Exercice

Deux processus A et B communiquent au moyen d'un tampon T pouvant contenir qu'un seul message à la fois. Ce tampon est utilisé, de façon alternée, pour la communication dans les deux sens (attention un seul processus utilise à la fois ce tampon). Le processus A dépose un message dans le tampon puis attend la réponse de B avant de déposer à nouveau un autre message et ainsi de suite….

Lorsque B reçoit un message de A, il dépose sa réponse dans le tampon puis se met en attente d'un autre message de A et ainsi de suite…

1. synchronisez au moyen de sémaphores les processus A et B (pour répondre à la question, complétez le code suivant)

Processus A	Processus B

{ char mess[256], rep[256] ;
 	
while (1)
{ /* 1 */
lire (mess);
depot (mess) ;
 	
/* 2*/
recuperer(rep) ;
/*3*/
}
 	
}
	{ char mess[256] , rep[256] ;
	
	while (1)
	{ /* 4 */
	
	recuperer(mess) ;
	reponse(mess,rep)
	/*5*/
	depot(rep);
	/* 6*/
	}
	
	}

2. Supposez maintenant qu'un troisième processus C veuille communiquer avec B en utilisant l'unique tampon T. Les processus A et C se comportent de la même manière. B peut donc recevoir un message de A ou C, la réponse doit être récupérée par le processus expéditeur du message. Synchroniser au moyen de sémaphores les processus A, B et C.

Solution

1/

```
semaphore  SA=0, SB=0 ;  /*0*/
char T[256] ;
void depot (char buf[] ) ;
void recuperer(char buf[] ) ;
```

Processus A

```
{  char mess[256] , rep[256];

        while (1)
        {       /* 1 */
             lire (mess);
             depot (mess ) ;


             /* 2*/ V(SB);
                    P(SA);
              recuperer(rep) ;
             /*3*/
        }

}
```

Processus B

```
{    char mess[256], rep[256]  ;

 while (1)
 {       /*  4 */
          P(SB);
         recuperer( mess) ;
          reponse(mess,rep);


          /*5*/
          depot(mess);
          /* 6*/ V(SA);
 }

}
```

2/

```
semaphore  SA=0, SB=0, SC=0, mutex=1 ;  /*0*/
char T [257] ;
void depot (char buf[] ) ;
void recuperer(char buf[] ) ;
```

Processus A

```
{
   char mess[257], rep[256];

   while (1)
   {          P(mutex);
              lire (mess); mess[256]='A',
              depot (mess ) ;

              V(SB);
              P(SA);
              recuperer(rep) ;
             V(mutex) ;
   }

}
```

Processus B

```
{
    char mess[257],rep[256];

  while (1)
  {
     P(SB);
      recuperer( mess) ;
      reponse(mess,rep);
      depot(rep);
      if (mess[256]='A') V(SA);
     else V(SC);

  }

}
```

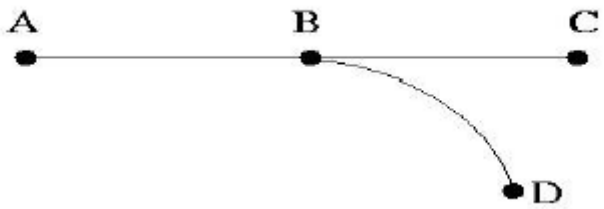
Processus C

```
   {
    char mess [257], rep[256] ;

        while (1)
        {    P(mutex);
             lire (mess); mess[256]='C',
             depot (mess ) ;
             V(SB);
             P(SC);
             recuperer(rep) ;
             V(mutex);
        }

    }
```

Exercice

Nous disposons d'une carte électronique basée sur un microcontrôleur, conçue pour contrôler et coordonner un ensemble de robots. Cette carte est accompagnée d'un logiciel sous Linux permettant de développer des programmes personnalisés. Ces programmes peuvent ensuite être transférés et stockés dans la mémoire non volatile de la carte via un port série.

Votre mission consiste à élaborer un pseudocode destiné à gérer le déplacement de plusieurs robots le long de trajectoires comme suit :



-> Les robots peuvent partir de A vers C ou de D vers A.

--> Pour éviter tout risque de collision, il faut s'assurer que chaque segment du chemin (segments AB, BC et DB) est utilisé par un robot au plus.

