TP3_PBT

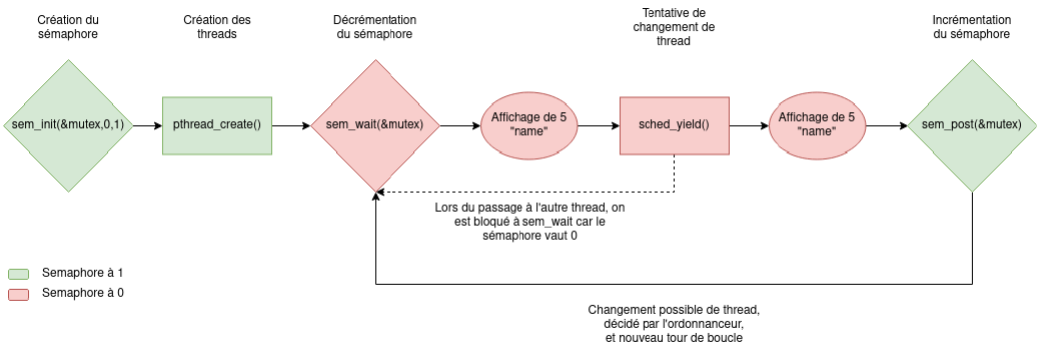
Night

March 2023

1 Sémaphore

1.1 q1

- Le programme crée deux threads, filsA et filsB, qui exécutent tous deux la même fonction affichage en parallèle.
- Le sémaphore sert à synchroniser l'accès à la sortie standard entre deux threads en parallèle. Les threads affichent des caractères en alternance, chacun affichant 10 caractères en tout.
- À chaque itération de la boucle for, les threads utilisent sem_wait(mutex) pour tenter de verrouiller le sémaphore mutex. Si le sémaphore a une valeur de 1, le verrouillage est réussi et le thread peut accéder à la section critique, qui est l'affichage sur la sortie standard.
- Une fois que le thread a verrouillé le sémaphore et a accédé à la section critique, il affiche 5 caractères correspondant au nom du thread (name).
- La fonction sched_yield est utilisée pour éviter qu'un thread ne monopolise le temps d'exécution du processeur.
- Le thread affiche ensuite à nouveau 5 caractères correspondant au nom du thread (name).
- Enfin, le thread utilise sem_post(mutex) pour déverrouiller le sémaphore mutex, ce qui permet à d'autres threads d'accéder à la section critique.
- Les threads répètent ce processus 20 fois avant de se terminer.



2 Mise en place des sémaphores dans mthread

Fichier **mthread_sem.c**:

2.1 q2

On définit un sémaphore **struct mthread_sem_s** par une lock, une value et un max.

```
int mthread_sem_init(mthread_sem_t *sem, unsigned int value)
{
    if (sem == NULL || value == 0) {
        return EINVAL;
    }

    sem->lock = 0;
    sem->value = value;
    sem->max = value;

    return 0;
}
```

sem_init renvoie EINVAL si le sémaphore passé en entrée est NULL ou si la value donnée est à 0, afin d'éviter de produire un sémaphore inutilisable. Dans le cas contraire, lock est mis à 0 , le sémaphore n'étant pas encore utilisé. value et max sont mis à la valeur donné en argument.

2.2 q3

sem_wait est utilisée pour tenter de verrouiller un sémaphore. Si le sémaphore est actuellement verrouillé par un autre thread, sem_wait mettra le thread en attente jusqu'à ce que le sémaphore soit déverrouillé.

```
int mthread_sem_wait(mthread_sem_t *sem)
{
    if (sem == NULL) {
        return EINVAL;
    }

    mthread_spinlock_lock(&sem->lock);

    while (sem->value == 0) {
        mthread_yield();
    }

    sem->value--;

    mthread_spinlock_unlock(&sem->lock);

    return 0;
}
```

sem_wait renvoie EINVAL si le sémaphore passé en entrée est NULL . Le lock est ensuité récupéré. Si value est encore non-nulle, elle est décrémentée, le lock est libéré et la fonction renvoie 0. Sinon, le thread attends que value devienne positive puis la décrémente à nouveau.

2.3 q4

```
int mthread_sem_post(mthread_sem_t *sem)
{
    if (sem == NULL) {
        return EINVAL;
    }

    mthread_spinlock_lock(&sem->lock);
```

```
    sem->value++;

    mthread_spinlock_unlock(&sem->lock);

    return 0;
}
```

sem_post renvoie EINVAL si le sémaphore passé en entrée est NULL . post incrémente value avoir pris le lock, puis le libère.

2.4 q5

```
int mthread_sem_destroy(mthread_sem_t *sem)
{
  if (sem == NULL)
    return EINVAL;

  mthread_spinlock_lock(&sem->lock);
  if (sem->value != sem->max) {
     mthread_spinlock_unlock(&sem->lock);
    return EBUSY;
  }

  mthread_spinlock_unlock(&sem->lock);

  return 0;
}
```

sem_destroy renvoie EINVAL si le sémaphore passé en entrée est NULL . Le sémaphore n'est libéré que si value est égale à max , afin de s'assurer qu'il ne soit pas détruit pendant utilisation, sinon la fonction retourne EBUSY (ainsi on a un comportement similaire à mthread_mutex_destroy).

2.5 q6

```
int mthread_sem_trywait(mthread_sem_t *sem)
{
  if (sem == NULL)
    return EINVAL;

  if (sem->value == 0)
    return EBUSY;

  return mthread_sem_wait(sem);
}
```

sem_trywait renvoie EINVAL si le sémaphore passé en entrée est NULL et renvoie EBUSY si value est à 0. Sinon la méthode agit comme sem_wait .

2.6 q7

```
int mthread_sem_getvalue(mthread_sem_t *sem, int *sval)
{
  if (sem == NULL || sval == NULL)
    return EINVAL;

  mthread_spinlock_lock(&sem->lock);
```

```
  *sval = sem->value;
  mthread_spinlock_unlock(&sem->lock);

  return 0;
}
```

sem_getvalue renvoie EINVAL si le sémaphore ou sval passé en entrée est NULL . Sinon la méthode place value dans *sval .

3 Démonstration

3.1 q8

Voir test.c