Cherokee, serveur http

ETNA IDV-AQL5

rousse_n, bouton_c, pastor_m, chedoz_m

Les objectifs

Construire un serveur http

Réseau

- Concevoir des architectures réseaux en C.
- Comprendre et manipuler les sockets et le protocole TCP

Performance

- Concevoir des architectures pour répondre à des besoins de performance.
- Implémenter des stratégies de performance optimale en respectant des contraintes

Benchmarks

- Mesurer et comparer les performances de différentes implémentations

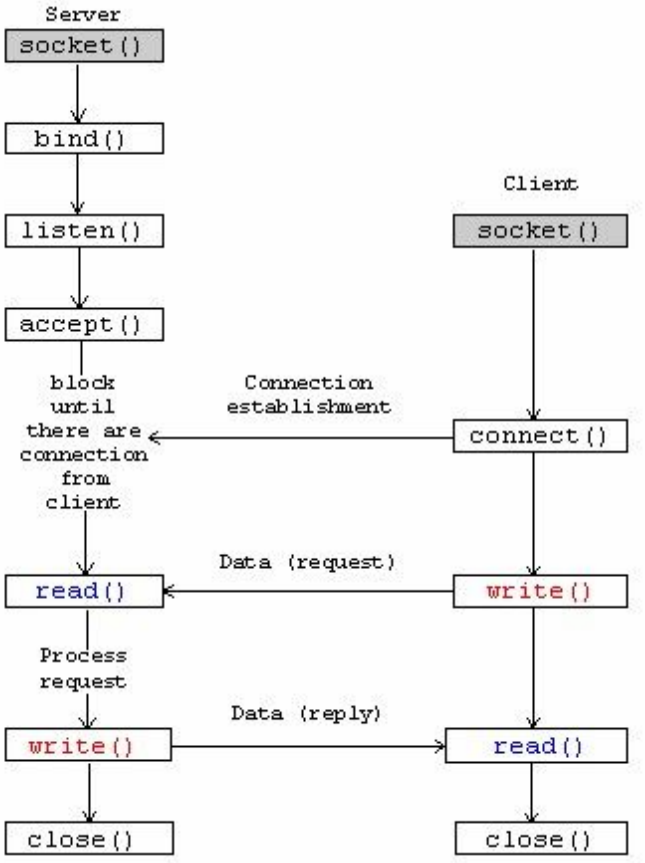
Tests

- Valider la conformité d'une implémentation en fonction de pré-requis

Réseau - Communication

Utilisation des sockets en mode TCP
Gestion des requêtes en asynchrone

Les sockets



Les sockets sont représentées par des descripteurs de fichiers. Il faut préciser le mode de connexion, ici le mode connecté avec le protocol TCP à la création.

La création d'une socket ne fait que réserver un emplacement dans la table des descripteurs du kernel.

Avant de pouvoir l'utiliser, il faut identifier la socket en définissant l'adresse de communication grâce à l'appel système *bind*.

Subtilité réseau côté serveur, *accept* ouvre une autre socket pour établir la connexion sur celle-ci, pour ne pas bloquer le port d'écoute du serveur.

Requêtes asynchrones: Epoll - Multiplexage d'I/O

Les entrée-sorties sont souvent sous contrôle du noyau et il est donc inutile d'utiliser des boucles d'attente active. Les appels système *select*, *poll* et *epoll* réveillent le processus appelant uniquement lorsque des données sont disponibles, permettant à l'application de relâcher complètement le processeur lors de l'attente. Ici nous avons choisit epoll.

Fonctionnement

Mode de notification par front ou par niveau. Dans notre cas nous utilisons le mode par niveau qui met à jour la liste des événements uniquement lorsqu'un file descriptor sous jacent est prêt.

