Moniteurs C/POSIX

Synchronisation à l'aide des Moniteurs Posix

Samia Bouzefrane

Maître de Conférences

CEDRIC –CNAM

samia.bouzefrane@cnam.fr
http://cedric.cnam.fr/~bouzefra

Mécanismes étudiés

- **Mutex Posix (sémaphore binaire)**

- **Moniteurs Posix : variables conditionnelles associées aux Mutex**

Gestion des Mutex

• Un « **Mutex** » est **un sémaphore binaire** pouvant prendre un des deux états

• **"lock"** (verrouillé) ou **"unlock"** (déverrouillé): valeur de sémaphore 1 ou 0

• Un « **Mutex** » ne peut être partagé que par des threads d'un même processus

• Un « **Mutex** » ne peut être verrouillé que par une seule thread à la fois.

• Une thread qui tente de verrouiller un « Mutex » déjà verrouillé est suspendu jusqu'à ce que le « Mutex » soit déverrouillé.

Déclaration et initialisation d'un Mutex

• Un mutex est une variable de type « **thread_mutex_t** »

• Il existe une constante **PTHREAD_MUTEX_INITIALIZER** de ce type permettant une déclaration avec initialisation statique du mutex (avec les valeurs de comportement par défaut)

```
pthread_mutex_t monMutex = THREAD_MUTEX_INITIALIZER;
```

• Un mutex peut également être initialisé par un appel de la primitive

```
int pthread_mutex_init(pthread_mutex_t *mutex,const
pthread_mutexattr_t *mutexattr);
```

avec une initialisation par défaut lorsque mutexattr vaut NULL

ex : `pthread_mutex_init(&monMutex, NULL);`

Prise (verrouillage) d'un mutex

• Un mutex peut être **verrouillé** par la primitive

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

• Si le mutex est **déverrouillé** il devient **verrouillé**

• Si le mutex est déjà verrouillé par une autre thread la tentative de verrouillage **suspend** l'appelant jusqu'à ce que le mutex soit déverrouillé.

Relâchement (déverrouillage) d'un mutex

• Un mutex peut être déverrouillé par la primitive

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

• Si le mutex est déjà déverrouillé, cet appel n'a aucun effet (comportement par défaut)

• Si le mutex est verrouillé, une des threads en attente obtient le mutex (qui reprend alors l'état verrouillé) et cette thread redevient active (elle n'est plus bloquée)

• L'opération est toujours *non bloquante pour l'appelant*

Exemple d'utilisation de mutex

```
#define N      10  /* Nb de cases du tampon */

pthread_mutex_t monMutex = PTHREAD_MUTEX_INITIALIZER;
int tampon[N];



void Deposer(int m){
pthread_mutex_lock(&mutex);
        // déposer m dans tampon;
        ……

pthread_mutex_unlock(&mutex);
}

……..
```

On verrouille le Mutex : accès exclusif

On déverrouille le Mutex

Moniteurs Posix

• **Un moniteur Posix est l'association**

– d'un mutex (type **`pthread_mutex_t`** **)** qui sert à protéger la partie de code où
l'on teste les conditions de progression
– et **d'une variable condition (** type **`pthread_cond_t`** **)** qui sert de point de signalisation :

• on se met en attente sur cette variable par la primitive :
Libération du mutex + Blocage systématique de l'appelant de manière atomique

```
pthread_cond_wait(&laVariableCondition,&leMutex);
```

• on est réveillé sur cette variable avec la primitive :
Réveil d'une thread en attente qui acquiert à nouveau le mutex

```
pthread_cond_signal(&laVariableCondition);
```

Schéma d'utilisation

- **Soit la condition de progression C,**
- **Le schéma d'utilisation des moniteurs Posix est le suivant :**

```
pthread_mutex_lock (&leMutex);
évaluer C;
while  ( ! C ) {
        pthread_cond_wait(&laVariableCondition,&leMutex);
        ré-évaluer C si nécessaire
}

Faire le travail;
pthread_mutex_unlock(&leMutex);
```

Exemple du Prod/Cons avec les moniteurs Posix

```
#include <pthread.h>

/* définition du tampon */
#define N      10  /* Nb de cases du tampon */
#define NbMess 20  /* Nb de messages échangés */
int NbPleins=0;
int tete=0, queue=0;
int tampon[N];

/* définition des conditions et du mutex */
pthread_cond_t  vide;
pthread_cond_t  plein;
pthread_mutex_t mutex;

pthread_t tid[2];
```

Exemple (suite)

```
void Deposer(int m){
pthread_mutex_lock(&mutex);
        if(NbPleins == N) pthread_cond_wait(&plein, &mutex);
        tampon[queue]=m;
        queue=(queue+1)%N;
        NbPleins++;
        pthread_cond_signal(&vide);
pthread_mutex_unlock(&mutex);
}

int Prelever(void){
int m;
pthread_mutex_lock(&mutex);
        if(NbPleins ==0) pthread_cond_wait(&vide, &mutex);
        m=tampon[tete];
        tete=(tete+1)%N;
        NbPleins--;
        pthread_cond_signal(&plein);
pthread_mutex_unlock(&mutex);
return m;
}
```

Exemple (suite)

```
void * Prod(void * k)          /********** PRODUCTEUR */
{
int i;
int mess;
srand(pthread_self());
for(i=0;i<=NbMess; i++){
        usleep(rand()%10000);  /* fabrication du message */
        mess=rand()%1000;
        Deposer(mess);
        printf("Mess depose: %d\n",mess);
        }
}
void * Cons(void * k)          /********** CONSOMMATEUR */
{
int i;
int mess;
srand(pthread_self());
for(i=0;i<=NbMess; i++){
        mess=Prelever();
        printf("\tMess preleve: %d\n",mess);
        usleep(rand()%1000000);  /* traitement du message */
        }
 }
```

Exemple (fin)

```
void main()                          /* M A I N */
{
int i,  num;
pthread_mutex_init(&mutex,0);
pthread_cond_init(&vide,0);
pthread_cond_init(&plein,0);

/* creation des threads */
pthread_create(tid,  0, (void * (*)()) Prod, NULL);
pthread_create(tid+1, 0, (void * (*)()) Cons, NULL);

// attente de la fin des threads
pthread_join(tid[0],NULL);
pthread_join(tid[1],NULL);

// libération des ressources
pthread_mutex_destroy(&mutex);
pthread_cond_destroy(&vide);
pthread_cond_destroy(&plein);

exit(0);
}
```

Références

Jean-François Peyre, supports de cours sur l'informatique industrielle-systèmes temps réel, CNAM(Paris).

Samia Bouzefrane, LES SYSTEMES D'EXPLOITATION: COURS ET EXERCICES CORRIGES UNIX, LINUX et WINDOWS XP avec C et JAVA (566 pages), Dunod Editeur, Octobre 2003, ISBN : 2 10 007 189 0.