Rapport TP Système

Exercice 01 : Test de primalité

Exercice 02 : Producteurs/Consommateurs

BENDJOUDI Ilyes

ROUGAB Imene

M1 SII G01

Exercice 1 : Test de primalité

Dans cet exercice, on souhaite vérifier si un nombre « n » est premier. Pour ce fait, on va tester tous les diviseurs inférieurs à sa racine.
Comme cette méthode engendre beaucoup de calcules, on va utiliser le parallélisme afin de diminuer le temps d'exécution.
Il s'agit de créer « p » threads, dont chacun va calculer les diviseurs de « n » se trouvant dans un intervalle donc les bornes seront calculées par les formules suivantes :
J= 0, 1.....p-1 ;
Borne_inf=2+j $n^{1/2}/p$
Borne_sup=$2+j\ (n+1)^{1/2}/p$

1. Méthode sans parallélisme:

Code Source :

```
#include<stdio.h>
#include<unistd.h>
#include<math.h>
#define nb 18446743979220271
int main (void)
{
register unsigned long long sqrt_nb ;
register unsigned long long i ;
sqrt_nb = sqrt(nb) ;
for(i=2;i<sqrt_nb; i++)
if (!(nb % i))
{
printf(" %llu est un diviseur de %llu \n",i,nb) ;
return 0 ;
}
return 1 ;
}
```

On va tester le programme sur 2 nombres, l'un est premier et l'autre non.
N= 130147795189 un nombre premier de 12 chiffres.
N=130147795190 un nombre non premier de 12 chiffres

```
imene@ubuntu:~/Tps$ time ./primalite2
le Nombre 130147795189 est premier

real    0m0.008s
user    0m0.004s
sys     0m0.000s
```

Temps d'exécution = 0.008 s

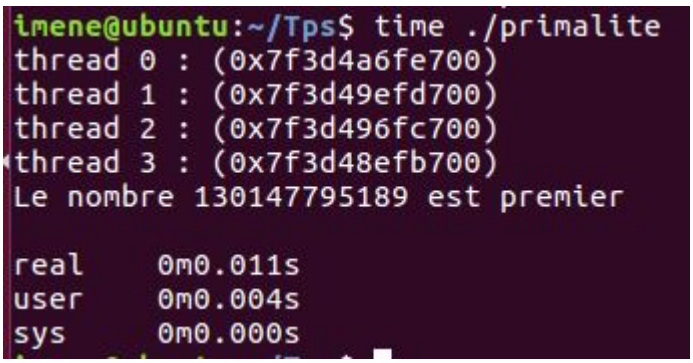
```
imene@ubuntu:~/Tps$ time ./primalite2
Le nombre 130147795190 N'est pas premier

real    0m0.001s
user    0m0.000s
sys     0m0.000s
```

Temps d'exécution = 0.001 s

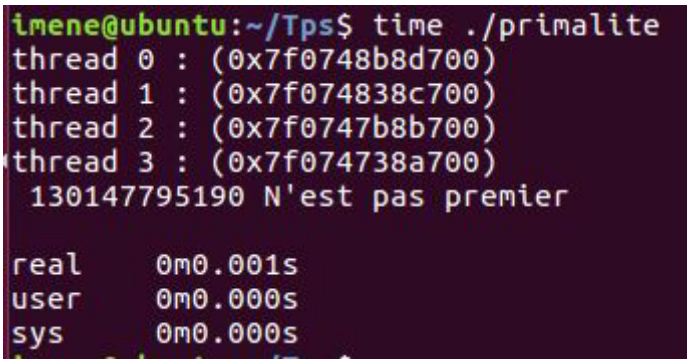
2. Méthode avec parallélisme:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <math.h>
#define nb 130147795190
int nb_thread; //Nombre de threads
pthread_t tid[10]; //tableau qui va contenir les threads créés

//Fonction de test de primalité
void *thread_function(void* arg) {
register unsigned long long sqrt_nb ;
register unsigned long long i ;
sqrt_nb = sqrt(nb) ;
int* b = (int *) arg ;
int j= *b;
unsigned long long debut =2+j*sqrt_nb/nb_thread ;
unsigned long long fin =2+(j+1)*sqrt_nb/nb_thread ;

for(i=debut;i<fin;i++)
 if (!(nb % i)){
        for(j=0;j<nb_thread;j++){
        if(tid[j] != pthread_self()) { pthread_cancel(tid[j]);}
      }
      pthread_exit( (void *)0); //si un diviseur est trouvé le thread retourne 0
 }
 pthread_exit( (void *)1); //si aucun diviseur n'est trouvé le thread retourne 1
}

int main(){
      int retour[10];
      int i;
      nb_thread= sysconf(_SC_NPROCESSORS_ONLN);
//Création des threads
      for (i = 0; i < nb_thread; ++i){
            pthread_create(&tid[i], NULL, &thread_function, (void *)&i);
            printf("thread %d : (%p) \n", i, (void*)tid[i]);
      }

