

Laboratoire 5 : Serveur - Phase 1

Auteurs : Oussama Lagha et Adam Zouari

Date : 11 Mai 2018

Choix du mécanisme

Durant cette phase du laboratoire, nous avons fait le choix d'utiliser les sémaphores afin d'implémenter le mécanisme de producteur-consommateur.

Implémentation

Nous avons commencer par créer une classe générique `producerconsumerbuffer` qui hérite de la classe générique `AbstractBuffer` fournie. Cette classe nous permet de gérer les tampons de requêtes et de réponses.

Afin d'implémenter le mécanisme de producteur-consommateur, nous avons choisis la version 1 de l'algorithme de tampon simple vu dans le cours. Ce choix a été motivé par soucis de simplicité du code.

Nous avons donc créer ces tampons dans `fileserver.cpp` :

- `requests = new producerconsumerbuffer<Request>();`
- `responses = new producerconsumerbuffer<Response>();`

Ensuite, nous avons créer la classe `requestdispatcherthread` qui permettras à un thread de gérer le traitement des requêtes. Ce thread lancera un thread de traitement pour chaque requête.

Nous avons donc créer ce thread dans `fileserver.cpp` :

- `reqDispatcher = new requestdispatcherthread(requests,responses,hasDebugLog);`

et nous le lançons `reqDispatcher->start();`

Nous avons crée la classe `requestHandler` qui permettra à un thread de traiter une requête. Le traitement est effectué par la fonction `handle()` fournie. Et la `Reponse` que nous retourne cette fonction, est ajouté au tampon de réponse.

Comparaison des perfomances

Contrairement à la version basique, nous pouvons voir que le traitement de 200 requetes s'étale ici sur mes 8 processeurs logiques.

