.



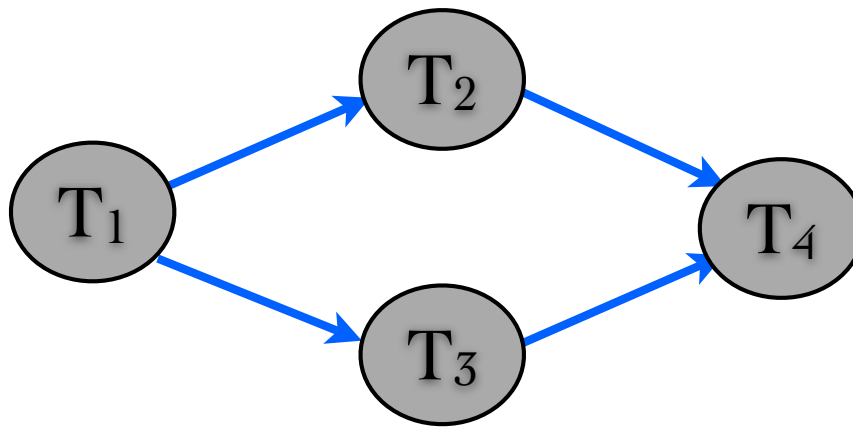
Graphe de précédence: $T_1 < T_2 < T_3$

- T_3 est successeur immédiate de T_2
- T_3 est successeur de T_1

- ▶ Un nœud est une tâche
- ▶ Une flèche entre tâches indique un ordre d'exécution

Décomposition d'un processus en tâches (2)

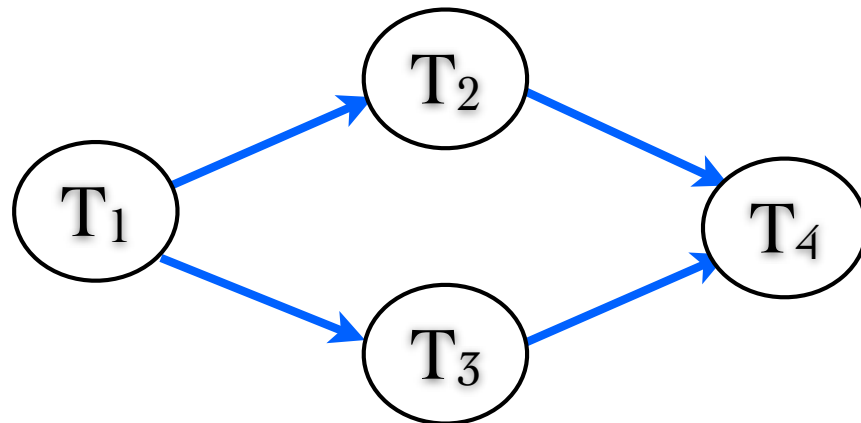
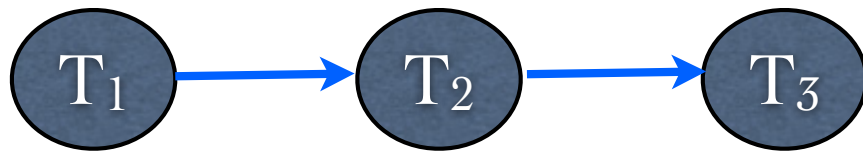
- Si ni $T_i < T_j$, ni $T_j < T_i$, alors T_i et T_j sont exécutables en parallèle.



- T_2 peut être successeur immédiate de T_1
- T_3 peut être successeur immédiate de T_1
- T_2 et T_3 sont exécutables en parallèle

Décomposition d'un processus en tâches

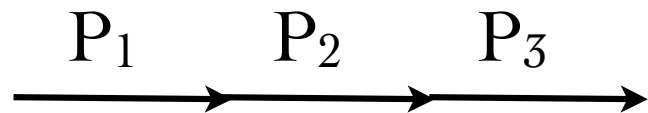
- ▶ Le langage d'un système de tâches $L(S)$: exprime toutes les scénarios possibles
- ▶ Alphabet $A = \{d_1f_1d_2f_2\dots d_nf_n\}$
 - ▶ d_i est le début de la tâche T_i et f_i la fin de la tâche T_i .
 - ▶ Si $T_i < T_j$, alors f_i vient avant d_j .



