Systèmes d'exploitation - Moniteurs II

Othmane AJDOR

2018-2019

Table des matières

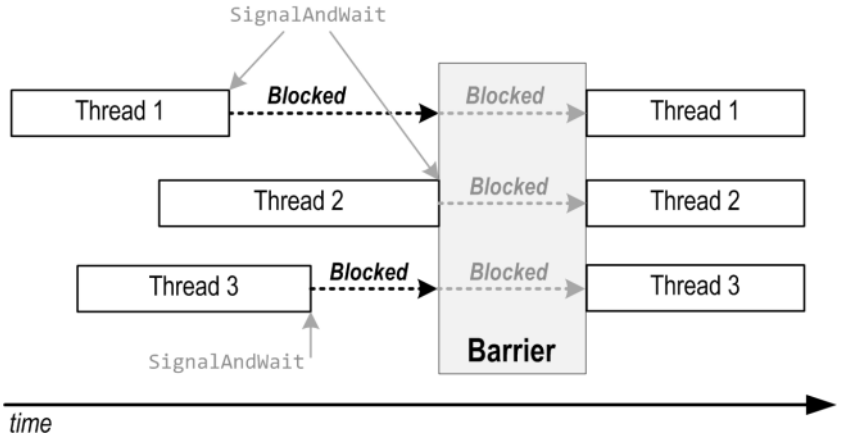
1 But

— Synchronisation de threads, 1 même programme , mémoire partagée
— Ensemble de fonctions en exclusion mutuelle (pas de problème de concurrence)

Le problème est de prendre des decisions alors qu'on est en exclusion mutuelle. On empeche le programme d'évoluer (mise en attente à l'aide de variables de condition), ainsi, on relache l'exclusion mutuelle et les threads sont placés dans une file d'attente.

2 Barrière à N

On veut que N threads appellent barriereN()



On utilise un mutex[lock|unlock], la fonction cnd_t[wait|signal].
cnd_signal(&c) réveille un thread.

```
int compteur = 0;
const N = 42;
mtx_t m;
cnd_t c;
barriereN(){
        mtx_lock();
        compteur++;
        while(compteur < N){
                cnd_wait(&c, &m);
        }
        // reveil en cascade, le N+1 reveil le premier et ainsi de suite
        cnd_signal(&c);
        mtx_unlock();
}
```

3 Allocateur

On se contente de reveiller tous les threads en attente.
cnd_broadcast(&c) reveille tous les threads.

```
const int resource = N;
mtx_t m;
cnd_t c;
allocation(int n){
        mtx_lock(&m);
        while(resource<n){
                cnd_wait(&c, &m);
        }
        resource -= n;
        mtx_unlock(&m);
}

liberation(int n){
        mtx_lock(&m);
        resource += n;
        cnd_broadcast(&c);
        mtx_unlock(&m);
}
```

