

CORRECTION

SÉRIE 5

SE

Madame Khaoula ElBedoui-Maktouf

2^{ème} année Ingénieur Informatique

Exercice 1

- On considère trois processus P1, P2 et P3 qui partagent une même variable (x initialisée à 3)

1) Que doit être la valeur initiale du sémaphore S ?

P1	P2	P3
(a) P(S) ; (b) $x = x + 1$; (c) V(S) ;	(d) P(S) ; (e) $x = x * 2$; (f) V(S) ;	(g) P(S) ; (h) $x = x - 4$; (i) V(S) ;

Exercice 1

- On considère trois processus P1, P2 et P3 qui partagent une même variable (x initialisée à 3)

1) Que doit être la valeur initiale du sémaphore S ?

P1	P2	P3
(a) $P(S)$; (b) $x = x + 1$; (c) $V(S)$;	(d) $P(S)$; (e) $x = x * 2$; (f) $V(S)$;	(g) $P(S)$; (h) $x = x - 4$; (i) $V(S)$;

- ❖ S est un sémaphore **d'exclusion mutuelle** sur la **variable partagée x** donc il doit être initialisé à **1**. C'est un sémaphore binaire.

Exercice 1

- ❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

Le scénario suivant est-il possible ?

- ❖ **a, d, b, g, c, e, f, h, i ?**

P1	P2	P3
(a) P(S) ; (b) $x = x + 1$; (c) V(S) ;	(d) P(S) ; (e) $x = x * 2$; (f) V(S) ;	(g) P(S) ; (h) $x = x - 4$; (i) V(S) ;

X = 3

Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

❖ **a, d, b, g, c, e, f, h, i ?**

❖ **a : $P(S) \rightarrow S.Val = 0$; P1 se branche sur b**

P1	P2	P3
(a) $P(S)$; (b) $x = x + 1$; (c) $V(S)$;	(d) $P(S)$; (e) $x = x * 2$; (f) $V(S)$;	(g) $P(S)$; (h) $x = x - 4$; (i) $V(S)$;

$X = 3$

Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

❖ **a, d, b, g, c, e, f, h, i ?**

❖ **a : $P(S) \rightarrow S.Val = 0$; P1 se branche sur b**

❖ **d : $P(S) \rightarrow S.Val = -1$; P2 est bloqué**

P1	P2	P3
(a) $P(S)$; (b) $x = x + 1$; (c) $V(S)$;	(d) $P(S)$; (e) $x = x * 2$; (f) $V(S)$;	(g) $P(S)$; (h) $x = x - 4$; (i) $V(S)$;

X = 3



Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

❖ **a, d, b, g, c, e, f, h, i ?**

❖ **a** : $P(S) \rightarrow S.Val = 0$; P1 se branche sur b

❖ **d** : $P(S) \rightarrow S.Val = -1$; P2 est bloqué

❖ **b** : $x = 4$

P1	P2	P3
(a) $P(S)$; (b) $x = x + 1$; (c) $V(S)$;	(d) $P(S)$; (e) $x = x * 2$; (f) $V(S)$;	(g) $P(S)$; (h) $x = x - 4$; (i) $V(S)$;

X = 4



Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

❖ **a, d, b, g, c, e, f, h, i ?**

❖ **a** : $P(S) \rightarrow S.Val = 0$; P1 se branche sur b

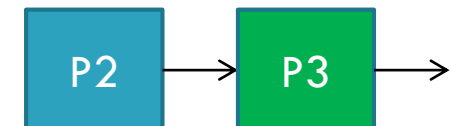
❖ **d** : $P(S) \rightarrow S.Val = -1$; P2 est bloqué

❖ **b** : $x = 4$

❖ **g** : $P(S) \rightarrow S.Val = -2$; P3 est bloqué

P1	P2	P3
(a) $P(S)$; (b) $x = x + 1$; (c) $V(S)$;	(d) $P(S)$; (e) $x = x * 2$; (f) $V(S)$;	(g) $P(S)$; (h) $x = x - 4$; (i) $V(S)$;

X = 4



Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

❖ **a, d, b, g, c, e, f, h, i ?**

❖ **a** : $P(S) \rightarrow S.Val = 0$; P1 se branche sur b

❖ **d** : $P(S) \rightarrow S.Val = -1$; P2 est bloqué

❖ **b** : $x = 4$

❖ **g** : $P(S) \rightarrow S.Val = -2$; P3 est bloqué

❖ **c** : $V(S) \rightarrow S.Val = -1$; P2 est débloquent

P1	P2	P3
(a) $P(S)$; (b) $x = x + 1$; (c) $V(S)$;	(d) $P(S)$; (e) $x = x * 2$; (f) $V(S)$;	(g) $P(S)$; (h) $x = x - 4$; (i) $V(S)$;

$x = 4$



Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

❖ **a, d, b, g, c, e, f, h, i ?**

❖ **a** : $P(S) \rightarrow S.Val = 0$; P1 se branche sur b

❖ **d** : $P(S) \rightarrow S.Val = -1$; P2 est bloqué

❖ **b** : $x = 4$

❖ **g** : $P(S) \rightarrow S.Val = -2$; P3 est bloqué

❖ **c** : $V(S) \rightarrow S.Val = -1$; P2 est débloquent

❖ **e** : $x = 8$

P1	P2	P3
(a) $P(S)$; (b) $x = x + 1$; (c) $V(S)$;	(d) $P(S)$; (e) $x = x * 2$; (f) $V(S)$;	(g) $P(S)$; (h) $x = x - 4$; (i) $V(S)$;

$x = 8$



Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

❖ **a, d, b, g, c, e, f, h, i ?**

❖ **a** : $P(S) \rightarrow S.Val = 0$; P1 se branche sur b

❖ **d** : $P(S) \rightarrow S.Val = -1$; P2 est bloqué

❖ **b** : $x = 4$

❖ **g** : $P(S) \rightarrow S.Val = -2$; P3 est bloqué

❖ **c** : $V(S) \rightarrow S.Val = -1$; P2 est débloquent

❖ **e** : $x = 8$

❖ **f** : $V(S) \rightarrow S.Val = 0$; P3 est débloquent

P1	P2	P3
(a) $P(S)$; (b) $x = x + 1$; (c) $V(S)$;	(d) $P(S)$; (e) $x = x * 2$; (f) $V(S)$;	(g) $P(S)$; (h) $x = x - 4$; (i) $V(S)$;

$x = 8$

Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

❖ **a, d, b, g, c, e, f, h, i ?**

❖ **a** : $P(S) \rightarrow S.Val = 0$; P1 se branche sur b

❖ **d** : $P(S) \rightarrow S.Val = -1$; P2 est bloqué

❖ **b** : $x = 4$

❖ **g** : $P(S) \rightarrow S.Val = -2$; P3 est bloqué

❖ **c** : $V(S) \rightarrow S.Val = -1$; P2 est débloquent

❖ **e** : $x = 8$

❖ **f** : $V(S) \rightarrow S.Val = 0$; P3 est débloquent

❖ **h** : $x = 4$

P1	P2	P3
(a) $P(S)$; (b) $x = x + 1$; (c) $V(S)$;	(d) $P(S)$; (e) $x = x * 2$; (f) $V(S)$;	(g) $P(S)$; (h) $x = x - 4$; (i) $V(S)$;

$x = 4$

Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

❖ **a, d, b, g, c, e, f, h, i ?**

❖ **a** : $P(S) \rightarrow S.Val = 0$; P1 se branche sur b

❖ **d** : $P(S) \rightarrow S.Val = -1$; P2 est bloqué

❖ **b** : $x = 4$

❖ **g** : $P(S) \rightarrow S.Val = -2$; P3 est bloqué

❖ **c** : $V(S) \rightarrow S.Val = -1$; P2 est débloquent

❖ **e** : $x = 8$

❖ **f** : $V(S) \rightarrow S.Val = 0$; P3 est débloquent

❖ **h** : $x = 4$

❖ **i** : $V(S) \rightarrow S.Val = 1$;

P1	P2	P3
(a) $P(S)$;	(d) $P(S)$;	(g) $P(S)$;
(b) $x = x + 1$;	(e) $x = x * 2$;	(h) $x = x - 4$;
(c) $V(S)$;	(f) $V(S)$;	(i) $V(S)$;

X = 4

Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

Le scénario suivant est-il possible ?

❖ **a, d, e, b, c, f, g, h, i ?**

P1	P2	P3
(a) $P(S)$; (b) $x = x + 1$; (c) $V(S)$;	(d) $P(S)$; (e) $x = x * 2$; (f) $V(S)$;	(g) $P(S)$; (h) $x = x - 4$; (i) $V(S)$;

$X = 3$

Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

❖ **a, d, e, b, c, f, g, h, i ?**

❖ **a : $P(S) \rightarrow S.Val = 0$; P1 se branche sur b**

P1	P2	P3
(a) $P(S)$; (b) $x = x + 1$; (c) $V(S)$;	(d) $P(S)$; (e) $x = x * 2$; (f) $V(S)$;	(g) $P(S)$; (h) $x = x - 4$; (i) $V(S)$;

$X = 3$

Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

❖ **a, d, e, b, c, f, g, h, i ?**

❖ **a : $P(S) \rightarrow S.Val = 0$; P1 se branche sur b**

❖ **d : $P(S) \rightarrow S.Val = -1$; P2 est bloqué**

P1	P2	P3
(a) $P(S)$; (b) $x = x + 1$; (c) $V(S)$;	(d) $P(S)$; (e) $x = x * 2$; (f) $V(S)$;	(g) $P(S)$; (h) $x = x - 4$; (i) $V(S)$;

X = 3



Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

❖ **a, d, e, b, c, f, g, h, i ?**

❖ **a** : $P(S) \rightarrow S.Val = 0$; P1 se branche sur b

❖ **d** : $P(S) \rightarrow S.Val = -1$; P2 est bloqué

❖ **e** : $x = x * 2$ **IMPOSSIBLE** car P2 est bloqué

P1	P2	P3
(a) $P(S)$; (b) $x = x + 1$; (c) $V(S)$;	(d) $P(S)$; (e) $x = x * 2$; (f) $V(S)$;	(g) $P(S)$; (h) $x = x - 4$; (i) $V(S)$;

X = 3



Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

❖ **Quelles sont les valeurs possibles de x ?**

P1	P2	P3
(a) $P(S)$; (b) $x = x + 1$; (c) $V(S)$;	(d) $P(S)$; (e) $x = x * 2$; (f) $V(S)$;	(g) $P(S)$; (h) $x = x - 4$; (i) $V(S)$;

Exercice 1

❖ Semaphore $S=1$ (exclusion mutuelle sur la variable partagée x)

❖ Les valeurs possibles de x :

❖ $P1, P2, P3 : x = 4$

❖ $P1, P3, P2 : x = 0$

❖ $P2, P1, P3 : x = 3$

❖ $P2, P3, P1 : x = 3$

❖ $P3, P1, P2 : x = 0$

❖ $P3, P2, P1 : x = -1$

P1	P2	P3
(a) P(S) ; (b) $x = x + 1$; (c) V(S) ;	(d) P(S) ; (e) $x = x * 2$; (f) V(S) ;	(g) P(S) ; (h) $x = x - 4$; (i) V(S) ;

Exercice 2

On considère deux processus P1 et P2 tel que:

- P1 est un processus qui fait augmenter de 1 la valeur d'une variable X et reste bloqué si cette valeur atteint 4 ;**
- P2 est un processus qui fait diminuer de 1 la valeur de la variable X et reste bloqué si $X=0$.**

On envisage d'utiliser des sémaphores pour assurer l'exclusion mutuelle et la synchronisation de ces deux processus. Une solution est la suivante :

Exercice 2

❖ $S1 = 4$, $S2 = 0$ et $exMut = 1$

**P1 fait
augmenter de 1
la valeur d'une
variable X et
reste bloqué si
cette valeur
atteint 4 ;**

Processus P1

Répéter indéfiniment

P(exMut)

P(S1)

Lire X

X=X+1

Ecrire X

V(S2)

V(exMut)

Processus P2

Répéter indéfiniment

P(exMut)

P(S2)

Lire X

X=X-1

Ecrire X

V(S1)

V(exMut)

**P2 fait
diminuer de 1
la valeur de la
variable X et
reste bloqué si
X=0.**

Exercice 2

❖ $S1 = 4, S2 = 0$ et $exMut = 1$

P1 fait
augmenter de 1
la valeur d'une
variable X et
reste bloqué si
cette valeur
atteint 4 ;

Processus P1

Répéter indéfiniment

P(exMut)

P(S1)

Lire X

X=X+1

Ecrire X

V(S2)

V(exMut)

Processus P2

Répéter indéfiniment

P(exMut)

P(S2)

Lire X

X=X-1

Ecrire X

V(S1)

V(exMut)

P2 fait
diminuer de 1
la valeur de la
variable X et
reste bloqué si
 $X=0$.

