CSEL

Rapports des TP

Git du projet : https://github.com/Mathistis/csel-workspace

Fribourg, 14 juin 2022

Auteurs
Macherel Rémy
remy.macherel@master.hes-so.ch
Raemy Mathis
mathis.raemy@master.hes-so.ch

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

Version: 1.0

Table des matières

Raemy Mathis, Macherel Rémy, CSEL

1. Introduction

Ce rapport décrit l'architecture ainsi que le développement et les fonctionnalités du mini-projet réalisé lors du cours *MA-CSEL* suivi lors du semestre de printemps 2022 du master MSE. Ce projet consiste à mettre en pratique les notions vues dans le cours par l'intermédiaire de l'implémentation d'un gestionnaire de ventilateur pour le processeur de la cible. Notre cible n'ayant pas de réel ventilateur, son fonctionnement sera simulé par le clignotement d'une LED symbolisant la fréquence de celui-ci ainsi qu'un écran OLED affichant quelques valeurs importantes.
Le but du travail est donc de concevoir une application permettant de simuler la gestion de la vitesse de rotation d'un ventilateur en fonction de la température du processeur. Les fonctionnalités suivantes seront donc implémentées :

— Supervision de la température du processeur et la gestion de la vitesse de clignotement de la LED à l'aide d'un module noyau.

— Un daemon en espace utilisateur qui offrira des services pour une gestion manuelle et prendra en compte la gestion des appuis sur les boutons afin d'augmenter la vitesse de rotation, de la diminuer et de passer du mode manuel à automatique.
Ce daemon pourra également, à l'aide d'une interface IPC, communiquer avec une application de type *CLI* pour la gestion du clignotement et du mode.

— Une application *CLI* pour piloter le système via l'interface IPC.

