CSEL – Rapport de travail pratique

Walpen Elliott, Arrigo Marc-Olivier

HES-SO – MSE Systèmes Embarqués

Semestre d’automne 2020

Table des matières

[Rapport Mini Projet 3](#_Toc61014166)

[Résumé 3](#_Toc61014167)

[Contexte 3](#_Toc61014168)

[Architecture 4](#_Toc61014169)

[Kernel 5](#_Toc61014170)

[Deamon 5](#_Toc61014171)

[App 5](#_Toc61014172)

[Utilisation 5](#_Toc61014173)

[Synthèse 6](#_Toc61014174)

[Feedback 6](#_Toc61014175)

[Annexes 6](#_Toc61014176)

# Rapport Mini Projet

## Résumé

Le mini projet vise à mettre en œuvre toutes les notions étudiées lors du semestre soit :

* Mise en œuvre d’un système embarqué sous Linux
* Développement de modules noyaux
* Développement de pilotes de périphériques
* Traitement des fichiers ordinaires et spéciaux
* Développement d’applications multitâches
* Gestion des ressources CPU
* Optimisation

## Contexte

Concevoir une application permettant de simuler la gestion de la vitesse de rotation d’un ventilateur en fonction de la température du processeur, via une gestion manuelle ou automatique. Le ventilateur n’est pas disponible sur notre cible, le nano-pi, donc celui-ci est simulé à travers le clignotement d’une LED.

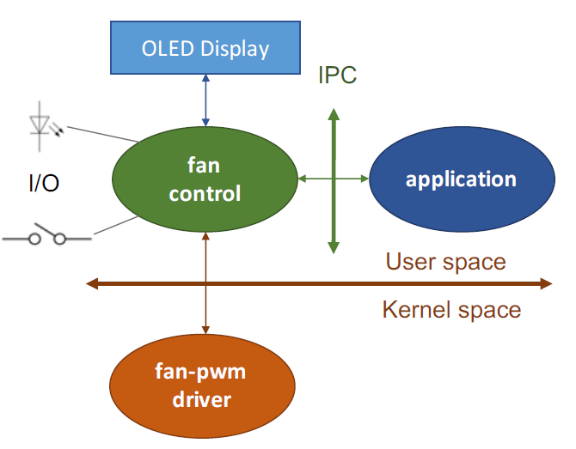
En premier lieu il faut réaliser un module (driver) contrôlant le clignotement de la LED et récupérant la température du processeur. Le module offre au travers du sysfs une interface permettant de choisir le mode manuel ou automatique et la fréquence de clignotement. Les conditions pour le clignotement sont :

* Température < 35°C => fréquence de 2Hz
* Température < 40°C => fréquence de 5Hz
* Température < 45°C => fréquence de 10Hz
* Température >= 40°C => fréquence de 20Hz

Deuxièmement un deamon doit être réaliser afin de visualiser les données sur l’écran LCD et de communiquer avec le driver au travers de deux interfaces :

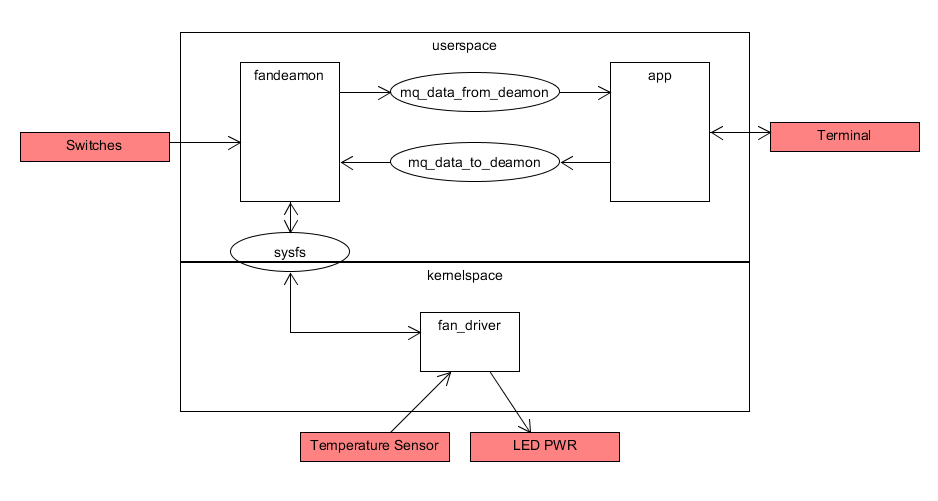
* Physiquement grâce aux boutons présents sur la carte avec :
  + S1 pour augmenter la fréquence
  + S2 pour diminuer la fréquence
  + S3 pour changer de mode
* Virtuellement avec une interface IPC au choix pour dialoguer avec une application utilisateur

Finalement une application fournit une interface en ligne de commande et dialogue avec le deamon.

