Université de Technologie de Compiègne

MI11

Rapport du TP2 linux embarqué

Clément BLANQUET et Rafik CHENNOUF

Juin 2017

Sommaire

Table des figures

Chapitre 1

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 exécute 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.

```
mi11@mi11-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**.

```
mi11@mi11-VirtualBox ~ $ source /opt/poky/1.7.3/environment-setup-armv7a-vfp-neo
n-poky-linux-gnueabi
mi11@mi11-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.

```
mi11@mi11-VirtualBox ~/Documents/tp6 $ $CC -o main main.c
mi11@mi11-VirtualBox ~/Documents/tp6 $ ls
main  main.c
mi11@mi11-VirtualBox ~/Documents/tp6 $ file main
main: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked (u
ses shared libs), for GNU/Linux 2.6.32, BuildID[sha1]=87a3c670badb9a2b8a6e1debc66fb
67b52fbbc8e, not stripped
```

FIGURE 1.3 – Commande *file*

1.4 Question 1.4

Voici le programme *Hello World* :

```c
#include <stdio.h>

int main()
{
    printf("Hello World !");
    return 0;
}
```

Ainsi que le résultat :

```
mi11@mi11-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 :

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

int main()
{
  int sys_led, user_led;

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

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

```
21    while(1)
      {
23      if(write(user_led, "0", 1) == -1) // éteingage de la LED user
          perror("Erreur d'ecriture");
25
        if(write(sys_led, "1", 1) == -1)    // allumage de la LED système
27        perror("Erreur d'ecriture");

29      sleep(1);              // attente d'une seconde

31      if(write(sys_led, "0", 1) == -1)  // éteingage de la LED système
          perror("Erreur d'ecriture");
33
        if(write(user_led, "1", 1) == -1)   // allumage de la LED user
35        perror("Erreur d'ecriture");

37      sleep(1);              // attente d'une seconde
      }
39
      close(sys_led);    // fermeture
41    close(user_led);   // des fichiers

43    return 0;
}
```

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)

```
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

Les valeurs sont soit à 1 soit à 0 selon l'état du bouton : pressé ou relâché.

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

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <linux/input.h>
#include <sys/types.h>
#include <sys/stat.h>

int main()
{
  struct input_event event_buttons;
  int sys_led, user_led, buttons;

  buttons = open("/dev/input/event1", O_RDONLY);

  if(buttons == -1)
  {
    perror("Erreur d'ouverture de l'un des fichiers");
    return 0;
  }

  sys_led = open("/sys/class/leds/sys_led/brightness", O_RDWR);
  user_led = open("/sys/class/leds/user_led/brightness", O_RDWR);

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

  while(1)
  {

    read(buttons, &event_buttons, sizeof(event_buttons));

    int code = (int)event_buttons.code;
    int value = (int)event_buttons.value;

    switch(code)
    {
      case KEY_F1: // MENU
        if(value == 1)
        {
          printf("MENU PUSH\n");
          if(write(sys_led, "1", 1) == -1)
            perror("Erreur d'ecriture");
        }
        else
        {
```

```
            printf("MENU RELEASE\n");
            if(write(sys_led, "0", 1) == -1)
              perror("Erreur d'ecriture");
          }
        break;

        case KEY_ESC: // BACK
          if(value == 1)
          {
            printf("BACK PUSH\n");
            if(write(user_led, "1", 1) == -1)
              perror("Erreur d'ecriture");
          }
          else
          {
            printf("BACK RELEASE\n");
            if(write(user_led, "0", 1) == -1)
              perror("Erreur d'ecriture");
          }
        break;

        case KEY_HOME: // HOME
          if(value == 1)
          {
            printf("HOME PUSH\n");
            if(write(user_led, "1", 1) == -1)
              perror("Erreur d'ecriture");
            if(write(sys_led, "1", 1) == -1)
              perror("Erreur d'ecriture");
          }
          else
          {
            printf("HOME RELEASE\n");
            if(write(user_led, "0", 1) == -1)
              perror("Erreur d'ecriture");
            if(write(sys_led, "0", 1) == -1)
              perror("Erreur d'ecriture");
          }
        break;
    }
  }
  if(close(sys_led) == -1)
    perror("Erreur de fermeture de l'un des fichiers");
  if(close(user_led) == -1)
    perror("Erreur de fermeture de l'un des fichiers");
  if(close(buttons) == -1)
    perror("Erreur de fermeture de l'un des fichiers");

  return 0;
```

```
98 }
```

Exercice 4 : Charge CPU

4.1 Question 4.1

Voici notre programme :

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>

#define TAILLE 10000

int main()
{
  int j = 0, tempstotal = 0;
  struct timeval debut, fin, res;
  gettimeofday(&debut, NULL);
  for(j = 0; j < TAILLE; j++)
  {
    usleep(1000);
  }
  gettimeofday(&fin, NULL);
  timersub(&fin, &debut, &res);
  int tempstotal = res.tv_sec* 1000000 + res.tv_usec;

  printf("Temps total : %d\n, tempstotal);

  return 0;
}
```

On relève un temps de 10,9 ms.

4.2 Question 4.2

Voici notre programme modifié :

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>

#define TAILLE 10000

int main()
{
  int j = 0, min, max, moy = 0, tempstotal = 0;
  int tab[TAILLE];
  struct timeval debut, fin, res;

  for(j = 0; j < TAILLE; j++)
  {
    gettimeofday(&debut, NULL);
    usleep(1000);
    gettimeofday(&fin, NULL);
    timersub(&fin, &debut, &res);
    tab[j] = res.tv_sec* 1000000 + res.tv_usec;
    moy += tab[j];
  }

  moy = moy / TAILLE - 1000;
  min = tab[0];
  max = tab[0];

  for(j = 1; j < TAILLE; j++)
  {
    if(min > tab[j])
      min = tab[j];

    if(max < tab[j])
      max = tab[j];
  }

  min-=1000;
  max-=1000;
  printf("Moyenne : %d\nMaximum : %d\nMinimum : %d\n", moy, max, min)
    ;
  return 0;
}
```

Voici les temps relevés :

— Sans stress :
  — Moyenne : 85 ms
  — Maximum : 772 ms
  — Minimum : 13 ms

```
root@devkit8600:~# ./cpu
Moyenne : 85
Maximum : 772
Minimum : 13
```

FIGURE 1.8 – Résultat sans stress

— Avec stress :
  — Moyenne : 766 ms
  — Maximum : 304397 ms
  — Minimum : 27 ms

```
root@devkit8600:~# stress --cpu 100 --timeout 100s &
[1] 1580
root@devkit8600:~# stress: info: [1580] dispatching hogs: 100 cpu, 0 io, 0 vm, 0 hdd

root@devkit8600:~# ./cpu
Moyenne : 766
Maximum : 304397
Minimum : 27
```

FIGURE 1.9 – Résultat avec stress

On se rend compte qu'avec un stress, le temps maximum et par conséquent le temps moyen explosent. En effet, il arrive très fréquemment que notre programme soit interrompu au profit du stress que l'on a lancé en même temps. Notre tâche est donc fortement ralentie. Le minimum, lui, augmente mais beaucoup moins que le maximum car il est aussi possible que notre programme ne soit beaucoup interrompu sur une itération.

On pourrait, pour améliorer ces résultats, rendre notre programme temps réel et lui assigner une priorité maximale.