- 1) Précisez l'utilité de chacun des sémaphores utilisés.
- 2) Que pensez-vous de cette solution :
 - Est-elle correcte ? Pourquoi ?
 - Faudrait-il un sémaphore supplémentaire ?
- 3) Proposez votre solution.

Exercice 2

❖ $S1 = 4$, $S2 = 0$ et $exMut = 1$

P1 fait
augmenter de 1
la valeur d'une
variable X et
reste bloqué si
cette valeur
atteint 4 ;

Processus P1

Répéter indéfiniment

P(exMut)

P(S1)

Lire X

X=X+1

Ecrire X

V(S2)

V(exMut)

Processus P2

Répéter indéfiniment

P(exMut)

P(S2)

Lire X

X=X-1

Ecrire X

V(S1)

V(exMut)

P2 fait
diminuer de 1
la valeur de la
variable X et
reste bloqué si
 $X=0$.

Examinons la solution

Exercice 2

❖ $S1 = 4, S2 = 0$ et $exMut = 1$

Processus P1

Répéter indéfiniment

P(exMut)

P(S1)

Lire X

X=X+1

Ecrire X

V(S2)

V(exMut)

Processus P2

Répéter indéfiniment

P(exMut)

P(S2)

Lire X

X=X-1

Ecrire X

V(S1)

V(exMut)

Section critique

Exercice 2

❖ $S1 = 4, S2 = 0$ et **exMut = 1**

Processus P1

Répéter indéfiniment

P(exMut)

P(S1)

Lire X

X=X+1

Ecrire X

V(S2)

V(exMut)

Processus P2

Répéter indéfiniment

P(exMut)

P(S2)

Lire X

X=X-1

Ecrire X

V(S1)

V(exMut)

**exMut est un sémaphore
d'exclusion mutuelle**

Exercice 2

❖ $S1 = 4$, $S2 = 0$ et $exMut = 1$

Processus P1

Répéter indéfiniment

P(exMut)

P(S1)

Lire X

X=X+1

Ecrire X

V(S2)

V(exMut)

Processus P2

Répéter indéfiniment

P(exMut)

P(S2)

Lire X

X=X-1

Ecrire X

V(S1)

V(exMut)

Mais il y a un Risque d'interblocage

Exercice 2

❖ $S1 = 4, S2 = 0$ et $exMut = 1$

Processus P1

Répéter indéfiniment

P(exMut)

P(S1)

Lire X

$X=X+1$

Ecrire X

V(S2)

V(exMut)

Processus P2

Répéter indéfiniment

P(exMut)

P(S2)

Lire X

$X=X-1$

Ecrire X

V(S1)

V(exMut)

Si le P2 fait
l'allocation de la
section critique avant
le P1

Alors il sera bloqué
au niveau de P(S2)
puisque S2 est déjà
égale à 0 (Ceci dit
que P(S2) va donner
un $S2 = -1$; donc P2
se bloque)

Mais il y a un Risque d'interblocage

Exercice 2

❖ $S1 = 4$, $S2 = 0$ et $exMut = 1$

Le P1 se bloque alors
au niveau de
 $P(exMut)$

Processus P1
Répéter indéfiniment
 $P(exMut)$
 $P(S1)$
Lire X
 $X=X+1$
Ecrire X
 $V(S2)$
 $V(exMut)$

Processus P2
Répéter indéfiniment
 $P(exMut)$
 $P(S2)$
Lire X
 $X=X-1$
Ecrire X
 $V(S1)$
 $V(exMut)$

Si le P2 fait
l'allocation de la
section critique avant
le P1
Alors il sera bloqué
au niveau de $P(S2)$
puisque $S2$ est déjà
égale à 0 (Ceci dit
que $P(S2)$ va donner
un $S2 = -1$; donc P2
se bloque)

Mais il y a un Risque d'interblocage

Exercice 2

❖ $S1 = 4$, $S2 = 0$ et $exMut = 1$

Processus P1	Processus P2
<i>Répéter indéfiniment</i>	<i>Répéter indéfiniment</i>
P(S1)	P(S2)
P(exMut)	P(exMut)
Lire X	Lire X
X=X+1	X=X-1
Ecrire X	Ecrire X
V(exMut)	V(exMut)
V(S2)	V(S1)

La **solution correcte** pour éviter l'interblocage est de permuter les positions des P et des V.

Les P et V du sémaphore binaire doivent protéger directement la section critique

Exercice 2

❖ $S1 = 4$, $S2 = 0$ et $exMut = 1$

P1 fait
augmenter de 1
la valeur d'une
variable X et
reste bloqué si
cette valeur
atteint 4 ;

Processus P1

Répéter indéfiniment

P(S1)

P(exMut)

Lire X

$X=X+1$

Ecrire X

V(exMut)

V(S2)

Processus P2

Répéter indéfiniment

P(S2)

P(exMut)

Lire X

$X=X-1$

Ecrire X

V(exMut)

V(S1)

P2 fait
diminuer de 1
la valeur de la
variable X et
reste bloqué si
 $X=0$.

Le sémaphore S1 permet de bloquer P1 si la valeur de x atteint 4

Exercice 2

❖ $S1 = 4$, $S2 = 0$ et $exMut = 1$

P1 fait
augmenter de 1
la valeur d'une
variable X et
reste bloqué si
cette valeur
atteint 4 ;

Processus P1

Répéter indéfiniment

P(S1)

P(exMut)

Lire X

X=X+1

Ecrire X

V(exMut)

V(S2)

Processus P2

Répéter indéfiniment

P(S2)

P(exMut)

Lire X

X=X-1

Ecrire X

V(exMut)

V(S1)

P2 fait
diminuer de 1
la valeur de la
variable X et
reste bloqué si
 $X=0$.

Le sémaphore S2 permet de bloquer P2 si la valeur de x atteint 0

Exercice 2

❖ $S1 = 4, S2 = 0$ et $exMut = 1$

Processus P1

Répéter indéfiniment

P(S1)
P(exMut)
Lire X
X=X+1
Ecrire X
V(exMut)
V(S2)

Processus P2

Répéter indéfiniment

P(S2)
P(exMut)
Lire X
X=X-1
Ecrire X
V(exMut)
V(S1)

Cette **solution est la solution correcte** pour le problème de synchronisation : il n'y a ni un besoin d'ajouter d'un autre sémaphore ni un besoin de proposer une autre solution.

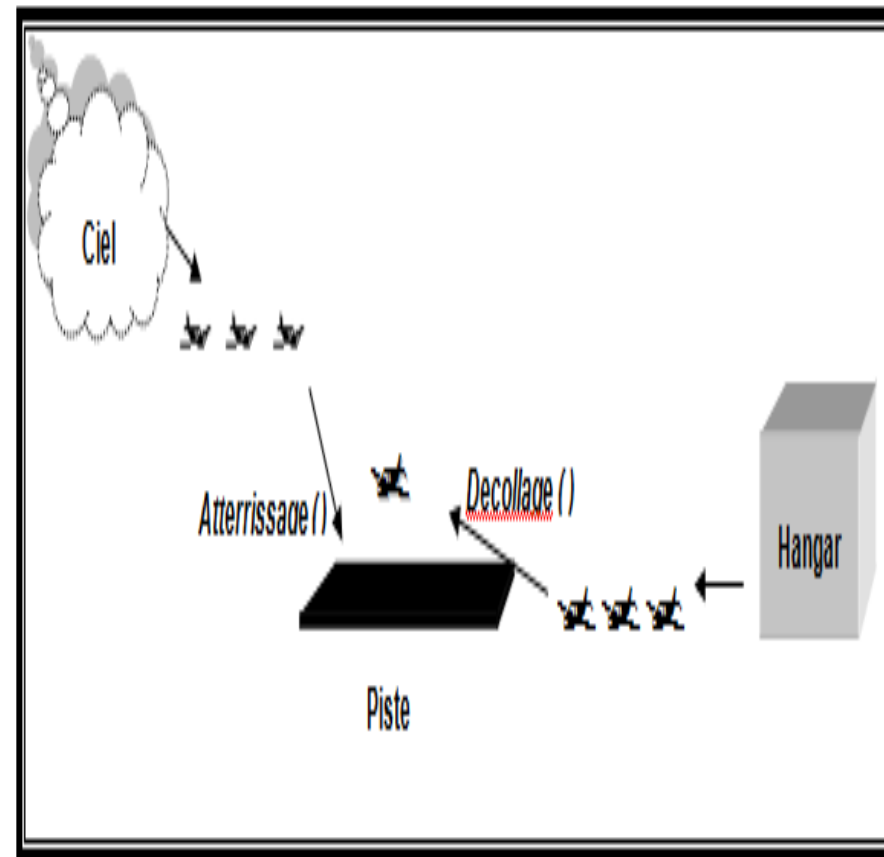
Exercice 3

Le but de cet exercice est la gestion du trafic aérien.

On ne dispose que **d'une seule piste à la fois d'atterrissage et de décollage**.