- $L(S) = \{d_1f_1d_2f_2d_3f_3\}$
- $L(S) = \{$
- $d_1f_1d_2f_2d_3f_3d_4f_4$
- $d_1f_1d_2d_3f_2f_3d_4f_4$
- $d_1f_1d_2d_3f_3f_2d_4f_4$
- $d_1f_1d_3f_3d_2f_2d_4f_4$
- $d_1f_1d_3d_2f_3f_2d_4f_4$
- $d_1f_1d_3d_2f_2f_3d_4f_4$
- $\}$

Spécification dans un langage évolué

- Représentation de la mise en séquence ou en parallèle
 - En série: Begin/End (ou SeqBegin/SeqEnd...)
 - En parallèle: CoBegin/CoEnd (ou ParBegin/ParEnd...)

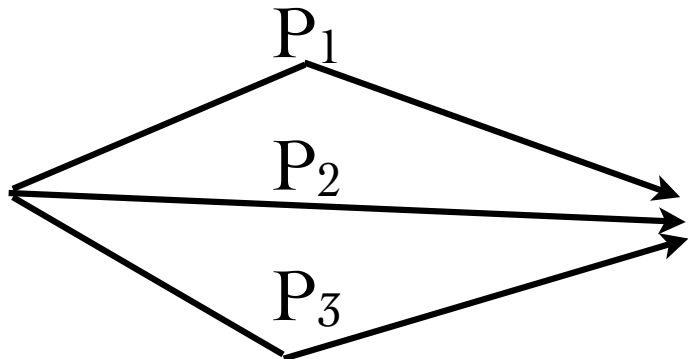


$S(P_1, P_2, P_3)$

Begin

P_1
 P_2
 P_3

End



$P(P_1, P_2, P_3)$

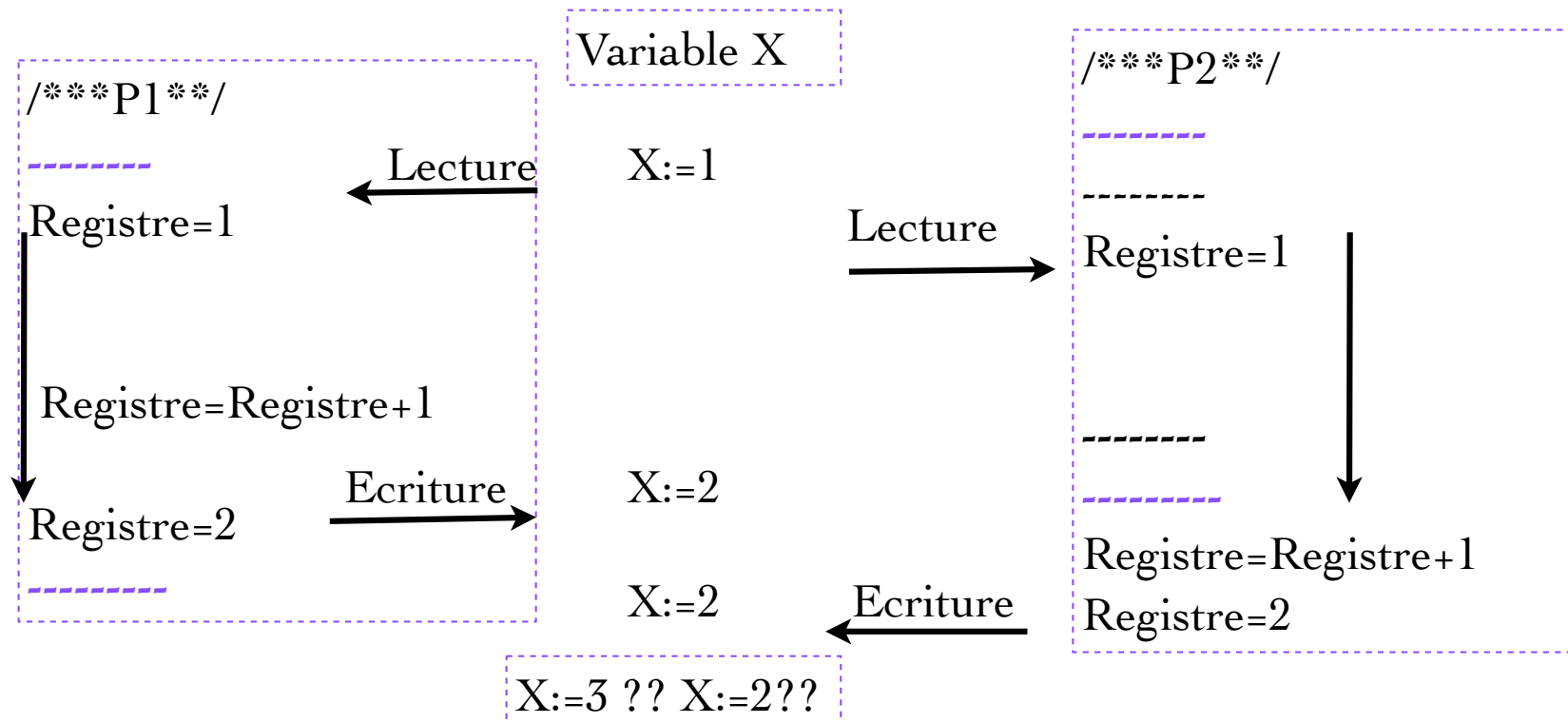
Cobegin

P_1
 P_2
 P_3

Coend

Problème de la corruption des données

- Soient deux processus P1 et P2
- Une variable X initialisée à 1: $X:=1$
- P1 et P2 exécutent le code $X:=X+1$



Problème de parallélisme des tâches (1)

► Exemple: $(a+b)*(c+d)-(e\div f)$

- Système de tâches S0
- 1 processus P_1

P_1

$T_1: t1=a+b$
 $T_2: t2=c+d$
 $T_3: t3=e/f$
 $T_4: t4=t1*t2$
 $T_5: t5=t4-t3$

- Système de tâches S1
- 2 processus P_1 et P_2

P_1

$T_1: t1=a+b$
 $T_2: t2=c+d$
 $T_3: t3=e/f$
 $T_5: t5=t4-t3$

P_2

$T_4: t4=t1*t2$

- T_4 ne doit pas s'exécuter avant T_2

Problème de parallélisme des tâches (2)

Exemple: Programme

T₁: a=2

T₂: b=3

T₃: a=a+b

T₄: c=a+b

T₅: Afficher c

► Système de tâches S0

► 1 processus P₁

P₁

T₁

T₂

T₃

T₄

T₅

► Système de tâches S1

► 2 processus P₁ et P₂

P₁

P₂

T₁

T₃

T₅

T₂:

T₄

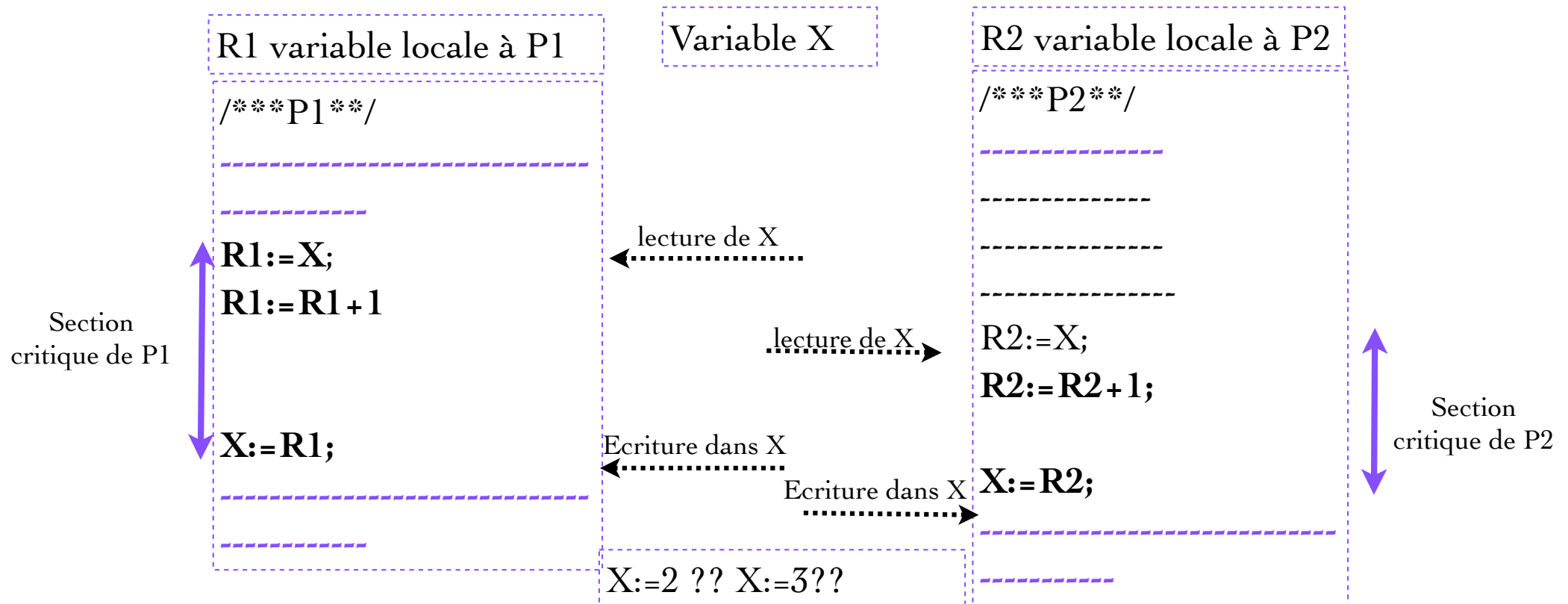
► S1 est indéterminé

► L'affichage de S0 et de S1 n'est pas forcément identique.

Section critique

- **Section critique:** partie du programme qui manipule les données communes

X variable globale initialisée à 1: $X:=1$



Exclusion mutuelle

- ▶ Chaque coroutine contient des sections critiques qui exigent une exclusion mutuelle
- ▶ Pour réaliser exclusion mutuelle:
 - ▶ Une seule coroutine dans la SC
 - ▶ Impossibilité de rester indéfiniment dans la SC
 - ▶ Pas d'attente indéfiniment sans entrer dans la SC
 - ▶ Eviter l'exécution simultanée de la SC et l'interblocage

Atomicité

- ▶ Dans le problème posé, le point critique se trouve entre la lecture d'une variable partagée et son écriture.
- ▶ Si un changement de contexte intervient entre ces deux actions, et qu'un autre processus écrive dans la variable: l'état de la mémoire du premier processus est incohérent
- ▶ Une instruction atomique, est une commande fournie par le SE
- ▶ Elle ne peut pas être interrompue par un changement de contexte

Définition

- ▶ Les sémaphores par Dijkstra en 1965
- ▶ Le sémaphore est une variable (généralement un entier non négatif) utilisée en commun par plusieurs processus
- ▶ Organise l'accès à des données communes
- ▶ Il peut être initialisé à une valeur
- ▶ 2 opérations **P** et **V** garanties atomiques par le SE sur les sémaphores
 - ▶ Proberen=tester
 - ▶ Verhogen=augmenter
- ▶ **P** et **V** s'excluent mutuellement et sont réalisées via *TestAndSet* (instruction matérielle ininterrompible) permettant de lire ou d'écrire le contenu d'un mot de la mémoire.
- ▶ Si **P** et **V** sont appelées en même temps, elles s'exécutent aléatoirement l'une après l'autre