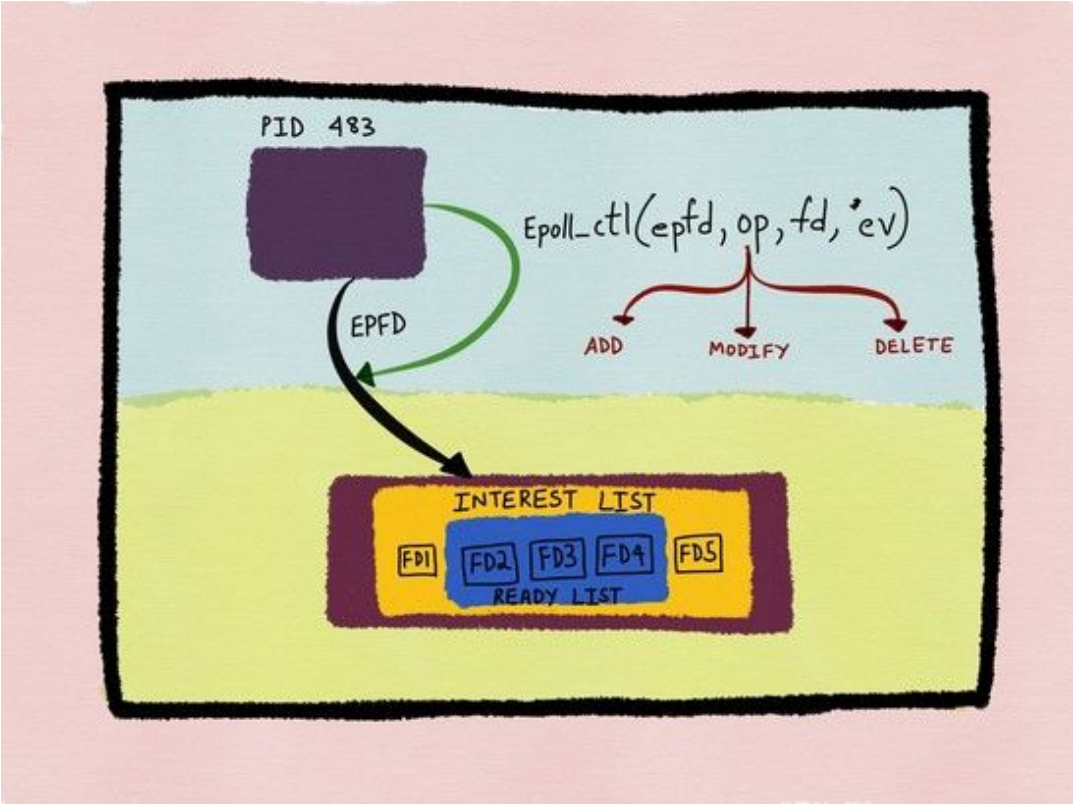
Avantages

Complexité de *epoll* est de O(nombre d'événements) contrairement à *select* et *poll* qui est de O(nombre de file descriptors monitorés)

Inconvénients

Spécifiques à Linux et donc pas destiné à des applications sur os SUSv4 (Unix).

Fonctionnement d’epoll



Performances

Gestion de la répartition de la charge
Optimisation du temps de réponse

Répartition de la charge - Pool de workers

Manager

Définit le nombre de worker

Bind la socket du server

Spawn les instances de workers (fork)

Worker

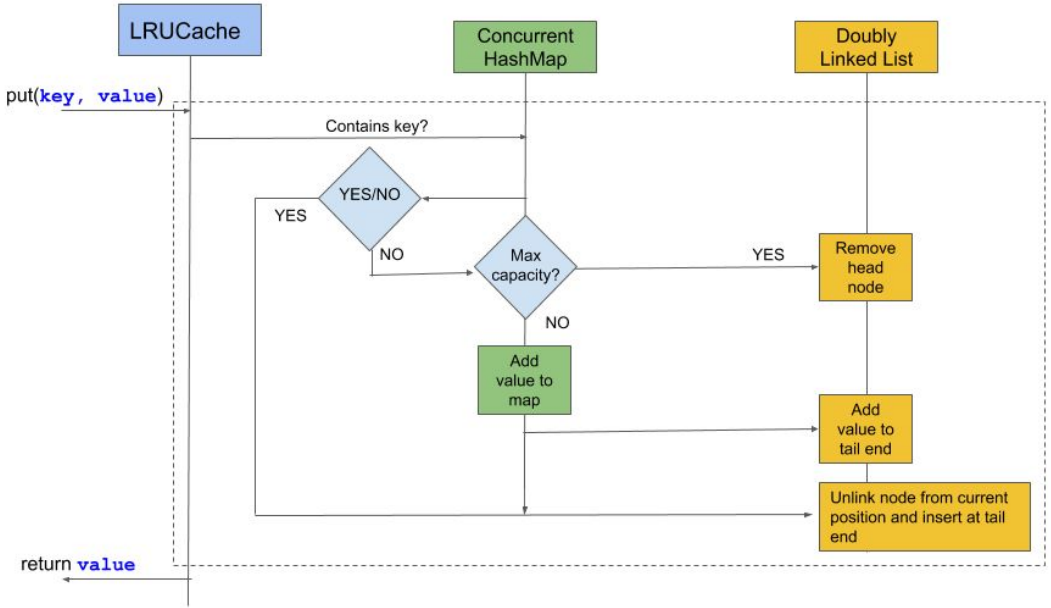
Hérite des files descriptors, accepte les connexions via epoll.

