



UNIVERSITÉ DE TECHNOLOGIE DE
COMPIÈGNE

MI11

MI11 - Rapport du TP 2 Linux embarqué

Clément BLANQUET et Rafik CHENNOUF

Juin 2017

Sommaire

1	Rapport TP 2 - Linux embarqué	3
	Exercice 1 : Hello World	3
	1.1 Question 1.1	3
	1.2 Question 1.2	3
	1.3 Question 1.3	3
	1.4 Question 1.4	4
	Exercice 2 : Clignotement des LEDs	4
	2.1 Question 2.1	4
	2.2 Question 2.2	5
	2.3 Question 2.3	5
	Exercice 3 : Boutons poussoirs	7
	3.1 Question 3.1	7
	3.2 Question 3.2	7
	3.3 Question 3.3	7
	3.4 Question 3.4	7
	Exercice 4 : Charge CPU	10
	4.1 Question 4.1	10
	4.2 Question 4.2	10

Table des figures

1.1	Commande <i>file</i>	3
1.2	Commande <i>source</i>	3
1.3	Commande <i>file</i>	4
1.4	Résultat <i>Hello World</i>	4
1.5	Fichiers des LEDs	5
1.6	Manipulation des LEDs	5
1.7	Résultat du evtest sur <code>/dev/input/event1</code>	7

Rapport TP 2 - Linux embarqué

Exercice 1 : Hello World

1.1 Question 1.1

Après avoir compilé notre programme *Hello World* avec **gcc** nous avons exécuté la commande `file` pour obtenir des informations sur l'exécutable. Nous nous sommes rendu compte que cet exécutable a été compilé pour une architecture classique x86/64 et non ARM. C'est pour cela que le programme ne peut pas s'exécuter sur la cible.

```
mill@mill-VirtualBox ~/Documents/tp6 $ file main
main: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=41339e3d9c8d801a732fb7004c7bc66e29f5eaca, not stripped
```

FIGURE 1.1 – Commande *file*

1.2 Question 1.2

Avant de pouvoir cross-compiler notre programme il faut activer l'environnement de cross-compilation via la commande `source` sur le fichier `/opt/poky/1.7.3/environment-setup-armv7a-vfp-neon-poky-linux-gnueabi`.

```
mill@mill-VirtualBox ~ $ source /opt/poky/1.7.3/environment-setup-armv7a-vfp-neon-poky-linux-gnueabi
mill@mill-VirtualBox ~ $ echo $CC
arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=softfp -mfpu=neon --sysroot=/opt/poky/1.7.3/sysroots/armv7a-vfp-neon-poky-linux-gnueabi
```

FIGURE 1.2 – Commande *source*

1.3 Question 1.3

En utilisant de nouveau la commande `file` après la cross-compilation, nous constatons que le nouveau exécutable a bien été compilé pour une architecture ARM.

```

mill@mill-VirtualBox ~/Documents/tp6 $ gcc -o main main.c
mill@mill-VirtualBox ~/Documents/tp6 $ ls
main  main.c
mill@mill-VirtualBox ~/Documents/tp6 $ file main
main: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.32, BuildID[sha1]=87a3c670badb9a2b8a6e1debc66fb67b52fbbc8e, not stripped

```

FIGURE 1.3 – Commande *file*

1.4 Question 1.4

Voici le programme *Hello World* :

```

1 #include <stdio.h>
3 int main()
4 {
5     printf("Hello World !");
6     return 0;
7 }

```

Ainsi que le résultat :

```

mill@mill-VirtualBox ~/Documents/tp6 $ ssh root@192.168.1.6
root@devkit8600:~# ls
libxml2 2.9.1-r0_armv7a-vfp-neon.ipk  main
root@devkit8600:~# ./main
Hello World !root@devkit8600:~#

```

FIGURE 1.4 – Résultat *Hello World*

Exercice 2 : Clignotement des LEDs

2.1 Question 2.1

Pour accéder aux LEDs il suffit de manipuler les fichiers
 /sys/class/leds/user_led/brightness
 pour la LED utilisateur et
 /sys/class/leds/sys_led/brightness
 pour la LED système.

```
root@devkit8600:~# cat /sys/class/leds/user_led/brightness
0
root@devkit8600:~# cat /sys/class/leds/sys_led/brightness
0
```

FIGURE 1.5 – Fichiers des LEDs

2.2 Question 2.2

L'allumage d'une LED se fait en écrivant le caractère '1' dans le fichier correspondant et l'éteignage se fait en écrivant le caractère '0' comme on peut le voir sur la copie d'écran suivante.

```
root@devkit8600:~# echo 1 > /sys/class/leds/sys_led/brightness
root@devkit8600:~# echo 0 > /sys/class/leds/sys_led/brightness
root@devkit8600:~# echo 1 > /sys/class/leds/user_led/brightness
root@devkit8600:~# echo 0 > /sys/class/leds/user_led/brightness
```

