


2009 - 2010

A set of four horizontal lines, with the second line from the top being a darker teal color and the others being a lighter teal color.

# Systemes d'exploitation 2

*TD n 3 : Applications classiques*

A set of four horizontal lines, with the second line from the top being a darker teal color and the others being a lighter teal color.

Mme Lilia SFAXI

# Systèmes d'exploitation 2

## TD n 3 : Applications classiques

### Exercice 1 : Les Barrières ---

#### Contexte commun

```
sémaphore mutex = 1, s = 0;
```

```
entier NbArrivés = 0;           /*nbre de processus arrivés au rendez-vous*/
```

#### Procédure RDV

Début

```
    down(mutex);
```

```
    NbArrivés = NbArrivés + 1;
```

```
    Si (NbArrivés < N) Alors           /* non tous arrivés */
```

```
        up(mutex);                     /* on libère mutex et */
```

```
        down(s);                       /* on se bloque */
```

Sinon

```
        up(mutex);                     /* le dernier arrivé libère mutex et */
```

```
        Pour i = 1 à N-1 Faire
```

```
            up(s);                     /* réveille les N-1 bloqués */
```

Finsi

Fin

## Exercice 2 : Le dîner des philosophes

---

Première non-solution :

```
# define N 5                                /* nombre de philosophes */
philosophe(int i) {                          /* i : numéro du philosophe */
    while (TRUE) {
        penser( ) ;
        prendre_fourchette(i) ;             /* prend la fourchette gauche */
        prendre_fourchette( (i+1)%N) ;      /* prend la fourchette droite */
        manger( ) ;
        poser_fourchette( i ) ;             /* repose la fourchette gauche */
        poser_fourchette((i+1)%N) ;         /* repose la fourchette droite */
    }
}
```

## Deuxième solution :

```
# define N 5 /* nombre de philosophes */
# define GAUCHE (i-1)%N /* numéro du voisin de gauche de i */
# define DROITE (i+1)%N /* numéro du voisin de droite de i */
# define PENSE 0 /* Le philosophe pense */
# define FAIM 1 /* Le philosophe a faim */
# define MANGE 2 /* Le philosophe mange */

typedef int sémaphore ;
int état [N] ; /* Tableau pour suivre l'état des philosophes */
sémaphore mutex = 1 ; /* Exclusion mutuelle pour la SC */
sémaphore s[N] ; /* Un sémaphore par philosophe */

philosophe(int i) { /* i : numéro du philosophe */

    while( TRUE){
        penser ( ) ;
        prendre-fourchette( i ) ; /* prend 2 fourchettes ou bloque */
        manger( ) ;
        poser-fourchette( i ) ; /* repose 2 fourchettes */
    }
}

prendre-fourchette(int i){
    down (mutex) ; /* entre en section critique */
    état[i] = FAIM ;
    test(i) ; /* tente de prendre les 2 fourchettes */
    up (mutex) ; /* quitte la section critique */
    down (s[i]) ; /* bloque s'il n'a pas pu prendre les fourchettes */
}

poser-fourchette(int i){
    down (mutex) ; /* entre en section critique */
    état[i] = PENSE ; /* philosophe a fini de manger */
    test(GAUCHE) ; /* regarde si voisin de gauche peut manger */
    test(DROITE) ; /* regarde si voisin de droite peut manger */
    up (mutex) ; /* quitte la section critique */
}

test(int i) {
    if (état[i]==FAIM AND état[GAUCHE]!=MANGE AND état[DROITE]!=MANGE ){
        état [i] = MANGE ;
        up( s[i] ) ;
    }
}
```

## Exercice 3 : Problème des lecteurs-rédacteurs

```

Semaphore mutex 1           // contrôle l'accès à nb_lect
Semaphore db 1              // contrôle l'accès à la base de données
int nb_lect = 0 ;          // var partagées entre lecteurs pour
                           // compter le nombre de lecteurs
                           // accédant actuellement à la BD

//lecteur
void lecture(){
    while (true) {          //boucle sans fin
        down (mutex);       // la modif de la var. partagée
        nb_lect est
        nb_lect ++;         // une section critique entre
        lecteurs
        if (nb_lect == 1) down (db); //si le premier lecteur
        up(mutex);          // libère l'accès exclusif à
        nb_lect
        lire_la_BD();       //accès à la BD
        down(mutex);
        nb_lect --;
        if (nb_lect == 0) up (db); //si le dernier lecteur
        up (mutex)
        utiliser_données (); //Section restante
    }
}

//rédacteur
void écriture(){
    while (true) {          //boucle sans fin
        créer_données ();   //Section restante

        down (db);
        écrire_dans_la_BD(); //accès à la BD
        up (db);
    }
}

```

## Exercice 4 : Le coiffeur endormi

Utilisation de 3 sémaphores et un compteur

- Semaphore Clients 0 : bloque le coiffeur s'il n'y a pas de clients
- Semaphore Mutex 1 : accès exclusif à la zone critique
- Semaphore Coiffeurs 0 : bloque le client si le coiffeur est occupé avec un autre client
- Int Attente = 0 : Le nombre de clients en attente

```
Semaphore Clients 0;
Semaphore Mutex 1;
Semaphore Coiffeurs 0;
Int Attente = 0 ;

//Coiffeur
void coiffeur() {
    while(1){
        down(Clients);
        down(Mutex);
        Attente = attente - 1;
        up(Coiffeurs);
        up(Mutex);
        Couper_cheveux();
    }
}

//Client
void client() {
    down(Mutex)
    if (Attente < Chaises) {
        Attente = attente + 1
        up(Clients)
        up(Mutex)
        down(Coiffeurs)
        Obtenir_coupe
    } else {
        up(Mutex)
    }
}
```