ENSM-SE

B-DAT

TP Données massives

9 mars 2018

Michel Beigbeder

Ce travail pratique sera à rendre par groupe de deux personnes : programme source avec les commentaires intégrés à l'intérieur. Date à définir. Les groupes seront imposés.

Exercice 1 Tri à grande échelle, tri externe

Le but du travail pratique est un programme qui permet de trier un fichier de très grande taille. Plus précisément, la taille du fichier est de plusieurs (minimum 5 fois, jusqu'à 10-100-1000 fois) la taille de la mémoire physique disponible. Les fichiers à trier contiendront des paires de deux entiers écrits en binaire.

Le programme réalisé ne devra pas charger les données depuis le fichier par des opérations de lecture (typiquement `read(2)` ou `fread(3)`) mais en « projetant » le fichier en mémoire (`mmap(2)`). Il faudra connaître la taille du fichier avant sa projection en mémoire ce que l'on obtiendra avec l'appel système `fstat(2)`.

La procédure générale est de trier des empans du fichier fourni d'une taille S (S comme *size*) inférieure à la taille de la mémoire centrale. Le tri se fera sur la projection et on fera en sorte qu'il n'y ait pas d'allocation mémoire dans cette phase. Pour cela étudiez les options de `mmap(2)` (et de `open(2)` qui est un préalable à `mmap(2)`). Comme effet peut-être indésirable, le fichier de données sera modifié.

Pour l'adaptation de votre programme à l'une ou l'autre machine qui n'ont pas toutes la même taille de mémoire centrale, votre programme admettra une option `-s` qui permettra de donner la taille S des empans à trier indépendamment.

Pour les opérations de tri en mémoire centrale on utilisera la fonction disponible dans toutes les distributions de Unix, à savoir `qsort(3)`. (**Nota bene :** malgré son nom qui pourrait faire croire que qsort implémente un tri rapide (*quick sort* en anglais) n'importe quel algorithme de tri pourrait y être utilisé. Cependant toutes les implémentations que j'ai rencontrées sont très efficaces et en $O(n \cdot \log n)$.)

Pour la fusion des empans, il est impératif d'utiliser un **tas** et donc de programmer les algorithmes de percolation dans un tas. Le résultat de la fusion sera écrit sur la sortie standard en binaire ou en ASCII si l'option `-B` est donnée.

Exercice 2 Documentation

- le polycopié du cours d'algorithmique de première année, *Algorithmes de tris* de **Roland Jégou**.
- *Le langage C, norme ANSI* **Brian W. Kernighan** et **Dennis M. Ritchie**, Dunod 2014. En bibliothèque.
- *Langage C* **Gerhard Willms** Micro application 1996. Un peu ancien par rapport aux normes et usages actuels. En bibliothèque.
- *UNIX* **Michael Wielsch** Micro application 1997. En bibliothèque.
- *Programmation système en C sous Linux* **Christophe Blaess** Eyrolles 2003. En bibliothèque.
- le manuel en ligne de **votre** machine, par exemple `man 2 mmap`, `man 2 open`, `man 2 fstat`, `man 3 qsort`, etc.
- `https://fr.wikipedia.org/wiki/Hi%C3%A9rarchie_de_m%C3%A9moire`
- `https://www.rambus.com/blogs/understanding-the-memory-storage-pyramid-2/`
- `https://fr.wikipedia.org/wiki/Solid-state_drive`
- `https://fr.wikipedia.org/wiki/Tas_(informatique)`
- `http://www.cs.nthu.edu.tw/~wkhon/algo08-tutorials/tutorial-ext-mem.pdf`, ces diapositives présentent la méthode avec des lectures (`read(2)`) lesquelles sont implicites avec la projection que nous utilisons.
- `http://www.csd.uoc.gr/~hy460/2017-2018-fall/tutorials/Assisting_4_C460SortMergeTutorial17.pdf`
- `http://www-inst.eecs.berkeley.edu/~cs186/sp08/notes/08-Sorting.pdf`

Exercice 3 Programmes fournis

Le programme `generate` génère des paires de nombres pseudo-aléatoires et les écrit sur la sortie standard soit formatté en décimal avec des chiffres ASCII, soit en binaire si l'option `-B` est donnée sur la ligne de commande. L'option `-s` permet de donner le nombre de paires souhaitées, par défaut 10. Le nombre de paires spécifié par l'option `-s` est soit un nombre en décimal, soit un nombre décimal suivi d'une lettre `k`, `M` et `G`, laquelle applique un coefficient multiplicateur de $1\,024$, $1\,024^2$, $1\,024^3$ respectivement. Dans l'écriture en ASCII, il y a aussi une numérotation décroissante des lignes écrites.

Le programme `bin2asc` lit sur son entrée standard les octets de nombres entiers binaires sur 32 bits (écrits avec `fwrite(2)`) et les écrit en décimal par paquet de n, nombre spécifié par l'option `-n`, par défaut 1.

Mode d'emploi : Extraire le contenu de l'archive (`tar zxvf BD-1-SORT.tgz`). Compiler les pro-

grammes (`make`).

Exercice 4 Interface de votre programme

Votre programme prendra un argument obligatoire : le nom du fichier à trier. Votre programme aura pour effet de bord de modifier le fichier indiqué (c'est très rare qu'une commande Unix non interactive ait cet effet de bord, il y a par exemple la commande `sed` utilisée avec l'option `-i` qui modifie les fichiers de données.) Cette modification est la contrepartie de l'absence d'allocation dans la première phase de l'algorithme.

De plus votre programme aura une option `-s` qui permettra de donner sur la ligne de commande la taille des empans qui sont triés avec `qsort(3)`. Et enfin l'écriture du résultat du tri se fera en ASCII ou en binaire si l'option `-B` est donnée.

Exemple d'utilisation :

```
$ time ./generate -s 1M -B > file.dat
[...]
$ ls -lh file.dat
-rw-r--r-- 1 user     group       8,0M mars   9 11:11 file.dat
$ ./bin2asc -n2 < file.dat | more
[...]
$ time ./largesort -s 1k -B file.dat > file.sorted
[...]
$ ls -lh file.sorted
-rw-r--r-- 1 user     group       8,0M mars   9 11:13 file.sorted
$ ./bin2asc -n2 < file.sorted | more
[...]
```

Exercice 5 Rendu du jour

Prévoir un découpage en tâches (cf. méthode agile) avec des « livrables » qui peuvent être des programmes de quelques lignes pour tester tel ou tel aspect, telle ou telle fonction, de la lecture (active !), etc.