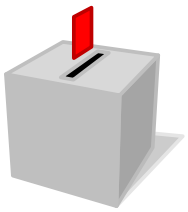
Votation

Ensimag 2A

1 Isoloirs

English translation of French voting vocabulary is given in the annexe A.



Plusieurs processus votent. Il faut les synchroniser. Pour voter, un processus doit passer dans un isoloir. Il y a N isoloirs. Il doit y avoir au plus 1 processus dans un isoloir à 1 instant donné.

On modélise les isoloirs par un tableau de booléen **isoloir_occupe**, avec la sémantique suivante :

- **isoloir_occupe[i] == faux** : l'isoloir est vide.
- **isoloir_occupe[i] == vrai** : un processus est dans l'isoloir.

Les isoloirs sont vides à l'origine, c'est-à-dire **bool isoloir[N] = {faux, ..., faux};**.

1. Écrivez un *moniteur* avec les primitives de synchronisations suivantes :

- **entrer_isoloir(int *i)**
- **sortir_isoloir(int i)**

où i est un index d'isoloir. **entrer_isoloir** permet au processus de rentrer dans un isoloir libre, et lui retourne l'index de l'isoloir. **sortir_isoloir** permet au processus de quitter l'isoloir i dans lequel il était entré. On considère qu'un processus appellera toujours **sortir_isoloir** avec l'index qu'il a récupéré avec **entrer_isoloir**.

Vous devez garantir qu'il ne peut y avoir au maximum qu'un seul processus par isoloir.

Vous devez définir les variables nécessaires à l'écriture du moniteur.

Solution:

```
mutex m;
cond c_isoloir;
bool isoloir_occupe[N] = {false, ..., false};
entrer_isoloir(int *i) {
   m.lock();
test_isoloir:
   for (j = 0; j < N ; j++ ) {
       if ( not(isoloir_occupe[j]) ) {
           isoloir_occupe[j] = true;
           *i = j;
           m.unlock();
           return ;
       }
   }
   c_isoloir.wait();
   goto test_isoloir;
}
sortir_isoloir(int i) {
   m.lock();
   isoloir_occupe[i] = false;
   c_isoloir.signal();
   m.unlock();
}
```

2 Bulletins

On considère maintenant qu'il y a des bulletins dans chaque isoloir. Ils sont représentés par un tableau d'entiers, il y a au départ B bulletins dans chaque isoloir.

int bulletins[N] = {B, ..., B};

A chaque fois qu'un processus vote, il consomme un bulletin dans l'isoloir où il se trouve. Un processus qui vote a donc le comportement suivant :

```
void vote(void) {
    int i;

    entrer_isoloir(&i);
    bulletins[i]--;
    /* ... choisi son candidat ... */
    sortir_isoloir(i);
}
```

De plus, il existe un processus chargé de remettre des bulletins dans les isoloirs. Ce processus entre dans un isoloir vide où il n'y a plus de bulletins, y dépose B bulletins, et sort de l'isoloir.

Le comportement de ce processus est le suivant, **trouver_isoloir_sans_bulletin** étant une nouvelle primitive de synchronisation du moniteur.

```
void remplir_bulletins() {
    int i;

    while(true) {
        trouver_isoloir_sans_bulletin(int *i);
        bulletins[i] = B;
        sortir_isoloir(i);
    }
}
```

1. Modifier **entrer_isoloir** et / ou **sortir_isoloir** de telle sorte que :
   — un processus qui veux voter ne peut pas entrer dans un isoloir où il n'y a pas de bulletins.
   — un processus qui vote et qui prend le dernier bulletin signale qu'il y a un isoloir à remplir.
   Écrivez aussi la primitive **trouver_isoloir_sans_bulletin**. Quand aucun isoloir n'a besoin d'être rempli, le processus se placera en attente. Attention, le processus qui veut remplir des bulletins ne doit pas entrer dans un isoloir qui n'est pas vide.

Solution:

```
cond c_remplir;

entrer_isoloir(int *i) {
    int j;
    m.lock();

test_isoloir:

    for (j = 0; j < N ; j++ ) {
        if ( not(isoloir_occupe[j])
             && (bulletins[i] > 0) /* nouveau */
            ) {
            isoloir_occupe[j] = true;
```

```
            *i = j;
            m.unlock();
            return ;
        }
    }

    c_isoloir.wait();
    goto test_isoloir;
}

sortir_isoloir(int i) {
    m.lock();

    isoloir_occupe[i] = false;

    /* si on a pris le dernier bulletin, on reveille le processus qui
     * rempli */
    if ( bulletins[i] == 0 ) {
        c_remplir.signal();
    } else {
        c_isoloir.signal();
    }

    m.unlock();
}


trouver_isoloir_sans_bulletin(int *i) {
    int j;
    m.lock();

test_isoloir:

    for (j = 0; j < N ; j++ ) {
        if ( not(isoloir_occupe[j])
             && (bulletins[i] == 0) /* on cherche un isoloir ou il n'y
                                     * a pas de bulletins */
            ) {
            isoloir_occupe[j] = true;
            *i = j;
            m.unlock();
            return ;
        }
    }

    /* si on en a trouve aucun, on attend sur c_remplir */
    c_remplir.wait();
    goto test_isoloir;
}
```

3 Urnes

Une fois qu'un processus votant a rempli son bulletin et quitté l'isoloir, il doit mettre son bulletin dans l'urne.