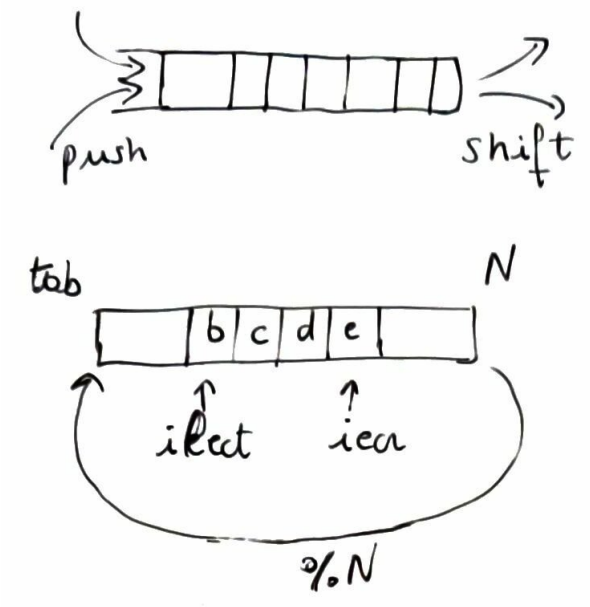
CODE CI-DESSOUS FAUX

Dans le cas où deux threads arrivent, ils se reveillent et s'endorment entre eux avec le code suivant :

```
const int resource = N;
mtx_t m;
cnd_t c;
allocation(int n){
        mtx_lock(&m);
        while(resource<n){
                cnd_wait(&c, &m);
                cnd_signal(&c);
        }
        resource -= n;
        mtx_unlock(&m);
}

liberation(int n){
        mtx_lock(&m);
        resource += n;
        cnd_signal(&c);
        mtx_unlock(&m);
}
```

4 Producteurs Consommateurs




```
int iecr = 0; // indice d'ecriture
int ilect = 0; // indice de lecture
int nb = 0;
mtx_t m;
cnd_t fp; // file de producteurs
cnd_t fc; // file de consomateurs
Msg Tab[N];

push(Msg msg){
        mtx_lock(&m);
        while (nb == N){
                cnd_wait(&fp, &m)
        }
        tab[iecr] = msg;
        nb++;
        iecr++;
        iecr %= N;
        cnd_signal(&fc)
        mtx_unlock(&m);
}
```

```
Msg shift(){
        mtx_lock(&m);
        while (nb == 0){
                cnd_wait(&c, &m);
        }
        Msg msg = tab[ilect];
        nb--;
        ilect++;
        ilect %= N;
        cnd_signal(&fp);
        mtx_unlock(&m);
        return msg;
}
```

Les modifications des indices ilect et iecr doivent être faites au même endroit pour lire/ecrire dans la même case du tableau.

5 Lecteurs-Redacteurs (shared mutex)

On dispose d'une base de données où on stock des informations, on autorise les lectures à plusieurs mais les ecritures seront limités à 1 thread à la fois et il sera tout seul (pas de lecture, pas d'autres redacteurs).

On utilise un compteur de lecteurs et un compteurs de redacteurs.
Lorsqu'un redacteur finit d'ecrire, il faut donner la priorité aux redacteurs. L'inconvenient de cela est lorsqu'il y a beaucoup d'ecrivains, les lecteurs ne passent jamais. Cette priorité peut donc causer une famine de lecture. On peut aussi créer une famine d'écriture si la priorité aux lecteurs est choisie (pire qu'une famine de lecture).

Pour reveiller la bonne personne, on a besoin de compter les lecteurs et redacteurs en attente.

```
int nblect;
bool ecr;
int nblect_att;
int nbecr_att;
mtx_t m;
cnd_t clect;
cnd_t cecr;

debut_lire(){
        mtx_lock(&m);
        while (nbecr_att > 0 || ecr){
                nblect_att++;
                cnd_wait(&clect, &m);
                nblect_att--;
        }
        nblect++;
        mtx_unlock(&m);
}

fin_lire(){
        mtx_lock(&m);
        if (nblect_att != 0 && nblect == 0){
                cnd_signal(&cecr);
        }
        mtx_unlock(&m);
}

debut_ecr(){
        mtx_lock(&m);
        while (ecr || nblect != 0){
                nbecr_att++;
                cnd_wait(&cecr, &m)
                nbecr_att--;
```

```
        }
        ecr = true;
        mtx_unlock(&m);
}

fin_ecr(){
        mtx_lock(&m);
        ecr = false;
        if (nbecr_att > 0){
                cnd_signal(&cecr);
        } else {
                if (nblect_att > 0){
                        cnd_broadcast(&clect);
                }
        }
        mtx_unlock(&m);
}
```

Si on retire les compteurs des lecteurs et redacteurs en attente, le code devient :

```
const int resource = N;
mtx_t m;
cnd_t c;
allocation(int n){
        mtx_lock(&m);
        while(resource<n){
                cnd_wait(&c, &m);
        }
        resource -= n;
        mtx_unlock(&m);
}

liberation(int n){
        mtx_lock(&m);
        resource += n;
        cnd_broadcast(&c);
        mtx_unlock(&m);
}
```