ED9 : Synchronisation par Sémaphores
Corrigé indicatif

Exercice 1

Question 1
Rappelez le concept des sémaphores en mettant en évidence le fonctionnement des primitives P et V.

Un sémaphore S est un objet partagé constitué de
- un entier **E** initialisé à une valeur ≥0
- une file d'attente **F** des processus bloqués

```
Primitive P(sémaphore S) :
     début
       S.E=S.E-1; //retrait d'une autorisation
       si S.E<0 alors état du processus appelant bloqué;
       placer son id dans la file F;
     fin
Primitive V(sémaphore S) :
     début
       S.E=S.E+1; //ajout d'une autorisation
       si S.E≤0 alors //il y a au moins un processus bloqué
       choix et retrait d'un processus de F;
       réveil du processus;finsi;
     fin
Primitive E0(sémaphore S,entier I) :
     début
       S.E=I; S.F=vide; //I≥0
     fin
```

Question 2 :
Montrez comment réaliser l'exclusion mutuelle avec deux tâches en utilisant les sémaphores. Vérifiez que votre solution garantit les propriétés de l'exclusion mutuelle.

E0(S,1) ;

Tache 1 : **P(S)** **Section_critique** **V(S)**	**Tache 2 :** **P(S)** **Section_critique** **V(S)**

Exercice 2

Question 1 :

Traduire la solution de l'exercice 1 (sur l'exclusion mutuelle) en utilisant l'API proposée dans cet exercice.

```
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include "Sem_Task.h"

SEM mutex;  /* semaphore d'exclusion mutuelle */

TASK_CODE Tache() {

            P(mutex);
            //Section_Critique();
            V(mutex);
}


int   main(void) {
      TASK t1, t2;

/* creation et initialisation des semaphores */
      mutex = newSEM();
      E0(mutex, 1);


/* creation et lancement des taches  */
      t1 = newTASK();
      launchTASK(t1, Tache);
      t2 = newTASK();
      launchTASK(t2, Tache);

/* attente de la fin des taches */

      waitTASK(t1);
      waitTASK(t2);

/* suppression du semaphore */
      deleteSEM(mutex);

return (0);
}
```

Question 2 :

On s'intéresse au modèle producteur/consommateur.

Écrire l'algorithme du producteur/consommateur vu en cours, en utilisant des sémaphores. On montrera que cet algorithme respecte bien les propriétés du modèle producteur /consommateur.

Contexte commun: *SnPlein*: **sémaphore** initialisé à ***0***; *SnVide*: **sémaphore** initialisé à **N**;
Tampon: Tableau de *N* messages;

Processus Producteur	*Processus Consommateur*
Queue: entier initialisé à 0;	*Tete*: entier initialisé à 0;
MessProd: message;	*MessCons*: message;
Début	Début
Tant que vrai faire	Tant que vrai faire
Fabriquer(*MessProd*);	**P(***SnPlein*);
P(*SnVide*);	*MessCons*=*Tampon*[*Tete*];
Tampon [*Queue*]=*MessProd* ;	Tete=(Tete +1) mod N ;
Queue=(Queue +1) mod N ;	**V(***SnVide*);
V(*SnPlein*);	Traiter (*MessCons*);
Fait ;	Fait ;
Fin	Fin

Question 3 :

En vous inspirant de l'algorithme du producteur/consommateur, compléter le programme suivant à l'aide des fonctions décrites dans l'API.

```
/* prodcons.c avec des threads*/
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include "Sem_Task.h"

#define Ncases 10   /* nbr de cases du tampon */


int   Tampon[Ncases];         /* Tampon a N cases*/
SEM Snvide, Snplein;    /* les semaphores */

TASK_CODE Producteur() {
      int i, queue=0, MessProd;

      srand(getpid());

      for(i=0; i<20; i++){
            sleep(rand()%3);   /* fabrique le message */
            MessProd = rand() % 10000;
            printf("Product %d\n",MessProd);

            P(Snvide);
            Tampon[queue]=MessProd;
            V(Snplein);
            queue=(queue+1)%Ncases;
      }

}
```

```
TASK_CODE Consommateur() {
      int tete=0, MessCons, i;

      srand(getpid());
      for(i=0; i<20; i++){
            P(Snplein);
            MessCons = Tampon[tete];
            V(Snvide);
            tete=(tete+1)%Ncases;
            printf("\t\tConsomm  %d \n",MessCons);
            sleep(rand()%3);   /* traite le message */
      }

}

int   main(void) {
      TASK t1, t2;

/* creation et initialisation des semaphores */
      Snvide = newSEM();
      E0(Snvide, Ncases);
      Snplein = newSEM();
      E0(Snplein, 0);



/* creation et lancement des taches  */
      t1 = newTASK();
      launchTASK(t1, Producteur);
      t2 = newTASK();
      launchTASK(t2, Consommateur);

/* attente de la fin des taches */

      waitTASK(t1);
      waitTASK(t2);

/* suppression des semaphores */
      deleteSEM(Snplein);
      deleteSEM(Snvide);
return (0);
}
```