// Jointures des threads et réccuperation de leurs valeurs de retour dans le
tableau retour[]
      for (i = 0; i < nb_thread; ++i){
            pthread_join(tid[i], (void **)&retour[i]);
      }

//Vérification des valeurs retournées par les threads
    for (i = 0; i < nb_thread; ++i){
            if(retour[i]==0) {printf("Le  nombre n est pas premier");
            exit(0);}
      }
    printf("Le nombre %lld est premier \n", nb );
    return 0;
}
```

Résultats de l'exécution du programme pour les mêmes nombres :



Temps d'exécution = 0.11s



Temps d'exécution = 0.01s

3. Comparaison entre les deux méthodes :

   Il est clair, d'après les résultats obtenus, que le temps est moindre quand on n'utilise pas le parallélisme.

   Pour N=130147795189 premier, on a :

   $$\begin{cases} \text{Temps d'exécution sans parallélisme} = 0.08\ s \\ \text{Temps d'exécution avec parallélisme} = 0.11s \end{cases}$$

   Ceci est dû au fait que la création des threads exige un temps qui n'est négligeable.

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.

La gestion du tampon se fait de manière circulaire :
Quand on arrive à la case N – 1 (case 9 dans notre cas) on repasse à la case 0 : les opérations de dépôt et de prélèvement se font modulo N.

On utilise deux indices :

- idxr : Pour le producteur (pour l'écriture d'une valeur)
- idxl : Pour le consommateur (pour la lecture d'une valeur)

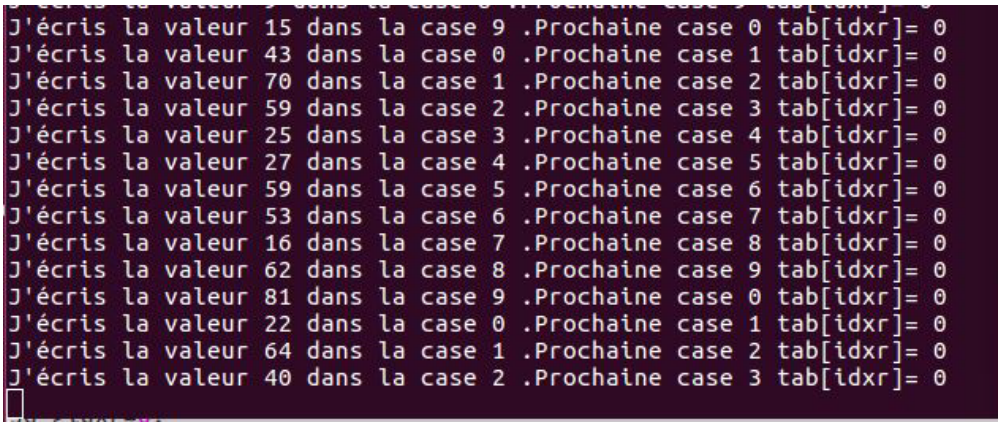
Le dépôt effectué par le producteur se traduit par tab[idxr] = valeur ;
Le prélèvement effectué par le consommateur se traduit par valeur = tab[idxl] ;

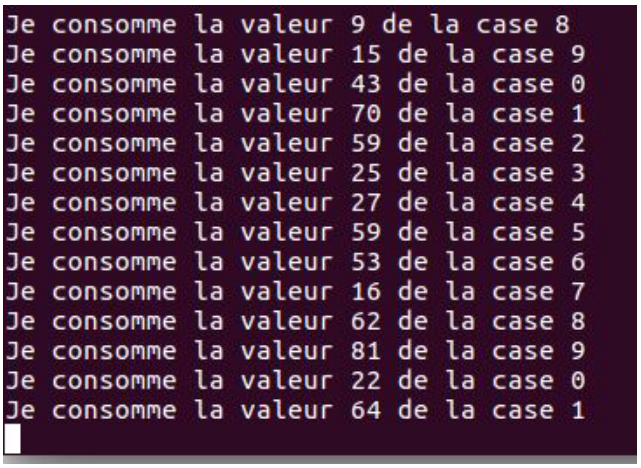
- Pour le producteur, le dépôt d'une valeur est possible s'il existe au moins une case vide. Il doit donc commencer par attendre une case vide.
- Pour le consommateur, le prélèvement d'une valeur est possible s'il existe au moins une case pleine. Il doit donc commencer par attendre une case pleine.

Au départ, toutes les cases sont vides et par conséquent, nous n'avons pas de cases pleines.
Sémaphore nvide = 10 ;
Sémaphore npleine = 0 ;
Tableau de valeurs tab[10] ;
Int idxl, idxr = 0 ;

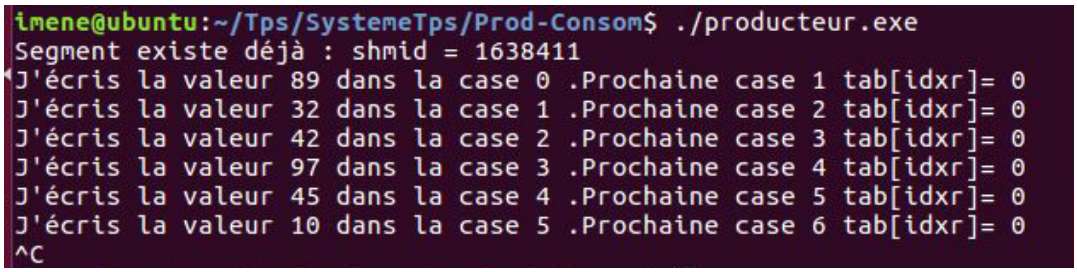
Résultats :

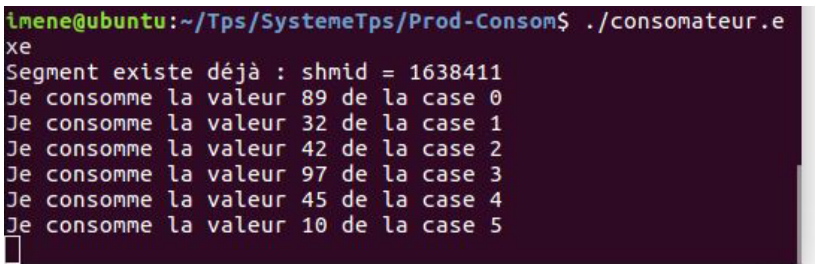
On voit bien que le producteur écrit dans les 10 cases et reprend à la case 1 quand toutes les cases sont pleines (sachant que le consommateur a consommé les valeurs produites, les cases sont donc vidées avant que le producteur ne revienne à la case 0).





Si on arrête le producteur, le consommateur se bloque quand toutes les cases seront vides.





2- N Producteurs / N Consommateurs

Pour ce cas on peut avoir les situations suivantes :

- 2 producteurs testent en même temps nvide.
  Ils passent tous les deux (le nombre des cases vides est supérieur à 1)

  - Le premier exécute tab[idxr] = valeur ;
  - Il est interrompu avant qu'il puisse exécuter l'instruction idxr = (idxr mod N)+1
  - le deuxième exécute tab[idxr] = valeur
  - Il écrase ainsi la valeur déposée par le producteur précédent parce qu'il n'a pas eu le temps de pointer la case libre suivante.

- Une situation similaire peut se produire pour les consommateurs conduisant à la consommation et l'utilisation d'un même message par plusieurs consommateurs.

Pour résoudre ce problème, il faut interdire à plusieurs producteurs d'accéder à une même case en même temps. De même, il faut interdire à plusieurs consommateurs d'accéder à une même case en même temps. En d'autres termes, il faut que l'accès à une case se fasse en exclusion mutuelle entre les producteurs lors du dépôt, et entre les consommateurs lors du prélèvement.

Sémaphore nvide = 10 ;
Sémaphore npleine = 0 ;
Sémaphore Mutexp = 1 ; //Pour l'exclusion mutuelle entre les producteurs
Sémaphore Mutexc = 1 ; //Pour l'exclusion mutuelle entre les consommateurs
Tableau de valeurs tab[10] ;
Int idxr, idxl = 0 ;

Résultats :