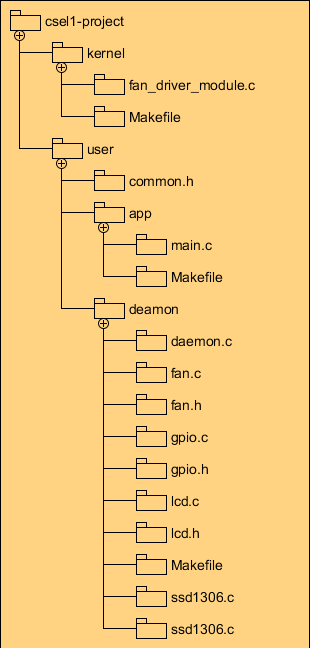


## Architecture

La figure ci-dessous représente l’architecture générale du système :



Avec comme fichiers sources l’arborescence ci-après.



### Kernel

Le module réalisé présente trois attributs sous */sys/devices/platform/fan-driver*, *freq, mode* en lecture et écriture et *temp* en lecture seule. La fréquence ne peut être enregistrée que en mode manuelle et le mode automatique régule la température tout seul grâce à un timer qui adapte sa période en fonction de la température pour plus de facilité. Le module est installé sur la cible depuis la machine hôte depuis */kernel* avec *sudo make install* afin de pouvoir gérer le module avec *modprobe*.

### Deamon

Dans le répertoire deamon les fichiers *fan.\**  présentent l’interface avec le module afin de lire et écrire les différents attributs, de plus des fonctions d’initialisation et déinitialisation s’occupe elles-mêmes de charger le module. Les fichiers *gpio.\** permettent de configurer les boutons et la led. Les fichiers *lcd.\** quand à eux dispose de l’interface pour le LCD et ainsi initialiser, mettre à jour et nettoyer l’écran.

Toute la logique est présente dans le fichier *deamon.c*. Au lancement du programme celui-ci configure le driver avec le mode par défaut qui est en automatique, après avoir chargé le driver. Le programme se démonise afin de passer en arrière-plan puis la LED et les boutons sont configurés ainsi qu’un timer qui s’occupe d’éteindre la LED lors de son timeout. Deux messages queues sont configurées afin de communiquer avec l’application et ainsi avoir un lien bidirectionnel. Les descripteurs de fichiers de la LED, des boutons, du timer et de la message queue faisant le lien application->deamon sont enregistrés pour le traitement d’événement. Finalement la boucle principale ne fait que d’appeler les callbacks nécessaires selon les événements survenus et mettre à jour le LCD quand un changement de valeurs arrive. Le traitement des événements est réalisé avec trois fonctions :

* ipc\_handler : prend en compte les données provenant de l’application, soit le mode, la fréquence, une demande de terminaison de programme et renvoie un feedback à l’application avec les données mise à jour.
* button\_handler : prend en compte les pressions de boutons et applique les changements de valeurs nécessaire puis notifie l’application des changements.
* timer\_handler : pour éteindre la LED après 200ms

### App

Le programme principal offre une simple ligne de commande avec :

*automatic* passer en mode automatique

*manual* passer en mode manuel

*<value>*  fréquence en mode manuel (2, 5, 10 ou 20)

*help* affiche les commandes

*exit* pour terminer le programme

Le programme se *fork* lui-même et créer un processus enfant :

* listen\_to\_deamon\_process : qui va continuellement lire les données de notification du deamon au travers de la message queue correspondante et afficher les données

Le parent est :

* shell\_process : qui réagit aux commandes entrées par l’utilisateur et va réaliser l’action nécessaire.

### Utilisation

Comme indiqué le module est chargé par le deamon, le deamon est à lancer manuellement ainsi que l’application. Lorsque la commande *exit* est reçue par l’application le deamon est alors terminé, lui-même s’occupe de décharger le module.

## Synthèse

## Feedback

## Annexes

Les codes sources sont disponible dans l’archive fournie.