USTHB

Faculté d'Informatique

Département d'Intelligence Artificielle et des Sciences de Données

SERIE D'EXERCICES N° 2

(Synchronisation par variable d'états et événements)

Le 23/09/2025 - Année 2025/2026

Exercice 1 : Le programme suivant (algorithme de Dekker) est proposé comme solution au problème de la section critique entre deux processus concurrents :

Initialement : Di := faux ; (i= 1,2) tour := 1 ou 2 ;

Processus Pi

Début

Répéter

Di := vrai :

Tant que Dj faire si tour = j Alors

Début Di := faux ; Tant que tour = j Faire <rien>Fait ; Di :=vrai Fin ;

Fait ;

<section critique i>

tour : =j ; Di : = faux ;

Jusqu'à faux ;

Fin.

- Vérifier les conditions de la section critique.

Exercice2: Le programme suivant (algorithme de Lamport) est proposé comme solution au problème de section critique entre deux processus concurrents ;

Choix : tableau [0.. 1] de booleen :=faux; Numero : tableau [0.. 1] de entier :=0 ;

Notation : (a,b) < (c,d) ⇔ (a < c) ou (a = c et b < d)

Processus Pi

Debut

Répeter

$Choix_i$= vrai ; $Numero_i$= max ($Numero_i$, $Numero_j$) +1 ; $Choix_i$= Faux ;

Tant que $choix_j$ faire rien fait ;

Tant que ($Numero_j \neq 0$) et (($Numero_j$, j) < ($Numero_i$, i)) faire <rien> fait;

<SC_i >

$Numero_i = 0$;

<Section $restante_i$ > ;

Jusqu'à faux ;

Fin ;

a/ Vérifier les conditions de la section critique

b/ Discuter le problème lié à la valeur numéro qui peut croître indéfiniment

c/ Réécrire la solution pour n processus

Exercice 3 :

On propose la solution suivante comme protocole d'exclusion mutuelle pour deux processus Pi et Pj.

Initialement Di=Dj=faux ; tour= i ou j ;

Processus Pi

Début

Répéter

Di := vrai :

Tantque tour = j **Faire**

Tantque Dj = vrai **Faire** <rien>**Fait** ; tour :=i

Fait ;

<Section Critique i>

Di : = faux ;
Jusqu'à faux
Fin.

1/ Donner la condition d'entrée en SC.
2/ Vérifier si cette solution peut être retenue comme protocole d'exclusion mutuelle.
3/ On modifie la solution comme suit :

Répéter
Di := vrai :
Tantque (tour = j) ou (Dj=vrai) **Faire**
Tantque Dj = vrai **Faire** <rien>**Fait** ; tour :=i
Fait ;
<Section Critique i>
Di : = faux ;
Jusqu'à faux

a- Donner la condition d'entrée en SC.
b- Vérifier les 4 conditions de la SC.

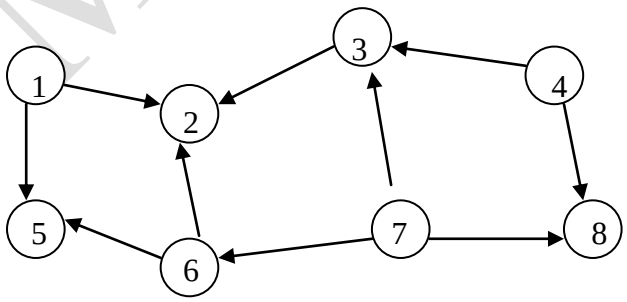
Exercice 4 :

On suppose un système constitué uniquement de deux événements mémorisés e1, e2.

- Ecrire une implémentation des *deux primitives* suivantes :
- **Attendre (e1 et e2)** qui permet d'attendre que les événements e1 et e2 se produisent.
- **Déclencher (ei)** qui permet de réveiller tous les processus en attente de cet événement si leurs conditions sont satisfaites, dans ce cas, l'événement est acquitté. Si aucun processus ne l'attend, l'événement est mémorisé.

Exercice 5: On s'intéresse aux événements mémorisés. On définit les primitives suivantes :
Wait(k,e1,…en) : permet d'attendre *k* événements sur *n* déclenchés par des processus quelconques.
Signal(e) : déclenche l'arrivée de l'événement *e*. Tous les processus en attente de cet événement sont activés si leurs conditions sont satisfaites et l'événement est acquitté. Si aucun processus ne l'attend ou les conditions des processus en attente de l'événement ne sont pas satisfaites, l'événement est mémorisé.
Utiliser cet outil pour programmer l'application suivante dans deux cas différents :
1- Soient 3 processus ayant un point de rendez-vous. Un processus arrivant au point de rendez-vous continue son exécution :
a/ si les 2 autres sont arrivés à ce point.
b/ si au moins 1 autre est arrivé.
2- Proposer une implémentation de ces primitives

Exercice n°6 : Soit le graphe suivant :



1/ Ce graphe est-il proprement lié ?
2/ Exprimer ce graphe à l'aide de Parbegin/Parend et éventuellement avec les sémaphores.

