Rapport AISE
Création d'un logiciel de monitoring

Thomas Trebosc
Aka Brou

12 mars 2021

Table des matières

0.1 Introduction

Le but du projet est de créer un logiciel de monitoring des processus en cours d'exécution sur une machine. Cependant une contrainte nous est imposée. Nous devons récupérer les informations d'un autre ordinateur connecté au même wifi pour les afficher sur la machine d'origine.
Ce projet est également complété par un dépôt GitHub (Github) ou vous pourrez trouver nos code ainsi qu'un README et un makefile. Le dépôt est décomposé en deux branches "main" et "master". La branche "main" contient l'implémentation avec GUI et "master" l'implémentation sans GUI.

0.2 Implémentation

0.2.1 Traitement des données

Notre modèle de départ sera la commande linux top. La première étape de notre projet consiste à reproduire le plus fidèlement possible son affichage. Nous avons donc commencé par implémenter une fonction temps() qui affiche l'heure à l'aide du type time_t et de la fonction time() de la librairie <time.h>.
Puis nous avons créé la fonction uptime(), qui affiche un chronomètre démarré à l'allumage du système. Dans cette fonction nous faisons appel aux I/0 FILE. Les valeurs nécessaires aux calculs se trouvent dans le fichier meminfo du répertoire proc sur linux.
Ensuite, nous avons affiché les informations sur la mémoire en lisant le fichier meminfo ou encore le fichier swaps, tous deux situés dans le répertoire proc. Nous calculons aussi le pourcentage cpu dans la fonction pcpu pour chaque processus.Tous ces fichiers, et tous les fichiers lus dans notre programme, sont des fichiers qui se mettent à jour en continu. La formule pour calculer l'utilisation du cpu de chaque processus est la suivante :

$$\%cpu = 100 * \frac{total\_time}{Hertz * seconds}$$

la variable seconds s'obtient à l'aide de la formule :

$$seconds = uptime - \frac{starttime}{Hertz}$$

La variable starttime stocke le temps du système au lancement du processus. Le calcul de Hertz s'obtient en appelant la fonction sysconf(SC_CLK_TCK) de la librairie unistd.h. Cette fonction permet de retourner la fréquence du cpu.
Enfin la dernière variable total_time est la somme des données utime, stime,

cutime, cstime que l'on trouve dans le fichier uptime du répertoire proc :
- utime correspond au temps passé par le code dans l'espace utilisateur.
- stime correspond au temps passé par le code dans l'espace noyau.
- cutime correspond au temps passé à attendre l'exécution d'un processus fils dans l'espace utilisateur.
- cstime correspond au temps passé à attendre l'exécution d'un processus fils dans l'espace root.

Un autre de nos capteurs est chargé d'évaluer la consommation mémoire de chaque processus.
La formule de la consommation mémoire est la suivante :

$$\%mem = 100 * \frac{RES}{RAM}$$

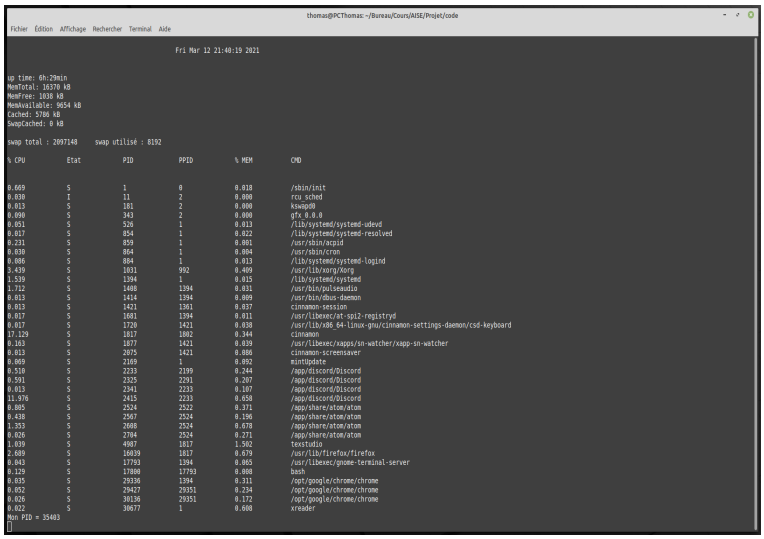
La consommation mémoire est donc issue du rapport en la RES(mémoire physique accessible par le processus) et la RAM disponible sur la machine.

