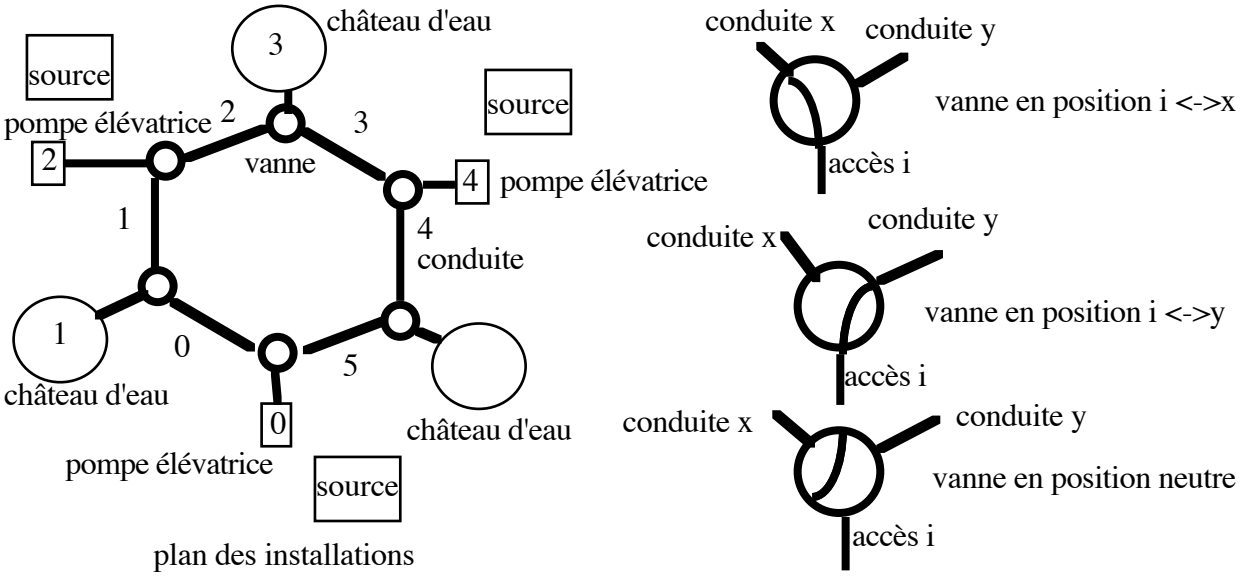
ED 11
Sémaphores privés

Dans une commune de grande étendue il y a 3 châteaux d'eau qui sont alimentés par 3 pompes élévatrices qui pompent l'eau de 3 points d'eau (sources ou réservoirs naturels). Pour éviter une pénurie d'eau en cas de tarissement d'un des points d'eau, le conseil municipal décide de construire des conduites pour pouvoir alimenter un château d'eau à partir de deux points d'eau. Mais par souci d'économie, il ne veut pas doubler les pompes élévatrices ; par contre installer les conduites n'est pas onéreux et au contraire apporte de l'emploi dans la commune. Le conseil municipal fait donc l'achat de 6 vannes et fait construire 6 conduites d'eau selon le plan ci-dessous.



Question1.

Le conseil municipal confie l'entretien des 6 conduites d'eau à 6 employés communaux. chacun d'eux est responsable d'une conduite et de son utilisation. Chaque employé peut donc actionner les deux vannes qui sont à chaque bout de la conduite, l'une vers un château d'eau, l'autre vers une pompe. Les vannes comportent trois positions (neutre, affectée à l'une ou l'autre des conduites), ce qui interdit à deux pompes d'alimenter le même château d'eau (il y aurait une surpression dangereuse), ou à une même pompe d'alimenter deux châteaux d'eau (il y aurait une dépression et arrêt de la pompe).

Montrer que si les actions des 6 employés communaux ne sont pas coordonnées, il peut y avoir interblocage.

Montrez que cet interblocage peut être évité par une gestion simple de l'emploi du temps de ces 6 employés.

Question 2.

Un membre informaticien du conseil municipal (ancien auditeur du CNAM, qui n'a pas tout oublié du cours système), décide d'étudier plusieurs règles d'action par ces employés sur les vannes et d'analyser leur efficacité par un programme de simulation .

