```
int sys_P (int sem) {
  if ( (v.vect_sem[sem].first == -1) ||
(v.vect_sem[sem].compt == - MAX_FILE) )

    /* sémaphore n'est pas initialisé ou file
pleine*/
     return -1;
  /* code à compléter - question 1.2 */
  ......
  return 0 ;
}
```

```
int sys_V (int sem) {
  if (v.vect_sem[sem].first == -1)
      /* sémaphore n'est pas initialisé */
     return -1;
  /* code à compléter - question 1.2*/
  ......
  return 0 ;
}
```

1.1

En considérant que les opérations sur un sémaphore doivent être **atomiques**, pourquoi n'est-t-il pas nécessaire de masquer les interruptions dans le corps des fonctions décrites ci-dessus?

1.2

Complétez les fonctions *int sys_P(int sem)* et int sys_V(sem).

Observation : Si vous le jugez nécessaire, vous pouvez ajouter d'autres champs à *struct sem*.

Supposons maintenant que nous ajoutons le champ *int compt_init* à *struct sem* afin de sauvegarder la valeur d'initialisation du sémaphore. Cette sauvegarde a été ajoutée au code de la fonction sys_*Init_sem* :

```
int sys_Init_sem (int sem, int N) {
.... /* même code que la version précedente */
   v.vect_sem[sem].compt_init=N;
   return 0;
}
```

1.3

Modifiez la fonction *int sys_Lib_sem (int sem)* pour qu'elle soit bloquante. Tant qu'il y a des processus utilisant un exemplaire de la ressource ou en attente pour un exemplaire, le processus qui a appelé *Lib_sem* restera bloqué et le sémaphore ne sera pas libéré.

Quelle fonction va réveiller ce processus ? Donnez le nouveau code de cette fonction (seulement les lignes ajoutée).

Observation : vous pouvez ajouter des champs à *struct sem*, si nécessaire. La priorité de réveil est PRISEM.

2. SIGNAUX (25 MIN – 6 POINTS)

Considérez le code du Noyau **non préemptif** vu en TD.

Nous voulons offrir aux utilisateurs l'appel système *int tsigwait (long ens)* qui renvoie le plus petit numéro du signal compris dans la liste de signaux pendants du processus appelant, appartenant aussi à l'ensemble *ens*. Le signal en question est retiré de la liste de signaux pendants. Cependant, s'il n'existe aucun signal pendant qui est compris dans *ens*, le processus appelant est bloqué jusqu'à réception d'un signal.

2.1

Programmez la fonction du noyau *int sys_tsigwait (long ens)* appelée par l'appel système *tsigwait* décrit ci-dessus. Si nécessaire, le processus doit s'endormir avec la priorité ***PRISIG*** (interruptible par des signaux) sur l'adresse de la variable du noyau qui sauvegarde les signaux pendants du processus appelant.

2.2

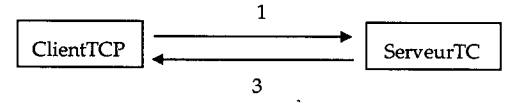
Supposons que le processus en appelant *tsigwait* s'endorme. Quand est-ce qu'il sera réveillé ? Quelle fonction va le réveiller ? Quel sera le code de renvoie de la fonction *tsigwait*?

3. UTILISATION D'UN SERVEUR TCP AVEC UDP (25 MIN. 4 POINTS)

Le but de cet exercice est de réutiliser le serveur de variable TCP écrit en TME, pour gérer des clients qui communiquent en UDP. Contrairement à la solution proposée dans le TME, on ne souhaite pas modifier le serveur écrit en TCP. On définit la structure des messages de la façon suivante :

```
#define MSGSET 0
#define MSGGET 1
strcut msg {
 int type ; /* MSGSET ou MSGGET
  char var[256] ;
  char val[256] ;
}
```

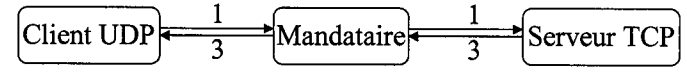
Le protocole original peut être symbolisé de la façon suivante.



Avec 1, 2 et 3 symbolisant les actions suivantes :

- 1 : msg avec msg.type = MSG**GET**, 2 : consultation de la variable, 3 : msg avec val la valeur de la variable.
- 1 : msg avec msg.type = MSG**SET**, 2 : modification de la variable, 3 : msg d'origine.

De manière à utiliser le serveur écrit en TCP avec des clients écrits en UDP, on interpose un nouveau serveur, appelé mandataire, entre les deux. Le mandataire s'occupe de traduire les requêtes UDP en TCP. Le mandataire permet donc de réutiliser le serveur TCP sans modification. La figure suivante symbolise le fonctionnement.



3.1

Brièvement, que se passe-t-il si le mandataire tombe en panne ? Est-ce que le service peut toujours être rendu ? Et si le serveur TCP tombe en panne ?

3.2

Combien de sockets doivent être créées sur le mandataire ?

3.3

Programmez le mandataire sachant que le serveur ne ferme pas sa socket de communication TCP après le traitement d'une requête. On supposera que le premier argument du main du mandataire est le nom de la machine du serveur et que le second argument est le numéro de port du serveur.