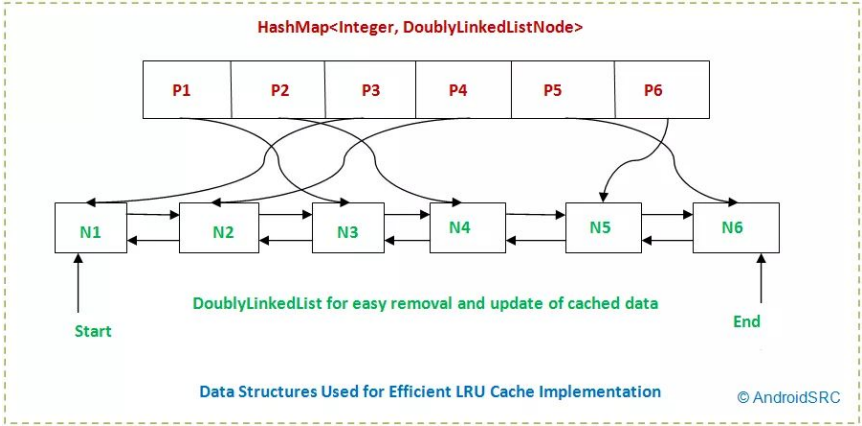
Configuration epoll exclusive pour ne réveiller qu'un seul processus

