

Rapport projet de système : Simulation d'un contrôleur de disque.

I. Fonctionnalités.

* Les trois programmes gèrent les erreurs de type “erreurs de frappe”, sur les arguments P, S, N, G, comme : P= “000”, “1a”, “-2”, “ab”, ou encore des arguments plus grand que INT_MAX (valeur max d'un int dans /usr/include/limits.h).

* Le programme client lance gnuplot sur le fichier de requêtes créé par défaut, cf. les fichiers setps.glt et exemple.ps pour conserver l'image en postscript. Avec l'option “-q” en fin de ligne de commande, client ne lance pas gnuplot.

* L'initialisation de srand() dans client se fait à la microseconde, via la fonction gettimeofday(). Avantage : pour le programme simulation, même quand les valeurs de P, S, N, G sont petites, les clients sont lancés avec des initialisations différentes de srand(). Inconvénients : la fonction gettimeofday() n'est pas POSIX, “seulement” SVr4, BSD 4.3, de plus elle n'a pas fonctionné chez moi à l'intérieur d'une autre fonction que main(). C'est pourquoi au cas où elle ne fonctionnerait pas on utilise times() (qui donne du centième de seconde, insuffisant pour deux clients lancés quasi-simultanément via simulation, mais bon...).

* Il est possible de placer l'éventuelle option “-S” ou “-C” où l'on veut sur la ligne de commande au lancement de serveur ou simulation.

II. Améliorations envisageables.

* Proposer d'autres options de tris dans serveur, voire trouver (et prouver) une option optimale suivant le client (bon courage...).

* Dans simulation donner le nombre de clients en argument, (pas seulement toujours trois clients).

III. Bugs.

Lancer à la ligne de commande : “serveur P S fichier > fichier_”. (avec fichier=fichier de requêtes).

Ceci tourne en boucle et remplit son quota de disque (avec fichier_). Mais il faut le vouloir...

IV. Comparaison des différents modes.

Le mode par défaut calcule les dates de traitement dans l'ordre dont sont produites les requêtes. C'est toujours lui le plus coûteux en temps (sauf cf. remarque).

Le mode SCAN trie auparavant les requêtes, selon l'ordre lexicographique :

$$(p, s) < (p', s') \Leftrightarrow S \times (p' - p) + (s' - s) > 0 \Leftrightarrow \begin{cases} - \text{soit } p' > p \\ - \text{soit } p' = p \text{ et } s' > s. \end{cases}$$

Le mode C-SCAN trie quant à lui les requêtes, selon un ordre “semi-lexicographique inversé” :

$$(p, s) < (p', s') \Leftrightarrow S \times (p' - p) + (s - s') > 0 \Leftrightarrow \begin{cases} - \text{soit } p' > p \\ - \text{soit } p' = p \text{ et } s > s'. \end{cases}$$

Donnons une liste de valeurs (en ms) obtenues avec simulation :

P S N G = 100 25 100 1	Défaut	SCAN	C-SCAN
	1818	1439	1320
	1840	1352	1254
	1863	1397	1286
	1795	1365	1308
P S N G = 100 25 100 5			
	1574	1502	1472
	1579	1500	1479
	1562	1474	1469
	1556	1479	1495
P S N G = 100 25 100 10			
	1542	1509	1532
	1528	1506	1521
	1544	1500	1522
	1535	1515	1504
P S N G = 100 25 100 15			
	1509	1493	1490
	1519	1514	1471
	1513	1507	1490
	1525	1509	1482

On pourrait en donner des pages en faisant aussi varier P et S, ou G plus finement entre 1 et 20. Nous avons constaté que quand G est petit, \forall les valeurs de P, S, N, le mode C-SCAN est le moins coûteux. Par contre quand P, S, G sont grands, c'est le mode SCAN le moins coûteux.

Remarque :

La formule (appliquée scrupuleusement dans serveur.c) donnée dans l'énoncé est :

$$\text{durée_du_déplacement} = 2 + 5 \times \left\lfloor \frac{\text{piste_départ} - \text{piste_arrivée}}{P} \right\rfloor \text{ ms.}$$

(On peut considérer qu'il y a une durée incompressible de 2 ms "d'information de la tête de lecture", ou de que-sais-je encore...)

D'aucuns m'ont dit qu'ils avaient considéré que :

$$\text{durée_du_déplacement} = 0, \text{ si } \text{piste_départ} = \text{piste_arrivée}.$$

Dans ce cas le mode C-scan proposé n'est pas le plus avantageux.