FIGURE 1.6 – Manipulation des LEDs

2.3 Question 2.3

Le programme suivant allume en alternance la LED utilisateur et la LED système chaque seconde :

```
1 #include <stdio.h>
  #include <unistd.h>
3 #include <fcntl.h>
  #include <sys/types.h>
5 #include <sys/stat.h>

7 int main()
  {
9   int sys_led, user_led;

11  sys_led = open("/sys/class/leds/sys_led/brightness", O_RDWR);
    // ouverture des fichiers
13  user_led = open("/sys/class/leds/user_led/brightness", O_RDWR);

15  if(sys_led == -1 || user_led == -1)
    {
17    perror("Erreur d'ouverture de l'un des fichiers");
      return 0;
19  }
```

```
21 while(1)
22 {
23     if(write(user_led, "0", 1) == -1) // Éteignage de la LED user
24         perror("Erreur d'écriture");
25
26     if(write(sys_led, "1", 1) == -1) // allumage de la LED
27         perror("Erreur d'écriture");
28
29     sleep(1); // attente d'une seconde
30
31     if(write(sys_led, "0", 1) == -1) // Éteignage de la LED
32         perror("Erreur d'écriture");
33
34     if(write(user_led, "1", 1) == -1) // allumage de la LED user
35         perror("Erreur d'écriture");
36
37     sleep(1); // attente d'une seconde
38 }
39
40 close(sys_led); // fermeture
41 close(user_led); // des fichiers
42
43 return 0;
44 }
```

```

root@devkit8600:~# evtest /dev/input/event1
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "gpio-keys"
Supported events:
  Event type 0 (EV_SYN)
  Event type 1 (EV_KEY)
    Event code 1 (KEY_ESC)
    Event code 59 (KEY_F1)
    Event code 102 (KEY_HOME)
Properties:
Testing ... (interrupt to exit)
Event: time 1492179011.627030, type 1 (EV_KEY), code 1 (KEY_ESC), value 1
Event: time 1492179011.627034, ----- SYN REPORT -----
Event: time 1492179011.806361, type 1 (EV_KEY), code 1 (KEY_ESC), value 0
Event: time 1492179011.806365, ----- SYN REPORT -----
Event: time 1492179020.838352, type 1 (EV_KEY), code 102 (KEY_HOME), value 1
Event: time 1492179020.838356, ----- SYN REPORT -----
Event: time 1492179021.034120, type 1 (EV_KEY), code 102 (KEY_HOME), value 0
Event: time 1492179021.034122, ----- SYN REPORT -----
Event: time 1492179022.975083, type 1 (EV_KEY), code 59 (KEY_F1), value 1
Event: time 1492179022.975091, ----- SYN REPORT -----
Event: time 1492179023.172103, type 1 (EV_KEY), code 59 (KEY_F1), value 0
Event: time 1492179023.172106, ----- SYN REPORT -----

```

FIGURE 1.7 – Résultat du `evtest` sur `/dev/input/event1`

Exercice 3 : Boutons poussoirs

3.1 Question 3.1

La cible dispose de trois boutons utilisateur (HOME, BACK, MENU) et un bouton de reset. On y accède via le fichier `"/dev/input/event1"` dans lequel toutes les informations quant aux pressions et relâchement de ces boutons seront écrites.

3.2 Question 3.2

Pour tester en ligne de commande, on peut utiliser `"evtest /dev/input/event1"` (voir figure 1.7)

Les valeurs sont soit 1 soit 0 selon l'état du bouton : pression ou relâchement.

3.3 Question 3.3

La structure `input_event` est déclarée dans le fichier suivant :
`" /opt/poky/1.7.3/sysroots/armv7a-vfp-neon-poky-linux-gnueabi/usr/include/linux/input.h"`

Au début de notre fichier C, on ajoute donc : `"#include <linux/input.h>".` Voici le programme que nous avons codé. Celui-ci lit les événements indiqués dans le fichier `"/dev/input/event1"`, affiche ces événements dans le terminal et allume :

- La `sys_led` en cas de pression sur le bouton MENU
- La `user_led` en cas de pression sur le bouton BACK
- Les deux LEDs en cas de pression sur la touche HOME