Les Sémaphores binaires

- ▶ Sémaphores binaires
 - ▶ 2 valeurs: 0 et 1
 - ▶ 1 -ouverture
 - ▶ 0 -fermeture
- ▶ Soit S un sémaphore binaire ($S=0$ ou $S=1$)
- ▶ Opérations sur S
 - ▶ $V(S) \Rightarrow S := S+1$
 - ▶ $P(S) \Rightarrow S := S-1$ (si $S \neq 0$)
- ▶ $P(S)$ empêche l'entrée dans la SC
- ▶ $V(S)$ autorise l'entrée dans la SC

Exemple de représentation des programmes

En programmation structurée, on peut encadrer par les mots-clés **Cobegin** et **Coend** les sections de tâches pouvant s'exécuter en parallèle ou simultanément

```
Semaphore Exmut; //variable globale de type sémaphore
```

```
Exmut := 1; //initialisation du sémaphore
```

```
Cobegin //début d'exécution des coroutines
```

```
  Pi begin //la coroutine Pi démarre
```

```
    -----
```

```
    -----
```

```
      P(Exmut)
```

```
      Section critique
```

```
      V(Exmut)
```

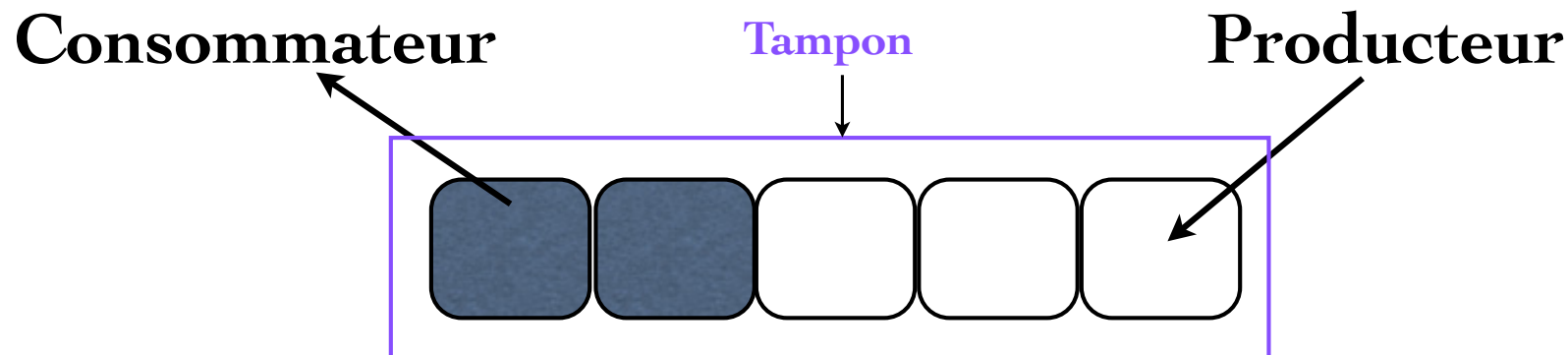
```
    -----
```

```
  end //fin d'exécution de la coroutine Pi
```

```
Coend //fin d'exécution des coroutines
```

Producteur/Consommateur(1)

- ▶ Soit une zone tampons (zone mémoire) contenant n tampons de même taille.
- ▶ Un processus produit un enregistrement et le place dans un tampon libre
- ▶ Un autre processus retire l'enregistrement du tampon et l'imprime.
- ▶ Les opérations d'écriture et de lecture se font de manière asynchrone.



