



Rapport TP Système

Exercice 02 : Producteurs/Consommateurs (Tableau géré comme pile)

BENDJOUDI Ilyes

ROUGAB Imene

M1 SII G01



Exercice 2 : Producteurs/Consommateurs

Un processus producteur et un processus consommateur se partagent l'accès à une zone mémoire permettant de stocker 10 valeurs entières que le producteur dépose et le consommateur récupère pour les afficher une à une.

1- 1 Producteur / 1 Consommateur

Considérons les deux processus suivants :

- P1 : producteur de valeurs
- P2 : consommateur de valeurs

Il s'agit de contrôler l'accès au tampon qui est un objet partagé.

Le Tampon est géré come une pile. L'ajout et le prélèvement se font au niveau du sommet.

On utilise deux indices :

- idxr : Pour l'écriture d'une valeur.
- idxl : Pour la lecture d'une valeur. Il indique aussi la dernière case écrite.

Le dépôt effectué par le producteur se traduit par `tab[idxr] = valeur ;`

Le prélèvement effectué par le consommateur se traduit par `valeur = tab[idxl] ;`

- Le producteur se bloque si la zone est pleine. Il est réveillé par le consommateur qui lit dans la zone et la libère.
- Le consommateur se bloque s'il n'y a aucune case écrite. Il est réveillé par le producteur lorsque il écrit (remplie la case)
- On a besoin de 3 sémaphores :
Sémaphore nvide initialisé à 10 ; //pour tester les cases vides
Sémaphore npleine initialisé à 0 ; //pour tester les cases pleines
Sémaphore binaire mutex initialisé à 1 ; //pour protéger idxl

Résultats : On lance le producteur et le consommateur en même temps dans 2 terminaux. Le consommateur se bloque quand on arrête la production et que toutes les valeurs ont déjà été consommées.

Producteur :

```
imene@ubuntu:~/Desktop/Prod-Consom$ ./producteur1.exe
Segment existe déjà : shmid = 1736715
J'écris la valeur 84 dans la case 0 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 58 dans la case 1 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 23 dans la case 1 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 23 dans la case 1 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 81 dans la case 1 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 94 dans la case 1 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 58 dans la case 1 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 59 dans la case 1 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 89 dans la case 1 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 20 dans la case 1 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 88 dans la case 1 .J'ai terminé. Prochaine case tab[idxr]= 0
^C
imene@ubuntu:~/Desktop/Prod-Consom$
```

Consommateur :

```
imene@ubuntu:~/Desktop/Prod-Consom$ ./consomateur.exe
Segment existe déjà : shmid = 1736715
Je consomme la valeur 58
Je consomme la valeur 23
Je consomme la valeur 23
Je consomme la valeur 81
Je consomme la valeur 94
Je consomme la valeur 58
Je consomme la valeur 59
Je consomme la valeur 89
Je consomme la valeur 20
Je consomme la valeur 88
Je consomme la valeur 84
```

2- N Producteurs / N Consommateurs

Pour ce cas il faut qu'il y ait, en plus de l'exclusion entre producteur et consommateur, une exclusion mutuelle entre les producteurs et les consommateurs.