3.4 Question 3.4

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <fcntl.h>
4 #include <linux/input.h>
5 #include <sys/types.h>
6 #include <sys/stat.h>
7
8 int main()
9 {
10     struct input_event event_buttons;
11     int sys_led, user_led, buttons;
12
13     buttons = open("/dev/input/event1", O_RDONLY);
14
15     if(buttons == -1)
16     {
17         perror("Erreur d'ouverture de l'un des fichiers");
18         return 0;
19     }
20
21     sys_led = open("/sys/class/leds/sys_led/brightness", O_RDWR);
22     user_led = open("/sys/class/leds/user_led/brightness", O_RDWR);
23
24     if(sys_led == -1 || user_led == -1)
25     {
26         perror("Erreur d'ouverture de l'un des fichiers");
27         return 0;
28     }
29
30     while(1)
31     {
32
33         read(buttons, &event_buttons, sizeof(event_buttons));
34
35         int code = (int)event_buttons.code;
36         int value = (int)event_buttons.value;
37
38         switch(code)
39         {
40             case KEY_F1: // MENU
41                 if(value == 1)
42                 {
43                     printf("MENU PUSH\n");
44                     if(write(sys_led, "1", 1) == -1)
45                         perror("Erreur d'écriture");
46                 }
47             else
```

```
48     {
50         printf("MENU RELEASE\n");
51         if(write(sys_led, "0", 1) == -1)
52             perror("Erreur d'écriture");
53     }
54     break;
55
56     case KEY_ESC: // BACK
57         if(value == 1)
58         {
59             printf("BACK PUSH\n");
60             if(write(user_led, "1", 1) == -1)
61                 perror("Erreur d'écriture");
62         }
63         else
64         {
65             printf("BACK RELEASE\n");
66             if(write(user_led, "0", 1) == -1)
67                 perror("Erreur d'écriture");
68         }
69         break;
70
71     case KEY_HOME: // HOME
72         if(value == 1)
73         {
74             printf("HOME PUSH\n");
75             if(write(user_led, "1", 1) == -1)
76                 perror("Erreur d'écriture");
77             if(write(sys_led, "1", 1) == -1)
78                 perror("Erreur d'écriture");
79         }
80         else
81         {
82             printf("HOME RELEASE\n");
83             if(write(user_led, "0", 1) == -1)
84                 perror("Erreur d'écriture");
85             if(write(sys_led, "0", 1) == -1)
86                 perror("Erreur d'écriture");
87         }
88         break;
89     }
90 }
91 if(close(sys_led) == -1)
92     perror("Erreur de fermeture de l'un des fichiers");
93 if(close(user_led) == -1)
94     perror("Erreur de fermeture de l'un des fichiers");
95 if(close(buttons) == -1)
96     perror("Erreur de fermeture de l'un des fichiers");
```

```
98 }  
    return 0;
```

Exercice 4 : Charge CPU

4.1 Question 4.1

Voici notre programme :

```
1 #include <stdio.h>  
2 #include <unistd.h>  
3 #include <fcntl.h>  
4 #include <sys/types.h>  
5 #include <sys/stat.h>  
6 #include <sys/time.h>  
  
8 #define TAILLE 10000  
  
10 int main()  
11 {  
12     int j = 0, tempstotal = 0;  
13     struct timeval debut, fin, res;  
14     gettimeofday(&debut, NULL);  
15     for(j = 0; j < TAILLE; j++)  
16     {  
17         usleep(1000);  
18     }  
19     gettimeofday(&fin, NULL);  
20     timersub(&fin, &debut, &res);  
21     int tempstotal = res.tv_sec * 1000000 + res.tv_usec;  
22  
23     printf("Temps total : %d\n", tempstotal);  
24  
25     return 0;  
26 }
```

A COMPLETER

4.2 Question 4.2

Voici notre programme modifié :

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <fcntl.h>
4 #include <sys/types.h>
5 #include <sys/stat.h>
6 #include <sys/time.h>
7
8 #define TAILLE 10000
9
10 int main()
11 {
12     int j = 0, min, max, moy = 0, tempstotal = 0;
13     int tab[TAILLE];
14     struct timeval debut, fin, res;
15
16     for (j = 0; j < TAILLE; j++)
17     {
18         gettimeofday(&debut, NULL);
19         usleep(1000);
20         gettimeofday(&fin, NULL);
21         timersub(&fin, &debut, &res);
22         tab[j] = res.tv_sec * 1000000 + res.tv_usec;
23         moy += tab[j];
24     }
25
26     moy = moy / TAILLE - 1000;
27     min = tab[0];
28     max = tab[0];
29
30     for (j = 1; j < TAILLE; j++)
31     {
32         if (min > tab[j])
33             min = tab[j];
34
35         if (max < tab[j])
36             max = tab[j];
37     }
38
39     min -= 1000;
40     max -= 1000;
41
42     printf("Moyenne : %d\nMaximum : %d\nMinimum : %d\n", moy, max, min)
43         ;
44
45     return 0;
46 }
```

Voici les temps relevés :

- Sans stress :
 - Moyenne : 85 ms
 - Maximum : 772 ms
 - Minimum : 13 ms
- Avec stress :
 - Moyenne : 766 ms
 - Maximum : 304397 ms
 - Minimum : 27 ms