Pour cela, il lance l'exécution de 6 tâches analogues. Les 6 tâches peuvent se dérouler en parallèle, il faut donc contrôler l'utilisation qu'ils font des vannes et compléter le comportement, ci-dessous, de chaque tâche par un prélude et un postlude réalisant l'évitement de l'interblocage :

Contexte commun

On suppose le type conduite prédéfini, c'est en fait le type ID_PROC

```
typedef ID_PROC enum {0,1,2,3,4,5} ;

SEM SEMPRIV[ID_PROC] ; // Un sémaphore binaire associé à chaque tâche

SEM MUTEX ; // Exclusion mutuelle pour l'accès aux variables d'état

// déclaration des variables d'état à compléter

  TASK_CODE PROCESSUS  (ID_PROC  I) {
  /* I : association d'un nom unique à la tâche */
  /* X = vanne à un bout de Z ; Y := vanne à l'autre bout de Z ;*/
  conduite  Z = I ;

  while true {
            prelude ;
            connecter X à la conduite Z ;
            connecter Y à la conduite Z ;
          simuler l'utilisation de la conduite pendant un délai aléatoire ;
            comptabiliser les durées d'attente et d'utilisation de Z ;
            déconnecter Y, déconnecter X ;
            postlude ;
            attendre un délai aléatoire;
              }

  }
```

Compléter le programme ci-dessus en introduisant des variables d'état et en appliquant la technique dite des sémaphores privés, avec un tableau de 6 sémaphores privés SEMPRIV[Z] où Z correspond au numéro de conduite. Ne pas oublier d'initialiser les sémaphores.

Question 3

On sait que cette méthode des sémaphores privés permet d'éviter l'interblocage, mais qu'elle ne prévient pas l'attente indéfinie d'une tâche par suite de coalition involontaire entre

d'autres tâches.

Montrer un exemple de coalition possible.

Question 4

Pour éviter cette coalition, on donne à chaque conduite une priorité qui croît avec l'attente de la tâche, et on ne connecte une vanne à une conduite que si la conduite voisine n'a pas une priorité plus forte.

Par exemple la priorité p[Z] d'une conduite Z vaut zéro quand la conduite est libre ou occupée. Que se passe-t-il quand la conduite Z est demandée ? Si les conduites Z-1 (modulo 6) et Z + 1 (modulo 6) ne sont pas occupées ou ne sont pas demandées avec des priorités plus grandes que p[Z], alors la conduite Z peut être attribuée et sa priorité p[Z] est remise à zéro. Si la conduite Z ne peut être attribuée, on en tient compte en augmentant de 1 la priorité p[Z] chaque fois que l'une des conduites Z-1 ou Z + 1 est libérée et lors de la demande de Z.

Reprendre le programme de la question 2 pour y introduire les priorités.

Question 5

Votre solution conduit-elle à un interblocage ? Expliquer pourquoi.

ED 11 Corrigé indicatif
Sémaphores privés

Schéma des sémaphores privés

Un sémaphore ne contient qu'un compteur, ce qui peut être insuffisant pour exprimer des contraintes de contrôle, qui peuvent être propres à chaque tâche concurrente.

Le principe est de définir un ensemble de données partagées qui reflète l'état du système en particulier l'état des tâches (blocage, non blocage) et l'état d'allocation des ressources.

A chaque tâche est associée une condition de blocage et un sémaphore pouvant prendre une valeur ≥-1. Ce sémaphore est dit sémaphore privé : seule la tâche à laquelle il est associé, peut effectuer un appel de la primitive P si la condition de blocage est réalisée. Les autres tâches effectueront éventuellement un V, pour débloquer la tâche . Ce sémaphore est initialisé à 0. Dans le schéma considéré, seule une autre tâche débloquera la tâche et donc la valeur du sémaphore sera 0 ou -1.

Les données partagées sont testées et modifiées en section critique.

Soient Sempriv(I) un sémaphore associé à la tâche Pi avec E0(Sempriv(I) , 0)

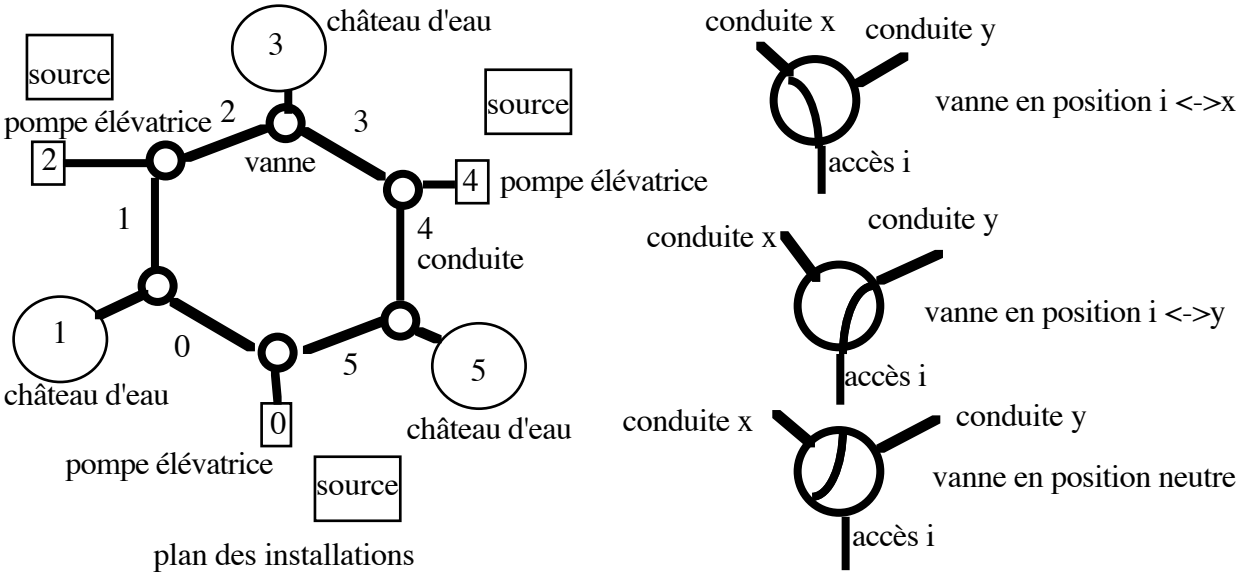
Mutex un sémaphore d'exclusion mutuelle E0(Mutex,1)

