Université de Versailles Saint-Quentin-en-Yvelines
Calcul Haut Performance et Simulation

Master Calcul Haute Performance et Simulation

Rapport de projet d' Architecture
interne des systèmes d'exploitation

Thème :Moniteur système

Responsable du module : M.Jean-Baptiste Besnard

Réalisé par :

FAOUZI LOUGANI
IHDENE CELIA

Année universitaire :2020/2021

SOMMAIRE

Date : 12 mars 2021

Avant propos :

Pour la réalisation du projet nous avons utilisé le service web d'hébergement et de gestion de développement de logiciels *GITHUB* pour synchronisier nous travaux , Projet disponible sur:
https://github.com/lougani-faouzi/AISE_21

Pour la rédaction de ce document nous avons utilisé en premier lieu Google Docs pour la rédaction collective du rapport.
Disponible sur : lien doc

Le projet comporte un makefile : Un *Makefile* est un fichier, regroupant une série de commandes permettant d'exécuter un ensemble d'actions, typiquement la compilation d'un projet ,il utilise un langage déclaratif qui décrit les cibles et leurs dépendances . [1]

Un Makefile peut être écrit à la main, ou généré automatiquement dans notre cas il est écrit à la main , Il est constitué de plusieurs règles,le compilateur choisi est GCC

Une cible clean permettant de nettoyer le projet a été ajouté , C'est une cible qui ne correspond pas à un fichier à produire

Le *make* sera appelé tout seul, c'est la première règle qui est construite , dans notre cas c'est ALL , Des variables ont été aussi ajouté lors de l'appel à make par exemple l'utilisation d'un variable *CC* pour le compilateur *GCC.*

- *CC* qui désigne le compilateur utilisé , comme énoncé précédemment dans notre cas cela sera GCC .
- *CFLAGS* qui regroupe les options de compilation , qui sera en Wall -O3 .
- *LDFLAG*S qui regroupe les options d'édition de liens .

un Help a été implémenté

- h pour afficher l'aide .
- c pour configurer comme client puis au choix les commandes .
- 4 pour les adresses ipv4 .
- 6 pour les adresses ipv6 .
- v pour les adresses ipv6 et ipv4 automatique .
- s pour configurer comme serveur puis au choix les commandes (4,6,v)

Cela a été fait en utilisant la fonction *getopt* qui analyse les arguments de la ligne de commande, Ses éléments argc et argv correspondent aux nombres et à la table d'arguments qui sont transmis à la fonction *main()* lors du lancement du programme. [2]

Politique de développement:

La méthode de développement choisie est la séparation des capteurs et des interfaces et enfin les éléments de mise en réseau .

Un capteur est créé dans une méthode , les résultats de ce dernier sont alors envoyés à la partie interface qui se charge de l'affichage dans un format ergonomique et adapté à la compression de l'utilisateur .

Les éléments que nous pensons avoir implémenté

Code	Composant
A1	Système de build
A2	Communication entre Capteur et Interface
A3	Premier Capteur
A4	Première Interface
A5	Rapport
B1	Client - avancé
B2	Multi client
B3	CLI complète via getopt/équivalent + help + doc

figure 1 : Tableau des éléments implémentés

I. Introduction

Dans le cadre du module aise ce rapport a été rédigé , nous avons à réaliser un outil de monitoring sous réseaux se rapprochant de l'outil Htop , permettant de récupérer toutes les métriques de performance de plusieurs machines.

Afin de mener à bien notre travail , le projet a été séparé en différentes parties , les capteurs mesurant l’activité d’une machine, et les interfaces qui seront le lien avec l’utilisateur.