1 - Pour répondre à cette question, complétez le pseudocode suivant afin que les règles cidessus soient respectées. Dans ce programme, chaque robot est commandé par un processus..

```
/*0*/
void TraverserSegAB ( ) ; // Traverser le segment AB
void TraverserSegBC ( ) ; // Traverser le segment BC
void TraverserSegBD ( ) ; // Traverser le segment BD
```

Processus RobotAC	Processus RobotDA
{	{
/* 1 */ TraverserSegAB ( ) ; /* 2*/ TraverserSegBC( ) ; /*3*/	/* 4 */ TraverserSegBD( ) ; /*5*/ TraverserSegAB( ); /* 6*/
}	}

Solution

```
/*0*/ Semaphore SAB =1, SBC=1, SBD=1 ;
void TraverserSegAB ( ) ; // Traverser le segment AB
void TraverserSegBC ( ) ; // Traverser le segment BC
void TraverserSegBD ( ) ; // Traverser le segment BD
```

Processus RobotAC	Processus RobotDA
{	{
/* 1 */ **P(SAB) ;** TraverserSegAB ( ) ; /* 2*/ **P(SBC) ; V(SAB) ;** TraverserSegBC( ) ; /*3*/ **V(SBC) :**	/* 4 */ **P(SBD) ;** TraverserSegBD( ) ; /*5*/ **P(SAB) ; V(SBD) ;** TraverserSegAB( ); /* 6*/ **V(SAB) ;**
}	}

Exercice

Soit trois processus P1, P2 et P3. Le processus P1 produit des objets qu'il dépose dans un tampon T1 de taille N1. P2 prélève les objets contenus dans T1, les traite puis dépose les objets traités dans un tampon T2 de taille N2. P3 prélève les objets contenus dans T2 et les consomme. L'accès aux deux tampons ne peut se faire que par un et un seul processus.

A l'aide des sémaphores, modifier le code source des processus P1, P2 et P3 afin de garantir le fonctionnement décrit ci-dessus.

```
void P1() {
  ProduireObjet();
  DéposerObjet(T1);
}
```

```
void P2() {
  RetirerObjet(T1);
  TraiterObjet();
  DéposerObjet(T2);
}
```

```
void P3() {
  RetirerObjet(T2);
  ConsommerObjet();
}
```

Solution

semaphore mutex_T1 = 1;
semaphore empty_T1 = N1;
semaphore full_T1 = 0;
semaphore mutex_T2 = 1;
semaphore empty_T2 = N2;
semaphore full_T2 = 0;

void P1() {
ProduireObjet();
empty_T1.P();
mutex_T1.P();
DéposerObjet(T1);
mutex_T1.V();
ull_T1.V();
}
	void P2() {
	full_T1.P();
	mutex_T1.P();
	RetirerObjet(T1);
	mutex_T1.V();
	empty_T1.V();
	TraiterObjet();
	empty_T2.P();
	mutex_T2.P();
	DéposerObjet(T2);
	mutex_T2.V();
	full_T2.V(); }
		void P3() {
		full_T2.P() ;;
		mutex_T2.P();
		RetirerObjet(T2);
		mutex_T2.V();
		empty_T2.V();
		ConsommerObjet();
		}

Exercice

Soient 2 processus séquentiels qui exécutent le programme suivant :

ProgA:	ProgB:
Debut Répéter Tant que « Vrai » T1 ; Fin Tant que Fin	Debut Répéter Tant que « Vrai » T2 ; Fin Tant que Fin

1. Utilisez deux sémaphores pour synchroniser les 2 processus de telle manière que les tâches se déroulent toujours dans l'ordre : T1T2T1T2T1T2...
2. Utilisez deux sémaphores pour synchroniser les 2 processus de telle manière que les tâches se déroulent toujours dans l'ordre : T1T1T2T2T1T1T2T2T1T1T2T2...
3. Utilisez deux sémaphores pour synchroniser les 2 processus de telle manière que les tâches se déroulent toujours dans l'ordre : T2T1T1T2T1T1T2T1T1...