Kernel

Répartit la charge des connexions sur les différents processus

Cache LRU - Least Recently Used

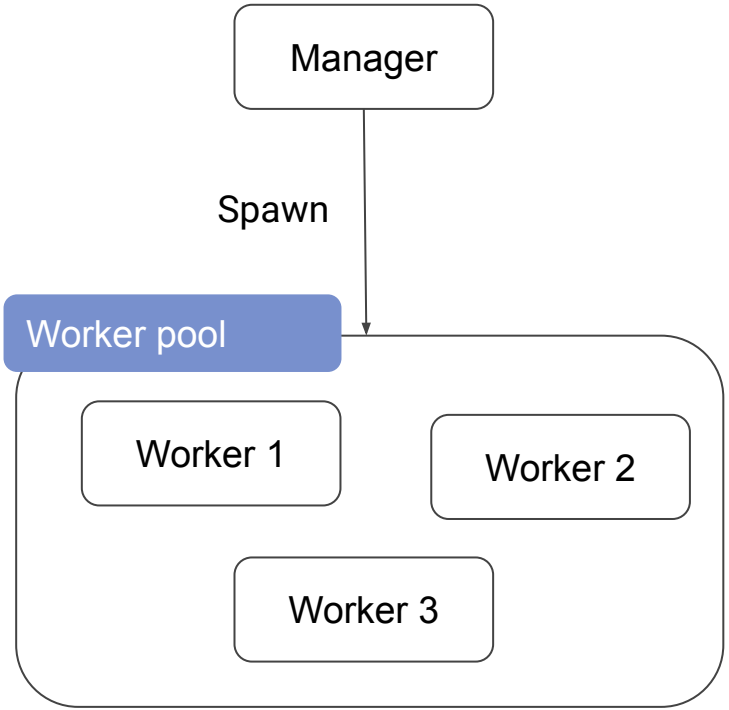




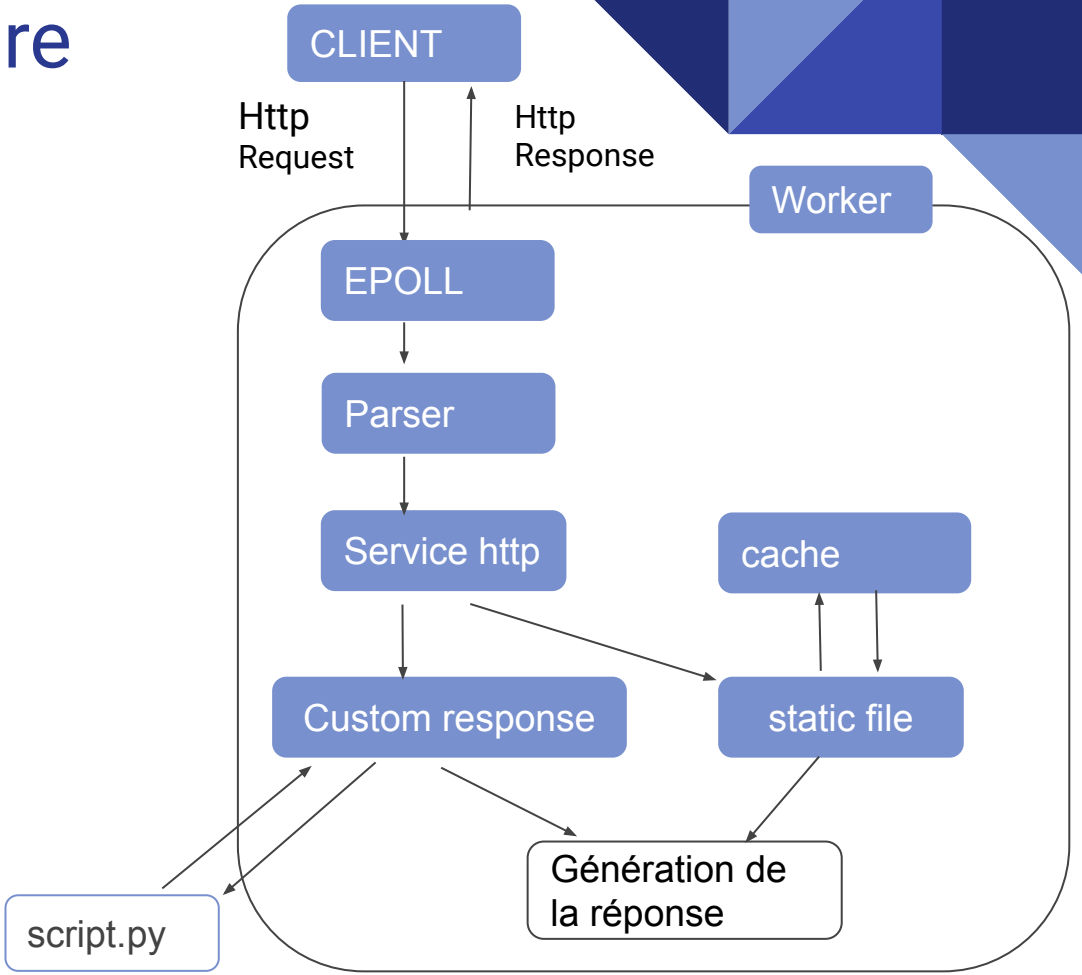
Deux structures de données à maintenir

Architecture et organisation

Schéma d'architecture



Le kernel répartit les connexions clients entre les workers



Prise en charge d’une requête

Lecture du contenu de la requête, génération de tokens, validation de la requête au niveau protocol http ainsi qu’au niveau cahier des charges (headers pris en charge).

Prise en charge de la requête par le module http et routage vers le handler adéquate.

Si fichier statique, demande au cache (hit or fault) et choix de la réponse.

Si réponse customisable, appel de la fonction liée, retours des valeurs, et choix de la réponse.

Formatage de la réponse, envoie au client, fermeture de la connexion.

CRUD

Génération de réponses customisables

Choix du langage
Python 3.xm

Configuration
Association d'une méthode et d'une url via un fichier de configuration.

Fonctionnement
La requête effectuée sur l'url appelle la méthode associée via l'interpréteur Python, effectue son traitement et renvoie un code http ainsi qu'un body.

Tests

Tests unitaires, tests fonctionnels, benchmarks

Tests unitaires sur les parties identifiées comme critiques, en l'occurrence couverture du parseur.

Tests fonctionnels tout au long du développement avec l'utilitaire curl et l'outils de développement web Postman.