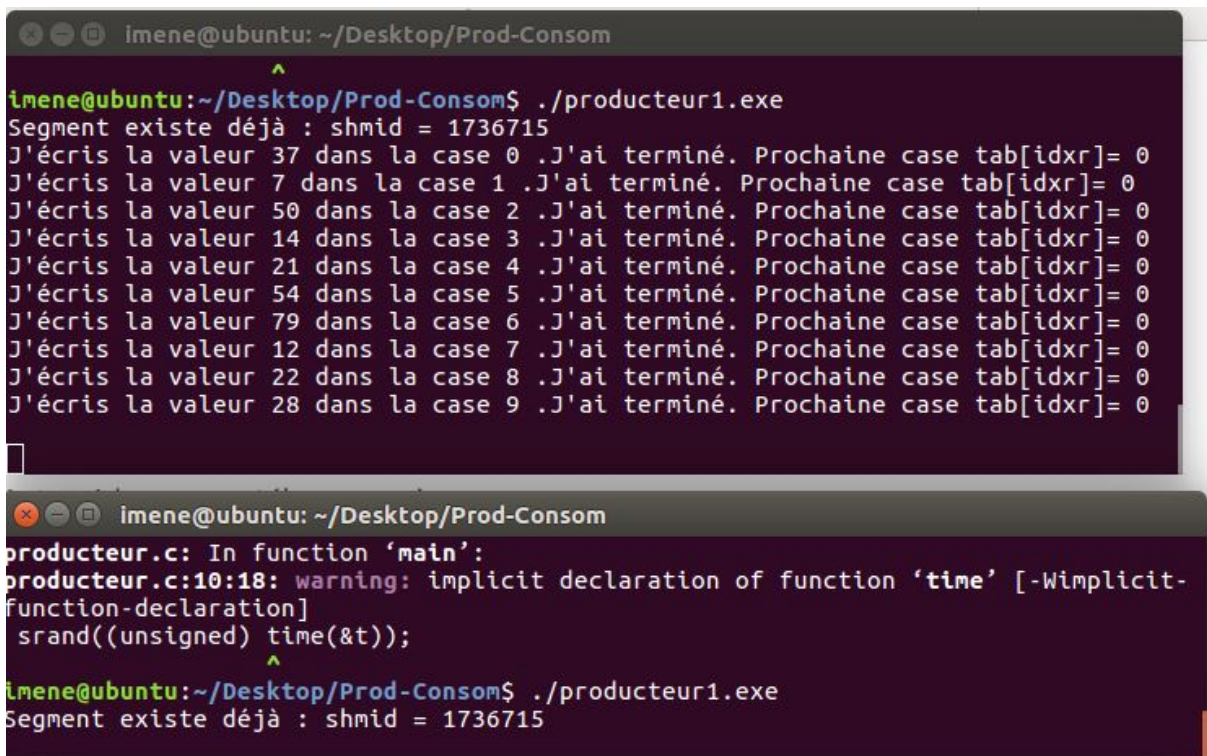
On va utiliser, en plus des sémaphores précédents, deux autres sémaphores binaires.

Sémaphore mutexp; //pour l'exclusion entre les producteurs

Sémaphore mutexc; //pour l'exclusion entre les consommateurs

Résultats :

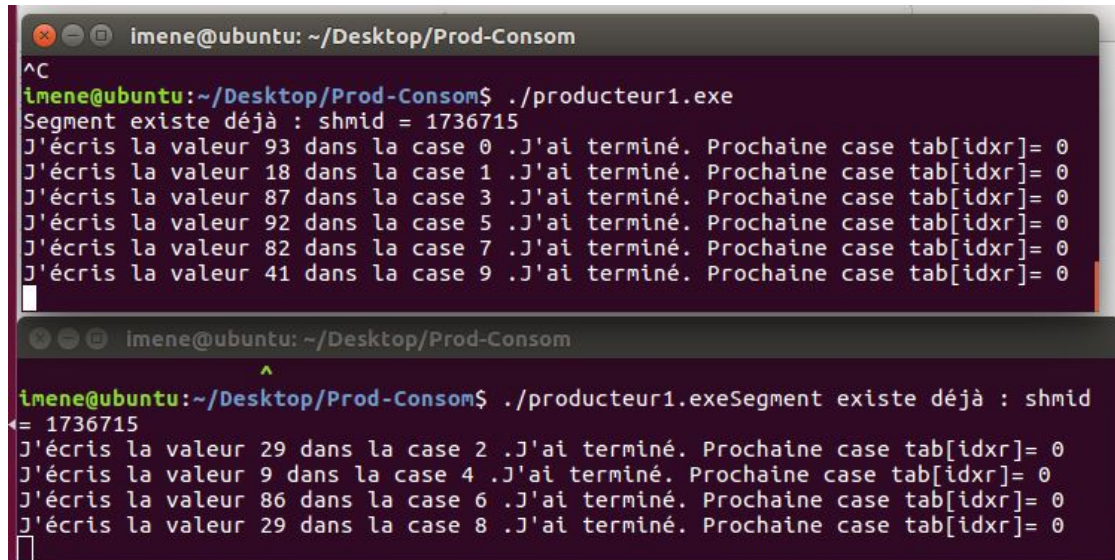
- On lance 2 producteurs pour tester l'exclusion mutuelle entre les producteurs. On ne va d'abord pas mettre de sleep().
On voit que le producteur 2 rester bloqué vu que toutes les cases ont été remplies par le producteur 1, et aucune n'a été consommée.



```
imene@ubuntu: ~/Desktop/Prod-Consom
^
imene@ubuntu:~/Desktop/Prod-Consom$ ./producteur1.exe
Segment existe déjà : shmid = 1736715
J'écris la valeur 37 dans la case 0 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 7 dans la case 1 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 50 dans la case 2 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 14 dans la case 3 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 21 dans la case 4 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 54 dans la case 5 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 79 dans la case 6 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 12 dans la case 7 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 22 dans la case 8 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 28 dans la case 9 .J'ai terminé. Prochaine case tab[idxr]= 0

producteur.c: In function 'main':
producteur.c:10:18: warning: implicit declaration of function 'time' [-Wimplicit-
function-declaration]
    srand((unsigned) time(&t));
                    ^
imene@ubuntu:~/Desktop/Prod-Consom$ ./producteur1.exe
Segment existe déjà : shmid = 1736715
```


On met un sleep dans la fonction producteur () afin d'afficher clairement que l'accès à une case ne peut pas être fait par deux producteurs. Ils accèdent l'un après l'autre et écrivent dans les cases séquentiellement.

