Systèmes

Temps

Réel

Liste des TD :

Séance n° 1 : Etude de l'ordonnancement sous Linux (PC sous Debian)
Séance n° 2 : Génération de signaux temps réel via des GPIO sous Linux (carte Beaglebone)
Séance n° 3 : Gestion des événements sous Linux (étude des temps de réponse)

Document initialement rédigé par G. Jacquet et modifié par A. Aubert

Consignes pour l'utilisation de la carte BeagleBone.

Dans cette série de TP, vous allez utiliser une carte BeagleBone (sauf pour la première séance). Cette carte permet un accès facile aux ports d'entrées/sorties du micro-contrôleur am335x de Texas Instrument. On pourra y travailler soit en baremetal , soit avec un système d'exploitation mono-tâche, soit sous Linux (système d'exploitation multi-tâches).

Le développement d'applications pour cette carte se fera à partir d'un PC avec la chaîne de développement croisée Buildroot sous Linux Debian. Ce système d'exploitation sera mise en œuvre sur une machine virtuelle « VirtualBox » à partir de Windows : le nom de la machine est elec6, vous devez la charger à partir du partage enseign. Le répertoire d'installation par défaut sera d:\MV\elec6.

Deux utilisateurs sont définis :

- root (superutilisateur avec tous les droits sur la machine) mot de passe admintest
- tpuser (utilisateur classique avec des droits réduits) mot de passe tpuser

Remarque importante : la carte Beaglebone s'alimente en 5V directement via le port USB.

Autre remarque importante : cette carte « low cost » ne possède pas de protections sur ses entrées/sorties, il faudra donc être très prudent sur la connexion des fils extérieurs ainsi que sur la programmation des ports pour ne pas réaliser de court-circuit.

Séance n°3 : Gestion des événements sous Linux (étude des temps de réponse)

Préalable

Sous votre répertoire de travail (/home/tpuser), copier le sous-répertoire TD3_event à partir du partage.

IV.1 Temps de réponse à un événement, gestion par polling.

IV.1.1 Utilisation du pseudo-répertoire /sys/class/gpio

Dans le sous-répertoire Q1_sys_button_poll, vous trouverez le fichier sys_button1.c.
Le programme suivant attend au sein d'une boucle de test une action sur un bouton poussoir relié au port gpio2_8. Lors de l'appui, le programme fait changer 10 fois l'état du port gpio2_6. .

```
#include <fcntl.h>
#include "port_beagle.h"

void write_sys(char *filename, char *str)
{
  int gpio_reg ;

  gpio_reg = open (filename, O_WRONLY);
  write (gpio_reg,str,strlen(str)+1) ;
  close (gpio_reg) ;
}

main()
{
  int i , bt;
  int bt_left ;
  char str[3] ;

  write_sys("/sys/class/gpio/export",GPIO2_6);
  write_sys("/sys/class/gpio/gpio70/direction","out") ;
  write_sys("/sys/class/gpio/export",GPIO2_8) ;
  write_sys("/sys/class/gpio/gpio72/direction","in") ;

  bt = 0 ;
  while (bt==0)
  {
//     usleep(1) ;
       bt_left = open ("/sys/class/gpio/gpio72/value", O_RDONLY);
       read (bt_left,str,2) ;
       close (bt_left) ;
       if (str[0]=='0') bt=1;
  }

  i = 0 ;
  while (i<10) {
       write_sys("/sys/class/gpio/gpio70/value","1") ;
       write_sys("/sys/class/gpio/gpio70/value","0") ;
       i++ ;
       }
  return (0) ;
}
```

Mesurer le temps de réponse entre le front descendant du port relié au bouton et le début des oscillations (on prendra soin de faire passer le port de sortie à son état actif d'abord, puis de le faire revenir à son état repos à la fin). Mesurer 10 fois ce temps, quelle est sa valeur moyenne, est-il constant ?

IV.1.2 Surveillance de 2 événements

Créer un deuxième exemplaire du programme précédent que vous nommerez sys_button2.c. Modifier ce dernier fichier pour attendre un événement sur le port gpio2_12, l'horloge sera générée sur le port gpio2_10. Lancer les 2 programmes de manière concurrente. Mesurer 10 fois le temps de réponse sur un appui sur le port gpio2_8, quelle la moyenne du temps mesuré, est-il constant ?

IV.1.3 Surveillance de 2 événements avec mise en veille de la tâche

Modifier les 2 fichiers précédents pour ajouter dans la boucle d'attente avant l'ouverture du pseudo-fichier associé au bouton un usleep(1) pour forcer la tâche à laisser sa place. Mesurer 10 fois le temps de réponse sur un appui sur le port gpio2_8, quelle la moyenne du temps mesuré, est-il constant ?

IV.2 Surveillance de 2 événements via un module noyau

Aller dans le répertoire Q2_LKM_button. A partir du code d'un module noyau déjà existant, écrire un programme réalisant la même opération qu'au III.1.1. Mesurer 10 fois le temps de réponse sur un appui sur le port gpio2_8, quelle la moyenne du temps mesuré, est-il constant ?
Créer un deuxième module noyau pour la surveillance du deuxième bouton poussoir. Lancer le aussi. Refaire les mêmes mesures.

IV.3 Surveillance de 2 événements en accès direct mémoire.

Aller dans le sous-répertoire Q3_button_mem. A partir du code déjà utilisé pour la création d'horloges avec la librairie tsegpio, écrire un programme (en espace utilisateur) pour surveiller le bouton relié au port gpio2_8. Mesurer 10 fois le temps de réponse sur un appui sur le port gpio2_8, quelle la moyenne du temps mesuré, est-il constant ?
De la même manière que précédemment, lancer aussi un programme pour surveiller le deuxième bouton poussoir.

IV.4 Temps de réponse à un événement, gestion par interruption.

IV.4.1 Déclenchement d'une interruption sur une modification d'état du pseudo-fichier /sys/class/gpio

Aller dans le répertoire Q4_1_button_select. En utilisant la fonction select(), surveiller l'action sur le bouton poussoir relié au port gpio2_8. Mesurer 10 fois le temps de réponse sur un appui sur le port gpio2_8, quelle la moyenne du temps mesuré, est-il constant ?
Lancer un deuxième programme surveillant l'autre bouton poussoir. Refaire les mêmes mesures.

IV.4.2 Déclenchement d'une interruption sur appui sur le bouton poussoir via un module noyau

Aller dans le sous-répertoire Q4_2_LKM_interrupt. Reprendre le code du développement de driver et réaliser la même fonction que précédemment.