1.3- Pilotage de GPIO en sortie en C

Allumer une LED

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
        FILE * value;
        FILE * direction;
        direction = fopen("/sys/class/gpio/gpio19/direction","w");
        fprintf(direction, "out");
        fclose(direction);
        value = fopen("/sys/class/gpio/gpio19/value","w");
        fprintf(value, "0");
        fclose(value);
        return EXIT_SUCCESS;
}
```

On utilise la fonction `fopen()` pour ouvrir le fichier direction du gpio 19 (LED) en mode écriture: `w` pour configurer la direction en out. On ferme le fichier avec `fclose()`.
On ouvre ensuite le fichier valeur du gpio 19 puis on écrit 0. La LED s'allume.

Eteindre une LED

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
        FILE *value = fopen("/sys/class/gpio/gpio19/value","w");
        fprintf(value,"1");
        fclose(value);

        return EXIT_SUCCESS;
}
```

On utilise la fonction `fopen()` pour ouvrir le fichier valeur du gpio 19 (LED) en mode écriture: `w`.
On écrit ensuite 1 dans le fichier valeur du gpio 19. La LED s'éteint donc.

Configuration d'un GPIO en sortie

On peut ensuite généraliser les 2 codes ci dessus:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char * argv []){

        char path1[25] = "/sys/class/gpio/gpio";
        char path2[25] = "/direction";

        char path3[25] = "/sys/class/gpio/gpio";
        char path4[25] = "/value";

        strcat(path1,argv[1]);
        strcat(path1,path2);

        strcat(path3,argv[1]);
        strcat(path3,path4);


        FILE * direction;
        direction = fopen(path1,"w");
        fprintf(direction, "out");
        fclose(direction);

        FILE * value;
        value = fopen(path3,"w");
        fprintf(value, argv[2]);
        fclose(value);




        return EXIT_SUCCESS;
}
```

Maintenant, on peut maintenant configurer n'importe quel port en sortie et lui donner une valeur.

Explications:

Déclarations de variables :

```c
char path1[25] = "/sys/class/gpio/gpio";
char path2[25] = "/direction";
char path3[25] = "/sys/class/gpio/gpio";
char path4[25] = "/value";
```

Quatre tableaux de caractères sont déclarés pour stocker les chemins vers les fichiers de contrôle GPIO.

Construction des chemins :

```c
strcat(path1, argv[1]);
strcat(path1, path2);

strcat(path3, argv[1]);
strcat(path3, path4);
```

Les chemins sont construits en concaténant la valeur du premier argument de la ligne de commande (`argv[1]` : le numéro de port) avec les différentes parties des chemins.

Exemple:
`./configGPIO_out.o 19 0` : on allume la LED du GPIO 19
`./configGPIO_out.o 19 1` : on éteint la LED

Clignoter une LED

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char * argv []){

        char path1[25] = "/sys/class/gpio/gpio";
        char path2[25] = "/direction";

        char path3[25] = "/sys/class/gpio/gpio";
        char path4[25] = "/value";

        strcat(path1,argv[1]);
        strcat(path1,path2);

        strcat(path3,argv[1]);
        strcat(path3,path4);


        FILE * direction;
        direction = fopen(path1,"w");
        fprintf(direction, "out");
        fclose(direction);

        FILE * value;
        while(1){
                value = fopen(path3,"w");
                fprintf(value, "0");
                fclose(value);
                sleep(2);
                value = fopen(path3,"w");
                fprintf(value, "1");
                fclose(value);
                sleep(2);
        }

        return EXIT_SUCCESS;
}
```

Le code est semblable à celui de la configuration GPIO en sortie. On ajoute seulement une boucle pour mettre 0 puis 1 dans la valeur du GPIO avec des pauses de 2 secondes.

Exemple:
`./clignLED.o 19` : on fait clignoter la LED GPIO19
`./clignLED.o 23` : on fait clignoter la LED GPIO23

Clignoter une LED avec une fréquence de clignotement

```c
int main(int argc, char * argv []){

        char path1[25] = "/sys/class/gpio/gpio";
        char path2[25] = "/direction";

        char path3[25] = "/sys/class/gpio/gpio";
        char path4[25] = "/value";

        strcat(path1,argv[1]);
        strcat(path1,path2);

        strcat(path3,argv[1]);
        strcat(path3,path4);

        struct timespec timesleep;
        if(atoi(argv[2])==1){
                timesleep.tv_sec=1;
                timesleep.tv_nsec=0L;
        }
        else{
                timesleep.tv_sec=0;
                timesleep.tv_nsec=1000000000L/atoi(argv[2]);
        }




        FILE * direction;
        direction = fopen(path1,"w");
        fprintf(direction, "out");
        fclose(direction);

        FILE * value;
        while(1){
                value = fopen(path3,"w");
                fprintf(value, "0");
                fclose(value);
                nanosleep(&timesleep, NULL);
                value = fopen(path3,"w");
                fprintf(value, "1");
                fclose(value);
                nanosleep(&timesleep,NULL);
        }

        return EXIT_SUCCESS;
}
```

Explications

Initialisation de la structure `timesleep` :

```c
struct timespec timesleep;
```

La structure `timesleep` est utilisée pour définir la durée de sommeil entre les changements d'état de la LED.

Condition pour définir `timesleep` :

```c
if (atoi(argv[2]) == 1) {
    timesleep.tv_sec = 1;
    timesleep.tv_nsec = 0L;
} else {
    timesleep.tv_sec = 0;
    timesleep.tv_nsec = 1000000000L / atoi(argv[2]);
}
```

Selon la valeur du deuxième argument de la ligne de commande (`argv[2]`), la durée de sommeil (`timesleep`) est définie. Si la valeur est égale à 1, la LED change d'état toutes les secondes. Sinon, elle change d'état toutes les nanosecondes.