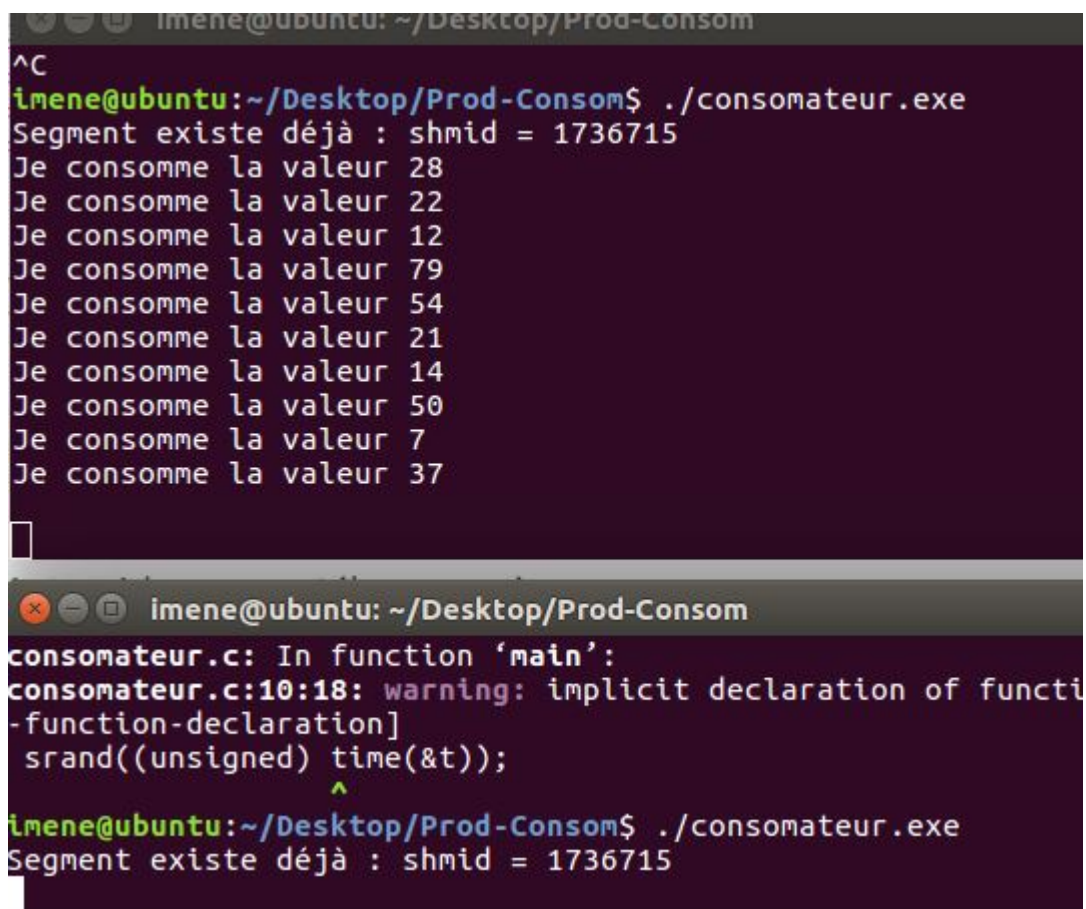


```
imene@ubuntu: ~/Desktop/Prod-Consom
^C
imene@ubuntu:~/Desktop/Prod-Consom$ ./producteur1.exe
Segment existe déjà : shmid = 1736715
J'écris la valeur 93 dans la case 0 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 18 dans la case 1 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 87 dans la case 3 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 92 dans la case 5 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 82 dans la case 7 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 41 dans la case 9 .J'ai terminé. Prochaine case tab[idxr]= 0

imene@ubuntu: ~/Desktop/Prod-Consom
^C
imene@ubuntu:~/Desktop/Prod-Consom$ ./producteur1.exeSegment existe déjà : shmid
= 1736715
J'écris la valeur 29 dans la case 2 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 9 dans la case 4 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 86 dans la case 6 .J'ai terminé. Prochaine case tab[idxr]= 0
J'écris la valeur 29 dans la case 8 .J'ai terminé. Prochaine case tab[idxr]= 0
```

