USTHB

Faculté d'Informatique

Département d'Intelligence Artificielle et des Sciences de Données

SERIE D'EXERCICES N° 3

Synchronisation (sémaphores, moniteurs & régions critiques)

Le 23/09/2025 Année : 2025/2026

Exercice N° 1 : Soit un système composé de trois processus fumeurs et d'un processus agent.
Chaque fumeur roule continuellement une cigarette et la fume. Pour pouvoir rouler une cigarette, trois ingrédients sont nécessaires : le tabac, le papier et les allumettes.
Un des processus fumeurs détient le papier, le deuxième processus le tabac et le troisième les allumettes.
Le processus agent a un approvisionnement infini des trois ingrédients. L'agent place deux ingrédients distincts sur la table ainsi le fumeur qui possède le troisième ingrédient peut rouler une cigarette, la fumer puis signale à l'agent la fin de son opération. L'agent remet sur la table deux autres ingrédients et ainsi le cycle se répète.

- Synchroniser les processus à l'aide des sémaphores
- Modifier la solution pour permettre à plus d'un fumeur d'opérer en même temps.

Exercice N° 2 : Des voitures venant du nord et du sud doivent traverser un pont. Sur ce pont ne peut passer qu'une seule file de voitures à la fois et dans un même sens.
- Ecrire un algorithme qui permet aux voitures de passer du nord vers le sud ou du sud vers le nord en se synchronisant à l'aide des sémaphores au niveau du pont.
- Discuter votre solution et proposer d'éventuelles modifications pour diverses situations.

Exercice N° 3 : Soit un système constitué d'un processus producteur et un processus consommateur qui se partagent deux tampons T1 et T2 de tailles respectives M et N, comme suit :
- Le producteur, à tout moment, ne peut déposer dans T2 que si le tampon T1 est plein.
- Le consommateur, à tout moment, ne peut prélever du tampon T2 que si le tampon T1 est vide.

1- Synchroniser les deux processus en utilisant les sémaphores.
2- Généraliser cette solution au cas de plusieurs producteurs et plusieurs consommateurs.

Exercice N°4 : Soient 2 processus producteurs P_1, P_2 et deux processus consommateurs C_1, C_2 qui partagent un tampon de taille fixe. On définit les contraintes suivantes d'accès au tampon :
- P_i constitue une classe CL_i,
- C_i ne peut consommer que les messages produits par le processus de la classe CL_i,
- A tout instant, le tampon ne contient que les messages produits par le processus d'une seule classe.

A/ A l'aide des sémaphores, synchroniser l'ensemble de ces processus parallèles pour l'accès au tampon.
B/ Quelles seront les modifications nécessaires à apporter à la solution,
a- si on ajoute des processus producteurs à chaque classe CLi ?
b- et de plus, si on ajoute une classe CL_3 et un consommateur C_3 ?

Exercice N°5 : Des processus « utilisation» et des processus « systèmes » se partagent n imprimantes. Les processus systèmes ont la priorité pour l'acquisition d'une imprimante.

- Décrire le comportement de ces deux classes de processus pour leur synchronisation en utilisant les moniteurs dans deux cas de demandes :
  A/ Une imprimante à la fois.
  B/ k imprimantes à la fois.

Exercice N° 6 : On considère 2 ressources appelées R1, R2. La ressource R1 existe en N1 exemplaires et la ressource R2 en N2 exemplaires.
On supposant que chaque processus peut demander :

- Soit 1 exemplaire de la ressource R1,
- Soit 1 exemplaire de la ressource R2,

- Soit 1 exemplaire de R1 et 1 exemplaire de R2

A/ Ecrire un moniteur gérant l'accès à ces ressources en donnant la priorité à celui qui exprime le troisième type de demande.
B/ Modifier la solution pour permettre l'accès FIFO.

