

Correction TD2

Programmation Système et Réseaux

Exercice 1

Ecrire un programme C qui permet de créer deux threads : le premier thread lit une chaîne de caractères au clavier et la passe à un deuxième thread qui se charge de l'afficher. Le thread principal (le père) se charge de la création de ses deux threads et de l'attente de leur terminaison.

Exercice 1

```
void *lecture (void * ch)
{   printf ("donner votre chaine \n");
    scanf ("%s", (char*) ch) ;
    pthread_exit (0);
}

void *ecriture (void * ch)
{   printf ("votre chaine est : %s \n", (char*) ch);
    pthread_exit (0);
}

void main (void)
{pthread_t th1, th2;
  void *ret;
  char ch[20];
  pthread_create (&th1, NULL, lecture, (void*) ch) ;
  pthread_join (th1, &ret);
  pthread_create (&th2, NULL, ecriture, (void*) ch) ;
  pthread_join (th2, &ret);
}
```

Exercice 1

```
void *lecture (void * ch)
{   printf ("donner votre chaine \n");
    scanf ("%s", (char*) ch) ;
    pthread_exit (0);
}

void *ecriture (void * ch)
{   printf ("votre chaine est : %s \n", (char*) ch);
    pthread_exit (0);
}

void main (void)
{pthread_t th1, th2;
  void *ret;
  char ch[20];
  pthread_create (&th1, NULL, lecture, (void*) ch) ;
  pthread_join (th1, &ret);
  pthread_create (&th2, NULL, ecriture, (void*) ch) ;
  pthread_join (th2, &ret);
}
```

// ajouter return ch;

// ret au lieu de ch

Exercice 2

Ecrire un programme C qui permet d'exécuter deux threads d'une même fonction: le premier thread affiche 5 fois le caractère 1 et le deuxième affiche 5 fois le caractère 2.

Exercice 2

```
void *fonction_affiche (void * arg)
{ int i;
  for (i = 0 ; i < 5 ; i++) {
    printf (" Le caractère est  %s: itération %d\n", (char*)arg, i);
    sleep (1);  }
  pthread_exit (0);}

void main (void)
{ pthread_t th1, th2;
  void *ret;
  if (pthread_create (&th1, NULL, fonction_affiche, "1") < 0)
    {      fprintf (stderr, "erreur de creation de thread 1\n");      exit (1);
      }
  if (pthread_create (&th2, NULL, fonction_affiche, "2") < 0)
    {      fprintf (stderr, " erreur de creation de thread 2\n");      exit (1);
      }

  pthread_join (th1, &ret);
  pthread_join (th2, &ret);
}
```

Exercice 3

Ecrire un programme C qui permet une transmission d'un flot de caractères à tour de rôle entre deux threads : le premier thread lit un caractère et donne la main à un deuxième thread pour l'afficher ; ensuite le deuxième thread donne la main au premier pour saisir un autre caractère et ainsi de suite. Cette transmission est effectuée d'une manière répétitive tant que le premier thread ne transmet pas la lettre F (pour dire Fin).

Exercice 3

```
volatile char LeCar = '\0';
volatile int  tour = 0;
void* lecture( )
{ do {
    while (tour == 1);           /* attendre mon tour */
    scanf("%c" , &LeCar);
    tour = 1;                    /* donner le tour à l'autre*/
  } while (LeCar != 'F');
  return NULL;
}

void* affichage ( )
{ int cpt = 0;
  do {
    while (tour == 0);           /* attendre mon tour */
    cpt ++;
    printf("Compteur = %d, caractère = %c \n", cpt, LeCar);
    tour = 0;                    /* donner le tour à l'autre*/
  } while (LeCar != 'F');
  return NULL; }
```


Exercise 3

```
int main (void)
{
    pthread_t thA, thB;
    if (pthread_create(&thA, NULL, affichage, NULL)) {
        perror("pthread_create");
        exit(EXIT_FAILURE);    }

    if (pthread_create(&thB, NULL, lecture, NULL)) {
        perror("pthread_create");
        exit(EXIT_FAILURE);    }

    if (pthread_join(thA, NULL))
        perror("pthread_join");

    if (pthread_join(thB, NULL))
        perror("pthread_join");

    printf("Fin du père\n") ;
    return (EXIT_SUCCESS);
}
```

Exercice 4

Ecrire un programme C qui permet de créer trois threads :

- **le premier thread saisit le contenu d'un tableau d'entiers**
- **le deuxième thread trie le tableau**
- **et le troisième calcule la moyenne du tableau**