- On lance 2 consommateurs en même temps :

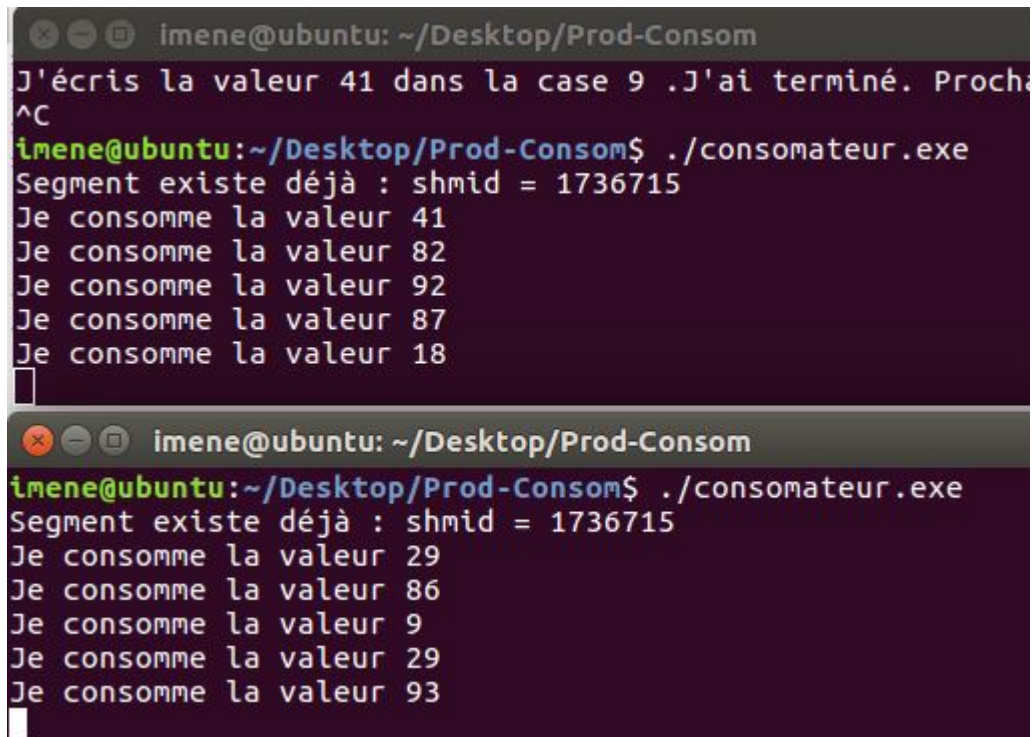
Sans utilisé de sleep, le consommateur 1 aura consommé toutes les valeurs avant qu'on ne lance le consommateur 2 :



```
imene@ubuntu: ~/Desktop/Prod-Consom
^C
imene@ubuntu:~/Desktop/Prod-Consom$ ./consomateur.exe
Segment existe déjà : shmid = 1736715
Je consomme la valeur 28
Je consomme la valeur 22
Je consomme la valeur 12
Je consomme la valeur 79
Je consomme la valeur 54
Je consomme la valeur 21
Je consomme la valeur 14
Je consomme la valeur 50
Je consomme la valeur 7
Je consomme la valeur 37

imene@ubuntu: ~/Desktop/Prod-Consom
consomateur.c: In function 'main':
consomateur.c:10:18: warning: implicit declaration of function
[-function-declaration]
srand((unsigned) time(&t));
^
imene@ubuntu:~/Desktop/Prod-Consom$ ./consomateur.exe
Segment existe déjà : shmid = 1736715
```

Utilisons désormais un `sleep()`, pour pouvoir lancer les 2 consommateurs et vérifier qu'ils ne peuvent pas consommer la valeur d'une même case.



```
imene@ubuntu: ~/Desktop/Prod-Consom
J'écris la valeur 41 dans la case 9 .J'ai terminé. Procha
^C
imene@ubuntu:~/Desktop/Prod-Consom$ ./consomateur.exe
Segment existe déjà : shmid = 1736715
Je consomme la valeur 41
Je consomme la valeur 82
Je consomme la valeur 92
Je consomme la valeur 87
Je consomme la valeur 18
█

imene@ubuntu: ~/Desktop/Prod-Consom
imene@ubuntu:~/Desktop/Prod-Consom$ ./consomateur.exe
Segment existe déjà : shmid = 1736715
Je consomme la valeur 29
Je consomme la valeur 86
Je consomme la valeur 9
Je consomme la valeur 29
Je consomme la valeur 93
█
```