En plus, cette piste ne peut accepter qu'un seul avion quelque soit la manœuvre (atterrissage ou décollage). Pour cela, on dispose de **deux files d'attente** :

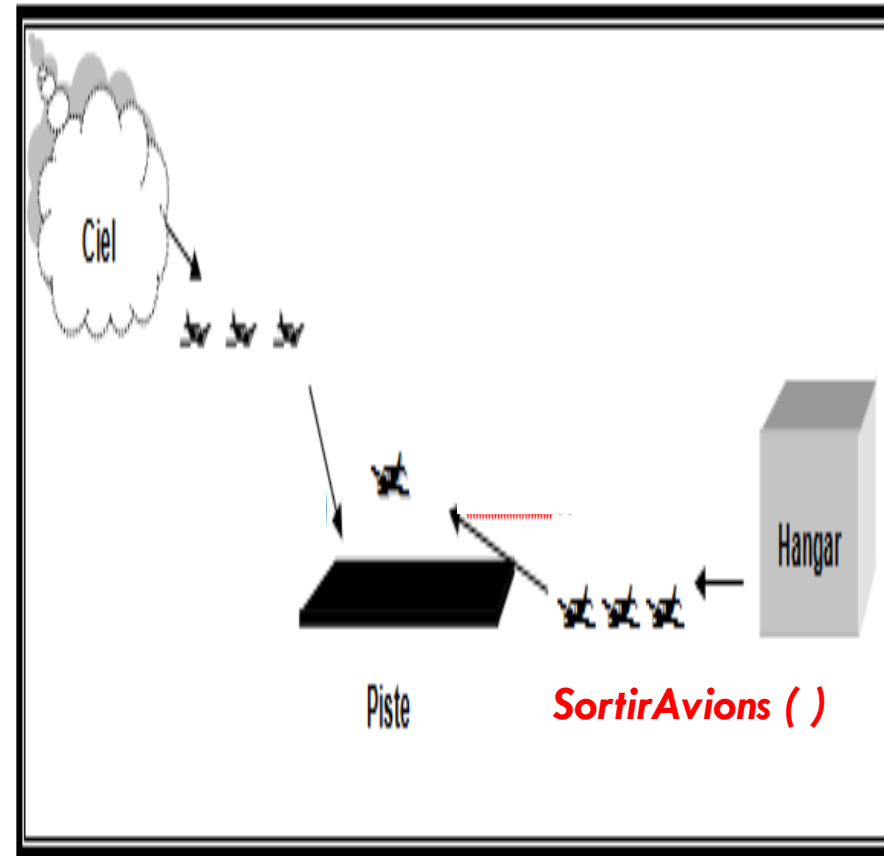
- ❖ **en air de taille N** pour les avions souhaitant atterrir
- ❖ **et au sol de taille M** pour les avions souhaitant décoller.



Exercice 3

La gestion de ce trafic des avions nécessite, alors, quatre fonctions :

- **une fonction *SortirAvions ()*** qui fait sortir les avions du hangar (dépôt) et les placer dans la file d'attente de décollage,



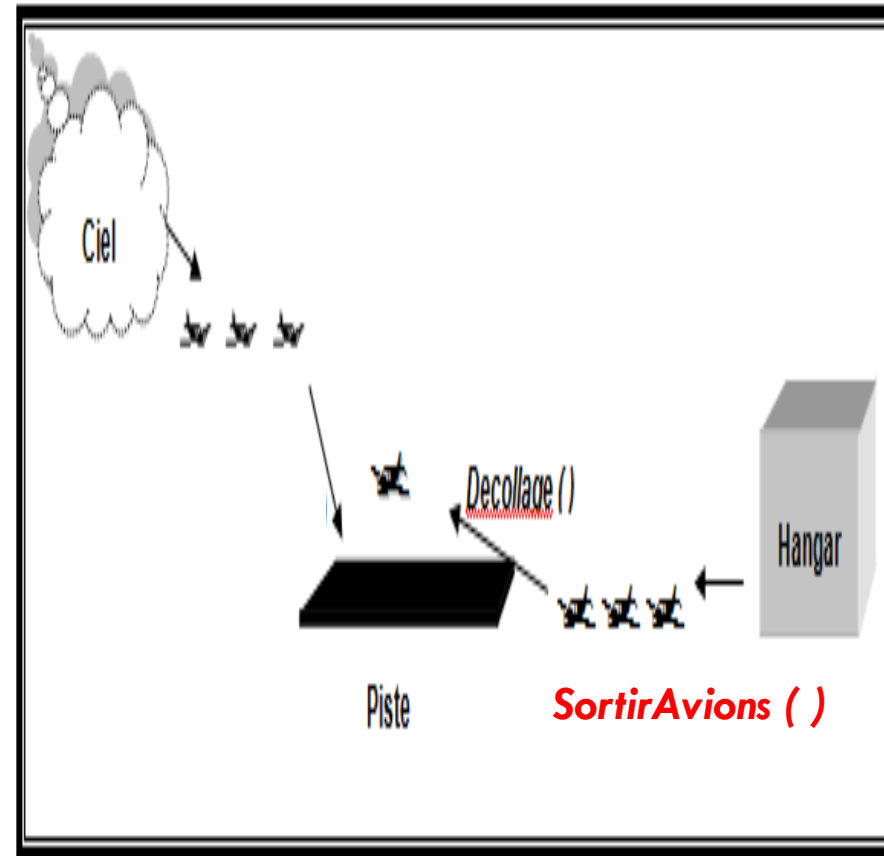
Exercice 3

SE

TD 5

La gestion de ce trafic des avions nécessite, alors, quatre fonctions :

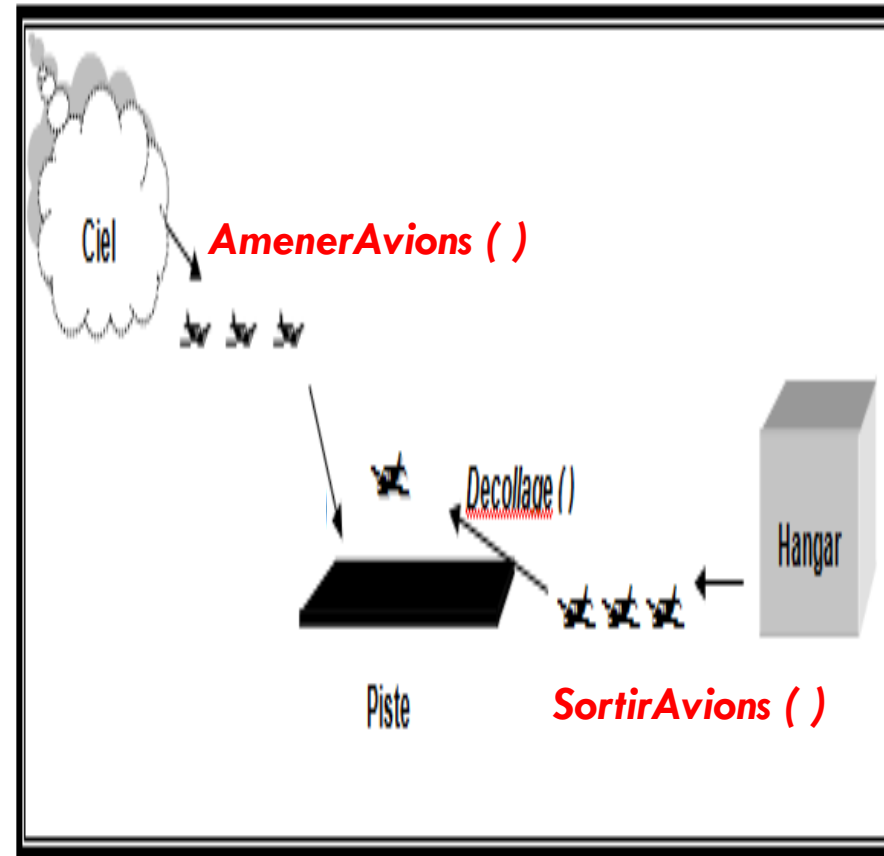
- **une fonction *SortirAvions ()*** qui fait sortir les avions du hangar (dépôt) et les placer dans la file d'attente de décollage,
- **une fonction *Decollage ()*** qui prend un avion cloué en sol dans la file d'attente de décollage et le fait décoller en utilisant la piste,



Exercice 3

La gestion de ce trafic des avions nécessite, alors, quatre fonctions :

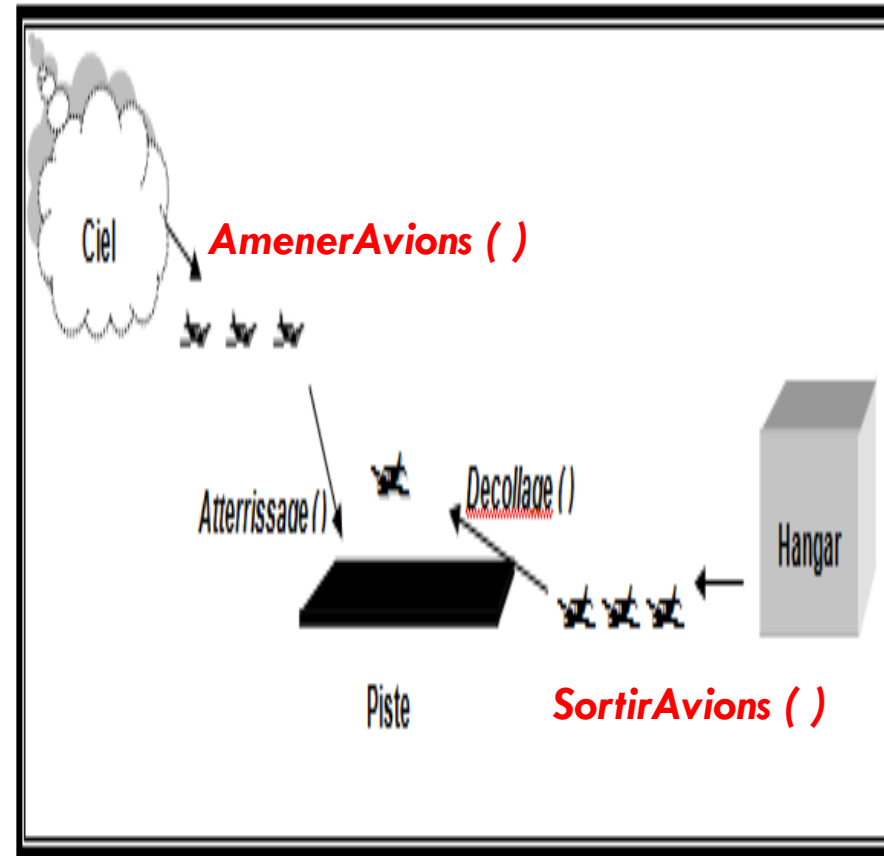
- **une fonction *SortirAvions ()*** qui fait sortir les avions du hangar (dépôt) et les placer dans la file d'attente de décollage,
- **une fonction *Decollage ()*** qui prend un avion cloué en sol dans la file d'attente de décollage et le fait décoller en utilisant la piste,
- **une fonction *AmenerAvions ()*** qui fait entrer, dans la file d'attente d'atterrissage, des avions en vol