Cond(I) la condition de blocage de Pi exprimée à partir des variables d'état

Le schéma d'une tâche est le suivant :

```
Tâche Pi :

OK : booléen =faux;

P(Mutex) ;

Si Cond(I) alors noter dans les variables d'état que Pi est bloqué ;

Sinon OK = vrai ; noter dans les variables d'état le non blocage de Pi ;

Fin si ;

V(Mutex) ;

Si non OK alors P(Sempriv(I)) ;finsi ;

-- blocage hors section critique pour éviter un interblocage ;

Utilisation des ressources ;

P(Mutex) ;

Mise à jour des variables d'état ;

Pour chaque tâche Pj bloquée faire

Si non Cond(J) alors  noter déblocage de Pj ; V(Sempriv(J)) ; fin si ;

V(Mutex) ;
```

Corrigé indicatif de l'exercice



Question1.

Montrer que si les actions des 6 employés communaux ne sont pas coordonnées, il peut y avoir interblocage.

Montrez que cet interblocage peut être évité par une gestion simple de l'emploi du temps de ces 6 employés.

Il y a interblocage si chaque employé actionne une vanne de telle sorte que toute conduite soit rattachée à un château ou bien à une pompe. Ce problème est analogue à celui des philosophes aux spaghettis où chaque philosophe prend une fourchette.

Si on n'autorise que 5 employés à actionner les vannes simultanément, au moins un employé pourra actionner deux vannes de telle sorte qu'une conduite fonctionne entre un château d'eau et une pompe.

Question 2.

Compléter le programme ci-dessous en introduisant des variables d'état et en appliquant la technique dite des sémaphores privés, avec un tableau de 6 sémaphores privés SEMPRIV[Z] où Z correspond au numéro de conduite.

La position d'une vanne correspond à un château ou une pompe ou libre. Une conduite, pour fonctionner nécessite que les deux vannes adjacentes soient disponibles. A chaque conduite Z, on associe une variable d'état A[Z] pouvant prendre les valeurs libre, occupée, demandée. Un processus simule le fonctionnement d'une conduite.

Contexte commun

On suppose le type conduite prédéfini, c'est en fait le type ID_PROC

```
typedef ID_PROC enum {0,1,2,3,4,5} ;

SEM SEMPRIV[ID_PROC] ; // Un sémaphore privé associé à chaque tâche

SEM MUTEX ; // Exclusion mutuelle pour l'accès aux variables d'état

// déclaration des variables d'état à compléter

typedef ETAT enum {libre,  demandee, occupee} ;

ETAT A[]={libre, libre, libre, libre, libre, libre} ;


// initialisation des sémaphores

E0(MUTEX,1) ;

for (i=0 ; i<=5 ;i++) E0(SEMPRIV[i],0) ;


TASK_CODE PROCESSUS  (ID_PROC  I) {
Z = I ; X = vanne à un bout de Z ; Y = vanne à l'autre bout de Z ;
while true {
            // prélude ;
            P(MUTEX);
            A[Z]=demandee;
            if (A[(Z+1) % 6)] !=occupee) && (A[(Z-1) % 6] !=occupee)
                  { -- la conduite peut fonctionner
                  A[Z]=occupee; V(SEMPRIV[Z]);

            }
            V(MUTEX);
            P(SEMPRIV[Z]);
            connecter X à la conduite Z ;
            connecter Y à la conduite Z ;
            simuler l'utilisation de la conduite pendant un délai aléatoire
            comptabiliser les durées d'attente et d'utilisation de Z
            déconnecter Y, déconnecter X ;
```

```
            //postlude ;
            P(MUTEX);

            A[Z]=libre ;
            if (A[(Z+1) % 6]== demandee) && (A[(Z+2) % 6] !=occupee)
                  {

                  A[(Z+1) % 6]=occupee; V(SEMPRIV[(Z+1) % 6] ;

                  }

            if (A[(Z-1) % 6]== demandee) && (A[(Z-2) % 6] !=occupee)
                  {

                  A[(Z-1) % 6]=occupee; V(SEMPRIV[(Z-1) % 6]);

                  }
            V(MUTEX);
            attendre un délai aléatoire;
            } // end while

} // end PROCESSUS
```

