**Projet de Programmation Système**

***Un ordonnanceur par work stealing***

- Architecture

*deque.h* et *deque.c* implémente la deque pour LIFO et WorkStealing

*sched.h* définit la structure scheduler

*sched.c* implémente LIFO

workstealing.c implémente Workstealing

*quicksort.c* contient main, la fonction quicksort et benchmark

*trace.gnuplot* définit comment dessiner le graph de benchmark

*generate.sh* génère le graph pour comparer LIFO et WorkStealing

- Implémentation

Deque : une méthode de push et deux méthodes de pop ( en haut et en bas), dans LIFO, tous les threads prennent des tâches en bas, dans Workstealing, quand ils volent, prend une tâche en haut. On utilise mutex dans pop et push.

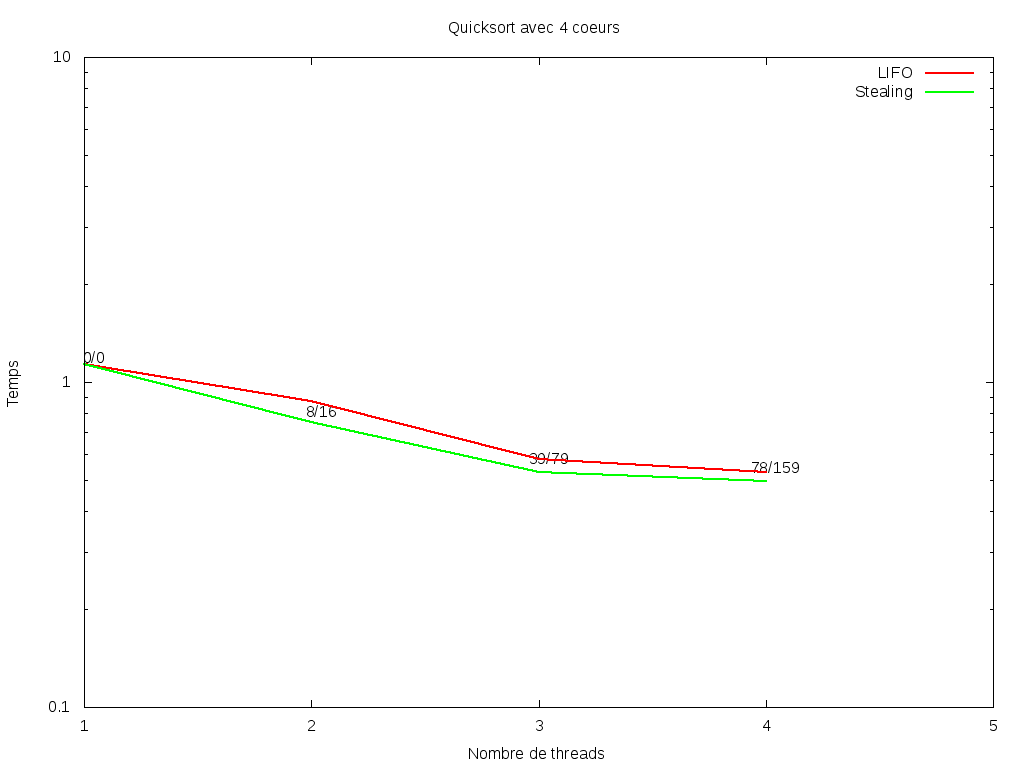
LIFO: seulement une deque(sched->deques[0]), push toutes les tâches dans cette deque, dans thread\_func(), tous les threads essaient de prendre une tâche de cette deque. S’il trouve pas et aucun thread est en train de travail, il va sortir de boule infini while.

WorkStealing: le nombre de deques dépend le nombre de threads, chaque thread a sa deque, quand on veut faire un push dans une deque, on va utliser pthread\_getspecific() à trouver quelle deque on doit l’enfiler(c’est à dire localité). dans thread\_func(), tous les threads d’abord essaient de trouver une tâche dans sa deque, s’il trouve pas il vont essayer de voler, s’il trouve pas encore est aucun thread est en train de travail, il va sortir le boucle infini while.

- Benchmark

La valeur à côté de la ligne verte(WorkStealing) : *M / N* (*M* - le nombre d’étapes de work stealing qui réussissent*, N* – *M* le nombre d’étapes de work stealing qui échouent ).

Avec la machine de 4 coeurs:



Avec la machine de 32 coeurs(chez serveur lulu):

