TP Temps Réel

Ludovic Saint-Bauzel <saintbauzel@isir.upmc.fr> and Jérôme Pouiller <jezz@sysmic.org>

Polytech Paris UPMC - 2018

Table des matières

1 Avant de commencer

Cette section est purement informative. Il n'y a pas de questions à l'intérieur.

1.1 Documentation

1.1.1 Références des pages de man

Comme le veut l'usage, les références des pages de man sont donnés avec le numéro de section entre parenthèses. Ainsi, *wait(2)* signifie que vous pouvez accéder à la documentation avec la commande `man 2 wait`. Si vous omettez le numéro de section, `man` recherchera la première page portant le nom `wait` (ce qui fonctionnera dans 95% des cas). Le numéro de section vous donne aussi une indication sur le type de documentation que vous aller trouver. D'après *man(1)*, voici les différentes sections :

1. Executable programs or shell commands
2. System calls (functions provided by the kernel)
3. Library calls (functions within program libraries)
4. Special files (usually found in /dev)
5. File formats and conventions (e.g. /etc/passwd)
6. Games
7. Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
8. System administration commands (usually only for root)
9. Kernel routines [Non standard]

Vous pouvez trouver une version en ligne de la plupart des pages de man sur The Friendly Manual (https://www.gnu.org/manual/manual.fr.html).

Vous trouverez en particulier les pages relatives à l'API du noyau Linux qui ne sont pas présentes en local sur vos machine.

Prenez néanmoins garde aux éventuelles différences de versions d'API (très rare et quasiment exclusivement sur les pages relatives à l'ABI du noyau).

1.1.2 Standards

Vous subissez aujourd'hui tout le poids de l'histoire de la norme Posix (30 ans d'histoire de l'informatique). Ne soyez pas étonnés de trouver de multiples interfaces pour une même fonctionnalité. C'est particulièrement vrai sur les fonction relatives au temps.

Nous basons ces exercice sur la norme POSIX.1-2001 que vous pourrez trouver sur http://www.unix-systems.org/version3/online.html

Vous trouverez plus d'informations sur l'histoire des différentes normes existantes sur *standards(7)*.

1.2 Erreurs

Pour indiquer quelle erreur s'est produite, la plupart des fonctions Posix utilisent :
- soit leur valeur de retour
- soit une variable globale de type `int` nommée `errno`

Utilisez *strerror(3)* pour obtenir le message d'erreur équivalent au code de retour.

Exemple :

```
1 if (err = pthread_create(&tid, NULL, f, NULL))
    printf("Task create: %s", strerror(err));
  if (-1 == pause())
    printf("Pause: %m");
```

1.3 Compilation

Pour la plupart des exercices, vous aurez besoin de linker [1] avec `-lpthread` et `-lrt`.

Par ailleurs, nous vous conseillons fortement de compiler avec l'option `-Wall` de `gcc` et d'utiliser un système de Makefile.

1. C'est à dire la phase de compilation où vous liez tous vous fichiers `.o` ensemble

2 Modules dans le noyau

2.1 module

Cette section se focalise sur la réalisation et la manipulation de modules dans le noyau.

Question 2.1. *Etudiez* `module1.c`*, puis compilez le et chargez le dans le noyau. Vous trouverez une description des actions à entreprendre pour compiler un module dans l'appendice A.*

Documentation utile: *insmod, rmmod, lsmod, dmesg*

Question 2.2. *Modifiez le code pour visualiser un texte "Hello World!" dans les log du noyau lorsque le module se charge.*

Documentation utile: *printk, dmesg*

Question 2.3. *Un module est capable de fournir des informations concernant l'auteur, la licence d'utilisation, on qualifie ces informations de méta-informations. Pour cela, on utilise des macros qui n'ont* ***pas besoins d'être dans une fonction****. Ces macros permettent avec l'utilisation du programme* `modinfo` *de montrer les meta-informations du module. Ajoutez ces méta-informations à votre module.*

Documentation utile: *modinfo, MODULE_AUTHOR, MODULE_DESCRIPTION, MODULE_LICENSE*

Question 2.4. *Le mécanisme de module permet un chargement avec des paramètres variables au chargement. Modifiez votre code précédent en incluant un paramètre entier statique (*`static int`*)* `parm` *et en modifiant le* `printk` *pour visualiser sa valeur.*

Documentation utile: *module_param, MODULE_PARM_DESC*

2.2 char device

Question 2.5. *Etudiez le fichier* `driver\char.c`*. Compilez-le et chargez-le. Il est à noter que vous risquez d'avoir à modifier le fichier* `Makefile` *pour inclure l'objectif de compilation* `char.o` *qui désigne le fichier* `char.c` *et génèrera le fichier* `char.ko`*.*

Observez dans le dossier `/dev`*, un fichier mychar est apparu. Quelle variable/constante dans le code défini ce nom de fichier ?*

