



REPUBLIQUE DU BENIN

UNIVERSITE D'ABOMEY-CALAVI (UAC)

ECOLE POLYTECHNIQUE D'ABOMEY CALAVI (EPAC)

GÉNIE INFORMATIQUE ET TÉLÉCOMMUNICATIONS

(4e Année- RII 1)

Matière: Système d'Exploitation Avancé (SEA)

RAPPORT D'EXERCICE SUR LA PROGRAMMATION SYSTÈME

Réalisé par le groupe 3:

- 1- HOUENOU Emmanuel
- 2- TOBOU Charmel
- 3- YAYA NADJO Sènanmie

Sous la supervision du :

Dr Emery ASSOGBA

Année Académique 2021-2022

ENONCÉ

Soit deux classes de processus : les producteurs et les consommateurs.

Les producteurs produisent de l'information et la place dans un buffer partagé. Les consommateurs traitent l'information en allant la chercher dans le buffer partagé. Comment coordonner ces processus ?

Implémenter dans le langage de votre choix l'algorithme. Les producteurs produisent toutes les 2 secondes tandis que les consommateurs consomment toutes les 6 secondes.

Vous devez donner le choix à l'utilisateur de saisir au démarrage du programme le nombre de producteurs, le nombre de consommateurs et le nombre d'éléments maximum à produire (c'est-à-dire la taille de la mémoire partagée).

Donner la possibilité d'afficher toutes les secondes le contenu de la mémoire partagée.

PRODUCTION

Pour résoudre ce problème, le producteur doit soit se mettre en veille, soit supprimer les données si le tampon est plein. La prochaine fois que le consommateur retire un article du tampon, il en informe le producteur, qui recommence à remplir le tampon. De la même manière, le consommateur peut s'endormir s'il trouve que le buffer est vide. La prochaine fois que le producteur place des données dans le tampon, il réveille le consommateur endormi.

Implémentons donc cette solution en utilisant les notions de sémaphore et mutex.

Les sémaphores permettent de limiter l'accès concurrent à une section critique à un certain nombre de processus.

Le Mutex est une primitive de synchronisation utilisée pour éviter que des ressources partagées d'un système ne soient utilisées en même temps.

Dans notre proposition de code, nous avons utilisé trois sémaphores : un pour initialiser le mutex, un autre pour initialiser le nombre de slots (places) dans le buffer et un dernier pour initialiser le nombre de slots occupés.

Proposition de code

```
C producer_consumer.c X
C producer_consumer.c > producer(void *)
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <string.h>
5  #include <unistd.h>
6  #include <pthread.h>
7  #include <semaphore.h>
8
9
10 pthread_t *producers;
11 pthread_t *consumers;
12
13 sem_t mutex;
14 sem_t empty;
15 sem_t full;
16
17 int *buf, buf_pos=-1, prod_count, con_count, buf_len;
18
19 int produce(pthread_t self){
20     int i = 0;
21     int p = 1 + rand()%40;
22     while(!pthread_equal(*(producers+i),self) && i < prod_count){
23         i++;
24     }
25     printf("Producteur %d produit %d \n",i+1,p);
26     return p;
27 }
28
29
30 void consume(int p, pthread_t self){
31     int i = 0;
32     while(!pthread_equal(*(consumers+i),self) && i < con count){
```

```

30 void consume(int p, pthread_t self){
31     int i = 0;
32     while(!pthread_equal(*(consumers+i),self) && i < con_count){
33         i++;
34     }
35
36     printf("Buffer:");
37     for(i=0;i≤buf_pos;++i)
38         printf("%d ",*(buf+i));
39     printf("\nConsommateur %d consomme %d\nTaille actuelle du buffer : %d\n",i+1,p,buf_pos);
40
41 }
42
43
44 void* producer(void *args){
45
46     while(1){
47         int p = produce(pthread_self());
48         sem_wait(&empty);
49         sem_wait(&mutex);
50         ++buf_pos;           // Section critique
51         *(buf + buf_pos) = p;
52         sem_post(&mutex);
53         sem_post(&full);
54         sleep(1 + rand()%3);
55     }
56
57     return NULL;
58 }
59

```

```

61 void* consumer(void *args){
62     int c;
63     while(1){
64         sem_wait(&full);
65         sem_wait(&mutex);
66         c = *(buf+buf_pos);
67         consume(c, pthread_self());
68         --buf_pos;
69         sem_post(&mutex);
70         sem_post(&empty);
71         sleep(1+rand()%5);
72     }
73
74     return NULL;
75 }
76
77 int main(void){
78
79     int i,err;
80
81     srand(time(NULL));
82
83     sem_init(&mutex,0,1);
84     sem_init(&full,0,0);
85
86     printf("Entrez le nombre de producteurs : ");
87     scanf("%d",&prod_count);
88     producers = (pthread_t*) malloc(prod_count*sizeof(pthread_t));
89
90     printf("Entrez le nombre de consommateurs : ");
91     scanf("%d",&con_count);