Exercice 3

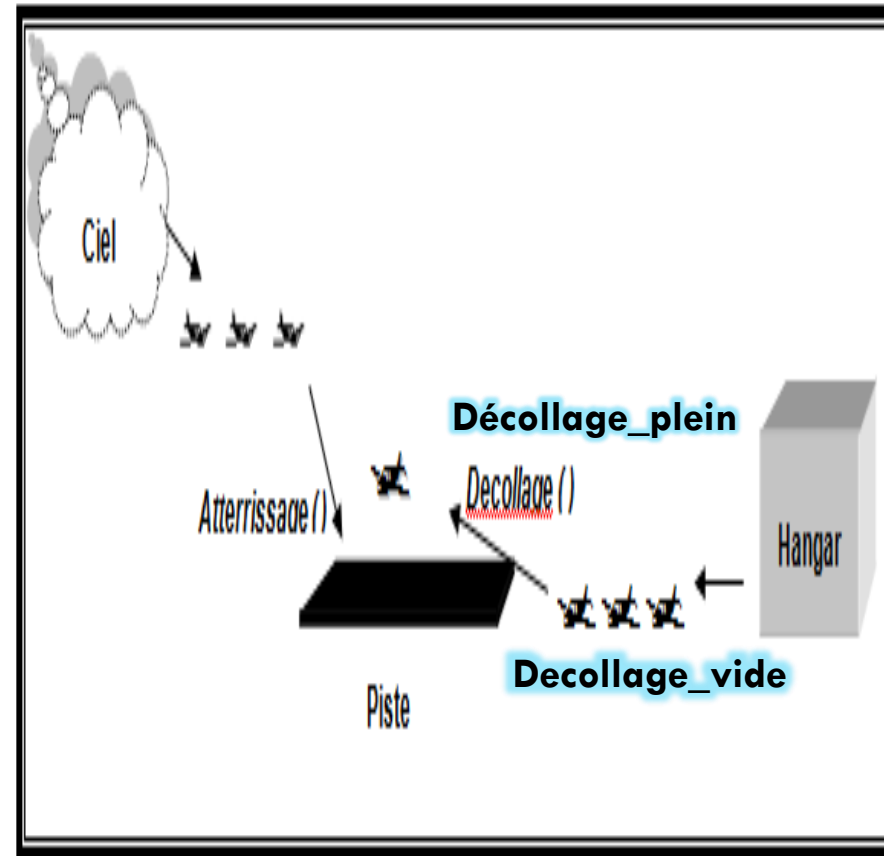
La gestion de ce trafic des avions nécessite, alors, quatre fonctions :

- **une fonction *SortirAvions ()*** qui fait sortir les avions du hangar (dépôt) et les placer dans la file d'attente de décollage,
- **une fonction *Decollage ()*** qui prend un avion cloué en sol dans la file d'attente de décollage et le fait décoller en utilisant la piste,
- **une fonction *AmenerAvions ()*** qui fait entrer, dans la file d'attente d'atterrissage, des avions en vol
- **une fonction *Atterrissage ()*** qui prend un avion de la file d'attente d'atterrissage et le fait atterrir en utilisant la piste.



Exercice 3

```
#define N 5
#define M 5
semaphore Decollage_vide, Décollage_plein,
Atterrissage_vide, Atterrissage_plein, Piste;
Sem_Init (Decollage_vide, M);
Sem_Init (Decollage_plein, 0);
Sem_Init (Atterrissage_vide, N);
Sem_Init (Atterrissage_plein, 0);
Sem_Init (Piste, 1);
```

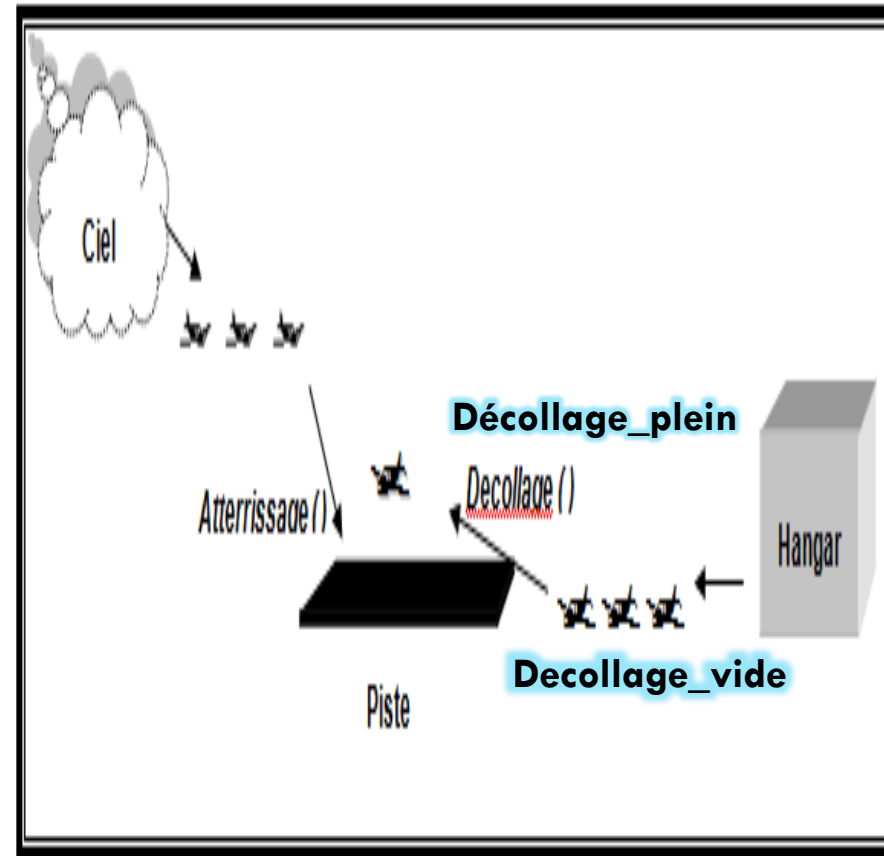


Exercice 3

SE

TD 5

```
Void SortirAvions( )  
{  
  
    Ajouter_un_avion_zone_decollage( )  
  
}  
  
Void Decolage( )  
{  
  
    Faire_decoller_un_avion ( );  
  
}
```



Exercice 3

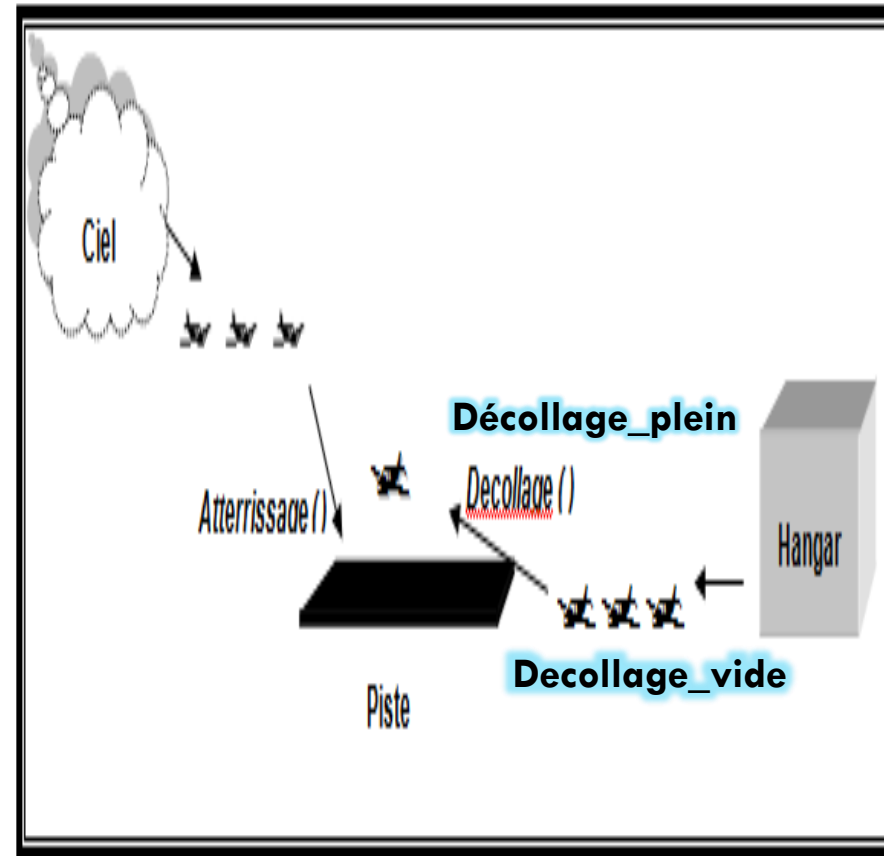
SE

TD 5

```
Void SortirAvions( )  
{  
  P (Decollage_vide)  
    Ajouter_un_avion_zone_decollage( )  
  V (Decollage_plein)  
}
```



```
Void Decolage( )  
{  
  P (Decollage_plein)  
  
  Faire_decoller_un_avion ( );  
  
  V (Decollage_vide)  
}
```



Exercice 3

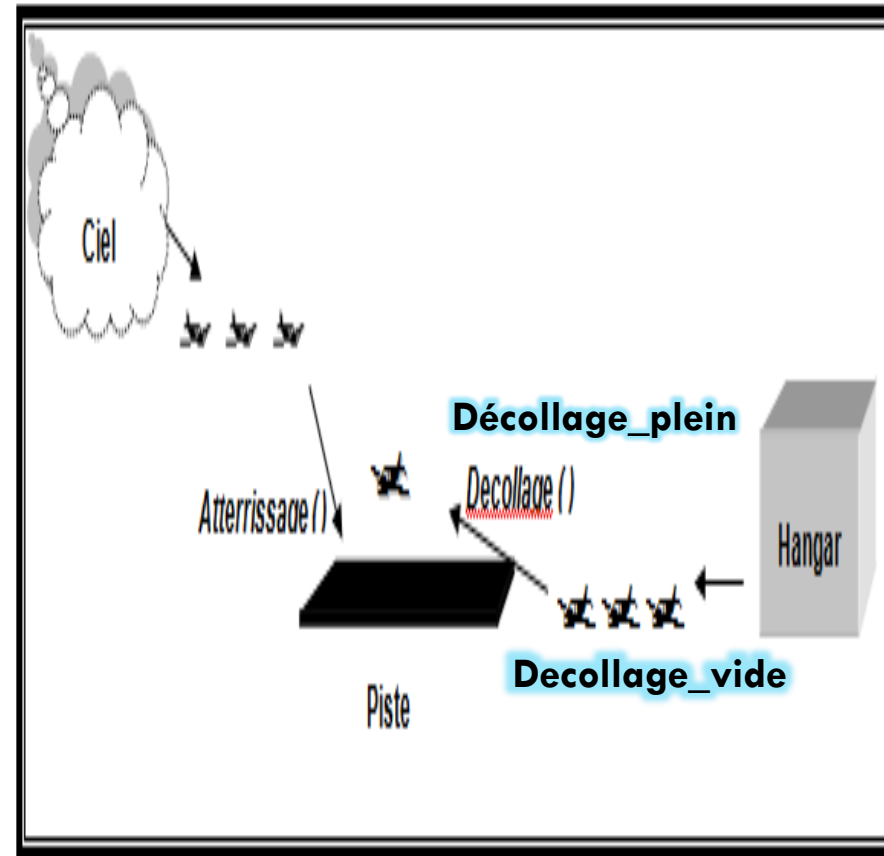
SE

TD 5

```
Void SortirAvions( )  
{  
  P (Decollage_vide)  
    Ajouter_un_avion_zone_decollage( )  
  V (Decollage_plein)  
}
```



```
Void Decolage( )  
{  
  P (Decollage_plein)  
  P (Piste)  
  Faire_decoller_un_avion ( );  
  V (Piste)  
  V (Decollage_vide)  
}
```