Nous avons essayé d'afficher les informations des processus en suivant cette même méthode de lecture de fichier manuelle. Nous avons utilisé la librairie dirent mais l'affichage ne nous a pas satisfait.
Nous avons donc décidé d'utiliser la librairie libprocps qui a des structures prédéfinis qui lisent les informations sur le processeur.
Pour plus de lisibilité, le fichier "sensors2.c" est inclut dans le fichier readproc.c dans lequel nous utilisons libprocps. Ainsi l'utilisation des fonctions implémentées précédemment est possible sans surcharger notre fichier. Dans ce fichier nous lisons les PID des processus, leur état, le PID du processus qui les a créé aussi appelé processus parent, ainsi que le nom des applications qui en sont à l'origine.
Pour l'affichage toutes ces informations sont stockées dans des variables de type ppinteur de char. Nous avons fait des allocations mémoire de 1000 éléments de char pour chaque variable car nous avions des problèmes de mémoire liés à la longueur des applications.
Nous avons utilisé la fonction calloc pour nos allocations mémoire. En théorie, cette fonction alloue de la mémoire pour un tableau du nombre d'éléments qu'on saisit en premier argument et de la taille de l'élément qu'on saisit en deuxième argument. Cependant, nous avons eu de nombreux messages d'erreurs de segmentation provenant de cette partie du code après analyse à l'aide de gdb. Nous avons donc pris l'initiative d'utiliser la fonction memset en plus de calloc et nous n'avons plus eu d'erreur de segmentation (du moins pour cette partie du code). Nous avions également tenté d'utiliser le combo malloc et memset, plus conventionnel, mais sans succès.

Pour éviter que l'affichage ne soit trop surchargé, nous avons insérer une condition sur la valeur du pourcentage cpu. Nous avons arbitrairement décidé de ne pas afficher les informations des processus dont l'usage cpu était inférieur à 0.01%.

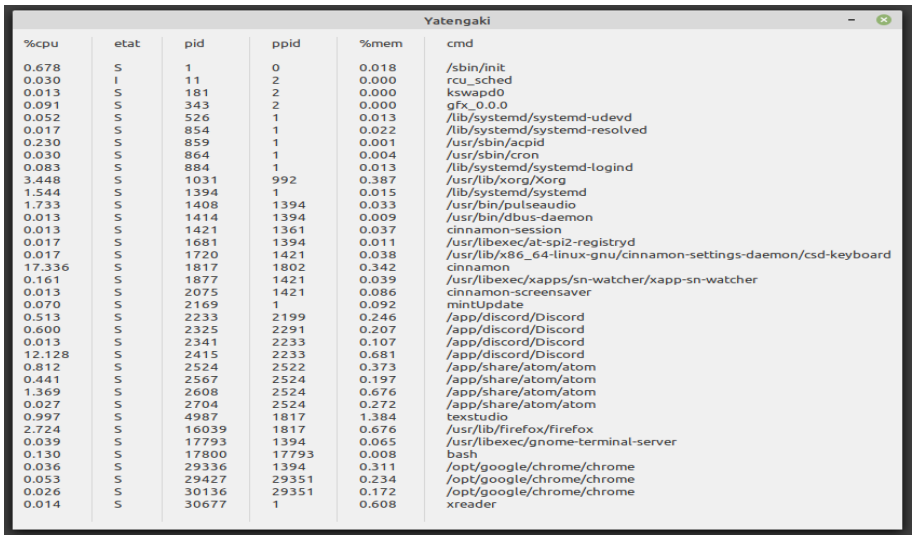
0.2.2 Affichage

Pour l'affichage des données nous avions d'abord décidé des les afficher dans le terminal de la façons la plus esthétique possible en affichant en premier la consommation CPU, l'état, le PID, le PPID, la consommation mémoire et le chemin ou l'exécutable comme ci contre :