```
imene@ubuntu:~/Tps/SystemeTps/Prod-Consom$ ./producteur.exe
Segment existe déjà : shmid = 1638411

Je suis le producteur 1 :J'écris la valeur 66 dans la case 0 .Prochaine case 1 tab[idxr]= 0

Je suis le producteur 2 :J'écris la valeur 99 dans la case 1 .Prochaine case 2 tab[idxr]= 0

Je suis le producteur 3 :J'écris la valeur 65 dans la case 2 .Prochaine case 3 tab[idxr]= 0

Je suis le producteur 4 :J'écris la valeur 50 dans la case 3 .Prochaine case 4 tab[idxr]= 0

Je suis le producteur 5 :J'écris la valeur 78 dans la case 4 .Prochaine case 5 tab[idxr]= 54

Je suis le producteur 6 :J'écris la valeur 31 dans la case 5 .Prochaine case 6 tab[idxr]= 43

Je suis le producteur 7 :J'écris la valeur 87 dans la case 6 .Prochaine case 7 tab[idxr]= 76
^C
imene@ubuntu:~/Tps/SystemeTps/Prod-Consom$
```

```
imene@ubuntu:~/Tps/SystemeTps/Prod-Consom$ ./consomateur.exe
Segment existe déjà : shmid = 1638411

Je suis le consommateur 0 :Je consomme la valeur 66 de la case 0

Je suis le consommateur 1 :Je consomme la valeur 99 de la case 1

Je suis le consommateur 2 :Je consomme la valeur 65 de la case 2

Je suis le consommateur 3 :Je consomme la valeur 50 de la case 3

Je suis le consommateur 4 :Je consomme la valeur 78 de la case 4

Je suis le consommateur 5 :Je consomme la valeur 31 de la case 5

Je suis le consommateur 6 :Je consomme la valeur 87 de la case 6

Je suis le consommateur 7 :Je consomme la valeur 76 de la case 7
```

Là aussi, si on arrête la production et que toutes les cases sont consommées, les consommateurs se bloquent.