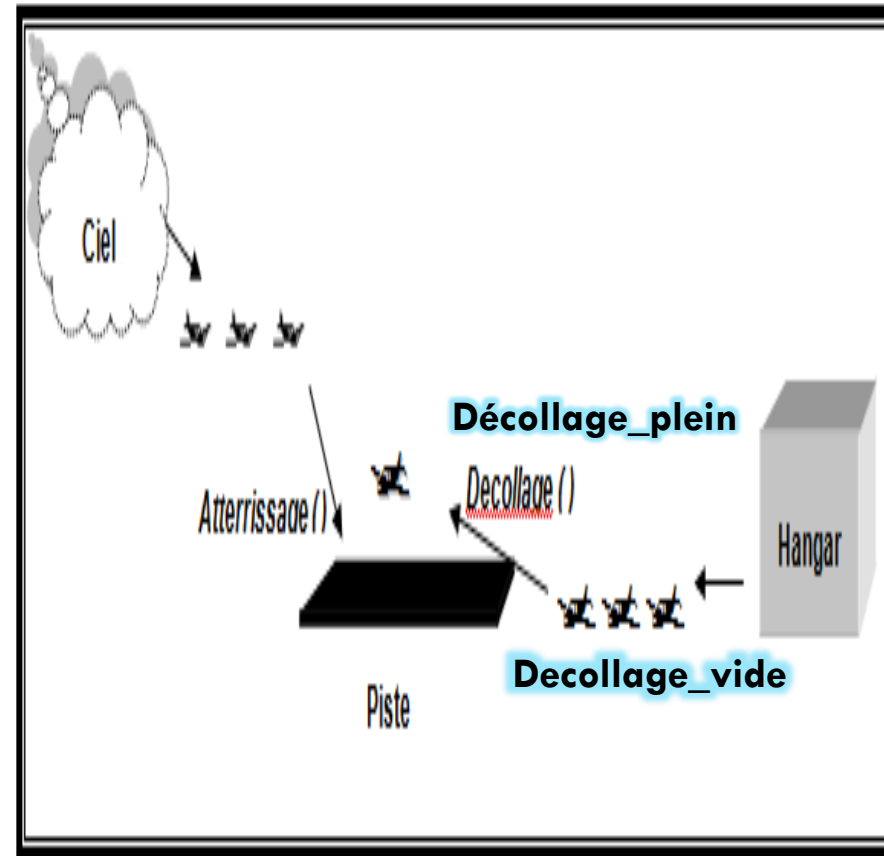


Exercice 3

Piste : Pour assurer l'exclusion mutuelle sur la piste

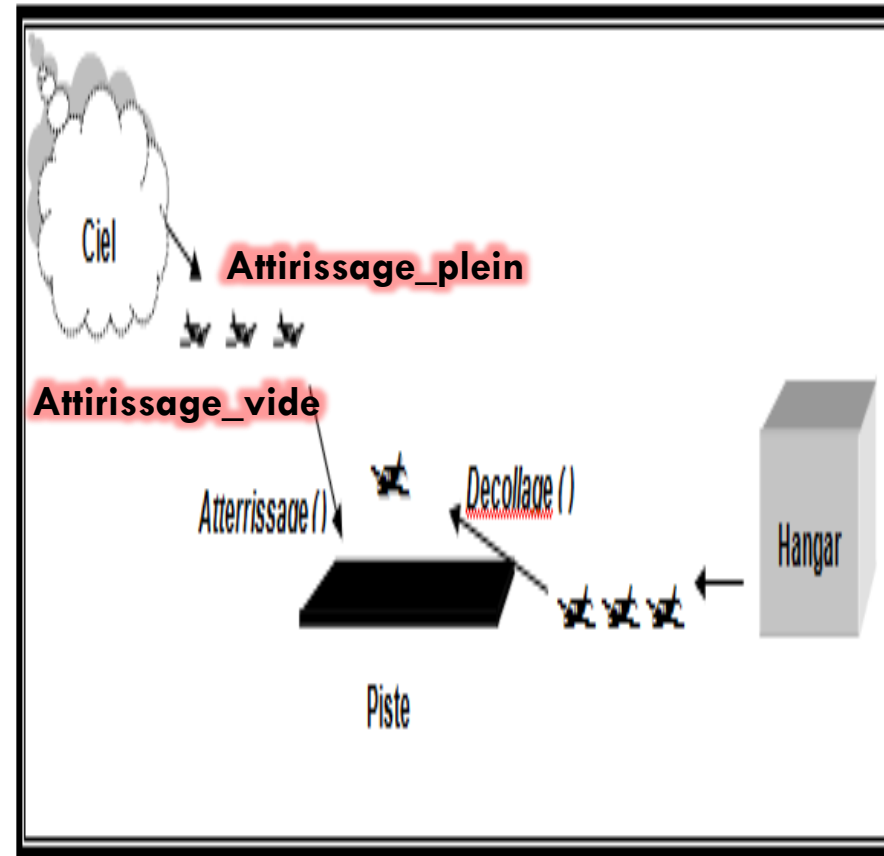
Decollage_vide : pour gérer le nombre de place vide au niveau de la file d'attente de décollage et bloquer un avion de joindre cette file d'attente s'il n'y pas de place vide

Decollage_plein : pour gérer le nombre de place pleine au niveau de la file d'attente de décollage et bloquer le décollage s'il n'y pas de place pleine dans la file de décollage (pas d'avion disponible)



Exercice 3

```
#define N 5
#define M 5
semaphore Decollage_vide, Décollage_plein,
Atterrissage_vide, Atterrissage_plein, Piste;
Sem_Init (Decollage_vide, M);
Sem_Init (Decollage_plein, 0);
Sem_Init (Atterrissage_vide, N);
Sem_Init (Atterrissage_plein, 0);
Sem_Init (Piste, 1);
```

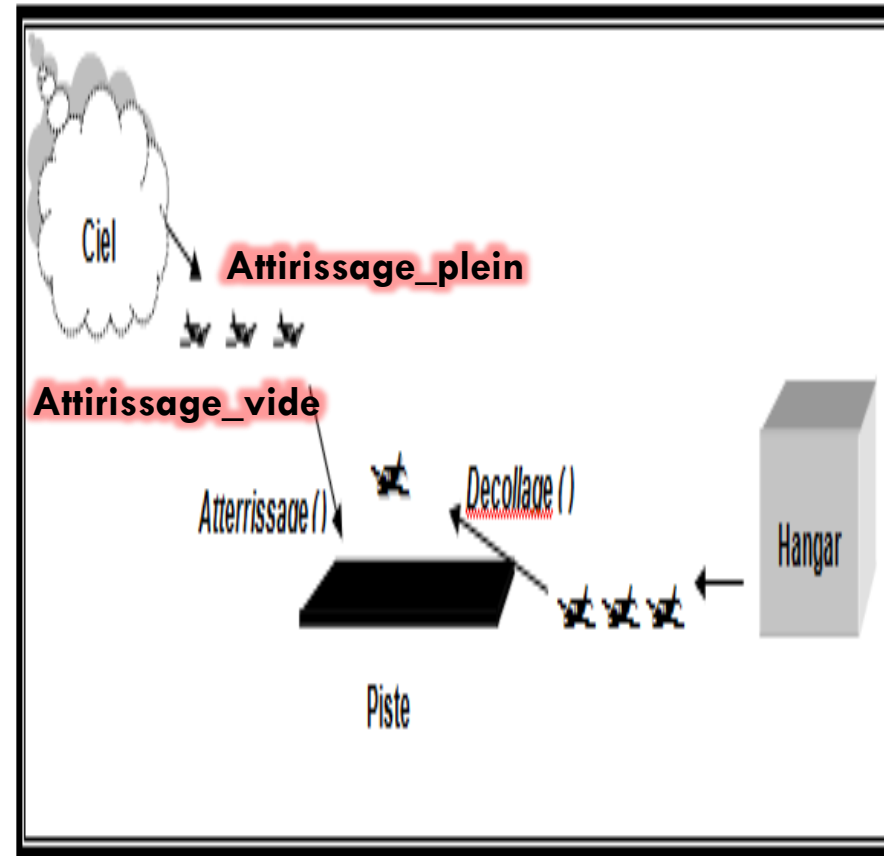


Exercice 3

SE

TD 5

```
Void AmenerAvions( )  
{  
  
    Ajouter_un_avion_zone_atterrissage( );  
  
}  
  
Void Atterrissage( )  
{  
  
    Faire_Atterrir_un_avion ( );  
  
}
```

