Rapport Projet Daemon

CLEMENT Pierre, COLLET François

Présentation du projet :

L’objectif du projet était de réaliser un daemon, un programme ne nécessitant pas beaucoup de ressources capables de tourner en tâche de fond jusqu’à ce qu’il soit appelé, ainsi que son invocateur, le programme qui va nous permettre d’appeler et de communiqué avec le daemon. Ce genre de programme peut être très utile par sa capacité à être constamment actif sans pour autant bloquer l’utilisation.

Pour que ce programme soit réellement utile, il est impératif que le daemon ne consomme pas beaucoup de ressources. Dans le cas contraire, cela pourrait ralentir la machine sur laquelle le deamon est exécuté et par conséquent gêner le confort de l’utilisateur. Notre deamon doit donc se limiter à des taches simples.

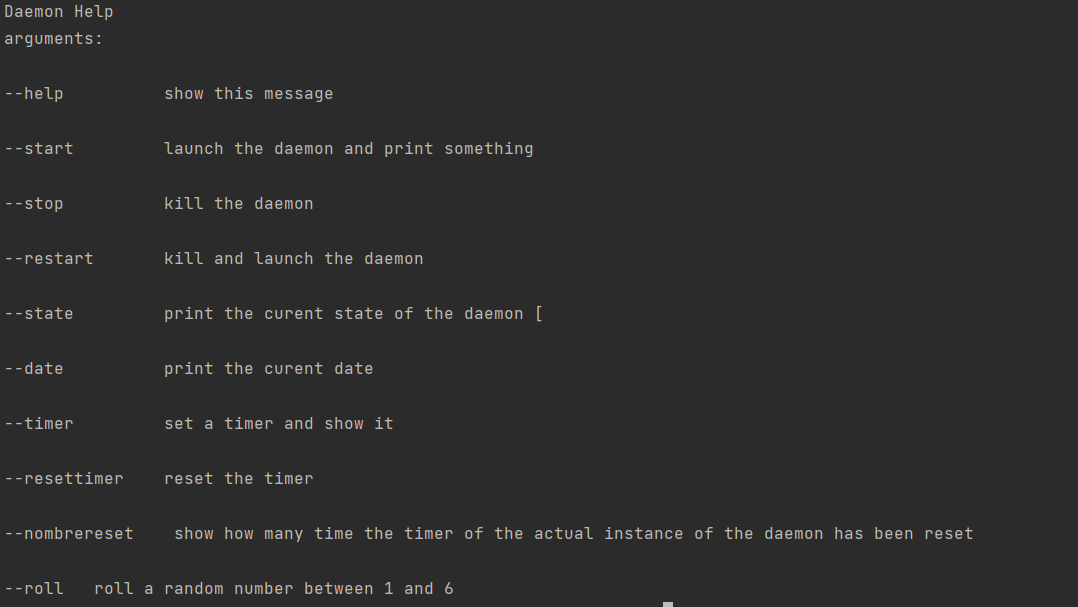
Le plus grand impératif de notre programme et de pouvoir être utilisable. Notre programme doit donc premièrement pouvoir créer un daemon mais également pouvoir le détruire. Il est en effet impensable de laisser un programme tourner indéfiniment sur une machine sans avoir moyen de de l’arrêter. Deuxièmement, le daemon devant s’exécuter en arrière-plan, il est impératif de pouvoir communiquer avec lui, sans le rendre bloquant. Troisièmement, pour qu’un programme soit utilisable, il faut qu’il soit compréhensible. Il doit y avoir une fonction permettant d’afficher une aide quant à l’utilisation du programme. Dernièrement, pour que le programmes soit utilisable, il doit être complet, et donc intégrer toutes les fonctions que l’on attend de lui. En l’occurrence, dans ce projet, notre programme doit pouvoir démarrer et arrêter le daemon, le faire redémarrer, afficher son état (démarré ou éteint), donner la date actuelle, affiché son compteur qui est initialisé lors de sa création, et réinitialiser ce compteur.

une méthode optionnelle permettant d’afficher le nombre de réinitialisation du compteur du daemon a aussi été implémentée.