Exercice 4

```
#define N 20;
int Tab [N];
void *Saisie ( )
{
    int i;
    for (i =0; i<N; i++)    scanf("%d" , &Tab [i]);
    pthread_exit (0);
}
void *Tri ( )
{
    /*n'importe quelle méthode de tri*/
    pthread_exit (0);
}
void *Moyenne(void* Moy )
{
    int i;
    for (i =0; i<N; i++)    *(float*)Moy+=Tab [i]/N);
    return (Moy);
}
void main (void)
{
    pthread_t th1, th2, th3;
    void *ret;
    float Moy;
    pthread_create (&th1, NULL, Saisie, NULL) ;

    pthread_create (&th2, NULL, Tri, NULL) ;
    pthread_create (&th3, NULL, Moyenne, (void*) Moy) ;
}
```

Exercice 4

```
#define N 20;
int Tab [N];
void *Saisie ( )
{
    int i;
    for (i =0; i<N; i++)    scanf("%d" , &Tab [i]);
    pthread_exit (0);
}
void *Tri ( )
{
    /*n'importe quelle méthode de tri*/
    pthread_exit (0);
}
void *Moyenne(void* Moy )
{
    int i;
    for (i =0; i<N; i++)    *(float*)Moy+=Tab [i]/N);
    return (Moy);
}
void main (void)
{
    pthread_t th1, th2, th3;
    void *ret;
    float Moy;
    pthread_create (&th1, NULL, Saisie, NULL) ;
    pthread_join (th1, &ret);
    pthread_create (&th2, NULL, Tri, NULL) ;
    pthread_create (&th3, NULL, Moyenne, (void*) Moy) ;
    pthread_join (th2, &ret);
    pthread_join (th3, &ret);
}
```

Exercice 4

```
#define N 20;
int Tab [N];
void *Saisie ( )
{
    int i;
    for (i =0; i<N; i++)    scanf("%d" , &Tab [i]);
    pthread_exit (0);
}
void *Tri ( )
{
    pthread_mutex_lock(&mutex); /*n'importe quelle méthode de tri*/ pthread_mutex_unlock(&mutex);
    pthread_exit (0);
}
void *Moyenne(void* Moy )
{
    int i;
    pthread_mutex_lock(&mutex); for (i =0; i<N; i++)    *(float*)Moy+=Tab [i]/N); pthread_mutex_unlock(&mutex);
    return (Moy);
}
void main (void)
{
    pthread_t th1, th2, th3;
    void *ret;
    float Moy;
    pthread_create (&th1, NULL, Saisie, NULL) ;
    pthread_join (th1, &ret);
    pthread_create (&th2, NULL, Tri, NULL) ;
    pthread_create (&th3, NULL, Moyenne, (void*) Moy) ;
    pthread_join (th2, &ret);
    pthread_join (th3, &ret);
}
```

Exercice 5

Ecrire le programme C qui permet de simuler le fonctionnement de lecteurs/rédacteur et qui permet de synchroniser l'accès de plusieurs lecteurs et un rédacteur à une variable partagée. (Base de données)

Plusieurs lecteurs peuvent accéder en même temps à la base.



À un instant donné, un seul écrivain peut exister seul dans la base.



Base de données

Mme K. ELBedoui-Maktouf

Exercice 5

Lecteur/rédacteur

```
void Lecteur ()  
{ while(1)  
  {
```

```
    lire_BD();
```

```
    traitement();
```

```
  }
```

```
}
```



```
void Ecrivain ()  
{
```

```
  while(1)  
  {
```

```
    creerDonnees()
```

```
    ecrire_BD();
```

```
  }
```

```
}
```

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

Exercice 5

Lecteur/rédacteur

```
void Lecteur ()  
{ while(1)  
  {
```

P(db);

lire_BD();

V(db);

traitement();

```
void Ecrivain ()  
{ while(1)  
  {
```

creerDonnees()

P(db);

ecrire_BD();

V(db);

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

Exercice 5

Lecteur/rédacteur

```
void Lecteur ()
{
    while(1)
    {
        NbL = NbL + 1;
        if (NbL == 1) P(db);

        lire_BD();

        NbL = NbL - 1;
        if (NbL == 0) V(db);

        traitement();
    }
}
```



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




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


Exercice 5

Lecteur/rédacteur

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



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

Exercice 5

```
#include <semaphore.h>
#include <pthread.h>
#define N_L 10
#define N_E 2
sem_t MUTEX, MUTEX_DB;

int main(void)
{
    int i;
    int NbL = 0;
    pthread_t lecteur, ecrivain;
    sem_init(&MUTEX, 0, 1);
    sem_init(&MUTEX_DB, 0, 1);

    for (i = 0; i < N_L; i++)
        pthread_create(&lecteur, NULL, lecture, NULL);

    for (i = 0; i < N_E; i++)
        pthread_create(&ecrivain, NULL, ecriture, NULL);

    pthread_join(lecteur, NULL);
    pthread_join(ecrivain, NULL);
    sem_destroy(&MUTEX);
    sem_destroy(&MUTEX_DB);
    exit(0);
}
```