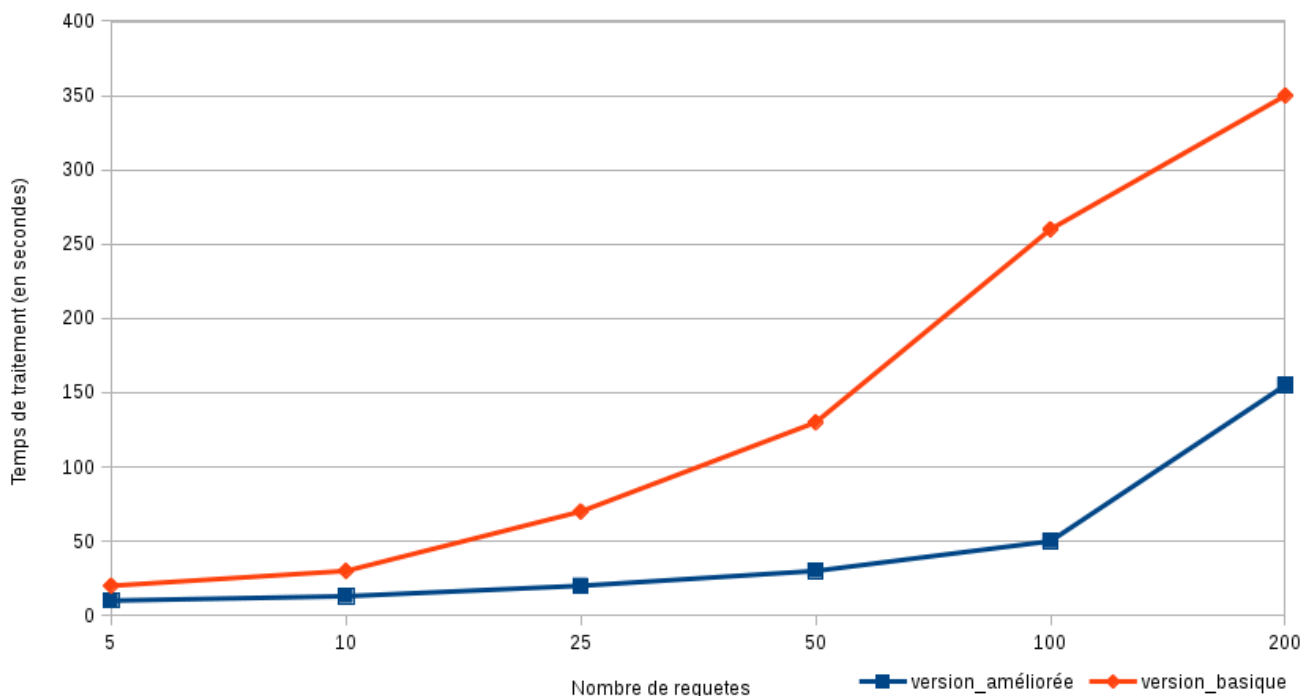
```
radame@xpsradame: ~
radame@xpsradame: ~ 90x32

1 [|||||] 100.0% 5 [|||||] 100.0%
2 [|||||] 100.0% 6 [|||||] 100.0%
3 [|||||] 100.0% 7 [|||||] 100.0%
4 [|||||] 100.0% 8 [|||||] 100.0%
Mem[|||||] 2.92G/15.1G Tasks: 122, 708 thr; 201 running
Swp[|||||] 0K/15.6G Load average: 113.89 31.26 11.29
Uptime: 05:48:28

  PID USER   PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
 1679 radame   20    0 25108  4500  3132 R   1.1  0.0  1:47.71 htop
13199 radame   20    0 5731M 12972 10588 R   3.8  0.1  0:01.97 ./fileserver
13200 radame   20    0 5731M 12972 10588 R   3.3  0.1  0:01.84 ./fileserver
13201 radame   20    0 5731M 12972 10588 R   3.8  0.1  0:02.13 ./fileserver
13202 radame   20    0 5731M 12972 10588 R   3.3  0.1  0:01.90 ./fileserver
13203 radame   20    0 5731M 12972 10588 R   4.3  0.1  0:02.02 ./fileserver
13204 radame   20    0 5731M 12972 10588 R   3.8  0.1  0:02.00 ./fileserver
13205 radame   20    0 5731M 12972 10588 R   4.3  0.1  0:01.92 ./fileserver
```

Afin de mieux se rendre compte de l'amélioration des performances, nous avons fait le graphique ci-dessous. Pour 100 requêtes, notre version les traite 5 fois plus rapidement que la version de base (50 secondes contre 260 secondes). En effet, nous constatons une amélioration significative.

Temps de traitement en fonction du nombre de requêtes



Limitations

Notre version comporte cependant quelques limitations.

Lorsque je lance 10 000 requêtes, ce message d'erreur survient. (Linux Debian)

Je l'expliquerais par le fait que l'OS limite le nombre de tubes (pipes) créés.



The screenshot shows a web browser interface at the top and a terminal window below it. The browser has a logo for 'ReDS Reconfigurable & Embedded Digital Systems' on the left. In the center, there is a text input field containing 'ws://localhost:1234' and a button labeled 'Number of requests' with the value '10000'. On the right, a portion of a URL is visible: '/home/radame/adam/modules/pco/labos/labo5/labo5_server/ph...'. The terminal window below shows the command prompt 'radame@xpsradame: ~/adam/modules/pco/labos/labo5/labo5_server/phase1/version-ameliore/filesserver-threadperreq' and a red banner indicating a terminal size of '110x12'. The terminal output shows two threads attempting to read a file named 'shakespeare.txt' from the same directory. At the bottom, a red error message is displayed: '(process:14593): GLib-ERROR **: Creating pipes for GWakeUp: Too many open files'.

On pourrait protéger le serveur de cet effet néfaste, en fixant un nombre de tubes accepté par le serveur, pour n'utiliser que ceux-ci. Il ne faudrait pas créer des tubes sans arrêt mais plutôt en réutilisant ceux qui ne sont plus utilisés.