Benchmarks des performances, utilisation de l'outils Apache Benchmark
https://httpd.apache.org/docs/2.4/fr/programs/ab.html

Résultats du benchmark

Pour un seul worker, 31ms de temps de réponse en moyenne

ab -n 5000 -c 500 http://localhost:8050/

```
Completed 3500 requests
Completed 4000 requests
Completed 4500 requests
Completed 5000 requests
Finished 5000 requests


Server Software:
Server Hostname:        localhost
Server Port:            8050

Document Path:          /
Document Length:        0 bytes

Concurrency Level:      500
Time taken for tests:   0.312 seconds
Complete requests:      5000
Failed requests:        0
Total transferred:      0 bytes
HTML transferred:       0 bytes
Requests per second:    16012.14 [#/sec] (mean)
Time per request:       31.226 [ms] (mean)
Time per request:       0.062 [ms] (mean, across all concurrent requests)
Transfer rate:          0.00 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    7   5.6      6      32
Processing:     0    7   5.1      6      25
Waiting:        0    0   0.0      0       0
Total:          1   13   9.3     13      39

Percentage of the requests served within a certain time (ms)
  50%     13
  66%     17
  75%     21
  80%     22
  90%     26
  95%     30
  98%     34
  99%     37
 100%     39 (longest request)
```

Améliorations

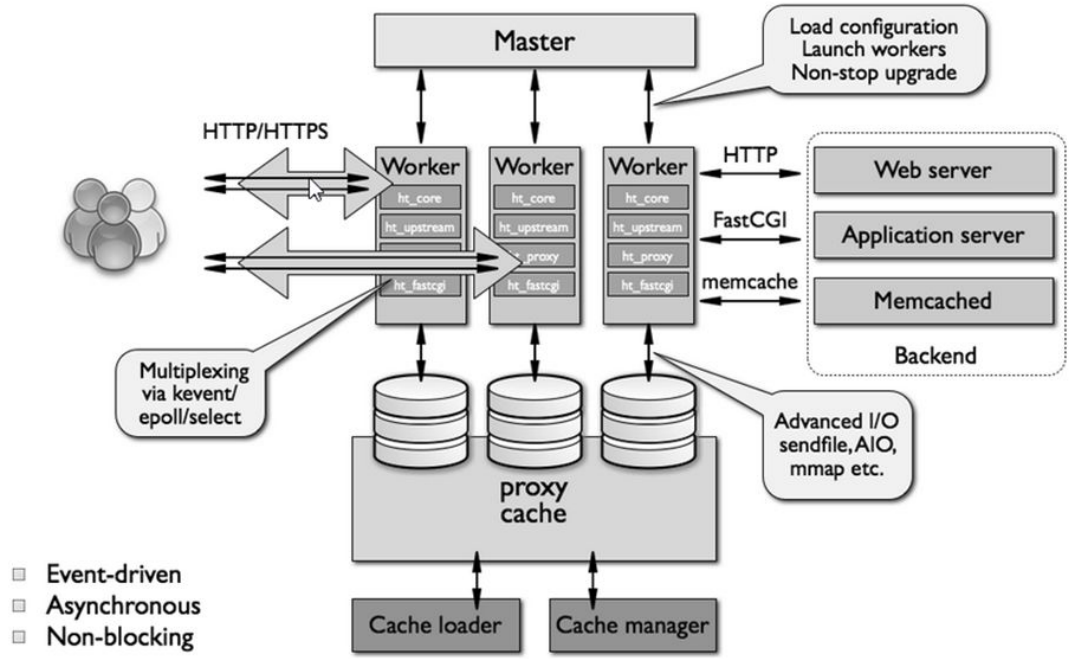
Performance:
Prise en charge de chaque requête par une thread pool.

Amélioration du taux de hit du cache en le plaçant en shared memory. Problème: il faut connaître à l'avance la taille du cache. Implémenter avec une table de hashage (*hcreate_r*)? Le mieux, un processus gère le cache et répond aux demandes des autres processus.

Fiabilité:
Gestion plus robuste de l'évaluation des réponses à fournir à l'aide d'un diagramme de décision (example: https://www.loggly.com/blog/http-status-code-diagram/)

Gestion des requêtes reçues partiellement.

Ressources



Architecture de nginx

https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_71/rzab6/example.htm

https://medium.com/@copyconstruct/the-method-to-epolls-madness-d9d2d6378642

http://aosabook.org/en/nginx.html

Merci