Cet affichage se renouvelait avec un temps choisit (1 seconde ici). Nous avons ensuite essayé d'implémenter une GUI via GTK. Notre objectif était d'avoir approximativement le même affichage qui se mettait à jour de la même manière (toute les secondes via un rafraîchissement de page automatique). Nous n'avons pas eu le temps de finir le rafraîchissement de page mais nous avons tout de même obtenue une GUI fonctionnelle et qui se met à jour en fermant la page (cela en ouvre automatiquement une autre). Le résultat

est donc le suivant :



0.2.3 Réseau

Pour pouvoir faire communiquer deux machines entre elles, nous avons choisi le Transmission Control Protocol plus connu comme TCP. Ce protocole permet l'envoi et la réception de messages entre un client et un serveur. Le client envoie une requête au serveur, puis si les conditions d'accès au serveur sont respectées le serveur répond à sa requête. La mise en place d'un tel protocole repose sur l'implémentation de deux programmes. L'un qu'on exécutera sur la machine jouant le rôle du client et l'autre sur la machine jouant le rôle du serveur.
Pour établir le lien entre ces deux programmes et donc entre les deux machines, nous devons créer un socket. Le socket est une interface de connexion réseau. Le socket assure la lecture des informations transmises par l'autre machine. On doit ensuite assigner une adresse à ce socket. Nous avons choisi de lui assigner une adresse IPV4, plus facile à lire. Nous lui avons donné le type sock_stream qui après quelques recherches rapides, s'est avéré être le type le plus couramment utilisé et le plus simple à utiliser. Après avoir saisi les propriétés du socket, il faut lui assigner une adresse. Une adresse de serveur se compose d'un type de communication, un numéro de port et une adresse. Nous avons utilisé INADDR_ANY pour ne pas avoir à choisir d'adresse particulière, la communication peut se faire par toutes les inter-

faces locales.
Une fois le socket configuré, il faut établir la connexion. Pour ce faire on utilise les fonctions bind et listen pour le serveur et la fonction connect pour le client.
On peut ensuite envoyer et recevoir des messages depuis les deux côtés du système de communication.
Notre volonté était de commander l'exécution depuis le client, puis que l'exécution ait lieu sur le serveur, pour qu'ensuite le serveur renvoie la sortie au client et qu'enfin l'affichage du programme ait lieu sur le client. Nous n'avons pas réussi malheureusement. Pour l'instant, notre réseau de communication ne permet que l'exécution sur le serveur.

0.3 Conclusion

Au travers de ce petit projet nous avons pu approfondir nos connaissances dans la langue C mais surtout apprendre énormément sur le monitoring et l'agencement des processus au sein de Linux ainsi que sur la création d'une GUI et d'un système réseau. Malgré que tous les objectifs n'aient pas pu être atteints ce projet a été riche en découverte et nous sommes plutôt satisfait du résultat pour n'avoir jamais codé en C auparavant.
Avec plus de temps nous aurions donc pu améliorer la GUI (avec rafraîchissement), avancé dans la partie réseau (communication et transmission entre deux appareils) et également comparer nos résultats avec la commande "top" via notamment la fonction "popen" qui permet de récupérer les sorties de commande bash (FILE* f = popen("top", "r")).

0.4 Bibliographie

Ressource	Lien	Utilité
YouTube	Programmer's Notes	Installer GTK
Blog	ewencumming	Installer/compiler libproc
Blog	Redhat	Informations dossiers dans "/proc"
Blog	Stackoverflow	% CPU
Blog	Carlchanet	% Mémoire
Blog	Medium	Comprendre TOP
Blog	memo linux	Comprendre TOP
YouTube	Idiot Developer	Serveur multi clients
YouTube	Eduonix Learning Solutions	Sockets
YouTube	Developpez	Utiliser GTK