Exercice n°7 : Etant donné un système contenant N processus évoluant de manière parallèle et dont le fonctionnement de chacun est donné comme suit :

```
Processus P (i : entier) ;   /* i : identité du processus pouvant prendre une valeur de 1 à N */
Var j : entier ; T : Tableau [1..N] de Booleen :=Faux ;
Debut -
Pour j :=1 à N Faire
        Envoyer (B, 'présent', i)
Fait ;
Repeter
        Recevoir (B , m, k) ;
        Si T[k]= Faux Alors T[k] := Vrai  Sinon Envoyer (B, m, k) Fsi ;
        j :=1 ;  Tantque (j<=N) et (T[j]= Vrai Faire j :=j+1 Fait
Jusqu'à (j>N)
Fin.
```

où B est une boite aux lettres commune (peut être utilisée par chaque processus en émission et en réception) supposée de capacité infinie.
- Envoyer (B, m, i) : permet d'envoyer un message m accompagné d'un entier i vers la boite aux lettres B.
- Recevoir (B, m, k) : permet de recevoir un message m accompagné d'un entier k dans la boite aux lettres B. cette primitive bloque le processus appelant jusqu'à la réception du message.

A/ Discuter cette solution.
- Si on suppose que chaque processus P(i) i=1, N possède sa propre boite aux lettres B(i) (donc utilisée uniquement pour la réception de messages),

B/ Réécrire la solution précédente avec cette nouvelle considération.
C/ Comparer les deux solutions.

Exercice 8: Dans un système d'exploitation, on dispose des primitives de communication par boites aux lettres suivantes :
- Send (B, message) : Dépôt de message dans la boite aux lettres B.
- Receive (B, message) : Attente et Retrait de message de la boite aux lettres B.
- Soit le processus suivant:

```
Processus P ;
Struct m, mess : ….. ;
Debut
-
Send( B1, ''vide'') ; Send (B2,''vide'') ; mess ← ''vide'';
Répéter
        Receive (B1, m) ;
        Si (m<>''vide'')  Alors mess ← m
                          Sinon  Send (B1,''vide'') ;
                                 Receive (B2, m) ;
                                 Si (m<>''vide'') Alors mess ←m
                                                  Sinon Send (B2,''vide'')
                                 Fsi
        Fsi
Jusqu'à (mess<>''vide'')
Fin.
```

B1 et B2 étant deux boites aux lettres communes à P et à d'autres processus.

1/ Expliquer le fonctionnement de la solution, déduire sa fonction.
- Remplaçons la primitive *Receive ()* par la fonction *Read (B)* qui retourne le premier message de la boite aux lettres B s'il existe sinon vide.
2/ Réécrire le processus P.

Exercice n°9

- Ecrire une solution à l'aide des régions critiques qui permet de réaliser le RDV de N processus parallèles de telle sorte qu' à l'arrivée de tous ces processus à ce point de rendez-vous, ils reprennent leurs exécutions dans un ordre prédéfini lié à leur ordre d'arrivé (on retiendra l'ordre inverse de leur ordre d'arrivée).

USTHB
Faculty of Computer Sciences
Department of artificial intelligence and data science
List of exorcises N° 03
Synchronisation (semaphores, monitors & critical regions)

The 23/09/2025 Year : 2025/2026

Exorcise N° 1 :
Consider a system composed of three smoking processes and one agent process.
Each smoker continuously rolls a cigarette and smokes it. To roll a cigarette, three ingredients are needed: tobacco, paper and matches.
One of the smoking processes holds the paper, the second the tobacco and the third the matches.
The agent process has an infinite provision of the three ingredients. The agent places two separate ingredients on the table, hence the smoker who has the third ingredient can roll a cigarette, smoke it and then signal to the agent the end of his operation. The agent places two more ingredients on the table and the cycle repeats again.
- Synchronise the processes using semaphores
- Modify the solution to allow more than one smoker to operate at the same time.
- Generalise the solution for many smokers.

Exorcise N° 2 : Cars coming from the north and south have to cross a bridge. Only one line of cars can cross the bridge at a time and in the same direction.
- Write an algorithm that allows the cars to pass from north to south or from south to north by synchronising themselves at the bridge using the semaphores.
- Discuss your solution and suggest possible modifications for different situations.