II.Les capteurs

La première partie est désignés par sensors qui regroupes les différents capteurs utilisé dans les méthodes :

1.Capteur: Processlist_sensor

Dans ce capteur nous récupérons la liste mais aussi des informations sur des processus qui s'exécutent à l' instant courant ,ainsi que leurs nombre totale en utilisant une structure processlist_info defini dans le fichier process liste.h cette dernière contient la taille ainsi que les informations sur le processus .
après cela nous avons une étape de désallocation qui sera gérée par la méthode *free_listprocess (),afin que notre capteur fonctionne un recours au packet procps est fait ,mais nécessitant une installation de libprocps-dev .*

2.Capteur : Uptime Sensor

L'Uptime permet d' indiquer depuis combien de temps le système fonctionne grâce à l'accès au fichier système dans le chemin suivant *" /proc/uptime "* en lecture seule pour le récupérer. La valeur récupérée est un double ,un cast en entier est évident pour l'afficher après sous la forme (heures:minutes:secondes)dans l'interface.

3.Capteur LoadAverage_sensor

Load Average est une métrique utilisée pour suivre les ressources système, elle représente la charge moyenne sur un processeur pour un intervalle de temps défini , elle est représentée par trois valeurs décimales différentes , La première valeur est sur l'interval de la dernière minute , la seconde est sur le dernier intervalle de 5 minutes,la troisième valeur nous donne la charge moyenne de 15 minutes .
Dans le programme elle sera récupéré dans le fichier du chemin "/proc/loadavg"

4.Capteur MemoryInfo sensor

Ce dernier capteur nous permets de récupérer depuis le chemin " */proc/meminfo "* différentes information sur l'utilisation de la mémoire tels que :
la mémoire totale ,mémoire libre , mémoire partagée et mémoire Swap ,mémoire swap libre ,ainsi que shmem , sreclaimable , mémoire utilisée .
Après avoir récupéré les valeurs pertinentes , un simple calcul est utilisé pour déduire certaines autres mesures, pour exemple ce qui suit :

- La mémoire utilisée sera le résultat de la soustraction entre la mémoire totale et la mémoire libre

```
usedMem = totalMem - freeMem
```

- La cached memory ou la mémoire cache sera le résultat de l'addition avec la mémoire SReclaimable (est la mémoire qui est utilisée par le noyau) et la soustraction de

```
cachedMem = cachedMem + sreclaimable - shmem
```

- La fraction de la mémoire non cache appelée Non_cache sur le buffer_memory sera le résultat de la soustraction de la mémoire utilisé calculée précédemment avec buffersMem et cachedMem

```
Non_cache / buffer_memory = usedMem- ( *buffersMem + *cachedMem);
```

- La mémoire swap utilisée sera le résultat de la soustraction de la mémoire swap total avec le swap libre

```
usedSwap =  totalSwap - swapFree
```

5. Performance

La performance joue un rôle important dans la programmation , afin d'analyser les performances de notre code nous avons utilisé RDTC pour chaque méthode un calcul a été fait afin d'établir une comparaison entre elles .
ci-dessous les différents résultats obtenus

Processliste sensor : Résultat de l'analyse des performance de cette méthode : 42472976.000000 cycles

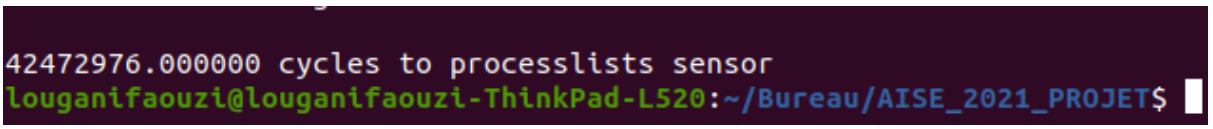

figure 2: performance Processliste sensor

Memoryinfo sensor : Résultat de l'analyse des performance de cette méthode : 422925.000000 cycles

```
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/AISE_2021_PROJET$ ./aisetop -cp
ipv configuration ok client
Mem[cached:780860 kb=0 G||Non_cache_buffer_memory: 2157892 kb=2 G||buffersMem: 30308 kb=0 G]
Swp[usedSwap:693584 kb=0 G]

422925.000000 cycles to memoryinfo sensor
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/AISE_2021_PROJET$
```

