Projet Master/Slaves

RS : Réseaux et Systèmes
Deuxième année

Évaluation de ce projet

Une fois n'est pas coutume, ce projet ne sera pas évalué de manière semi-automatique au travers d'un jeu de tests. Il sera tout de même compilé et testé manuellement. Un projet ne compilant pas correctement sera sanctionné par une note adéquate. Il s'agit d'un travail personnel (et non en binôme).

Vous devez rendre un mini-rapport de projet (5 pages maximum, format pdf). Vous y détaillerez les difficultés auxquelles vous avez été confronté, et comment vous les avez résolues. Vous indiquerez également le nombre d'heures passées sur les différentes étapes de ce projet (conception, codage, tests, rédaction du rapport).

La tricherie sera **sévèrement punie**. Voir `http://www.loria.fr/~quinson/teaching.html#triche`

Comment rendre votre projet

En plus du mini-rapport et de vos sources, vous devez fournir un makefile dont la cible par défaut compilera votre projet. Pour rendre votre «copie», vous devez placer les fichiers nécessaires dans un répertoire sur neptune, vous placer dedans, et invoquer `/home/EqPedag/quinson/bin/rendre_projet RS` (seuls les fichiers sources (et le pdf) sont copiés)

Avant le lundi 26 novembre à 23:59
(la commande ne fonctionnera plus après)

Voir `http://www.loria.fr/~quinson/teach-RS.html` d'éventuelles informations complémentaires.

1 Projet MasterSlaves

Une manière de déterminer les nombres premiers dans un intervalle est de découper l'espace de recherche en plusieurs sous-intervalles et d'affecter la recherche dans chaque intervalle à un processus créé par le père. Ainsi si on découpe l'intervalle $[1, N]$ en p sous-intervalles, le processus père lance p fils et le fils k recherche l'intervalle $[kN/p + 1, (k + 1)N/p], k = 0, ..., p - 1$. Cette technique permet, si on dispose d'une machine équipée de p processeurs, de paralléliser la recherche (et d'essayer de retourner le résultat p fois plus vite). Elle a cependant l'inconvénient d'affecter des espaces de recherche dont les temps d'exploration sont inégaux. Ainsi le processus 0 terminera la recherche sur $[1, N/p]$ bien avant le processus $p-1$. Il (et donc un des processeurs de la machine) sera donc oisif jusqu'à la fin du programme. La charge de travail entre les processus (et donc les processeurs) est inégalement repartie.

Une solution (celle que vous devez programmer) pour repartir la charge de travail, consiste a utiliser un « pool » de p processus travailleurs à qui un processus maître affecte successivement des travaux de recherche de nombres premiers dans de petits intervalles (taille $T << N/p$). Quand un travailleur a fini sa recherche, le maître lui affecte la recherche dans un intervalle encore inexploré.

Notez que l'algorithme important ici est celui de répartition du travail et non celui de recherche des nombres premiers (qui n'est qu'une excuse). La façon préconisée pour déterminer si le nombre k est premier est donc de calculer le reste de sa division entière avec tous les nombres i tels que $1 < i < k$.

1.1 Pistes pour l'implémentation

- Le maître est le processus père, et les p travailleurs sont des fils lancés par le père grâce à `fork()`.
- Le père utilise un tube différent pour parler à chacun de ses fils, tandis que tous les fils utilisent le même tube pour parler au maître.
- Le protocole de communication est le suivant :

  `père → fils` deux entiers représentant l'intervalle de recherche

  `fils → père` chaque nombre premier trouvé (0 quand il a terminé l'exploration de l'intervalle alloué)

  Pour s'identifier auprès du père, un fils précède chaque nombre envoyé par son numéro
- Le père récupère des arguments de la ligne de commande l'intervale à faire explorer par l'ensemble de ses fils, ainsi que la quantité de fils à lancer.
- Quand tout l'intervale est exploré, le père tue ses fils d'un envoi de signal approprié.