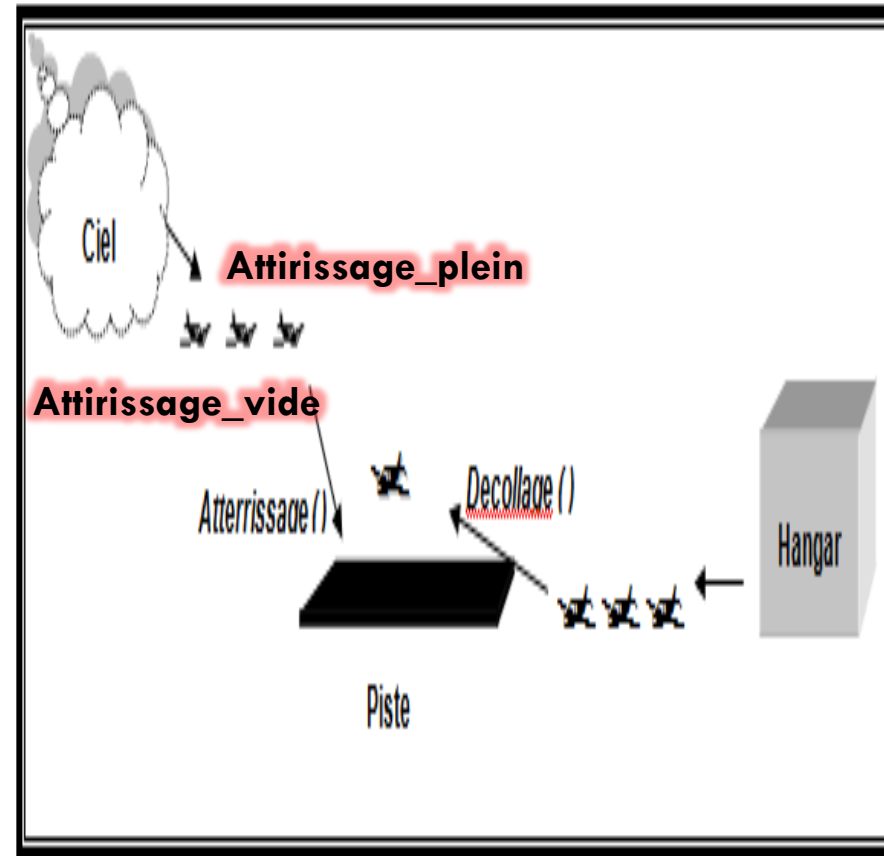


Exercice 3

SE

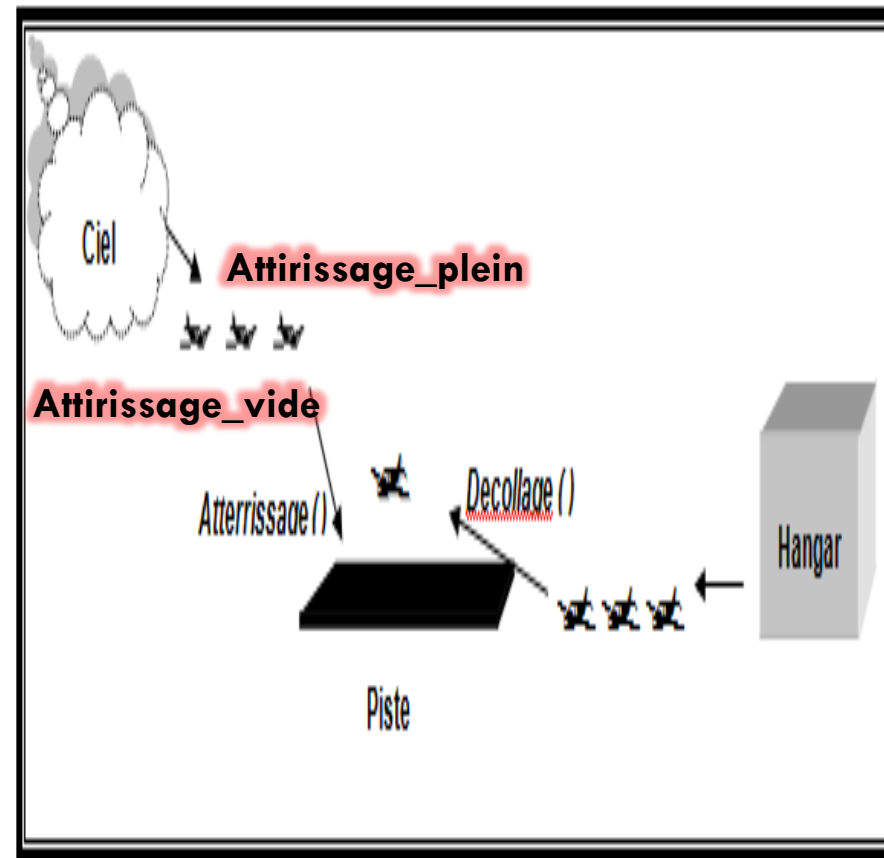
TD 5

```
Void AmenerAvions( )  
{  
  P (Atterrissage_vide)  
    Ajouter_un_avion_zone_atterrissage( );  
  V (Atterrissage_plein)  
}  
  
Void Atterrissage( )  
{  
  P (Atterrissage_plein)  
  
    Faire_Atterrir_un_avion ( );  
  
  V (Atterrissage_vide)  
}
```



Exercice 3

```
Void AmenerAvions( )  
{  
  P (Atterrissage_vide)  
    Ajouter_un_avion_zone_atterrissage( );  
  V (Atterrissage_plein)  
}  
  
Void Atterrissage( )  
{  
  P (Atterrissage_plein)  
  P (Piste)  
    Faire_Atterrir_un_avion ( );  
  V (Piste)  
  V (Atterrissage_vide)  
}
```

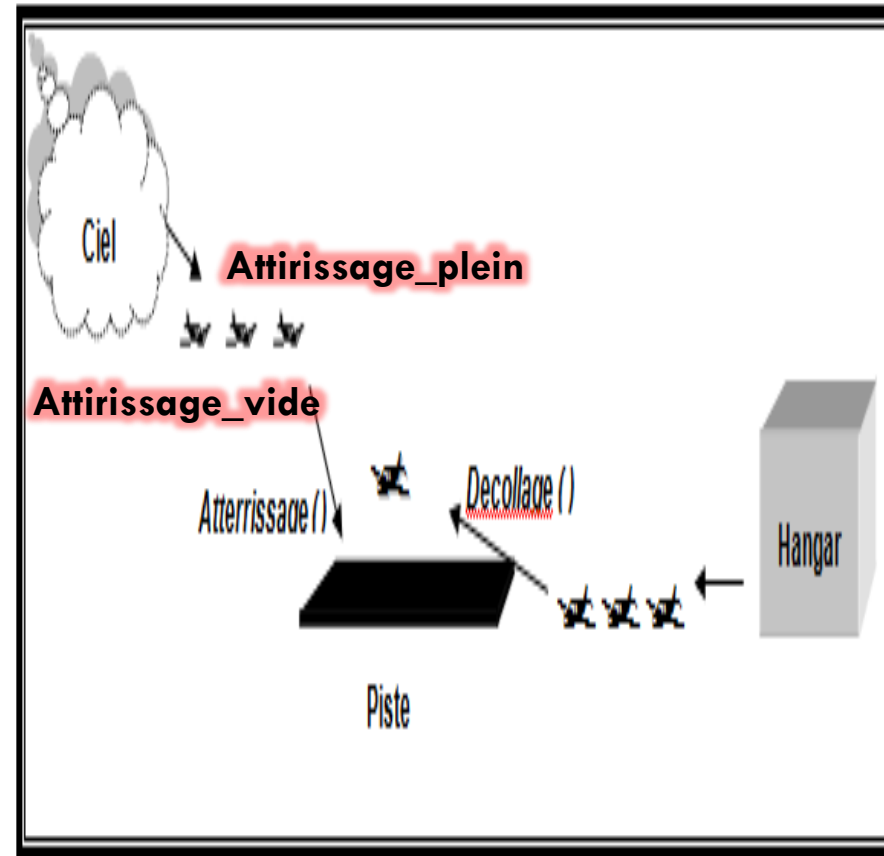


Exercice 3

Piste : Pour assurer l'exclusion mutuelle sur la piste

Attirissage_vide : pour gérer le nombre de place vide au niveau de la file d'attente d'atterrissage et bloquer un avion de joindre cette file d'attente s'il n'y a pas de place vide

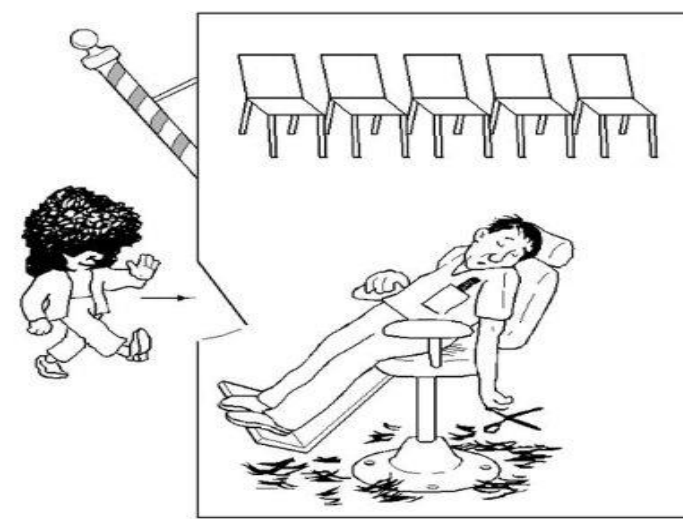
Attirissage_plein : pour gérer le nombre de place pleine au niveau de la file d'attente d'atterrissage et bloquer l'atterrissage s'il n'y a pas de place pleine dans la file d'atterrissage (pas d'avion disponible)



Exercice 4

SE

Une illustration classique du problème de la synchronisation est celui du salon de coiffure.



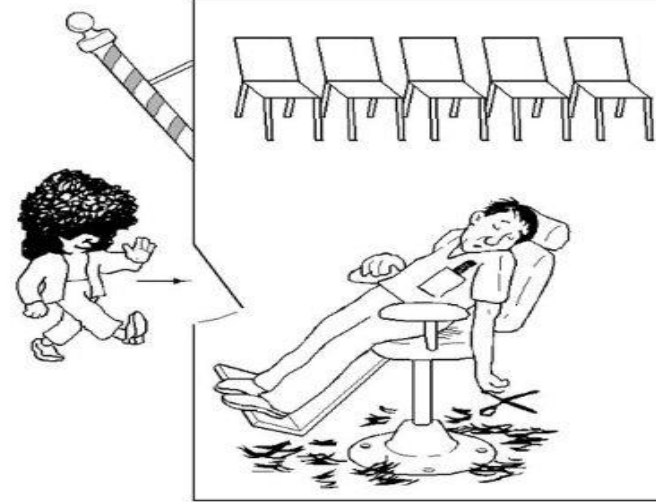
Dans ce salon, il y a un coiffeur, un *fauteuil* dans lequel se met le client pour être coiffé et N sièges pour attendre.

- **En absence de clients, le coiffeur somnole dans le fauteuil.**
- **Quand le premier client arrive, il doit réveiller le coiffeur. Ce dernier se lève et le client s'assoit alors pour se faire coiffer.**
- **Cependant si un client arrive pendant que le coiffeur est entrain de travailler :**
 - **il s'assoit et attend si un des N sièges est libre,**
 - **sinon il sort.**

Il s'agit donc de synchroniser les activités du coiffeur et de ses clients avec des sémaphores.

Exercice 4

SE



```
#define N 5
```

```
semaphore SClient, SCoiffeur, exMut ;
```

```
Sem_Init (SClient, 0);
```

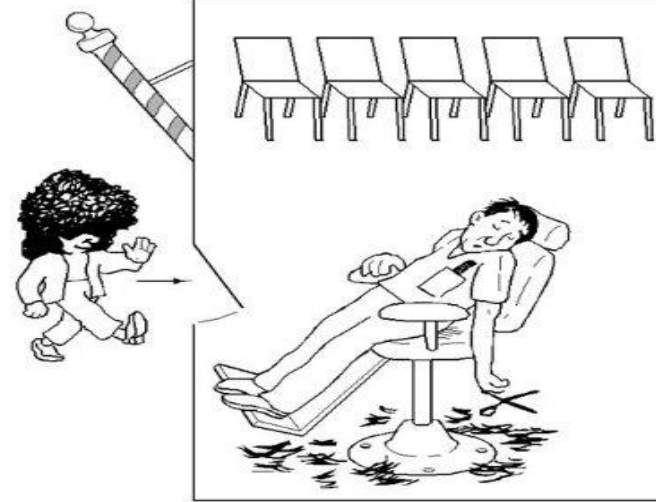
```
Sem_Init (SCoiffeur, 0);
```

```
Sem_Init (exMut, 1);
```

```
int waiting = 0 ;    //nombre de clients en attente
```

Exercise 4

SE



```
void Coiffeur()
```

```
{
```

```
    while (TRUE)
```

```
    {
```

```
        waiting --;
```

```
        coiffer();
```

```
    }
```

```
}
```

```
void Client()
```

```
{
```

```
    if (waiting < N)
```

```
    {
```

```
        waiting ++;
```

```
        se_faire_Coiffer();
```

```
    }
```

```
    else
```

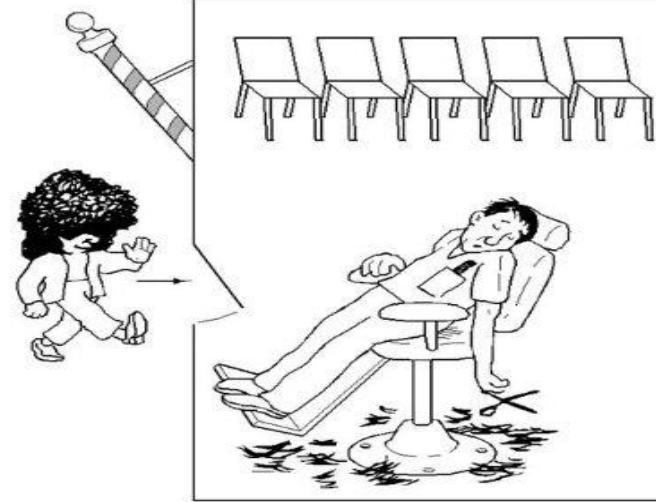
```
    {
```

```
    }
```

```
}
```

Exercise 4

SE



```
void Coiffeur()
{
    while (TRUE)
    {
        P (SClient);

        waiting --;

        coiffer();
    }
}
```