Faites `ls -l` *dans ce dossier et notez les informations de majeure et mineure pour ce fichier. De la même manière retrouvez le numéro de majeure dans le fichier* `/proc/devices`*. Quelle commande/ligne de code d'après vous provoque choisi cette majeure ?*

Question 2.6. *Modifiez ce code pour le rendre plus verbeux (*`printk`*) quand les fonctions char_open, char_close, char_read, char_write, char_ioctl s'exécutent.*

Recompilez et rechargez votre char device. Malheureusement vous n'avez pas les droits pour écrire ou lire sur le fichier que vous venez de créer. Ce n'est pas grave, on peut créer un nouveau fichier virtuel qui pointe sur le même driver et auquel on ajoute les droits. La commande à faire est la suivante :

```
sudo mknod -m 666 /tmp/mychar c XXX 0
```

où XXX est le numéro de majeur que vous avez identifié avec `ls -l` *ou dans le fichier* `/proc/devices`*.*

Question 2.7. *Commençons par utiliser la fonction* `ioctl`*. Faites un code "user" en C qui exécute l'action* `ioctl`*. Cela doit provoquer un* `printk` *dans les logs du noyau.*

Documentation utile: *open(2), ioctl(2)*

Question 2.8. *De même vous pouvez faire un write puis un read dans ce code "user" pour envoyer "hello world!" dans le fichier virtuel.*

Documentation utile: *read(2),write(2)*

Question 2.9. *A l'aide d'un buffer interne au driver, faites en sorte que le driver retienne les données du write. Et que le read permette de voir le contenu de ce que vous avez écrit.*

Documentation utile: *copy_from_user, copy_to_user*

Question 2.10. *Le but maintenant c'est de rendre le fichier bloquant en lecture lorsqu'il n'y a pas de données dans le buffer. Donc dans un premier temps faites 2 executables (un qui lit et un qui ecrit). Ensuite, faites la pause adéquat.*

Documentation utile: *wait_event_interruptible, wake_up_interruptible*

Question 2.11. *Maintenant, faites en sorte que le buffer soit dynamiquement créé.*

Question 2.12. *Nous pouvons utiliser la mineur pour gérer plusieurs fois le driver. Faites un code capable de gérer 2 mineures. Et donc qui vérifie la mineure avant de rentrer des données.*

Documentation utile: *iminor, MINOR*

A Écriture des modules noyau

A.1 Compilation

La manière classique de compiler un module noyau est de créer un Makefile contenant :

```
obj-m += module.o
```

Puis lancer la compilation avec :

```
make -C /lib/modules/$(uname -r)/build SUBDIRS=$(pwd) modules
```

Le système de compilation du noyau se chargera de compiler votre module avec les bonnes options.

Nous vous fournissons le fichier `Makefile` reprenant ces lignes.

A.2 Exécution

Les modules ne sont pas des applications. Ils n'ont pas de fonction `main`. Comme c'est souvent le cas lorsque l'on souhaite étendre une application en cours de fonctionnement, vous avez la possibilité de déclarer des fonctions qui seront appelées lors de certain évènements. Les fonctions les plus utiles sont déclarées par `MODULE_INIT` (chargement du module) et `MODULE_EXIT` (déchargement du module).

Vous pourrez charger le module à l'aide de *insmod(1)* et le décharger à l'aide de *rmmod(1)*.

Nous vous fournissons le fichier `module.c` qui vous donne un squelette de module pour le noyau Linux.

A.3 Log Buffer

Le noyau possède un buffer circulaire permettant de logger des messages. Il est possible d'écrire dans le buffer depuis un module à l'aide de *printk(9)* et des macros *pr_debug*, *pr_info*, *pr_notice*, *pr_warning*, *pr_err*, etc.... Il est possible de dumper le contenu du buffer avec la commande *dmesg(1)* (utilisant l'appel système *syslog(2)*). Il est aussi possible de suivre l'évolution du buffer avec la commande :

```
tail -f /var/log/kern.log
```

B Modalités de rendu

Votre rendu sera effectué sur la plateforme Moodle (https://moodle-sciences.upmc.fr/moodle-2019).

Vous devez rendre votre travail dans une archive gzippée. L'archive devra se nommer `TR_ROB5-nom.tar.gz` et se décompresser dans le répertoire du même nom.

Votre archives ne doit pas contenir de fichiers binaires. Il ne doit contenir que vos sources (fichiers sources, fichiers entêtes, Makefile, etc...).

Exemple de création de rendu :

```
make clean
cd ..
tar cvzf TR_ROB5-pouiller.tar.gz TR_ROB5-pouiller
```

Vous devrez envoyer déposer une version **à la fin de la séance** et vous pouvez faire évoluer ce document jusqu'au début du prochain TP.