Informatique embarquée : TP n°5

Luxon JEAN-PIERRE

Temps estimé : ~ 6 heures.

Temps passé : ~ 7 heures.

Objectif :

L'idée est de créer un programme qui puisse faire un inversion de priorité, c'est-à-dire de faire en sorte que le thread A (de faible priorité) prenne le verrou avant C (de haute priorité), que C soit bloqué lors de sa tentative de prise du mutex, et que A soit préempté par B, pendant que C est bloqué sur le mutex.

Méthode de travail :

Dans un premier temps, je fais un programme avec un thread principal qui va créer trois threads enfant A, B et C. Chaque thread enfant aura une priorité spécifique. A aura une priorité faible, B aura une priorité moyenne, et C aura une haute priorité.

La mise en place des priorités a été faite en utilisant la fonction *pthread_attr_setschedparam*. J'ai appliqué la politique d'ordonnancement de type priorité fixe FIFO via l'appel de fonction *pthread_attr_setschedpolicy*. Il m'a également fallu désactiver l'héritage de priorité afin que les priorités définies pour chacun des 3 threads soit bien appliquées.

Dans les threads A et C, je lance une fonction qui va prendre un mutex, allez en section critique et exécuter du code, puis relâcher le mutex. Le thread B va, de son côté, exécuter son code, indépendamment de ce que font A et B.

Au niveau des threads, je définie une variable globale *v* (initialisé à 0) qui va stocker une valeur relative à l'état de C. Lorsque C va essayer de prendre le mutex, *v* vaut 1. Si C parvient à avoir le mutex, alors *v* vaudra 2. Et lorsque C quitte la section critique, v est remis à 0.

Au niveau du thread principal, je créer les trois threads dans l'ordre suivant : A, C , B. L'application de la politique d'ordonnancement de type priorité fixe FIFO fait que les threads A, B, et C seront bien exécutés dans l'ordre indiqué plus haut.

Critiques :

Il est possible que le programme ne donne pas le résultat attendu sur des machines à processeur multicœurs. Pour autant, lorsque j'ai lancé le programme en indiquant