USTHB
Faculty of Computer Sciences
Department of artificial intelligence and data science
List of exorcises N° 02
(Synchronisation by state variables and events)

The 23/09/2025 - Year 2025/2026

Exorcise 1 : The following program (Dekker algorithm) is proposed as a solution to the problem of the critical section between two concurrent processes :

Initially : *Di := false (i= 1,2) ; turn := 1 or 2 ;*
Process *Pi () ;*
Begin
Repeat
Di := vrai :
While *Dj* ***Do If*** *turn = j* ***Then***
Begin *Di := false ;* ***While*** *turn = j* ***Do*** *<nothing>* ***Ewhile*** *; Di :=true* ***End*** *;*
Ewhile *;*
<Critical Section i>
turn : =j ; Di : = false ;
Until *false ;*
End*.*
- Check the conditions for the critical section.

Exorcise 2: The following program (Lamport algorithm) is proposed as a solution to the critical section problem between two concurrent processes;

choice : ***array*** *[0.. 1]* ***of*** *boolean :=false; Number :* ***array*** *[0.. 1]* ***of*** *integer :=0 ;*
Notation : $(a,b) < (c,d) \Leftrightarrow (a < c) \text{ or } (a = c \text{ and } b < d)$
Process *Pi () ;*
Begin
Repeat
$choice_i = true$; $Number_i = Max(Number_i, Number_j) + 1$; $choice_i = false$;
While $choice_j$ ***Do*** *<nothing>* ***Ewhile*** *;*
While $(Number_j \neq 0)$ ***and*** $((Number_j, j) < (Number_i, i))$ ***Do*** *<nothing>* ***Ewhile****;*
$<CS_i>$
$Number_i = 0$;
$< Remaining\ section_i >$;
Until *false ;*
End *;*
a/ Verify the conditions of the critical section.
b/ Discuss the problem linked to the value of *number* which can grow indefinitely.
c/ Rewrite the solution for *n* processes.

Exercice 3 :
The following solution is proposed as a mutual exclusion protocol for two processes Pi and Pj.
Initially Di=Dj=false ; turn= i or j ;

Process *Pi*
Begin
Repeat
Di := true :
While *turn = j* ***Do***
While *Dj = vrai* ***Do*** *<nothing>* ***Ewhile*** *; turn :=i*
Ewhile *;*

<Critical Section i>
Di : = false ;
Until *false*
End.

1/ Give the condition to enter the CS.
2/ Check whether this solution can be used as a mutual exclusion protocol.
3/ The solution is modified as follows:
-Repeat
Di := vrai :
While (turn = j) or (Dj=true) Do
While Dj = true **Do** <nothing>**Ewhile** ; turn :=i
Ewhile ;
<Critical Section i>
Di : = false ;
Until false ;
-
a- Give the condition to enter the CS.
b- Check the 4 CS conditions.

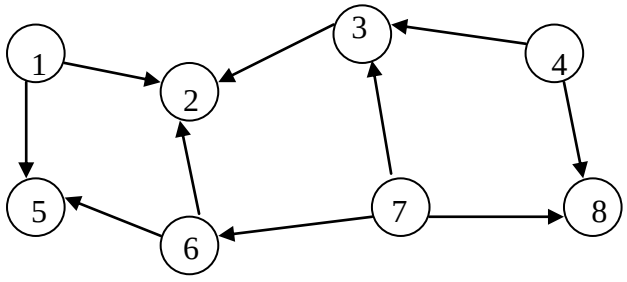
Exorcise 4 :
Assume a system consisting only of two memorised events e1, e2.

- Write an implementation of the following two primitives:
- ***Wait (e1 and e2)***, which waits for events e1 and e2 to occur.
- ***Signal (ei)***, which wakes up all the processes waiting for this event if their conditions are met, in which case the event is acknowledged. If no process is waiting for it, the event is memorised.

Exorcise 5:
We are interested in memorised events. The following primitives are defined:
Wait(k,e1,...en): used to wait for *k* events out of *n* triggered by any processes.
Signal(e): triggers the arrival of event *e*. All processes waiting for this event are activated if their conditions are satisfied and the event is acknowledged. If no process is waiting for it, or if the conditions of the processes waiting for the event are not satisfied, the event is memorised.

1- Use this tool to program the following application in two different cases:
Given 3 processes with a rendezvous point. A process arriving at the rendezvous point continues its execution:
a/ if the 2 others have arrived at this point.
b/ if at least 1 other has arrived.
2- Propose an implementation of these primitives

Exercice n°6 : Given the following graphe :



1/ Is this graph properly linked?
2/ Express this graph using *Parbegin/Parend* and possibly semaphores.