Les moyens d’utiliser ce projet

Notre projet consiste en un programme que l’on appelle avec la commande « ./programme » suivi d’un argument.

Les arguments sont tous construit de la même manière, il commence par «--»et sont suivi du nom de l’argument.

Le premier argument est l’argument « --help » permettant d’afficher l’aide comme présenté ci-dessous. 

Cette aide comprend le nom de tous les arguments existant avec leur utilité. On peut utiliser cet argument pour chercher un argument dont on connait l’effet mais dont on ignore le nom, ou inversement, pour chercher l’effet d’un argument dont on ne connait que le nom.

Cet argument peut être utilisé quel que soit l’état actuel du daemon.

Le deuxième argument est l’argument « --start » qui permet de démarrer le daemon. Lorsque le daemon est lancé il va initialiser son chronomètre ainsi que le nombre de fois que se chronomètre a été réinitialisé à 0. Il va également créer le pipe nommée qui va servir à la communication entre l’invocateur et le daemon. Il va également créer un fichier dans lequel il va stocker son pid. Ensuite, le daemon va atteindre de recevoir par l’intermédiaire de ce dit pipe des ordres de la part de l’invocateur. Cet argument ne peut être utilisé que lorsque le daemon est éteint.

Le troisième argument est l’argument « --stop » qui va permettre d’éteindre le deamon. Il détruira au passage le fichier stockant le pid du daemon ainsi que le pipe nommé. Cet argument ne peut s’utiliser que lorsque le daemon est démarré.

Le quatrième argument est l’argument « --restart » qui va éteindre puis démarrer le daemon. Pour ce faire il va dans un premier temps exécuter la méthode utilisé par l’argument «--stop» puis il va appeler la méthode utilisé par l’argument « --start ». Cet argument ne peut être utilisé que lorsque le daemon est démarré.

Le cinquième argument est l’argument « --state » qui va afficher l’état actuel du daemon (démarré ou éteint). Cet argument peut être utilisé quel que soit l’état du daemon.

Le sixième argument est l’argument « --date » qui va demander au daemon via le pipe nommé la date actuelle et qui va l’afficher. Cet argument ne peut être utilisé que lorsque le daemon est démarré.

Le septième argument est l’argument « --timer » qui va demander au daemon via le pipe nommé la valeur actuelle du chronomètre et qui va l’afficher. Cet argument ne peut être utilisé que lorsque le daemon est démarré.

Le septième argument est l’argument « --resettimer » qui va réinitialiser le chronomètre et incrémenter de 1 la valeur du compteur de réinitialisation. Cet argument ne peut être utilisé que lorsque le daemon est démarré.

Le huitième argument est l’argument « --nombrereset » qui va demander au daemon via le pipe nommé la valeur actuelle du compteur de réinitialisation et qui va l’afficher. Cet argument ne peut être utilisé que lorsque le daemon est démarré.

Le dernier argument est l’argument « --roll » qui va effectuer un jet de dés à 6 faces et afficher le résultat. Cet argument ne peut être utilisé que lorsque le daemon est démarré.

Voici le déroulé du développement :   
nous avons commencé par créer un programme daemon, son invocateur et des fonctions start, stop et restart pour allumer éteindre et redémarrer le daemon *cf. invocateur ligne 17.*

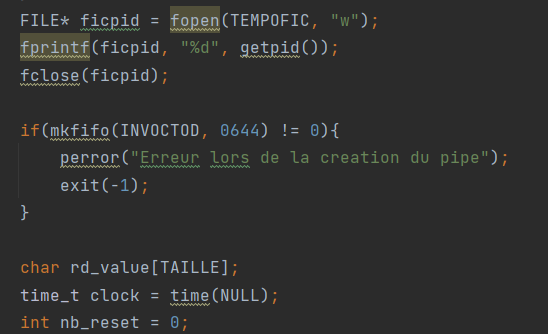
Pour demarrer le demon, on regarde tout d’abord si un demon existe déjà(methode isalive()), afin de ne pas en recréer un. Si il n’y a pas de demon d’actif, nous allons lancer le fontion startdaemon().Une image contenant texte

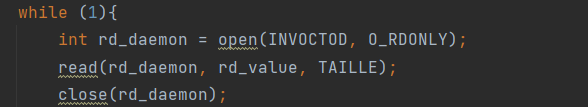
Description générée automatiquement La fonction startdaemon() utilise la fonction fork qui crée un fils et détache notre daemon du terminal avec lequel il a été appelle grâce à la fonction setsid *cf. daemon ligne 25*. Le daemon va stocker son pid dans un fichier temporaire défini dans le fichier invocateur.h.