Exorcise N° 3 :
Consider a system consisting of a producer process and a consumer process which share two buffers *T1* and *T2* of respective sizes *M* and *N*, as follows:
- The producer, at any time, can only deposit in *T2* if buffer *T1* is full.
- The consumer, at any time, can only extract from buffer *T2* if buffer *T1* is empty.
1- Synchronise the two processes using semaphores.
2- Generalise this solution to the case of several producers and several consumers.

Exorcise N°4 :
Consider two producer processes P_1 and P_2 and two consumer processes C_1, C_2 which share a buffer of fixed size. We define the following buffer access constraints:
- P_i forms a class CL_i,
- C_i can only consume messages produced by the process of class CL_i,
- At any given time, the buffer only contains messages produced by the process of a single class.

A/ Using semaphores, synchronise all these parallel processes for access to the buffer.
B/ What changes will need to be made to the solution,
a- if we add producer processes to each class CL_i?
b- and also, if we add a CL_3 class and a consumer C_3?

Exorcise N°5 :
Consider "*user*" processes and '*'system''* processes that share *n* printers.
The *system* processes have priority for acquiring a printer.
- Describe the behaviors of these two classes of processes for their synchronization using the monitors in two cases of requests:
A/ One printer at a time.
B/ *k* printers at a time.

Exorcise N° 6 :
Consider *2* resources called R_1, R_2. Resource R_1 exists in N_1 copies and resource R_2 in N_2 copies.
Assuming that each process can request :
- Either *1* copy of resource R_1,
- Either *1* copy of resource R_2,
- Either *1* copy of R_1 and 1 copy of R_2

A/ Write a monitor to manage access to these resources, giving priority to the process expressing the third type of request.
B/ Modify the solution to allow FIFO access.

Exorcise N°7 :
Given a system containing *N* processes running in parallel, the functioning of each of them is given as follows:

```
Process P (i : integer) ;   /* i : identity of the process which can take a value from 1 to N */
Var j : integer ; T : Array [1..N] of Boolean :=false ;
Begin -
For j :=1 to N Do
        send (B, 'present', i)
EndFor ;
Repeat
        receive (B , m, k) ;
        If T[k]= false Then T[k] := true  Else send (B, m, k) EndIf ;
        j :=1 ;  While (j<=N) and (T[j]= true Do j :=j+1 EWhile
Until (j>N)
End.
```

where *B* is a common mailbox (which can be used by each sending and receiving process) assumed to have infinite capacity.
- *send (B, m, i)*: sends a message *m* with an integer *i* to mailbox *B*.
- *receive (B, m, k)*: receives a message *m* with an integer *k* in mailbox *B*. This primitive blocks the calling process until the message is received.

A/ Discuss this solution.
- Assuming that each process *P(i), i=1..N* has its own mailbox *B(i)* (used only for receiving messages),

B/ Rewrite the previous solution with this new consideration.
C/ Compare the two solutions.

Exorcise N°8:
In an operating system, consider the following mailbox communication primitives :
- *send (B, message)* : Places *message* in mailbox *B.*
- *receive (B, message):* Waits for and removes *message* from mailbox *B.*

Consider the following process:

```
Process P () ;
Struct m, mess : ….. ;
Begin
    -
    send( B1, ''empty'') ; send (B2,''empty'') ; mess ← ''empty'';
    Repeat
        receive (B1, m) ;
        If (m<>''empty'')  Then mess ← m
                           Else  send (B1,''empty'') ; receive (B2, m) ;
                                 If (m<>''empty'') Then mess ←m
                                                   Else send (B2,''empty'')
                                 EndIf
```

```
            EndIf
    Until (mess<>''empty'')
End.
```

B_1 and B_2 being two mailboxes shared by *P* and other processes.
1/ Explain how the solution works and deduce its function.
- Replace the primitive *receive ()* with the function *read (B)*, which returns the first message from mailbox *B* if it exists, otherwise *empty*.
2/ Rewrite the process *P*.

Exorcise N°9

- Write a solution using critical regions that allows *N* parallel processes to make the rendezvous in such a way that when all these processes arrive at this rendezvous point, they resume their executions in a predefined order linked to their order of arrival (we will use the reverse order of their order of arrival).