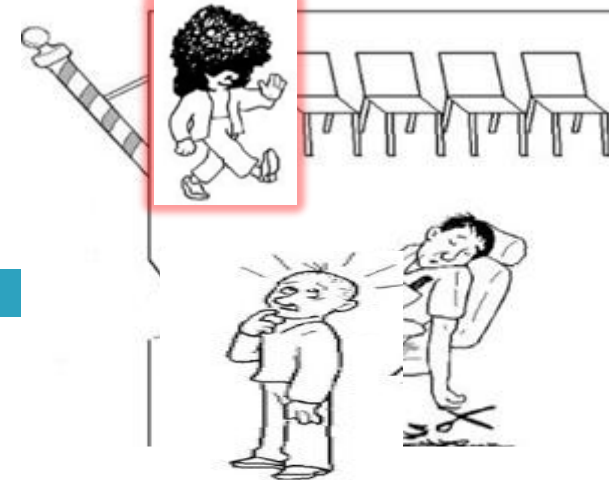
```
void Client()
{
    if (waiting < N)
    {
        waiting ++;
        V (SClient);

        se_faire_Coiffer();
    }
    else
    {
    }
}
```

Le coiffeur se bloque s'il n'y a pas de client

Exercise 4

SE



```
void Coiffeur()
{
    while (TRUE)
    {
        P (SClient);

        waiting --;
        V (SCoiffeur);

        coiffer();
    }
}
```

```
void Client()
{
    if (waiting < N)
    {
        waiting ++;
        V (SClient);

        P (SCoiffeur);
        se_faire_Coiffer();
    }
    else
    {
    }
}
```

Le client se bloque si le coiffeur est occupé

Exercise 4

```
void Coiffeur()
{
    while (TRUE)
    {
        P (SClient);
        P (exMut);
        waiting --;
        V (SCoiffeur);
        V (exMut);
        coiffer();
    }
}
```

```
void Client()
{
    P (exMut);
    if (waiting < N)
    {
        waiting ++;
        V (SClient);
        V (exMut);
        P (SCoiffeur);
        se_faire_Coiffer();
    }
    else
    {
        V (exMut);
    }
}
```

La variable **waiting** est une ressource critique partagée donc elle doit être utilisée en exclusion mutuelle

Exercice 4

```
void Coiffeur()
{
    while (TRUE)
    {
        P (SClient);
        P (exMut);
        waiting --;
        V (SCoiffeur);
        V (exMut);

        coiffer();
    }
}
```

```
void Client()
{
    P (exMut);
    if (waiting < N)
    {
        waiting ++;
        V (SClient);
        V (exMut);
        P (SCoiffeur);
        se_faire_Coiffer();
    }
    else
    {
        V (exMut);
    }
}
```

exMut sert à gérer l'exclusion mutuelle sur la variable **waiting** qui permet de contrôler la taille maximale de la file d'attente.

SClient gère la file des clients en attente. Indique le nombre de ressources pour le coiffeur, c'est à dire le nombre de clients en attente de se faire coiffer

SCoiffeur gère l'accès au fauteuil du coiffeur. Indique si le coiffeur est prêt à travailler ou non (c.a.d reflète si le coiffeur est actif (1) ou pas (0)).

Exercise 5

$S1 = 0, S2 = 0$

```
void P1 ( )  
{  
  procedure A1;  
  V(S2);  
  P(S1);  
  procedure B1;  
}
```

```
void P2 ( )  
{  
  procedure A2;  
  V(S1);  
  P(S2);  
  procedure B2;  
}
```

Quelle synchronisation a-t-on imposé sur les exécutions des procédures A1, A2, B1 et B2?

Donner le graphe de précedence correspondant.

Exercise 5

$S1 = 0, S2 = 0$

```
void P1 ( )  
{  
    procedure A1;  
    V(S2);  
    P(S1);  
    procedure B1;  
}
```

```
void P2 ( )  
{  
    procedure A2;  
    V(S1);  
    P(S2);  
    procedure B2;  
}
```

Il faut examiner les valeurs des sémaphores et leurs positions

Exercise 5

SE

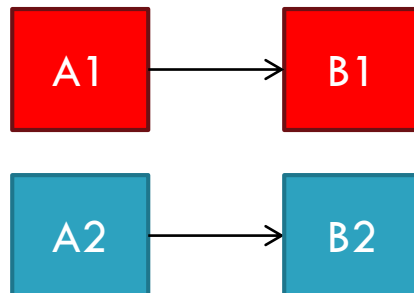
TD 5

$S1 = 0, S2 = 0$

```
void P1 ( )  
{  
    procedure A1;  
    V(S2);  
    P(S1);  
    procedure B1;  
}
```

```
void P2 ( )  
{  
    procedure A2;  
    V(S1);  
    P(S2);  
    procedure B2;  
}
```

Graphe de précedence



Ordre séquentiel

Exercise 5

SE

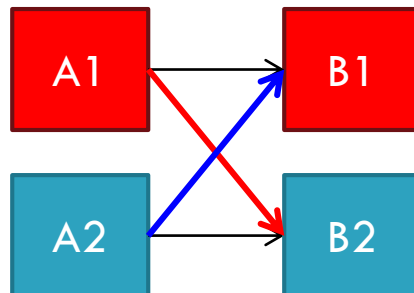
TD 5

$S1 = 0, S2 = 0$

```
void P1 ( )  
{  
  procedure A1;  
  V(S2);  
  P(S1);  
  procedure B1;  
}
```

```
void P2 ( )  
{  
  procedure A2;  
  V(S1);  
  P(S2);  
  procedure B2;  
}
```

Graphe de précedence



Ordre de synchronisation

Exercice 5

SE

TD 5

Cas 1

```
void P1 ( )
```

```
{
```

```
  procedure A1;
```

```
  procedure B1;
```

```
}
```

```
void P2 ( )
```

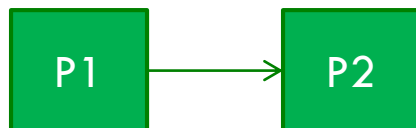
```
{
```

```
  procedure A2;
```

```
  procedure B2;
```

```
}
```

Modifier les programmes afin d'avoir le graphe de précedence suivant :



Exercise 5

SE

TD 5

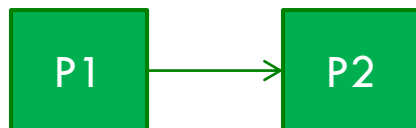
$S = 0$

Cas 1

```
void P1 ( )  
{  
  
    procedure A1;  
    procedure B1;  
    V(S);  
}
```

```
void P2 ( )  
{  
    P(S);  
    procedure A2;  
    procedure B2;  
}
```

Graphe de précedence



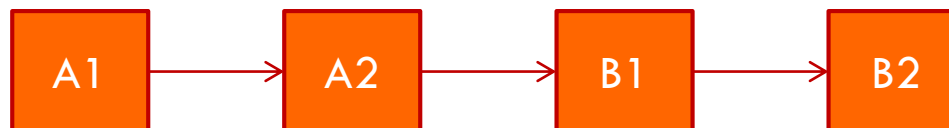
Exercise 5

Cas 2

```
void P1 ( )  
{  
  procedure A1;  
  
  procedure B1;  
}
```

```
void P2 ( )  
{  
  procedure A2;  
  
  procedure B2;  
}
```

Modifier les programmes afin d'avoir :



Exercise 5

SE

TD 5

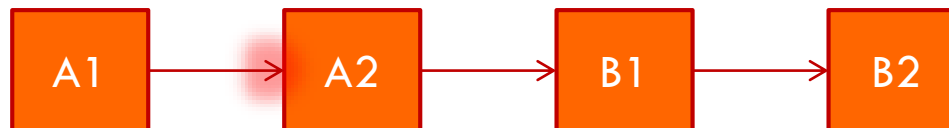
S1 = 0,

Cas 2

```
void P1 ( )  
{  
  procedure A1;  
  V(S1);  
  
  procedure B1;  
  
}
```

```
void P2 ( )  
{  
  P(S1);  
  procedure A2;  
  
  procedure B2;  
  
}
```

Graphe de précédence



Exercise 5

SE

TD 5

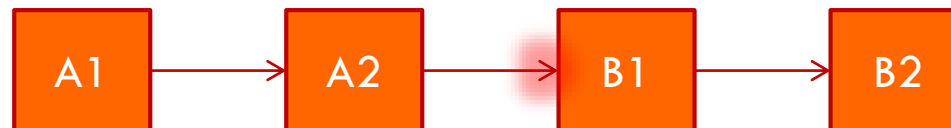
S1 = 0, **S2 = 0**

Cas 2

```
void P1 ( )  
{  
  procedure A1;  
  V(S1);  
  P(S2);  
  procedure B1;  
}
```

```
void P2 ( )  
{  
  P(S1);  
  procedure A2;  
  V(S2);  
  procedure B2;  
}
```

Graphe de précedence



Exercise 5

SE

TD 5

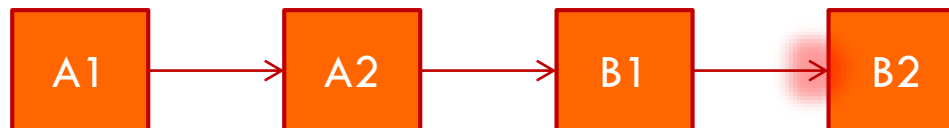
S1 = 0, S2 = 0, **S3=0**

Cas 2

```
void P1 ( )  
{  
  procedure A1;  
  V(S1);  
  P(S2);  
  procedure B1;  
  V(S3);  
}
```

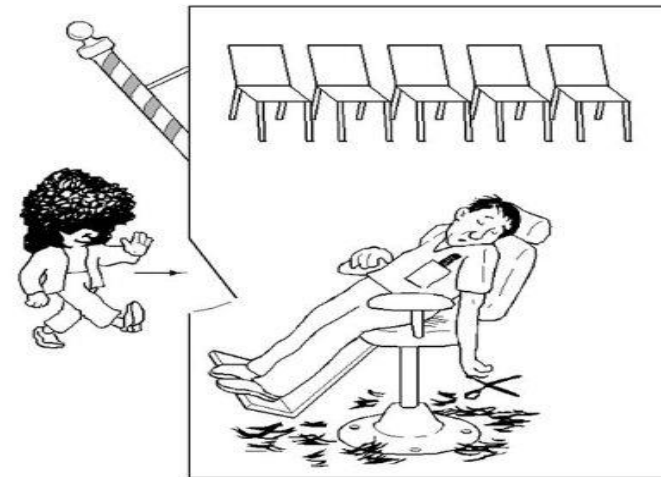
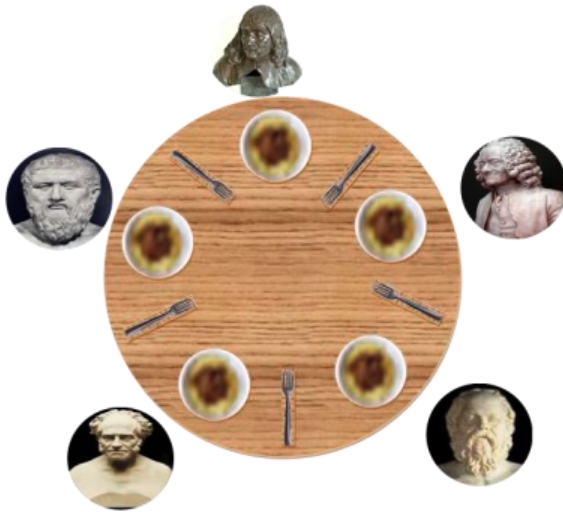
```
void P2 ( )  
{  
  P(S1);  
  procedure A2;  
  V(S2);  
  P(S3);  
  procedure B2;  
}
```

Graphe de précedence

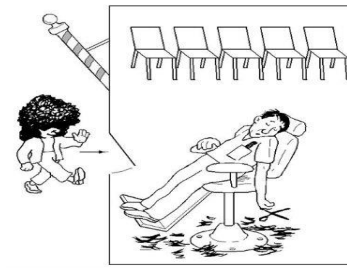


Exercice 6

- Proposez une solution avec les **moniteurs** pour chacun des problèmes classiques vus en cours (Dîner des philosophes et les lecteurs/rédacteurs) ainsi que le problème du coiffeur endormi.