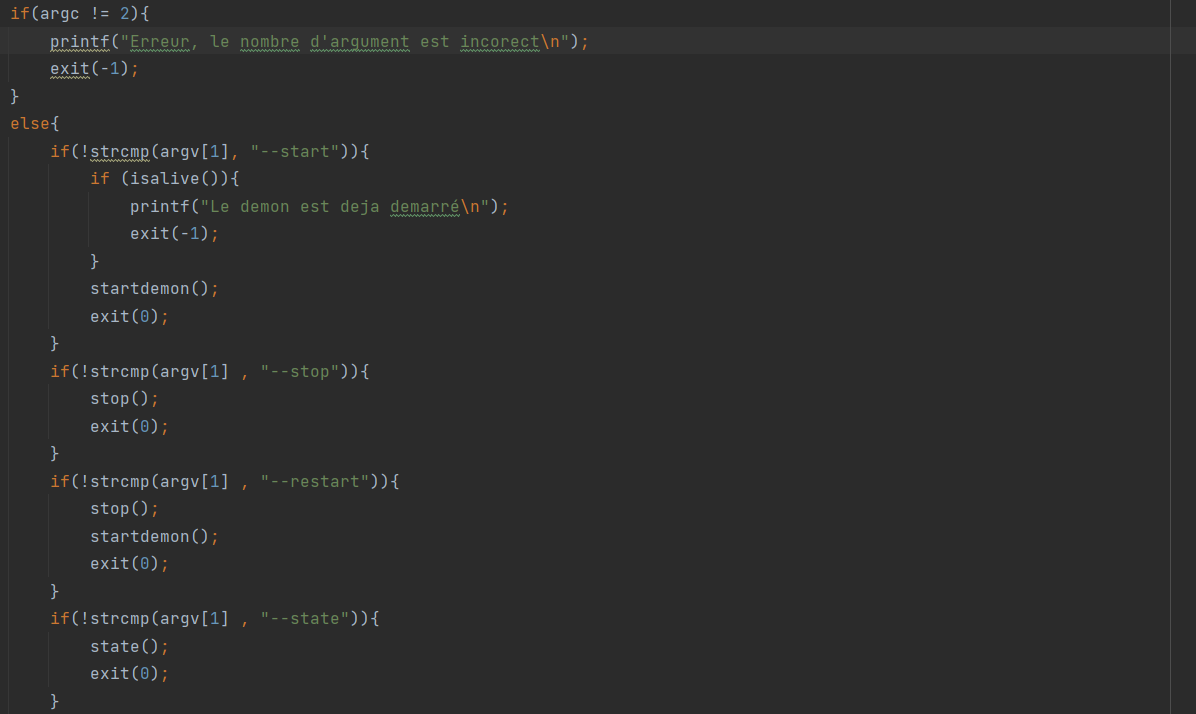


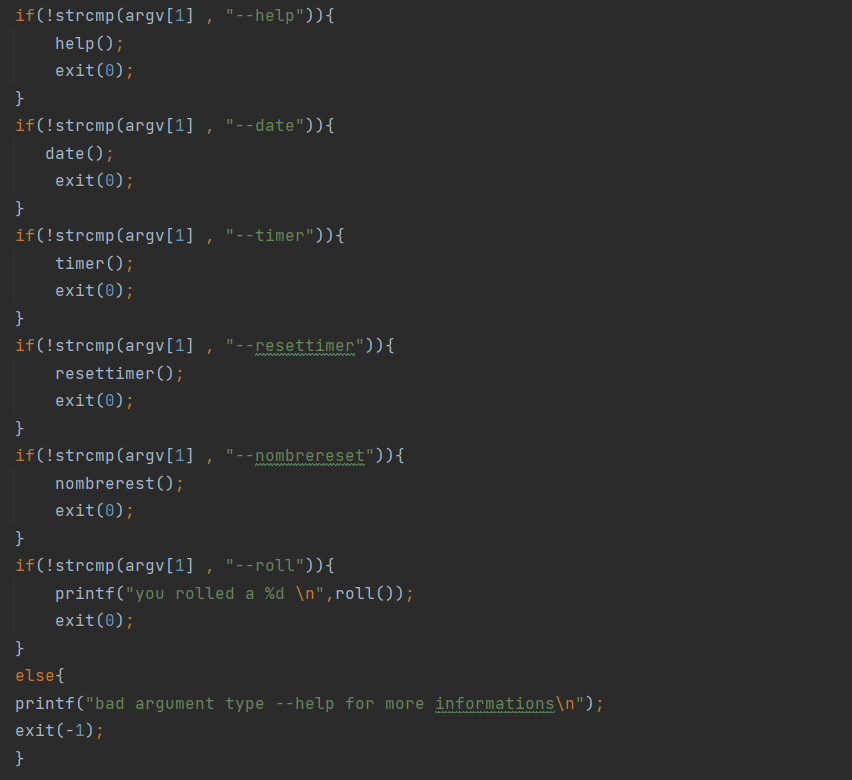
Une fois cela fait, le daemon va créer le tube nommé et va déclarer la variable qui va lire le contenu de pipe. Il va également initialiser le compteur et le nombre de reset de se compteur.

Une image contenant texte

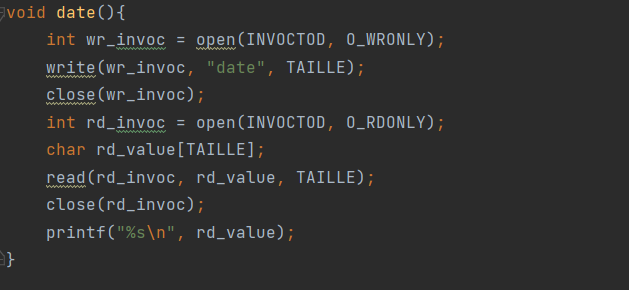
Description générée automatiquement  


Une fois démarré notre daemon se met en attente avec une boucle while(1) car en C toute valeur différent de 0 est considéré comme true. Dans cette boucle while, il va en permanece lire le contenu du pipe pour voir s’il a reçu un ordre. 

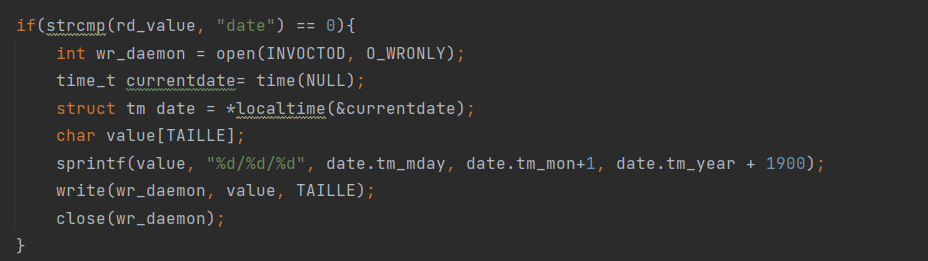
Lorsque l’on va appeler notre programme avec un argument, nous allons dans le main vérifier si il n’y a bien qu’un seul argument, et si l’argument transmis correspond à un des arguments figurant de l’aide. Si ce n’est pas le cas, le programme va lever une erreur, sinon, il va appeler la fonction correspondant à l’argument fourni. 



Toutes les fonctions ne nécessitent pas forcément d’appeler le daemon. Par exemple, la fonction state va simplement regarder si le fichier stockant le pid du daemon existe. Ce fichier étant créé au lancement du daemon et détruit lorsque la fonction stop est appelé, son existence ou non nous permet de déterminer l’état du daemon. Pour les fonctions nécessitant de communiqué avec le daemon, on commence par définir la variable qui va nous permettre d’écrire dans le tuyau nommé. Ensuite, on va ouvrir ce tuyau en écriture, puis marqué l’ordre dedans. Le tuyau va ensuite être fermé puis, si le daemon doit renvoyer quelque chose, il va être réouvert en mode lecture. La valeur de retour si elle existe va être récupérer dans une variable qui va ensuite être afficher.