Remarques :

1 On pourrait utiliser :

```
boolean test (ID_PROC Z) {

      if ((A[Z]==demandee) && (A[(Z+1)% 6]!=occupee) &&
      (A[(Z-1)% 6]!=occupee)) return true;

      else return false;

}
```

2. D'autre part, on peut utiliser une variante du schéma des sémaphores privés avec une variable locale OK, évitant un blocage en section critique.

```
// meme contexte commun que précédemment


void prelude(ID_PROC Z) {
boolean OK;

      P(MUTEX);
      A[Z]=demandee;OK=test(Z);
      if OK a[Z]=occupee;
      V(MUTEX);
      if (!OK) P(SEMPRIV[Z]);
} //end prelude


void postlude(ID_PROC Z) {

      P(MUTEX);
      A[Z]=libre;
      if test((Z+1)% 6) { A[(Z+1)% 6]=occupee; V(SEMPRIV[(Z+1)% 6]);}
```

```
      if test((Z-1)% 6) { A[(Z-1)% 6]=occupee; V(SEMPRIV[(Z-1)% 6]);}
      V(MUTEX);
}
```

Question 3

Montrer un exemple de coalition possible.

Les processus 0, 2, 4 s'exécutent sans jamais libérer simultanément les vannes correspondantes, les autres processus ne s'exécutent jamais. Par contre, il n'y a jamais interblocage car on connecte les deux vannes en une seule fois.

Question 4

Reprendre le programme de la question 2 pour y introduire les priorités.

L'idée est celle des priorités dynamiques croissant en fonction du temps écoulé depuis la demande (style allocation du processeur dans UNIX).

On ajoute dans le contexte commun les déclarations suivantes :

```
typedef int priorite;

priorite P[ID_PROC];
```

Les fonctions deviennent :

```
boolean test(IDPROC Z) {


      if ((A[Z]==demandee) && (A[(Z+1)% 6]!=occupee) && (A[(Z-1)%
6]!=occupee) && (P[(Z+1)% 6] <= P[Z]) && (P[(Z-1)% 6]<=P[Z])) return true;

      else return false;
} // end test



void prelude(ID_PROC Z) {
boolean OK;

      P(MUTEX);
      A[Z]=demandee;

      OK=test(Z);
      if OK {

            A[Z]=occupee;

            P[Z]=0;

            }

      else {

            P[Z]=P[Z]+1;
```

```
            }
      V(MUTEX);
      if (!OK) P(SEMPRIV[Z]);


}//end prelude


void postlude(ID_PROC Z) {

      P(MUTEX);
      A[Z]=libre;
      if test((Z+1)% 6) {

            A[(Z+1)% 6]=occupee;

            V(SEMPRIV[(Z+1)% 6]);

            P[(Z+1)% 6]=0;

      }

      else {
            /* on ne peut faire fonctionner Z+1, si Z+1 était demandee,
alors il faut augmenter sa priorité */
            if (A[(Z+1)% 6]==demandee) { P[(Z+1)% 6]=P[(Z+1)% 6]+1;}

      } // fin du if then else

      if test((Z-1)% 6) {

            A[(Z-1)% 6]=occupee;

            V(SEMPRIV[(Z-1)% 6]);

            P[(Z-1)% 6]=0;

      }

      else {
            if (A[(Z-1)% 6]==demandee) { P[(Z-1)% 6]=P[(Z-1)% 6]+1; }

      } // fin du if then else

V(MUTEX);


}// end  postlude
```

Question 5

Votre solution conduit-elle à un interblocage ? Expliquer pourquoi.

En ordonnançant les demandes de conduites en cas de conflit, on évite bien la coalition (famine). L'évitement de l'interblocage est maintenu : une condition nécessaire d'interblocage serait que pour tout Z (P[Z-1]>P[Z] ou P[Z+1]>P[Z], ce qui est impossible car conduisant à P[Z] < P[Z] (attente circulaire).

Par contre, si on avait programmé dans test : P[Z+1] <= P[Z] ..., il y aurait risque d'interblocage : une condition nécessaire d'interblocage serait pour tout Z (P[Z-1]>=P[Z] ou P[Z+1]>=P[Z]) ce qui implique P[Z]<=P[Z]. Il peut donc y avoir attente circulaire.