Exercise 5

```
void *lecture( )
{
    sem_wait(MUTEX);
    NbL++;
    if (NbL == 1) sem_wait(MUTEX_DB);
    sem_post(MUTEX);

    lire_BD();

    sem_wait(MUTEX);
    NbL--;
    if (NbL == 0) sem_post(MUTEX_DB);
    sem_post(MUTEX);

    pthread_exit (0);
}
```

```
void *ecriture( )
{
    sem_wait(MUTEX_DB);

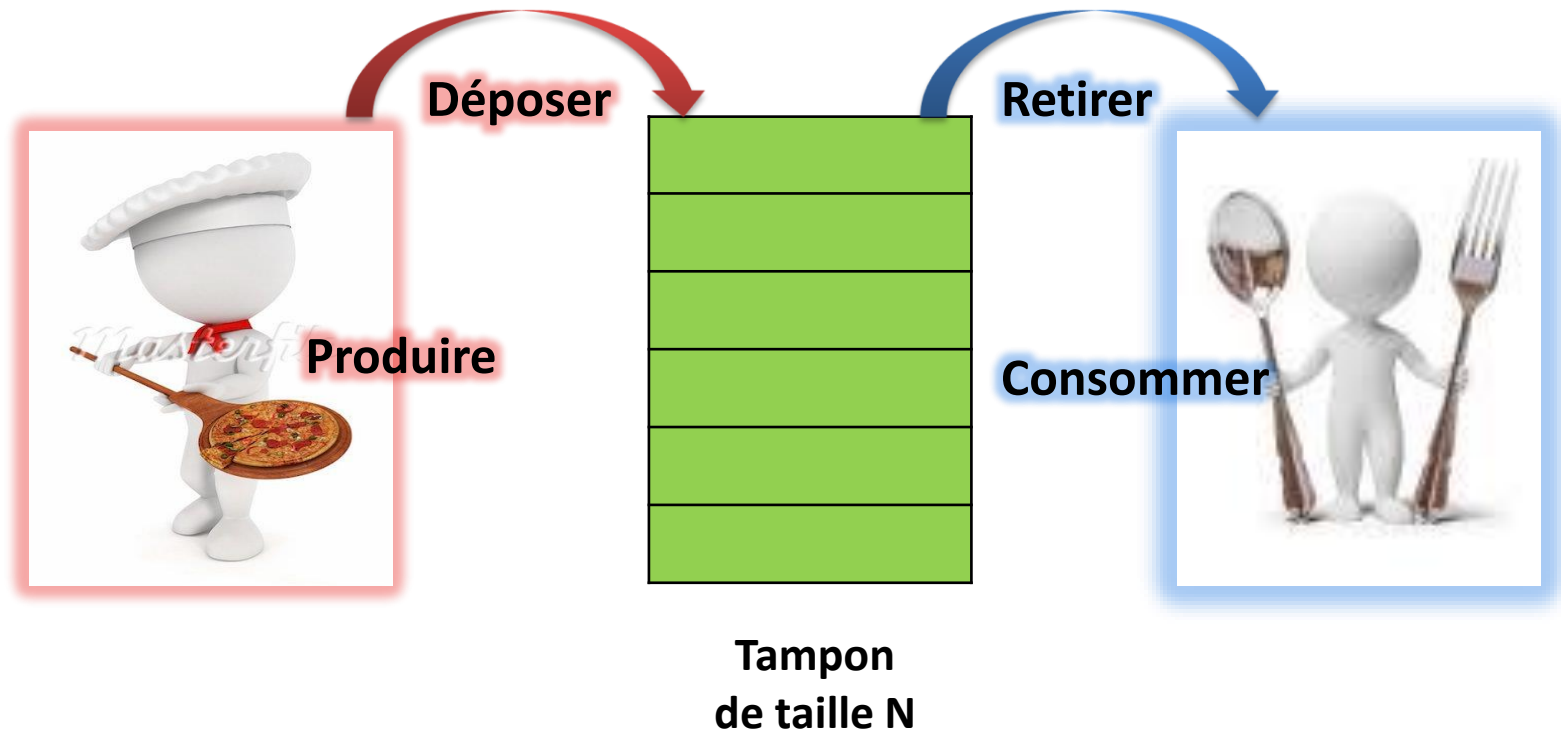
    ecrire_BD();

    sem_post(MUTEX_DB);

    pthread_exit (0);
}
```

Exercice 6

Ecrire le programme C qui permet de simuler le fonctionnement de Producteurs/consommateurs qui permet de synchroniser l'accès de plusieurs producteurs et plusieurs consommateurs à un tampon de taille limitée.