figure 3: performance memoryinfo sensor

Uptime sensor : Résultat de l'analyse des performance de cette méthode : 196950.000000 cycle

```
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/AISE_2021_PROJET$ ./aisetop -s4
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/AISE_2021_PROJET$ ./aisetop -c4
ipv configuration ok client
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/AISE_2021_PROJET$ ./aisetop -cp
ipv configuration ok client

                                                    Uptime:0,23,43

196950.000000 cycles to uptime sensor
```

figure 4: performance uptime sensor

Loard_average sensor :Résultat de l'analyse des performance de cette méthode : 3246565.000000 cycle

```
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/AISE_2021_PROJET$ ./aisetop -cp
ipv configuration ok client
                                                Load average: 1.19 1.44 1.61


324656.000000 cycles to load_average sensor
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/AISE_2021_PROJET$
```

figure 5: performance Load average sensor

On remarque que le capteur **Processlist Sensor** nécessite plus de cycles car nous avons une manipulation importante de donnés lors de la récupération d'informations (allocation de la structure ,plusieurs informations pour chaque processus,désallocation).

On déduit,que la complexité dans notre cas dépend de la quantité d'informations traitées .Ce qui influence sur la performance de notre programme .

III. Les Interfaces

La deuxième partie est la partie IHM qui est représentée par 2 fichiers ihm.h et ihm.c , cette partie gère l'ergonomie d'affichage sur le terminal pour une meilleure lisibilité et une compréhension plus facile par l'utilisateur final .
Pour cela plusieurs fonctions et méthodes sont implémentées.

1. La méthode print processlist

Comme son nom l'indique c'est une méthode d'affichage , elle affiche la liste des processus en cours ainsi que les informations déjà recueillies dans l'étape précédente cette méthode affiche aussi une entête

- User : Désigne le nom d'utilisateur du processus.
- TID : Thread ID généralement qui est égale au PID.
- PRIO : Désigne la priorité.
- PPID: L'identifiant du processus père.
- RSS : Est la taille définie par le résident et est utilisée pour indiquer la quantité de mémoire allouée à ce processus et se trouvant dans la RAM.