Solution

1/

semaphore sem_T1 = 1;
semaphore sem_T2 = 0;

Debut
Répéter Tant que « Vrai »
sem_T1.P();
T1;
sem_T2.V();
Fin Tant que
Fin
	Debut
	Répéter Tant que « Vrai »
	sem_T2.P();
	T2;
	sem_T1.V();
	Fin Tant que
	Fin

3/

semaphore sem_T1 = 0;
semaphore sem_T2 = 1;

Debut
Répéter Tant que « Vrai »
sem_T1.P();
T1;
sem_T2.V();
Fin Tant que
Fin
	Debut
	Répéter Tant que « Vrai »
	sem_T2.P();
	T2;
	sem_T1.V();
	sem_T1.V();
	Fin Tant que
	Fin

Exercice

Synchronisez au moyen de sémaphores l'enchaînement des opérations de fabrication de stylos à bille. Chaque stylo est formé d'un corps, d'une cartouche, d'un bouchon arrière et d'un capuchon. Les opérations à effectuer sont les suivantes :

• remplissage de la cartouche avec l'encre (opération RC),
• assemblage du bouchon arrière et du corps (opération BO),
• assemblage de la cartouche avec le corps et le capuchon (opération AS),
• emballage (opération EM).

Chaque opération est effectuée par une machine spécialisée (mRC, mBO, mAS, mEM). Les stocks de pièces détachées et d'encre sont supposés disponibles quand la machine est disponible. Les opérations RC et BO se font en parallèle. L'opération AS doit être effectuée, après ces deux opérations, en prélevant directement les éléments sur les machines mRC et

mBO. Le produit assemblé est déposé dans un stock en attente de l'opération EM. L'opération EM se fait donc après AS, à partir du stock. Le stock est supposé de taille N et de discipline FIFO.

```
mRC( )
{  while (1)
   {
      RC( ) ;

   }
}
```

```
mBO( )
{   while (1)
   {

       BO( ) ;

   }
}
```

```
mAS( )
{    while (1)
   {
      AS( ) ;
      }
}
```

```
mEM( )
{     while (1)
         {
            EM() ;

         }
}
```

Solution

```
Semaphore SRC=1, SBO=1, SAS1=0, SAS2=0, libre=N, occupe=0;
```

```
mRC( )
{  while (1)
   {  P(SRC) ;
      RC( ) ;
      V(SAS1) ;
   }
}
```

```
mBO( )
{   while (1)
   {
      P(SBO) ;
       BO( ) ;
       V(SAS2) ;
   }
}
```

```
mAS( )
{    while (1)
   {
      P(SAS1) ;
      P(SAS2) ;
       P(libre) ;
       AS( ) ;
      V(SRC) ;
      V(SBO) ;
       V(occupe) ;
   }
}
```

```
mEM( )
{
while (1)
{
  P(occupe) ;
   EM() ;
   V(libre) ;
}
}
```

Exercice

Complétez, en ajoutant les sémaphores et les opérations P et V nécessaires, les codes du producteur et du consommateur suivants. Le producteur produit plusieurs ressources à la fois alors que le consommateur consomme une seule ressource à la fois.

```
Producteur {
int ip=0, M;
char ch[N];
 Repeter
 {
        M=Lire(ch,N);

        Deposer(ch, M, ip);


        ip = (ip + M) % N;

  }
}
```

```
Consommateur {
int ic=0;
char c;
 Repeter
 {

        c = Retirer( ic);
        ic = (ic+1) %N

        Traiter(c);
  }

}
```

La fonction « int Lire(char ch[], int N); » construit, dans ch, une chaîne de caractères de

longueur comprise entre 1 et N inclusivement. Elle retourne la longueur de la chaîne.
La fonction « void Deposer(char ch[], int M, int ip); » insère, dans le tampon T, la chaîne de caractères ch. M est la longueur de la chaîne.
La fonction « char Retirer(int ic); » retire un caractère du tampon T. Elle retourne le caractère retiré.
La fonction « void Traiter(char c); » traite le caractère.

