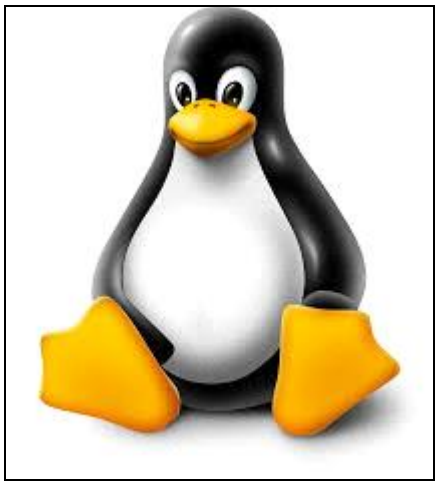
جامعة سيدي محمد بن عبدالله
كلية العلوم ظهر المهراز
Université Sidi Mohamed Ben Abdellah
Faculté des Sciences Dhar Mahraz

Compte Rendu De Travaux Pratique

System d'Exploitation

Travaille Pratique 6 : Les Tubes



SMI – S4

Réaliser par :

Prénom :	**CNE :**	**Nom :**
Achraf	1513755449	Tazi
Oussama	1311778906	Nadrani

Exercice1 :

Ecrire un programme qui crée deux threads dont chacune appelle la même fonction nommée 'doSomeThing'.Cette dernière imprime au début de son exécution un message du commencement de l'exécution ainsi que la valeur de la variable 'counter' qui correspond à un compteur. Après cette impression, la fonction effectue un certain traitement et imprime la fin de son exécution par un message accompagné de la valeur du 'counter'. Les deux fonctions partagent la variable 'counter'. Cette variable s'incrémente chaque fois que 'doSomeThing' est appelé par l'un ou l'autre des deux threads. Que remarquez vous ?

```
#include<unistd.h>
#include<string.h>
#include<pthread.h>

pthread_t tid[2];
int counter;
void *doSomeThing(void *arg){
        unsigned long i;
        counter += 1;
          printf("\n Job %d started \n", counter);//deb
        for(i=0;i<100;i++) printf("\nle job %d est entrain de
s'executer",counter);//execution
          printf("\n\n Job %d Finihed \n", counter);//fin
                return NULL;
}

int main(void){
        int i=0;
        int pid1,pid2;
        pid1 = pthread_create(&tid[0    ],NULL,&doSomeThing,NULL);
    pid2 = pthread_create(&tid[1],NULL,&doSomeThing,NULL);
        if(pid1 != 0 || pid2 != 0)
                printf("\n can't create thread ");
        pthread_join(tid[0],NULL);
        pthread_join(tid[1],NULL);
        return 0;
}
```

Résultat :

```
nadrani@nadrani-pc:~/sys_tp/TpSys8$ ./Tp8_1

 Job 1 started

le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
```

Remarque :

On remarque que la variable globale **counter** a était incrémentais avant la fin du premier thread

```
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
```

Exercice2 :

Modifier le programme de l'exercice 1 en introduisant les fonctions de synchronisation les mutex afin d'afficher les bons résultats.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<pthread.h>

pthread_t tid[2];
int counter;
pthread_mutex_t lock;
void *doSomeThing(void *arg){
        pthread_mutex_lock(&lock);
        unsigned long i;
        counter += 1;
         printf("\n Job %d started \n", counter);//deb
         for(i=0;i<100;i++)
          printf("\nle job %d est entrain de s'executer",counter);//execution
         printf("\n\n Job %d Finihed \n", counter);//fin
        pthread_mutex_unlock(&lock);
                return NULL;
}

int main(void){
        int i=0;
        int pid1,pid2;
        if(pthread_mutex_init(&lock,NULL) != 0)
        {
                printf("\n mutex init failed \n");
                return 1;
        }
        pid1 = pthread_create(&tid[1],NULL,&doSomeThing,NULL);
    pid2 = pthread_create(&tid[2],NULL,&doSomeThing,NULL);
        if(pid1 != 0 || pid2 != 0)
                printf("\n can't create thread ");
        pthread_join(tid[0],NULL);
        pthread_join(tid[1],NULL);
        pthread_mutex_destroy(&lock);
        return 0;
}
```

Résultat :

```
nadrani@nadrani-pc:~/sys_tp/TpSys8$ gcc -pthread ex2.c -o TP8_2
nadrani@nadrani-pc:~/sys_tp/TpSys8$ ./TP8_2

 Job 1 started

le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
```

```
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer
le job 1 est entrain de s'executer

 Job 1 Finihed

 Job 2 started

le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
```

```
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer
le job 2 est entrain de s'executer

 Job 2 Finihed
```

Exercice3 :

En utilisant les fonctions de synchronisation de mutex, écrire un programme C qui crée deux threads qui appellent deux fonctions (a) 'write_tab_process' et (b) 'read_tab_process'. La 1ère fonction (a) remplie un tableau tab de 10 éléments dont chaque case contient le double de son indice (2*i). La 2ème fonction (b) affiche le contenu de tab sur la sortie standard. Il est à préciser que les deux threads partagent entre elles le tableau tab.

```c
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
int tab[10];
pthread_mutex_t lock;
void *read_tab_process(void *args){//fonction lire
   int i;
   pthread_mutex_lock(&lock);
   for(i=0; i <10; i++){
      printf("Read procces [%d] = %d \n",i,tab[i]);
   }
   pthread_mutex_unlock(&lock);
   pthread_exit(NULL);
}
void *write_tab_process(void *args){//fonction ecrire
   pthread_mutex_lock(&lock);
   int i;
   for(i = 0 ; i < 10 ; i++){
      tab[i] = i * 2;
      printf("Write procces %d\n",i);
      sleep(1);
   }
   pthread_mutex_unlock(&lock);
   pthread_exit(NULL);
}
int main(){
  int err;
  pthread_t write;
  pthread_t read;
  pthread_mutex_init(&lock,NULL);
  err = pthread_create(&read,NULL,read_tab_process,NULL);
  if(err < 0){
     printf("Error create read thread \n");
  }
  err = pthread_create(&write,NULL,write_tab_process,NULL);
  if(err < 0){
     printf("Error create write thread \n");
  }
  pthread_join(write,NULL);
  pthread_join(read,NULL);
  pthread_mutex_destroy(&lock);
  return 0;
}
```

Résultat :

```
nadrani@nadrani-pc:~/sys_tp/TpSys8$ gcc -pthread ex3.c -o TP8_3
nadrani@nadrani-pc:~/sys_tp/TpSys8$ ./TP8_3
Write procces 0
Write procces 1
Write procces 2
Write procces 3
Write procces 4
Write procces 5
Write procces 6
Write procces 7
Write procces 8
Write procces 9
Read procces [0] = 0
Read procces [1] = 2
Read procces [2] = 4
Read procces [3] = 6
Read procces [4] = 8
Read procces [5] = 10
Read procces [6] = 12
Read procces [7] = 14
Read procces [8] = 16
Read procces [9] = 18
```