Producteur/Consommateur(2)

- ▶ Solution: utiliser 3 sémaphores
- ▶ 1 sémaphore pour indiquer l'exclusion mutuelle
- ▶ **bufsem**: sémaphore binaire
- ▶ 2 sémaphores pour la comptabilisation des ressources :
 - ▶ **bufvide** : voir combien de tampons sont vides
 - ▶ Initialisé à N
 - ▶ **bufplein** : voir combien de tampons sont pleins
 - ▶ Initialisé à 0

Producteur/Consommateur(3)

```
Semaphore: bufsem, bufvide, bufplein; //déclaration des sémaphores  
bufsem:=1, bufvide:=N, bufplein:=0; //initialisation des sémaphores
```

Cobegin //début d'exécution des coroutines

```
/******Producteur*****/
```

begin

```
Adr1: Produire_enregistrementsSuivant();
```

```
-----
```

```
P(bufvide);
```

```
P(bufsem);
```

```
EcrireDansBuffer();
```

```
V(bufsem);
```

```
V(bufplein);
```

```
-----
```

```
Aller à Adr1
```

```
end //fin d'exécution du Producteur
```

```
/******Consommateur*****/
```

begin

```
-----
```

```
Adr2: P(bufplein);
```

```
P(bufsem);
```

```
RetirerDuBuffer();
```

```
V(bufsem);
```

```
V(bufvide);
```

```
-----
```

```
Consommer_enregistrements();/*imprimer*/
```

```
Aller à Adr2
```

```
end //fin d'exécution du Consommateur
```

Coend //fin d'exécution des coroutines

Synchronisation avec les sémaphores

Semaphore: S; //déclaration du sémaphore
S:=0, //initialisation

Cobegin //début d'exécution des coroutines

P1
begin

Attend de P2
Signal d'un événement A;
P(S)

end

P2
begin

Envoyer le signal de l'événement A à P1;
V(S)

end

Coend //fin d'exécution des coroutines

Exemple: coroutines/sémaphores

- Soient 6 coroutines chargées en mémoire et les processus initiés correspondants ($P_1, P_2, P_3, P_4, P_5, P_6$). P_1 commence à se développer immédiatement, les autres étant en attente. P_2, P_3 et P_4 se développent lorsqu'ils sont libérés par P_1 . P_5 est libéré par P_2 ou P_3 alors que P_6 l'est par P_5 et P_4 . Chaque processus signale à ses successeurs qu'ils peuvent démarrer

Semaphore: $S_1, S_2, S_3, S_4, S_5, S_6$; //déclaration des sémaphores

$S_1:=0, S_2:=0, S_3:=0, S_4:=0, S_5:=0, S_6:=0$; //initialisation des sémaphores

Cobegin //début d'exécution des coroutines

P1
begin

V(S_1)
V(S_2)
V(S_3)

end

P2
begin

P(S_1)

V(S_4)

end

P3
begin

P(S_2)

V(S_4)

end

P4
begin

P(S_3)

V(S_5)

end

P5
begin

P(S_4)

V(S_6)

end

P6
begin

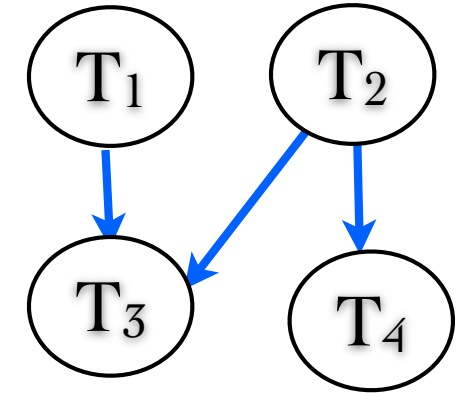
P(S_5)
P(S_6)

end

Coend //fin d'exécution des coroutines

Exemple: tâches/sémaphores (1)

- Soit le système à tâches (T_1, T_2, T_3, T_4) du graphe de précedence ci-contre
- Chaque tâche signale à ses successeurs qu'ils peuvent démarrer
- T_3 est en attente de la complétion de T_1 , et T_2 , tandis que T_4 est en attente de la complétion de T_2 .