Solution

```
char T[N]; // tableau de N caractères
Semaphore Plein =0, Vide=N, Mutex=1

Producteur {
 int ip=0, M;
 char ch[N];
  Repeter
  {
        M=Lire(ch,N);
        Pour i=1 à M Pas 1 faire P(Vide);
        P(Mutex);
        Deposer(ch, M, ip);
        V(mutex);
        Pour i=1 à M Pas 1 faire V(Plein);
        ip = (ip + M) % N;
  }
}
```

```
Consommateur {
 int ic=0;
 char c;
 Repeter
  {
        P(Plein);
        P(Mutex);
        c = Retirer( ic);
        V(Mutex);
        V(Vide);
        ic = (ic+1) %N ;
        Traiter(c);

  }
}
```

Exercice

Deux processus A et B communiquent au moyen d'un tampon T pouvant contenir qu'un seul message à la fois. Ce tampon est utilisé, de façon alternée, pour la communication dans les deux sens (attention un seul processus utilise à la fois ce tampon). Le processus A dépose un message dans le tampon puis attend la réponse de B avant de déposer à nouveau un autre message et ainsi de suite….
Lorsque B reçoit un message de A, il dépose sa réponse dans le tampon puis se met en attente d'un autre message de A et ainsi de suite…
1. synchronisez au moyen de sémaphores les processus A et B (pour répondre à la question, complétez le code suivant)

Processus A	Processus B

{ char mess[256], rep[256] ;
 	
while (1)
{ /* 1 */
lire (mess);
depot (mess) ;
 	
/* 2*/
recuperer(rep) ;
/*3*/
}
 	
}
	{ char mess[256] , rep[256] ;
	
	while (1)
	{ /* 4 */
	
	recuperer(mess) ;
	reponse(mess,rep)
	/*5*/
	depot(rep);
	/* 6*/
	}
	
	}

2. Supposez maintenant qu'un troisième processus C veuille communiquer avec B en utilisant l'unique tampon T. Les processus A et C se comportent de la même manière. B peut donc recevoir un message de A ou C, la réponse doit être récupérée par le processus expéditeur du message.
Synchroniser au moyen de sémaphores les processus A, B et C

Solution

1.

semaphore **SA=0, SB=0**; // initialisation des sémaphores
char T[256] ;
void depot (char buf[]) ;
void recuperer(char buf[]) ;

Processus A	Processus B

{ char mess[256], rep[256] ;
while (1)
{
lire (mess);
depot (mess) ;
V(SB);
P(SA);
recuperer(rep) ;
}
}
	{ char mess[256] , rep[256] ;
	while (1)
	{
	P(SB)
	recuperer(mess) ;
	reponse(mess,rep);.
	depot(rep);
	V(SA);
	}
	}

2.

semaphore … **SA=0, SB=0, SC=0, mutex=1** ; // initialisation des sémaphores
char T[256] ;
void depot (char buf[]) ;
void recuperer(char buf[]) ;

Processus A	Processus B	Processus C

{ char mess[256], rep[256] ;
while (1)
{ **P(mutex);**
lire (mess); **mess[256]='A';**
depot (mess) ;
V(SB);
P(SA);
recuperer(rep) ;
V(mutex);
}
}
	{ char mess[256] , rep[256] ;
	while (1)
	{ **P(SB);**
	recuperer(mess) ;
	reponse(mess,rep);
	depot(rep);
	if(mess[0]=='A') V(SA)
	else V(SC)
	}
	}
		{ char mess[256], rep[256] ;
		while (1)
		{ **P(mutex);**
		lire (mess); **mess[256]='C';**
		depot (mess) ;
		V(SB);
		P(SC);
		recuperer(rep) ;
		V(mutex);
		}
		}

Exercice

Une piscine peut accueillir au maximum N nageurs. Ce nombre N est le nombre de paniers disponibles pour les habits des nageurs. A l'entrée comme à la sortie les nageurs entrent en compétition pour l'acquisition d'une cabine pour se déshabiller et se rhabiller. La piscine dispose uniquement de C cabines (C << N).
Chaque nageur effectue les opérations :

void nageur() {
se_déshabiller(); /* nécessite un panier et une cabine */
nager();
se_rhabiller(); /* nécessite une cabine */
}

Nous pouvons assimiler ces nageurs à des processus concurrents; les cabines et les paniers sont les ressources partagées.

1. Quels sont les sémaphores nécessaires pour assurer la gestion de la piscine et e bon partage des ressources.
2. Modifier le code du processus nageur pour assurer le fonctionnement souhaité.

Solution

```
semaphore sem_paniers = N; // Nombre maximal de paniers disponibles
semaphore sem_cabines = C; // Nombre maximal de cabines disponibles

void nageur() {
  sem_cabines.P() ;
  sem_paniers.P();
  se_déshabiller();
  sem_cabines.V();
   nager();
   sem_cabines.P(); // Attendre qu'une cabine soit disponible
 se_rhabiller();
 sem_cabines.V();
 sem_paniers.V();}
```

Exercice

Rendez vous à N processus.

