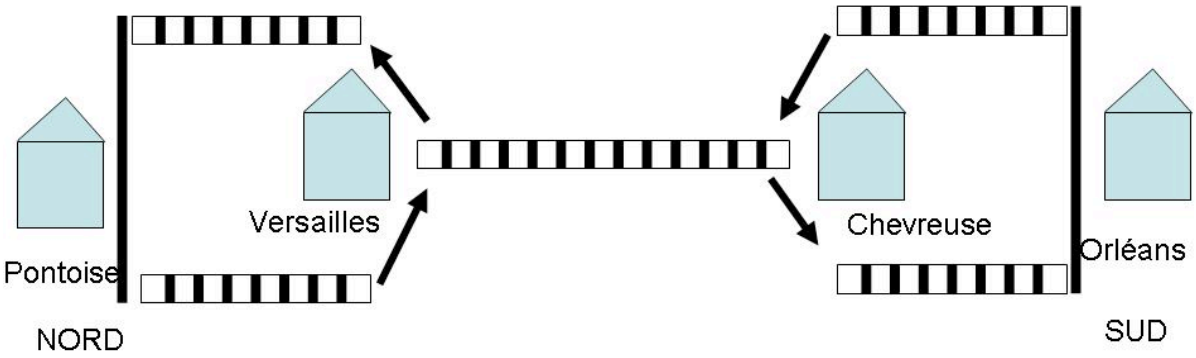
ED 10

Exercice 1 : Compétition d'accès pour une classe de processus

Le problème étudié est celui de l'exclusion mutuelle entre deux classes de processus, les processus d'une même classe ne s'excluant pas mutuellement. On pourra se référer au problème classique des « lecteurs et rédacteurs », en considérant deux classes de « lecteurs » en compétition d'accès pour une ressource.
La logne de chemin de fer PONTOISE-ORLEANS comporte un tronçon à voie unique de VERSAILLES à CHEVREUSE.



Les règles de circulation sur la voie unique sont les suivantes :

1. Le tronçon ne doit jamais être emprunté simultanément par deux trains allant en sens inverse ;
2. le tronçon peut être emprunté par un ou plusieurs trains allant tous dans le même sens ;
3. il n'y a pas de sens de parcours prioritaire.

Pour étudier la rentabilité de cette ligne, on désire effectuer une simulation du trafic, pour cela, on introduit deux classes de processus : les trains PONTOISE-ORLEANS et les trains ORLEANS-PONTOISE, qui se décrivent comme suit :

Tâche PONTOISE-ORLEANS :
Début
Parcours (PONTOISE, VERSAILLES) ;
Entrée_nord ;
Parcours (VERSAILLES,CHEVREUSE) ;
Sortie_sud ;
Parcours (CHEVREUSE, ORLEANS) ;
Fin

Tâche ORLEANS-PONTOISE:
Début
Parcours (ORLEANS, CHEVREUSE) ;
Entrée_sud ;
Parcours (CHEVREUSE, VERSAILLES) ;
Sortie_nord;

Parcours (VERSAILLES, PONTOISE) ;
Fin

Question :
On considère le cas où toute coalition est autorisée. Les trains allant dans un sens peuvent attendre indéfiniment, s'il y a constamment des trains dans l'autre sens sur le tronçon.

En utilisant l'API proposée à l'ED9, on demande de programmer les quatre fonctions : Entrée_nord, Sortie_sud, Entrée_sud et Sortie_nord, de façon à ce que les processus respectent les règles de circulation sur la voie unique.

Exercice 2 Duo de producteurs pour un consommateur

On veut installer une variante du producteur consommateur qui permette à deux producteurs P1 et P2 (processus clients) de partager un même consommateur C (tâche serveur), tout en ayant chacun son tampon de dépôt de message (chacun a un tampon de 5 cases). On ajoute en plus que les messages déposés par P1 sont prioritaires par rapport aux messages déposés par P2.