- Avec autant de sémaphores qu'il y a de transitions de précedence

Semaphore: S_1, S_2, S_3 ; //déclaration
 $S_1:=0, S_2:=0, S_3:=0$; //initialisation

Cobegin //début d'exécution des tâches

begin
 T_1
 $V(S_1)$
end

begin
 T_2
 $V(S_2)$
 $V(S_3)$
end

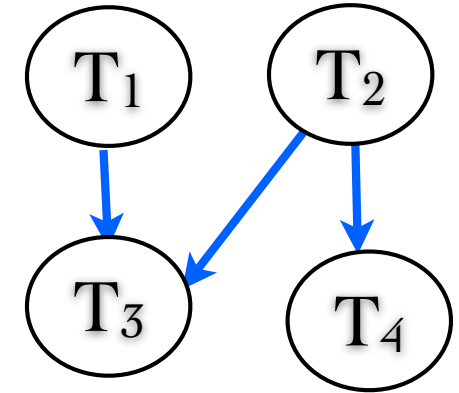
begin
 $P(S_1)$
 $P(S_2)$
 T_3
end

begin
 $P(S_3)$
 T_4
end

Coend //fin d'exécution des tâches

Exemple: tâches/sémaphores (2)

- ▶ Soient le système à tâches (T_1, T_2, T_3, T_4) du graphe de précédence ci-contre
- ▶ Chaque tâche signale à ses successeurs qu'ils peuvent démarrer
- ▶ T_3 est en attente de la complétion de T_1 , et T_2 , tandis que T_4 est en attente de la complétion de T_2 .



- ▶ Possibilité avec un seul sémaphore
 - ▶ Diviser pour régner
 - ▶ Supprimer une relation de précédence ($T_2 \rightarrow T_3$),
 - ▶ Programmer le graphe obtenu sans sémaphore
 - ▶ Ajouter le sémaphore pour la synchronisation de T_2 et T_3

Semaphore: $S:=0$; //déclaration/initialisation

Cobegin //début d'exécution des tâches

begin
 T_2
 $V(S)$
 T_4
end

begin
 T_1
 $P(S)$
 T_3
end

Coend //fin d'exécution des tâches

Exemple: tâches/sémaphores (3)

► Exemple: $(a+b)*(c+d)-(e \div f)$

- Soient 3 processus P_1, P_2, P_3 chargés de calculer $(a+b) * (c+d) - (e/f)$
- P_1 calcule $t1=a+b$, $t4=t1*t2$ et le résultat $t5=t4-t3$, P_2 calcule $t2=c+d$, P_3 calcule $t3=e/f$.

Semaphore: $S_1:=0, S_2:=0$; //déclaration/initialisation

P_1

$t1=a+b$

$P(S1)$

$t4=t1*t2$

$P(S2)$

$t5=t4-t3$

P_2

$t2=c+d$

$V(S1)$

P_3

$t3=e/f$

$V(S2)$

- NB: comme P_1 fait $t1=a+b$ et $t4=t1*t2$, $t5=t4-t3$, il ne pourra jamais utiliser $t4$ en même temps, donc on n'a pas besoin d'une exclusion mutuelle sur $t4$ au sein de $P1$

Les moniteurs

► Un moniteur est un programme constitué de variables et de procédures

Exemple de moniteur

Soit une ressource partagée

Un processus qui veut accéder à la ressource exécute la procédure DemanderRessource() puis LibererRessource() quand il finit d'utiliser la ressource

Soient deux variables: **libre**, **occupe**;

condition **libre**; //libre est une variable de type condition (type prédéfinis)

booléen **occupe**; //occupe une variable booléenne

Begin

occupe:=false; /**ressource non occupée**/

Procédure DemanderRessource();

begin

if(**occupe**) then WAIT(**libre**); /**WAIT() a pour effet de bloquer le processus appelant**/

occupe:=true

autorisation d'accès (**section critique**)

end

Procédure LibererRessource();

begin

occupe:=false

SIGNAL(**libre**) /**SIGNAL suspend le processus appelant**/

end

End