Série d'exercices #7

IFT-2245

19 février 2019

7.1 Famine

Comment un système peut-il détecter que certains de ses threads sont en état de famine, ou sinon comment peut-il éviter ou tenter de gérer ce problème ?

7.2 État sûr et interblocage

Développer un scénario où un système commence dans un état *unsafe* et où malgré cela, l'exécution des threads termine sans rencontrer d'interblocage.

7.3 Lockfree 1

Soit le code suivant qui permet d'insérer des éléments dans un arbre binaire :

```
struct tree {
  int key;
  void *val;
  struct tree *smaller, *larger;
}

struct tree *tree_insert (struct tree *t, int key, void *val)
{
  struct tree *n = malloc (sizeof (struct tree));
  if (t == NULL) {
    n->key = key; n->val = val;
    n->smaller = NULL; n->larger = NULL;
  } else if (key == t->key) {
    n->key = key; n->val = val;
    n->smaller = t->smaller; n->larger = t->larger;
  } else if (key < t->key) {
    n->key = t->key; n->val = t->val;
    n->larger = t->larger;
    n->smaller = tree_insert (t->smaller, key, val);
    return n;
  } else { /* key > t->key */
    n->key = t->key; n->val = t->val;
```

```
    n->smaller = t->smaller;
    n->larger = tree_insert (t->larger, key, val);
  }
  return n;
}

void tree_set (struct tree **t, int key, void *val)
{
  *t = tree_insert (*t, key, val);
}
```

1. Trouver les conditions de course présentes si le code est utilisé dans une application à plusieurs *threads*.
2. Corriger ces conditions de course en ajoutant les verrous et opérations correspondantes nécessaires. Décrire clairement quelles données sont protégées par chaque verrou.
3. Corriger ces mêmes conditions de course sans utiliser de verrous, en utilisant à la place une approche de synchronisation optimiste, en utilisant l'opération `compare&swap`.