On suppose le type Message prédéfini .

```
//Contexte commun
Message T1[4] ; Message T2[4] ;// tampons de messages
Int Nombre=0 ; //nombre de messages dans T1 prioritaires
Int Tete1=0, Queue1=0, Tete2=0, Queue2=0 //mod 5 gestion des tampons
// Déclarations des sémaphores à faire
```

On utilise les fonctions Deposer1(Message M), Deposer2(Message M) et Message Retirer () respectivement pour le dépôt dans T1 par P1, pour le dépôt dans T2 par P2 et le retrait d'un message dans T1 ou T2 par C.

Algorithme de Déposer1 :
Si T1 est plein alors blocage jusquà T1 non plein;
Déposer le message dans T1 ;
Incrémenter le nombre de messages prioritaires ;

Algorithme de Déposer2 :
Si T2 est plein alors blocage ;
Déposer le message dans T2;

Algorithme de Retirer
S'il y a un message a retirer alors
S'il y a un message prioritaire alors
Retirer un message de T1 ; décrementer le nombre de messages prioritaires ;
Sinon retirer un message de T2 ; fin si ;
Sinon blocage jusqu'à ce qu'il y ait un message a retirer dans T1 ou T2 ; finsi ;

Question

En utilisant l'API proposée en ED, compléter le contexte commun par la déclaration et l'initialisation des sémaphores et programmer les trois fonctions proposées. On explicitera la gestion des tampons.

ED 10

Corrigé indicatif

Exercice 1 : Compétition d'accès pour une classe de processus

```
/* train.c */
#include    <stdio.h>
#include    <unistd.h>
#include    <pthread.h>
#include    <semaphore.h>
#include    "Sem_Task.h"

// contexte commun
SEM mutexTroncon;
SEM mutexN1;
SEM mutexN2;
int n1=0;
int n2=0;

/* procedure entree_nord */
void entree_nord() {
P(mutexN1);
n1=n1+1;
if (n1==1) /* le premier train dans le sens nord-sud */
    P(mutexTroncon);
V(mutexN1);
}

/* procedure sortie_sud */
void sortie_sud () {

P(mutexN1);
n1=n1-1;
if (n1==0) /* liberer l'acces au troncon */
      V(mutexTroncon);
V(mutexN1);

}

/* procedure entree_sud */
void entree_sud() {
P(mutexN2);
n2=n2+1;
if (n2==1) /* le premier train dans le sens nord-sud */
    P(mutexTroncon);
V(mutexN2);

}

/* procedure sortie_nord */
void sortie_nord () {

P(mutexN2);
n2=n2-1;
if (n2==0) /* liberer l'acces au troncon */
```

```
      V(mutexTroncon);
V(mutexN2);

}

TASK_CODE   Train_1(int myNum) {
      srand(getpid());

        printf("Train %d (sens 1) parcourt PONTOISE VERSAILLES\n", myNum);
      sleep(1+random()%2);  /* parcours */
      entree_nord();
      printf("Train %d (sens 1) parcourt VERSAILLES CHEVREUSE\n", myNum);
      sleep(1+random()%2);  /* parcours */
      sortie_sud();
      printf("Train %d (sens 1) parcourt CHEVREUSE ORLEANS\n", myNum);
      sleep(1+random()%2); /* parcours */

}

TASK_CODE Train_2(int myNum) {

      srand(getpid());
        printf("\tTrain %d (sens 2) parcourt ORLEANS CHEVREUSE\n", myNum);
      sleep(1+random()%2);  /* parcours */
      entree_sud();
      printf("\tTrain %d (sens 2) parcourt CHEVREUSE VERSAILLES\n", myNum);
      sleep(1+random()%2);   /* parcours */
      sortie_nord();
      printf("\tTrain %d (sens 2) parcourt VERSAILLES PONTOISE\n", myNum);
      sleep(1+random()%2);   /* parcours */

}

int   main(void) {
      int i;
      TASK t1[5], t2[7];


      /* Creation des semaphores */
      mutexTroncon=newSEM();
mutexN1=newSEM();
mutexN2=newSEM();

      /* Initialisation des semaphores a 1 */
      E0(mutexTroncon,1);
      E0(mutexN1,1);
      E0(mutexN2,1);


      /* Creation  et lancement des trains type 1*/
      for (i=0; i<5; i++)
        {
          t1[i]=newTASK();
          launchTASK(t1[i], Train_1(i));
        }

      /* Creation  des trains type 2*/
      for (i=0; i<7; i++)
        {
          T2[i]=newTASK();
          launchTASK(t2[i], Train_2(i));
```

```
        }

      /* attente de terminaison */
      for (i=0; i<5; i++)  waitTASK(t[i]);
      for (i=0;i<7;i++) waitTASK(t[i]);

      printf("********* Fin des taches trains\n");

      /* suppression des semaphores  */
deleteSEM(mutexTroncon);
deleteSEM(mutexN1);
deleteSEM(mutexN2);

      return (0);
}
```