```

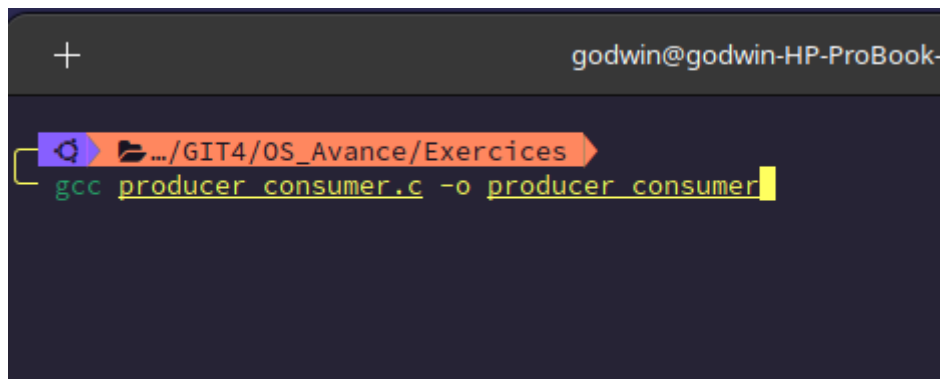
```

90     printf("Entrez le nombre de consommateurs : ");
91     scanf("%d",&con_count);
92     consumers = (pthread_t*) malloc(con_count*sizeof(pthread_t));
93
94     printf("Entrez le nombre d'éléments maximum à produire : ");
95     scanf("%d",&buf_len);
96     buf = (int*) malloc(buf_len*sizeof(int));
97
98     sem_init(&empty,0,buf_len);
99
100    for(i=0;i<prod_count;i++){
101        err = pthread_create(producers+i,NULL,&producer,NULL);
102        if(err != 0){
103            printf("Erreur lors de la création du producteur %d: %s\n",i+1,strerror(err));
104        }else{
105            printf("Producteur %d créé ... \n",i+1);
106        }
107    }
108
109    for(i=0;i<con_count;i++){
110        err = pthread_create(consumers+i,NULL,&consumer,NULL);
111        if(err != 0){
112            printf("Erreur lors de la création du consommateur %d: %s\n",i+1,strerror(err));
113        }else{
114            printf("Consommateur %d créé ... \n",i+1);
115        }
116    }
117

```

```
117
118     for(i=0;i<prod_count;i++){
119         pthread_join(*(producers+i),NULL);
120     }
121     for(i=0;i<con_count;i++){
122         pthread_join(*(consumers+i),NULL);
123     }
124
125     return 0;
126 }
127
```

Compilation



A terminal window with a dark background. The title bar shows a plus sign and the text "godwin@godwin-HP-ProBook-". The terminal shows a directory path ".../GIT4/OS_Avance/Exercices" and a command "gcc producer_consumer.c -o producer_consumer" being entered at the prompt.

```
+ godwin@godwin-HP-ProBook-
.../GIT4/OS_Avance/Exercices
gcc producer_consumer.c -o producer_consumer
```

Exécution

```
+ ./.producer_consumer
[~/GIT4/OS_Avance/Exercices] (anaconda3) godwin
./producer_consumer
Entrez le nombre de producteurs : 3
Entrez le nombre de consommateurs : 3
Entrez le nombre d'éléments maximum à produire : 5
Producteur 1 créé...
Producteur 1 produit 13
Producteur 2 créé...
Producteur 2 produit 25
Producteur 3 créé...
Producteur 3 produit 32
Consommateur 1 créé...
Buffer:13 25 32
Consommateur 4 consomme 32
Taille actuelle du buffer : 2
Consommateur 2 créé...
Buffer:13 25
Consommateur 3 consomme 25
Taille actuelle du buffer : 1
Consommateur 3 créé...
Buffer:13
Consommateur 2 consomme 13
Taille actuelle du buffer : 0
Producteur 2 produit 16
Producteur 3 produit 11
Buffer:16 11
Consommateur 3 consomme 11
Taille actuelle du buffer : 1
Buffer:16
Consommateur 2 consomme 16
Taille actuelle du buffer : 0
Producteur 1 produit 37
Producteur 2 produit 20
Buffer:37 20
Consommateur 3 consomme 20
Taille actuelle du buffer : 1
Producteur 3 produit 28
Buffer:37 28
Consommateur 3 consomme 28
Taille actuelle du buffer : 1
```