2. Architecture logicielle

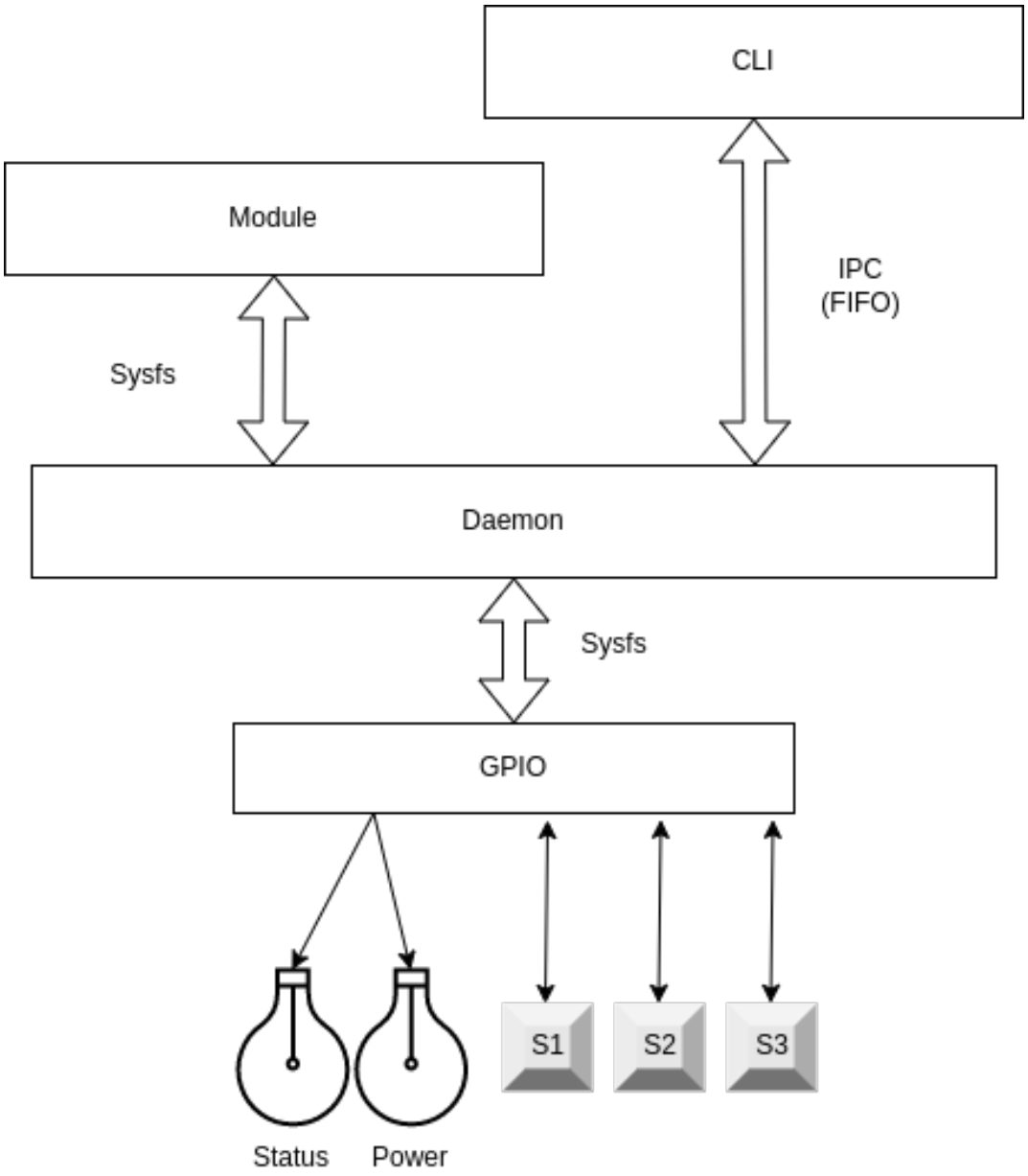

FIGURE 2.1: Diagramme de principe de l'application

3. Conception du module

Ce chapitre traite du développement du module noyau permettant de surveiller la température du processeur ainsi que la gestion du clignotement de la LED status. Ce module utilise le sysfs afin de gérer les deux modes de fonctionnement (manuel/automatique) et la fréquence de clignotement de la led Status. Selon les spécifications, les différentes fréquences de clignotement de la led sont :

- Température $< 35^{\circ}C \rightarrow$ fréquence de 2Hz
- Température $< 40^{\circ}C \rightarrow$ fréquence de 5Hz
- Température $< 45^{\circ}C \rightarrow$ fréquence de 10Hz
- Température $\geq 45^{\circ}C \rightarrow$ fréquence de 20Hz

FIGURE 3.1: Fréquences pour la led en mode automatique

Dans le module, nous avons séparé les codes en deux parties, un fichier *controller.c* (avec son .h dédié) qui représente une bibliothèque permettant la gestion des modes du programme. Ce controller permet d'initialiser les différents éléments utilisés dans le module (comme gpio, capteur de température, timers, etc.), il sert en quelque sorte d'interface de gestion des éléments du module noyau.
Le deuxième fichier *gpio.c* (également accompagné de son .h) permet quand à lui la gestion des gpio ainsi que leur initialisation.
Un autre code présent dans l'utilisation du module est le *temp_controller*, celui-ci permet à l'aide de la bibliothèque *linux/thermal.h* de récupérer la température actuelle du processeur.
Toutes ces fonctions sont alors utilisées (via le controller) dans le fichier *skeleton.c* dans celui-ci se trouvent les différentes méthodes permettant d'actualiser via le sysfs les états des leds.
Ce module va également créer et initialiser dans le sysfs les fichiers nécessaires à la transmission de nos informations.

```c
static struct class *sysfs_class;
static struct device *sysfs_device;

DEVICE_ATTR(is_manu, 0664, sysfs_show_is_manu, sysfs_store_is_manu);
DEVICE_ATTR(freq, 0664, sysfs_show_freq, sysfs_store_freq);

static int __init skeleton_init(void)
{
    int status = 0;

    sysfs_class = class_create(THIS_MODULE, "led_class");
    sysfs_device = device_create(sysfs_class, NULL, 0, NULL, "led_device");
    if (status == 0)
        status = device_create_file(sysfs_device, &dev_attr_is_manu);
    if (status == 0)
        status = device_create_file(sysfs_device, &dev_attr_freq);
    if (status == 0)
        status = init_controller(&is_manu, &freq_manual);
    set_mode(MODE_AUTO);
    pr_info("Linux module skeleton loaded\n");
    return 0;
}
```

FIGURE 3.2: Création des fichiers nécessaires dans le sysfs

4. Conception du daemon

Ce chapitre traite du développement du deamon offrant les services permettant la gestion de la fréquence ainsi que le choix du mode.