Exercice 6



SE

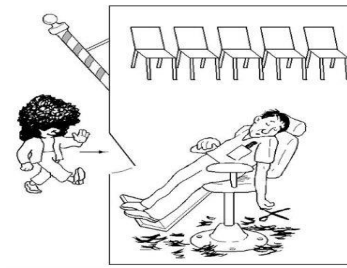
TD 5

1. Salon de coiffure avec Sémaphore

```
void Coiffeur()
{
    while (TRUE)
    {
        P (SClient);
        P (exMut);
        waiting - -;
        V (SCoiffeur);
        V (exMut);
        Coiffer();
    }
}
```

```
void Client()
{
    P (exMut);
    if (waiting < N)
    {
        waiting ++;
        V (SClient);
        V (exMut);
        P (SCoiffeur);
        Se_Faire_Coiffer();
    }
    else
    {
        V (exMut);
    }
}
```

Exercice 6



SE

TD 5

1. Salon de coiffure avec Moniteur

monitor SalonCoiffure

```
{  
    int waiting=0;  
    condition SCoiffeur, SClient;
```

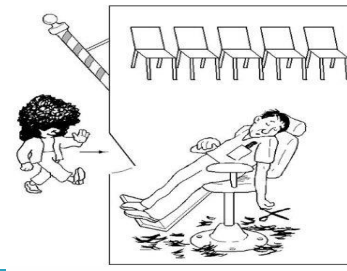
PrendreClient ()

```
{  
    if (waiting==0) wait (SClient);  
    waiting - -;  
    signal (SCoiffeur);  
    Coiffer();  
}
```

ObtenirCoupe()

```
{  
    if (waiting < N)  
    {  
        waiting ++;  
        signal (SClient);  
        wait (SCoiffeur);  
        Se_Faire_Coiffer();  
    }  
}
```

Exercice 6



SE

TD 5

1. Salon de coiffure avec Moniteur

```
monitor SalonCoiffure
{
    int waiting=0;
    condition SCoiffeur, SClient;
```

```
PrendreClient ( )
{
    if (waiting==0) wait (SClient);
    waiting - -;
    signal (SCoiffeur);
    Coiffer();
}
```

```
ObtenirCoupe( )
{
    if (waiting < N)
    {
        waiting ++;
        signal (SClient);
        wait (SCoiffeur);
        Se_Faire_Coiffer();
    }
}
```

```
void Coiffeur()
{
    while (TRUE)
        SalonCoiffure.PrendreClient();
}


void Client()
{
    SalonCoiffure.ObtenirCoupe();
}
```

Exercice 6

SE

TD 5

2. Lecteur/rédacteur avec Sémaphore



```
void lecteur ()
{
    while(1)
    {
        P(mutex);
        NbL = NbL + 1;
        if (NbL == 1) P(db);
        V(mutex);
        lire_BD();
        P(mutex);
        NbL = NbL - 1;
        if (NbL == 0) V(db);
        V(mutex);
        traitement();
    }
}
```



```
void redacteur()
{
    while(1)
    {
        creerDonnees()
        P(db);
        ecrire_BD();
        V(db);
    }
}
```




semaphore mutex = 1; // un sémaphore pour l'accès à la variable globale partagée NbL

Exercice 6

SE

TD 5

2. Lecteur/rédacteur avec Sémaphore



```
void lecteur ()
{
    while(1)
    {
        P(mutex);
        NbL = NbL + 1;
        if (NbL == 1) P(db);
        V(mutex);
        lire_BD();
        P(mutex);
        NbL = NbL - 1;
        if (NbL == 0) V(db);
        V(mutex);
        traitement();
    }
}
```



```
void redacteur()
{
    while(1)
    {
        creerDonnees()
        P(db);
        ecrire_BD();
        V(db);
    }
}
```



semaphore mutex = 1; // un sémaphore pour l'accès à la variable globale partagée NbL

Exercice 6

SE

TD 5

2. Lecteur/rédacteur vers le changement

```
void lecteur ()  
{ while(1)  
  {
```

```
    Debut_Lire();  
    lire_BD();  
    Fin_Lire();
```

```
    traitement();
```

```
  }  
}
```



```
void redacteur()  
{
```

```
  while(1)  
  {
```

```
    creerDonnees()  
    Debut_Ecrire();  
    ecrire_BD();  
    Fin_Ecrire(),
```

```
  }  
}
```



Exercice 6



SE

TD 5

2. Lecteur/rédacteur avec Moniteur

monitor LecteursRedacteurs

```
{ int NbL ;  
  bool ecr=false;  
  //indique si on est en train d'écrire dans la BD  
  condition f_lect , f_ecr;  
Debut_Lire( )  
{ NbL ++;  
  if (ecr)  
    wait (f_lect ); //s'il y a un écrivain on attend  
  
  signal (f_lect); // Les autres peuvent lire aussi  
}  
Fin_Lire( )  
{ NbL --;  
  if (NbL ==0)  
    signal (f_ecr );  
}
```

```
Debut_Ecrire( )  
{  
  if (NbL > 0 || ecr)  
    wait (f_ecr );  
  ecr =true;  
}  
Fin_Ecrire( )  
{ ecr = false;  
  if (NbL > 0)  
    signal (f_lect);  
  else  
    signal (f_ecr );  
}
```

Priorité aux lecteurs

Exercice 6



SE

TD 5

2. Lecteur/rédacteur avec Moniteur

monitor LecteursRedacteurs

```
{ int nb_lec_base, nb_lec_att, nb_ecr_att;
  bool ecr=false;
  //indique si on est en train d'écrire dans la BD
  condition f_lect, f_ecr;
Debut_Lire( )
{ nb_lec_att ++;
  if (ecr || nb_ecr_att > 0)
    wait (f_lect); //s'il y a un écrivain on attend
  nb_lec_att --;
  signal (f_lect); // Les autres peuvent lire aussi
  nb_lec_base ++;
}
Fin_Lire( )
{ nb_lec_base --;
  if (nb_lec_base ==0){
    signal (f_ecr);
  }
}
```

```
Debut_Ecrire( )
{ nb_ecr_att ++;
  if (nb_lec_base > 0 || ecr)
    wait (f_ecr);
  ecr =true;
  nb_ecr_att --;
}
Fin_Ecrire( )
{ ecr = false;
  if ( nb_ecr_att >0)
    signal (f_ecr);
  elsif (nb_lec_att >0)
    signal (f_lect);
}
}
```


Priorité aux écrivains

Exercice 6

SE

TD 5


2. Lecteur/rédacteur avec Moniteur



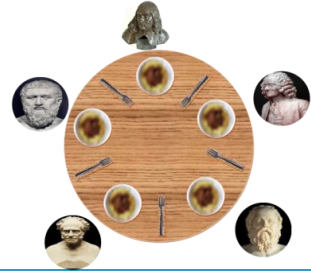
```
void lecteur ()
{
    while(1)
    {
        LecteursReacteurs.Debut_Lire();
        lire_BD();
        LecteursReacteurs.Fin_Lire();
        traitement();
    }
}
```



```
void redacteur()
{
    while(1)
    {
        creerDonnees()
        LecteursReacteurs.Debut_Ecrire();
        ecrire_BD();
        LecteursReacteurs.Fin_Ecrire(),
    }
}
```



Exercice 6



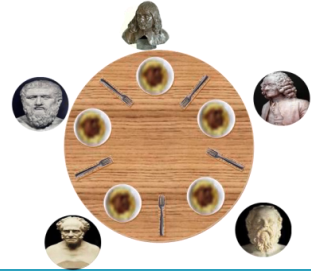
SE

TD 5

3. Repas des philosophes avec **sémaphore**

```
void philosophe (int i) // i est le numéro du philosophe
{
    while(1)
    {
        penser();
        Prendre_Fourchettes(i); // il prend deux fourchette ou se bloque
        manger();
        Poser_Fourchettes(i); // il pose les deux fourchettes sur la table
    }
}
```

Exercice 6



SE

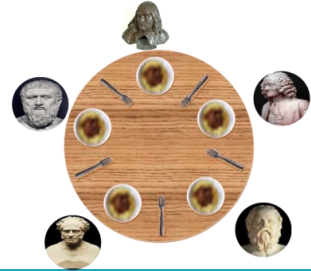
TD 5

3. Repas des philosophes avec **sémaphore**

```
void philosophe (int i) // i est le numéro du philosophe
{
    while(1)
    {
        penser();
        Prendre_Fourchettes(i);
        manger();
        Poser_Fourchettes(i);
    }
}
```

```
void test (int i)
{
    if (state[i] == FAIM
        && state[GAUCHE] != MANGE
        && state[DROITE] != MANGE )
    {
        state[i] = MANGE;
        V(s[i]);
    }
}
```

Exercice 6



SE

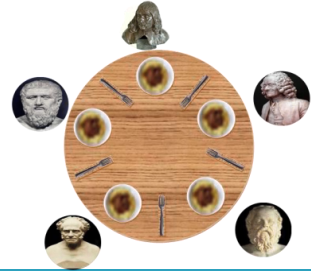
TD 5

3. Repas des philosophes avec Moniteur

```
monitor RepasPhilosophes
{
    int state[N];
    condition Phil[N];
Prendre_Fourchettes(int i)
{
    state[i] = FAIM;
    tester(i);
    if(state[i]!=MANGE)
        wait (Phil[i]);
}
Poser_Fourchettes(int i)
{
    state [i] = PENSE;
    tester(GAUCHE);
    tester(DROITE);
}
```

```
tester(int i)
{
    if (state [i] == FAIM)
        && state [GAUCHE] != MANGE
        && state[DROITE] != MANGE)
        {
            state[i] = MANGE;
            signal (Phil[i]);
        }
}
init()
{
    for i = 0 to 4
        state[i] = PENSE;
}
}
```

Exercice 6



SE

TD 5

3. Repas des philosophes avec **Moniteur**

```
void philosophe (int i) // i est le numéro du philosophe
{
    while(1)
    {
        penser();
        RepasPhilosophes.Prendre_Fourchettes(i);
        manger();
        RepasPhilosophes.Poser_Fourchettes(i);
    }
}
```

FIN

SE

Madame Khaoula ElBedoui-Maktouf
2^{ème} année Ingénieur Informatique