Exercice 2 Duo de producteurs pour un consommateur

```
//Contexte commun
Message T1[4] ; Message T2[4] ;// tampons de messages
Int Nombre=0 ; //nombre de messages dans T1 prioritaires
Int Tete1=0, Queue1=0, Tete2=0, Queue2=0 //mod 5 gestion des tampons
SEM Plein, Vide1, Vide2, Mutex ;
/*Initialisation des semaphores*/
E0(Plein,0) ; // gere le nombre total de messages dans T1 et T2
E0(Vide1,5) ; //gere le nombre de cases vides dans T1
E0(Vide2,5) ; //gere le nombre de cases vides dans T2
E0(Mutex,1) ; //gere l'acces a la variable partagee Nombre

// fonction Deposer1
void Deposer1(Message M) {
      P(Vide1) ;
      T1[Queue1]=M ;
      Queue1=(Queue1+1)%5 ;
      P(Mutex) ;
      Nombre++ ;
      V(Mutex) ;
      V(Plein) ;

}

// Fonction deposer2
void Deposer2(Message M) {
      P(Vide2) ;
      T2[Queue2]=M ;
      Queue2=(Queue2+1)%5 ;
      V(Plein) ;
}

// fonction Retirer
Message Retirer() {
Message M ;
      P(Plein) ; // Y-a-t-il un message a retirer ?
      P(Mutex) ;
      // y-a-t-il des messages prioritaires ?
      if (Nombre >0) {
            Nombre - - ;
            V(Mutex) // sortie de section critique
            M = T1[Tete1] ; // on retire dans T1
            Tete1 = (Tete1 + 1)%5 ;
            V(Vide1) ;
```

```
	}
	else {
		V(Mutex) // Sortie de section critique
		M = T2[Tete2] ; // On retire dans T2
		Tete2 = (Tete2 + 1)%5 ;
		V(Vide1) ;
	}
	Return (M) ;
}
```

Variante

On suppose que les opérations d'incrémentation, décrémentation sont atomiques
Le producteur P1 augmente nombre, le consommateur C diminue nombre.
L'exclusion mutuelle pour nombre n'est pas utile, il n'y a qu'un seul producteur et un seul consommateur.

```
// fonction Deposer1
void Deposer1(Message M) {
	P(Vide1) ;
	T1[Queue1]=M ;
	Queue1=(Queue1+1)%5 ;
	Nombre++ ;
	V(Plein) ;
}

// Fonction deposer2
void Deposer2(Message M) {
	P(Vide2) ;
	T2[Queue2]=M ;
	Queue2=(Queue2+1)%5 ;
	V(Plein) ;
}

// fonction Retirer
Message Retirer() {
Message M ;
	P(Plein) ; // Y-a-t-il un message a retirer ?
	// y-a-t-il des messages prioritaires ?
	if (Nombre >0) {
		Nombre - - ;
		M = T1[Tete1] ; // on retire dans T1
		Tete1 = (Tete1 + 1)%5 ;
		V(Vide1) ;
	}
	else {
		M = T2[Tete2] ; // On retire dans T2
		Tete2 = (Tete2 + 1)%5 ;
		V(Vide1) ;
	}
	Return (M) ;
}
```