Sujet systèmes embarqués – examen M2

J.-M Friedt, 3 novembre 2018

Tous les documents sont autorisés, toutes les consultations sur internet sont autorisées mais les communications par internet ou téléphone mobile sont **proscrites**. Les connexions internet sont enregistrées aux cours de l'examen.

Nous nous proposons d'étudier l'environnement de communication IIO et en particulier les interfaces créées pour divers périphériques.

1. Quelle commande permet de rechercher récursivement la présence d'une chaîne de caractères dans un répertoire et ses sous-répertoires ? Plusieurs réponses possibles
2. Nous proposons dans `/home/jmfriedt/exam2017_M2.tar.gz` une archive contenant un pilote IIO fonctionnel qui expose une interface de type *source de tension*. Identifier le fichier d'entête contenant les définitions des périphériques de type IIO (`/usr/src/linux-headers-XXX/include/` serait un bon point de départ pour la recherche, avec `XXX` la version du noyau).
   — Quelle est la version du noyau ?
   — Fournir 4 autres types d'interfaces matérielles reconnues par cette API.
3. Modifier le pilote fourni à `/home/jmfriedt/exam2017_M2.tar.gz` afin d'implémenter un accéléromètre : quelle modification effectuez-vous ?
4. Démontrer la compilation et le chargement du pilote sur PC (il se pourrait qu'il faille définir une variable d'environnement de façon appropriée, en fonction de la version du noyau, pour compiler ce pilote) en fournissant la liste des points de communication créés lors du chargement du pilote.
5. Démontrer la capacité à lire l'identifiant de ce périphérique : indiquer quel pseudo-fichier contient cette information ainsi que le chemin pour y accéder. Quel est le nom du pilote ?

Nous désirons maintenant porter ce pilote que nous venons de modifier à l'environnement de la carte Olinuxino A13 et son processeur ARM, dont les périphériques sont décrits par le mécanisme du *devicetree*

6. récupérer sur le PC une copie du *devicetree* se trouvant sur la première partition de la carte SD montée sur la carte A13, et afficher le code source (version du fichier `.dts`) de ce fichier (on pourra donner comme réponse les 3 premières lignes du code source correspondant)
7. ajouter une entrée qui décrit l'accéléromètre en fournissant par exemple une variable `adresse` qui donnerait l'adresse de ce périphérique sur son bus de communication
8. modifier le pilote proposé ci-dessus pour lire cette variable dans le devicetree
9. compiler le devicetree, placer l'image binaire ainsi générée sur la première partition de la carte SD (on pourra prendre soin de conserver une copie du fichier original avant de l'écraser), rebooter la carte A13
10. démontrer la capacité du pilote compilé pour processeur ARM à lire le contenu de la variable dans le *devicetree*.
11. renvoyer cette information lors de la lecture sur un pseudo-fichier approprié créé par le pilote IIO.

1 Questions de cours

12. Quel sous-répertoire de buildroot contient les applications pour l'hôte (PC) ?
13. Quel est l'appel système qui ne respecte pas la philosophie de "tout est fichier" dans un pilote classique (`/dev`) unix ?
14. Quelle structure de données contient les fonctions appelées lors de l'accès à un module au travers de `/dev` ?
15. Comment afficher la liste des modules chargés et liés au noyau ?
16. Quelle fonction est appelée lors de l'initialisation d'un pilote IIO ?
17. Quel mécanisme permet de créer le point de communication entre l'espace utilisateur et un module noyau ?
18. À quelle déficience un système temps-réel tente-t-il de pallier ?
19. Quel mécanisme protège l'accès concurrent à une variable ou une structure de données ?
20. Donner deux implémentations, dans le noyau Linux, du principe mentionné dans la question 8 : qu'est-ce qui les différencie ?