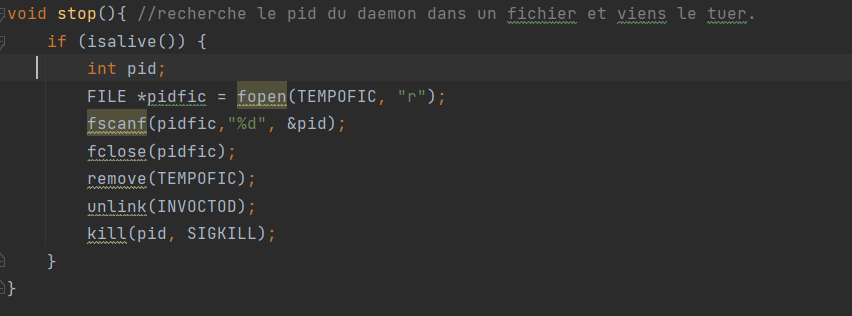


Lorsque le daemon va lire le tuyau et que la valeur qu’il va lire correspond à une valeur défini, il va fermer le tuyau, exécuter la fonction associé à l’ordre qu’il reçut, puis si il a besoin de renvoyer un valeur, il va rouvrir le tuyaux en mode écriture pour écrire la valeur de retour. Une fois la valeur de retour, si elle existe, renvoyé, il va refermer le tuyau, puis lorsqu’il sortira de l’instruction conditionnelle if, il va le rouvrir en mode lecture.



Le deamon lira constamment le contenu du pipe jusqu’à ce que l’argument « --stop » ou l’argument « --restart » soit entrée.

Pour éteindre le daemon, l’instruction stop va récupérer le pid contenu dans le fichier, dont le chemin est defini dans la constante TEMPOFIC, et va tuer ce PID avec la fonction Kill(). Une fois le pid tué, le fichier va être supprimé.



Pour réaliser ce projet, nous nous sommes, séparer les tâches. François s’est occupé du daemon et du main tandis que Pierre à réaliser les fonctions de l’invocateur. Le fichier était partagé sur un GitHub afin de pouvoir travailler ensemble à distance.

**CONCLUSION**

Ce projet remplie sa fonction, il comporte bien un deamon et invocateur capable de communiqué entre eux. Le daemon s’exécute bien en arrière-plan et l’invocateur est bien capable de communiqué avec le daemon. Le projet implémente toutes les méthodes demandées mais également des méthodes bonus. Le projet est fonctionnel, cependant des difficultés ont été rencontrées. La principale a été rencontrés au niveau du pipe. Au début du projet, Il y avait deux pipes nommées, un dans lequel l’invocateur écrivait ses ordres et que le daemon lisait pour récupérer l’ordre, et un dans lequel le daemon écrivait la valeur de retour si elle existait et que l’invocateur lisait pour récupérer la valeur afin de l’afficher. Ces pipes était également ouvert dans des fichiers. Pour des raisons de synchronisation, un seul pipe à été gardé. Pour alléger visuellement le code, le pipe n’utilise plus de fichier, car cela nous permet de nous affranchir des fonctions fdopen() et fclose(). Cette solution nous oblige tout de même à utiliser la fonction sprintf() afin d’avoir le bon format de texte pour la valeur de retour . Initialement, les fonctions de l’invocateur étaient définies dans le fichier main. Pour des raisons de confort visuel, ces fonctions sont maintenant défini dans un fichier invocateur.c, et le main se contente d’appeler ses fonction.

Il nous aurait été compliqué voire impossible de réalisé ce projet sans l’aide de nos camarades, qui nous ont notamment aidé pour dissocier le daemon du terminal, et pour comprendre le fonctionnement des pipes nommés.

Ce projet nous a permis d’approfondir nos capacités à travailler en équipe. Il nous a également permis de mieux comprendre, grâce à l’aide nos camarades, le concept de tube nommé qui nous semblait réellement compliqué auparavant.