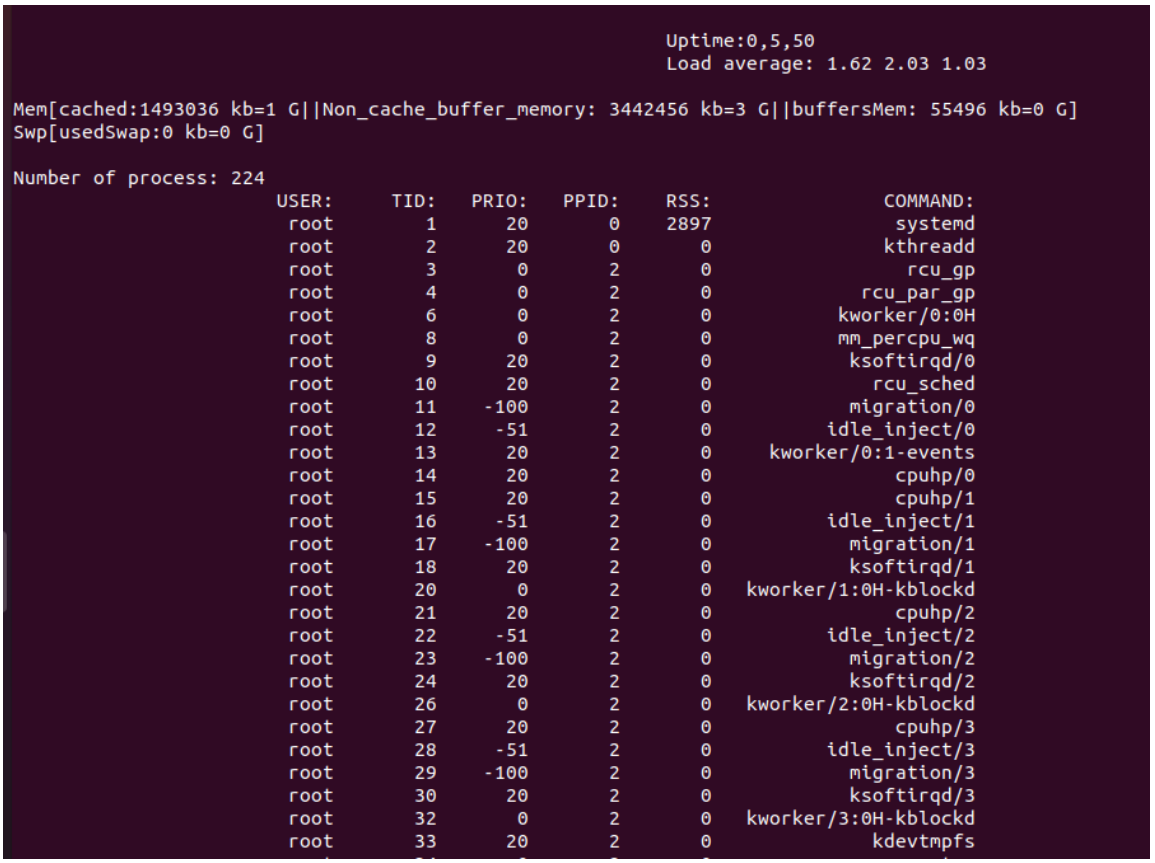

figure 6 : Interface du terminal

2 La méthode void print_Uptime

Cette méthode permet d'afficher la valeur de Uptime en unité beaucoup plus lisible, en effet la valeur sera convertie en secondes et minutes et heures.

3 La méthode print Load Average

Affiche les 3 valeurs de load average définies précédemment .

4 La methode print_memory_result

Cette dernière affiche dans notre interface , l'état des différentes mémoires la quantités utilisée et libre .

IV. La mise en réseaux

Afin de réaliser la deuxième partie du du projet qui est la mise en réseaux , nous nous aidons des notions vue en cours mais aussi des démonstrations des différents travaux pratique (TP)

Dans un premier temps nous aurons un serveur et plusieurs client qui pourront s'y connecter c'est à dire une architecture client/serveur pour cela , des Sockets seront utilisés , un socket représente une interface de communication logicielle avec le qui permet d'exploiter les services d'un protocole réseau et par laquelle une application peut envoyer et recevoir des données. C'est donc un mécanisme de communication bidirectionnelle entre processus , plus simplement une socket est un point de communication par lequel un processus peut émettre et recevoir des données. [4]

Nous présentant les grandes lignes du code source :

- **Coté client**

Un socket sera créé

```
int socket(int domain, int type, int protocol);
```

Une attribution d'adresse ip , de type d'adresse et de port sera faite pour des raisons de conflit le port utilisé sera différents des ports habituellement utilisé par exemple nous n'utilisons pas le port n° 80 qui est associé au service HTTP ou bien le 110 associé au service Post Office **Protocol** v3 (pop3)

Une connexion sera effectuée, une fois la connexion effectuée un échange de message pourra avoir lieu .

- **Coté serveur**

Un socket sera créé
Un assignement de la socket à une adresse sera faite grace a Bind()
le serveur sera en attente de connexion ensuite la connexion du client sera accepté puis un échange pourra se faire grâce à send et recv des fonctions d'envoi et de réception

- **Gestion de Client Multiple**

Pour effectuer la connexion de plusieurs clients à notre serveur la notion de thread sera utilisée .
afin de créer un thread, on utilise la fonction

```
pthread_create( pthread_t *thread,  const pthread_attr_t *attr,  void *(*start_routine) (void *),   void *arg);
```

un nombre maximum de client pourra être défini pour des raisons de sécurité ou de fiabilité ainsi une limite de client à accepter pourrait être établie bien qu'elle soit facultative , pour cela une boucle avec un compteur sera implémenté par exemple .

V. Référence

Bibliographie et Webographie

- [1] https://ensiwiki.ensimag.fr/images/e/eb/Makefile.pdf
- [2] http://manpagesfr.free.fr/man/man3/getopt.3.html
- [3] https://fossies.org/linux/procps-ng/proc/readproc.h
- [4] : Cours Programmation Réseau : Socket TCP/UDP © 2012-2020

Liste des figures