Soient N processus parallèles ayant un point de rendez-vous. Un processus arrivant au point de rendez-vous se met en attente s'il existe au moins un autre processus qui n'y est pas arrivé.

Le dernier arrivé réveillera les processus bloqués.

Proposer (avec explications) un code du processus qui permet de résoudre ce problème en utilisant des sémaphores.

Solution

sémaphore mutex = 1, s= 0;
entier NbArrivés = 0; /*nbre de processus arrivés au rendez-vous*/
Procédure RDV

```
Début
mutex.P();
NbArrivés = NbArrivés + 1;
Si (NBArrivés < N) Alors /* non tous arrivés */
    mutex.V(); /* on libère mutex et */
    s.V(); /* on se bloque */
Sinon
   Mutex.V(); /* le dernier arrivé libère mutex et */
   Pour i = 1 à N-1 Faire
    s.V(); /* réveille les N-1 bloqués */
Finsi Fin
```

Exercice

Considérez un système multicouche composé de trois couches P0, P1 et P2. Les couches sont des processus concurrents qui communiquent au moyen de deux tampons T0 et T1 de même taille N :

- P0 et P1 partagent le tampon T0 et
- P1 et P2 partagent le tampon T1.

Chaque couche se charge d'un traitement particulier :

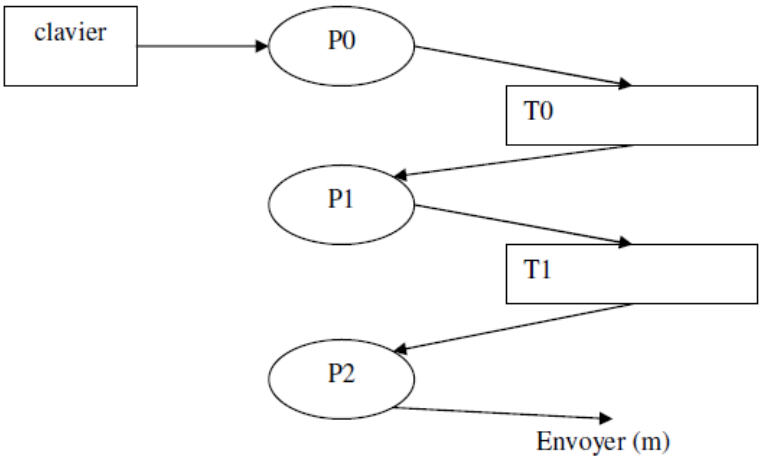
- Le processus P0 se charge de lire du clavier des messages qu'il traite avant de les déposer dans le tampon T0. Le traitement d'un message par la couche P1 consiste à l'encrypter. Il est réalisé par la fonction Encrypter suivante :

Message Encrypter (Message);

La fonction Message Lire (); permet de lire un message du clavier.

- Le processus P1 se charge de transférer directement les messages du tampon T0 vers le tampon T1.
- Le processus P2 récupère les messages du tampon T1 pour les envoyer à un destinataire.

L'envoi d'un message est réalisé par la fonction Envoyer : Envoyer (Message);



Solution

mutex1=1, mutex0=1, Vide0=N, Vide1=N, Plein0 = 0, Plein1 = 0;

P0

```
Message m, mc;
int ip=0;
Répéter
{       m= lire();
        mc = Encrypter(m);
        P(Vide0);
        P(mutex0);
        T0[ip] = mc;
        V(mutex0);
        ip = ip+1 mod(N)
        V(Plein0);
}
```

P1

```
int icp=0;
Répéter
{       P(Plein0);
        P(Vide1)
        P(mutex0);
        P(mutex1)
        T1[icp] = T0[icp];
        V(mutex1);
        V(mutex0)
        icp = icp+1 mod(N)
        V(Plein1);
        V(Vide0);
}
```

P2

```
Message mc;
int ic=0;
Répéter
{       P(Plein1)
        P(mutex1);
        mc = T1[ic];
        V(mutex1);
        ic = ic+1 mod(N)
        V(Vide1);
        Envoyer(mc);
}
```