Lecteurs/rédacteurs

09 février 2024

On souhaite gérer un système de concurrence lecteurs/rédacteurs où plusieurs lecteurs et plusieurs rédacteurs agissent en parallèle sur un document. Les lecteurs se contentent de lire le document et les rédacteurs le modifient. Les lecteurs peuvent donc être plusieurs à lire simultanément sans que cela n'implique de problème de concurrence à condition qu'aucun rédacteur ne soit en train de modifier le document. Un seul rédacteur à la fois peut travailler sur le document et si un rédacteur est en train de travailler sur le document alors aucun lecteur ne peut plus y accéder. On propose l'utilisation de la structure C suivante où on dispose d'un sémaphore binaire (initialisé à 1). La variable `nbreaders` est partagée et compte le nombre de lecteurs en train de consulter le document.

```
typedef struct rwlock_t{
   sem_t writelock;
   int nbreaders;
} rwlock_t;

void rwlock_init(rwlock_t* l){
   sem_init(&(l->writelock),0,1);
   nbreaders=0;
}
```

Nous allons écrire une fonction d'accès (`rwlock_acquire_readlock`) et une fonction de sortie de la section de lecture (`rwlock_release_readlock`) ainsi qu'une fonction d'accès (`rwlock_acquire_writelock`) et une fonction de sortie (`rwlock_release_writelock`) d'une section d'écriture qui vont respecter les contraintes. Voici une première proposition :

```
1void rwlock_acquire_readlock(rwlock_t *lock) {
2
3 lock->nbreaders++;
4 if (lock->nbreaders == 1)
5      sem_wait(&lock->writelock);
6
7}
8
9 void rwlock_release_readlock(rwlock_t *lock) {
10
11 lock->nbreaders--;
12 if (lock->nbreaders == 0)
13     sem_post(&lock->writelock);
14
15}
16
17 void rwlock_acquire_writelock(rwlock_t *lock) {
18    sem_wait(&lock->writelock);
19 }
10
21 void rwlock_release_writelock(rwlock_t *lock) {
22    sem_post(&lock->writelock);
23 }
```

1. Une erreur s'est glissée dans la fonction `rwlock_init`. L'identifier et la corriger.
2. Aurait-on pu utiliser un autre type de variable à la place du sémaphore `write_lock` ?
3. Expliquer, en français, les rôles des lignes 4 et 5 ainsi que 12 et 13.
4. Expliquer pourquoi deux rédacteurs ne pourront pas écrire simultanément ?

5. Pourquoi est ce que tous les agents (lecteurs et rédacteurs) utilisent le même sémaphore ?
6. Quel problème de concurrence apparait dans le code proposé ? Expliquer comment modifier le code pour résoudre ce problème.
7. On considère maintenant les fonctions `reader` et `writer` suivantes. Que représente la variable `counter` ?

```
int read_loops;
int write_loops;
int counter = 0;

rwlock_t mutex;

void* reader(void *arg) {
    int i;
    int local = 0;
    for (i = 0; i < read_loops; i++) {
        rwlock_acquire_readlock(&mutex);
        local = counter;
        rwlock_release_readlock(&mutex);
        printf("read %d\n", local);
    }
   printf("read done\n");
   return NULL;
}

void* writer(void *arg) {
    int i;
    for (i = 0; i < write_loops; i++) {
         rwlock_acquire_writelock(&mutex);
         counter++;
         rwlock_release_writelock(&mutex);
    }
    printf("write done\n");
    return NULL;
}
```

8. Recopier et compléter le `main` pour lancer les fonctions précédentes :

```
1 int main(int argc, char *argv[]) {
2    if (argc != 3) {
3          fprintf(stderr, "usage: rwlock readloops writeloops\n");
4          exit(1);
5    }
6    read_loops = atoi(argv[1]);
7    write_loops = atoi(argv[2]);
8
9    rwlock_init(&mutex);
10
11   pthread_create(           , NULL,             , NULL);
12   pthread_create(           , NULL,             , NULL);
13
14
 15  return 0;
16 }
```

9. Supposons que `read_loops=5` et `write_loops=3`. Parmis les affichages suivants, dire, en justifiant, lesquels correspondent à une exécution possible :

(a)
```
read 0
read 3
read 3
read 3
read 3
read 3
read done
write done
```

(b)
```
read 3
read 3
read 3
read 3
read 3
read done
```

```
write done
```

(c)
```
read 3
read 3
write done
read 3
read 3
read 3
read done
```

(d)
```
read 0
read 1
read 2
read 2
read 3
read done
write done
```

(e)
```
read 1
read 3
read 2
read 3
read 3
read done
write done
```

(f)
```
read 0
read 0
read 0
read 0
read 0
read done
write done
```

(g)
```
read 0
read 2
write done
read 3
read 3
read 3
read done
```

(h)
```
read 0
read 2
write done
read 2
read 3
read 3
read done
```