Exercice 6

Producteur/Consommateur

define N 100 // taille du tampon

semaphore mutex = 1; // assure l'exclusion mutuelle sur le tampon (deposer/retirer)

semaphore vide = N; // nombre de places vides

semaphore plein = 0; // nombre de places pleines

void producteur ()

{

int objet;

while (1)

{

Produire(&objet);

Deposer (objet);

}

}

void consommateur ()

{

int objet;

while (1)

{

Retirer (objet);

Consommer(&objet);

}

}

Exercice 6

Producteur/Consommateur

define N 100 // taille du tampon

semaphore mutex = 1; // assure l'exclusion mutuelle sur le tampon (deposer/retirer)

semaphore vide = N; // nombre de places vides

semaphore plein = 0; // nombre de places pleines

void producteur ()

{

int objet;

while (1)

{

Produire(&objet);

P(mutex);

Deposer (objet);

V(mutex);

}

}

void consommateur ()

{

int objet;

while (1)

{

P(mutex);

Retirer (objet);

V(mutex);

Consommer(&objet);

}

}

Exercice 6

Producteur/Consommateur

define N 100 // taille du tampon

semaphore mutex = 1; // assure l'exclusion mutuelle sur le tampon (deposer/retirer)

semaphore vide = N; // nombre de places vides

semaphore plein = 0; // nombre de places pleines

void producteur ()

{

int objet;

while (1)

{

Produire(&objet);

P(vide);

P(mutex);

Deposer (objet);

V(mutex);

}

}

void consommateur ()

{

int objet;

while (1)

{

P(mutex);

Retirer (objet);

V(mutex);

V(vide);

Consommer(&objet);

}

}

Exercice 6

Producteur/Consommateur

define N 100 // taille du tampon

semaphore mutex = 1; // assure l'exclusion mutuelle sur le tampon (deposer/retirer)

semaphore vide = N; // nombre de places vides

semaphore plein = 0; // nombre de places pleines

void producteur ()

{

int objet;

while (1)

{

Produire(&objet);

P(vide);

P(mutex);

Deposer (objet);

V(mutex);

V(plein);

}

}

void consommateur ()

{

int objet;

while (1)

{

P(plein);

P(mutex);

Retirer (objet);

V(mutex);

V(vide);

Consommer(&objet);

}

}

Exercice 6

```
#include <semaphore.h>
#include <pthread.h>
#define N_P 10
#define N_C 10
#define N 50 /* taille tampon */
sem_t vide, plein, MUTEX;
int main(void)
{
    int i;
    pthread_t producteur, consommateur;
    sem_init(&vide, 0, N);
    sem_init(&plein, 0, 0);
    sem_init(&MUTEX, 0, 1);
    for (i = 0; i < N_P; i++)
        pthread_create(&producteur, NULL, production, NULL);
    for (i = 0; i < N_C; i++)
        pthread_create(&consommateur, NULL, consommation, NULL);
    pthread_join(producteur, NULL);
    pthread_join(consommateur, NULL);
    sem_destroy(&vide);
    sem_destroy(&plein);
    sem_destroy(&MUTEX);
    exit(0);
}
```

Exercise 6

```
void *production(void *bidon)
{
    int objet;
    for (;;)
    {
        Produire(&objet);
        sem_wait(&vide);

        sem_wait(&MUTEX);
        Deposer(objet);
        sem_post(&MUTEX);

        sem_post(&plein);
    }
    return NULL;
}
```

```
void *consommation(void *bidon)
{
    int objet;
    for (;;)
    {
        sem_wait(&plein);

        sem_wait(&MUTEX);
        Retirer(objet);
        sem_post(&MUTEX);

        sem_post(&vide);
        Consommer(&objet);
    }
    return NULL;
}
```

Exercice 7

Ecrire le programme C qui permet de gérer un compte bancaire et ce via deux fonctions :

- **la fonction débiter qui permet de retirer une somme d'argent**
- **et la fonction créditer qui permet de verser une somme d'argent.**

Exercice 7

```
int ValCompte =100;
void *Credit (void * C)
{
    pthread_mutex_lock(&mutex);
    ValCompte += * ((int*) C) ;
    pthread_mutex_unlock(&mutex);
    pthread_exit (0);
}

void *Debit (void * D)
{
    pthread_mutex_lock(&mutex);
    if (ValCompte <*(int*)D)        printf("IMPOSSIBLE");
    else                            ValCompte - = * ((int*) D) ;
    pthread_mutex_unlock(&mutex);
    pthread_exit (0);
}

void main (void)
{ pthread_t th1, th2;
  int * C, *D;
  void *ret;
  scanf("%d%d", C, D);
  pthread_create (&th1, NULL, Credit, (void*) C) ;
  pthread_create (&th2, NULL, Debit, (void*) D) ;
  pthread_join (th1, &ret);
  pthread_join (th2, &ret);
}
```