On veut garantir qu'il n'y a pas de fraude quand les processus qui votent mettent leur bulletin dans l'urne.

Pour garantir qu'il n'y a pas de fraude, un processus doit mettre son bulletin dans l'urne uniquement si il y a au moins 2 autres processus prêts à mettre leur bulletin dans l'urne. Quand un groupe de processus a commencé de mettre ses bulletins dans l'urne, les autres processus doivent attendre qu'ils aient fini avant de se présenter devant l'urne.

Voici un petit exemple qui illustre ce mécanisme :

— au départ, aucun processus n'est prêt à déposer son bulletin.
— les processus $p1$ puis $p2$ arrivent devant l'urne et veulent mettre leur bulletin, mais ils ne peuvent pas car ils ont besoin d'être au moins 3.
— le processus $p3$ arrive devant l'urne, il va pouvoir permettre à $p1$, $p2$ et lui-même $p3$ à mettre leur bulletin dans l'urne.
— pendant que $p1$, $p2$ et $p3$ mettent leur bulletin, les processus $p4$, $p5$, $p6$ et $p7$ arrivent devant l'urne. Ils doivent attendre que $p1$, $p2$ et $p3$ aient fini avant de savoir s'ils peuvent mettre leur bulletins.
— une fois que $p1$, $p2$ et $p3$ ont fini, $p4$, $p5$, $p6$ et $p7$ constatent qu'ils sont suffisamment nombreux pour déposer leur bulletins. Ils commencent à déposer leur bulletin. Les processus $p8$ et $p9$ arrivent et se mettent en attente.
— $p4$, $p5$, $p6$ et $p7$ terminent de déposer leur bulletin et s'en vont. $p8$ et $p9$ ne sont pas assez nombreux pour déposer leur bulletins, ils attendent.

On considère qu'à la fin de la journée, des processus en charge de l'organisation des votes viendront compléter les derniers processus en attente de dépose de bulletins.

Pour résumer, le comportement d'un processus est maintenant le suivant :

```
vote() {
    int i;

    entrer_isoloir(&i);
    bulletins[i]--;
    /* ... choisi son candidat ... */
    sortir_isoloir(i);

    attente_devant_urne();
    /* ... depose son bulletin dans l'urne ... */
    quitter_urne();
}
```

On a un nouveau moniteur qui protège l'accès à l'urne.

1. Écrivez les primitives de moniteur **attente_devant_urne** et **quitter_urne** de manière à ce que les processus attendent d'être au moins 3 pour pouvoir déposer leur bulletin. Vous pouvez utiliser les variables globales que vous désirez.

Solution: HE : ici l'idée est de leur faire écrire un sas, où seul des groupes de plus de 3 votant peuvent passer, et si un groupe est en train de passer alors le sas est fermé.

HE : on peut aussi imaginer que l'urne doit etre vidée de temps en tant, uniquement quand aucun groupe n'est en train de mettre ses bulletins dedans. On refait un lecteurs / redacteur en fait.

HE : on pourrait peut etre leur demander de faire cette partie avec des sémaphores ?

```
vote() {
    int i;

    entrer_isoloir(&i);

    bulletins[i]--;

    /* ... choisi son candidat ... */
```

```
    sortir_isoloir(i);

    aller_devant_urne();
    deposer_bulletin();
    quitter_urne();
}


/* les noms de variables sont bof ici. */

bool sas_vide = true;

int attend_sas = 0;
int devant_urne = 0;

cond c_pas_assez;

aller_devant_urne() {
   m.lock();

    attend_sas++;

    /* bloc et reveil en cascade sur le nombre suffisant de
     * processus */
    if ( attend_sas < 3 ) {
        c_pas_assez.wait();
    }

    c_pas_assez.signal();

    /* bloque et reveil en cascade sur la disponibilite de l'urne */
    if ( not(sas_vide) ) {
        c_sas.wait();
    }

    c_sas.signal();

    sas_vide = false;
    attend_sas--;
    devant_urne++;

   m.unlock();
}

quitter_urne() {
   m.lock();

    devant_urne--;

    if( devant_urne == 0 ) {
        sas_vide = true;
        c_sas.signal();
    }

   m.unlock();
```

```
}
```

A Translation

isoloir	→	voting cubicle, voting booth
bulletin	→	ballot paper
urne	→	ballot box