Solutions :

1. `find . -name '*' -exec grep chaine {} \;` ou `grep -r chaine *`
2. depuis `/usr/src/linux-headers-4.8.0-2-common`, la commande `grep -r VOLTAGE * | grep IIO` nous informe que `include/uapi/linux/iio/types.h` contient la définition des types de canaux respectant l'interface IIO. La version du noyau est fournie par `uname -a` ou `uname -r`. Parmi les autres méthodes connues de IIO, `uapi/linux/iio/types.h` nous informe de l'existence en plus de `IIO_VOLTAGE`, de `IIO_ANGL_VEL` pour les gyromètres, `IIO_MAGN` pour les magnétomètres, `IIO_LIGHT` pour les capteurs d'intensité lumineuse, `IIO_INTENSITY` pour les ampèremètres, `IIO_PROXIMITY` pour les indicateurs de distance, `IIO_TEMP` pour les thermomètres, ...
3. remplacer `IIO_VOLTAGE` par `IIO_ACCEL`
4. `export REPERTOIRE=/usr/src/linux-headers-4.8.0-2-amd64/` pour définir le répertoire contenant les sources du noyau, puis `make` et charger le pilote ainsi que la plateforme afin de créer le périphérique. Initialement le pilote crée

   ```
   $ ls /sys/bus/iio/devices/iio\:device0
   dev  in_voltage101_raw  name  power  subsystem  uevent
   ```

   et après ajout de l'accéléromètre

   ```
   const struct iio_chan_spec composant_channels[] = {
    {.type = IIO_VOLTAGE,                          \
     .channel = (101),                             \
     .info_mask_separate = BIT(IIO_CHAN_INFO_RAW), \
     .address = (1),                               \
     .indexed = 1,                                 \
    },
    {.type = IIO_ACCEL,                            \
     .channel = (102),                             \
     .info_mask_separate = BIT(IIO_CHAN_INFO_RAW), \
     .address = (1),                               \
     .indexed = 1,                                 \
    }
   };
   ```

   en précisant qu'il y a deux canaux (`indio_dev->num_channels = 2;`) :

   ```
   dev  in_accel102_raw  in_voltage101_raw  name  power  subsystem  uevent
   ```

5. ```
   $ cat /sys/bus/iio/devices/iio\:device0/name
   pilote exam
   ```
6. sur A13, après avoir configuré la liaison IP sur USB :

   ```
   # mount /dev/mmcblk0p1 /mnt/
   ```

   et copier depuis le PC `/mnt/sun5i-a13-olinuxino-micro.dtb` soit le fichier devicetree compilé en format binaire (dtb), qui se décompile en son code source par `dtc -I dtb -O dts fichier.dtb`.
7. on ajoute dans le dts

   ```
   dummy_entry {compatible="composant1"; adresse=<56>;};
   ```

   avec `composant1` le nom du composant associé au pilote, donné dans le module `composant.c` par

   ```
   static const struct platform_device_id composant_id[] = {
          {"composant1", ID_COMPOSANT12}, {} };
   ```

8. dans `static int composant_probe(struct platform_device *pdev)` :

   ```
   u32 val32;
   of_property_read_u32(np, "adresse", &val32);
   ```

9. `dtc -O dtb -o fichier.dtb fichier.dts` puis copier le fichier binaire résultant du PC vers le répertoire `/mnt` sur lequel la première partition de la carte SD a été montée.
10. démontrer la capacité du pilote compilé pour processeur ARM à lire le contenu de la variable dans le *devicetree*.
11. renvoyer cette information lors de la lecture sur un pseudo-fichier approprié créé par le pilote IIO.
12. `host` ou `host/usr/bin`
13. `ioctl`
   ```

14. `static struct file_operations`
15. `lsmod`
16. `probe`
17. `unregister_chrdev` ou `misc_register` ou `platform_device_register_simple` selon la nature du pilote et son mode de création du point d'entrée dans `/dev`.
18. latence non-bornée – intervalle de temps inconnu entre le déclenchement d'un évènement (interruption par exemple) et prise en compte de cet évènement.
19. accès mutuellement exclusif – `mutex`
20. `mutex`, `semaphore` ou `spinlock`. Dans les deux premiers cas la tâche quitte la queue de l'ordonnanceur, dans le dernier cas la boucle est active. Par ailleurs, mutex est binaire, tandis que semaphore est un compteur.