S. Lavirotte
Polytech Nice Sophia
SI5 - Master 2 Ing Info

TD n° 3 et 4
Pilotes et Périphériques Logiciels

Le but de ce TD consiste à définir un pilote (`/dev/cutbuf`) où chaque périphérique qui lui est associé se comporte comme une sorte de cut-buffer dans lequel on peut stocker des caractères. Pour simplifier, la taille de chaque buffer sera fixée à 4ko (le buffer sera alloué lors du premier accès au périphérique).
Pour tester votre pilote, vous aurez besoin de quelques programmes utilitaires (`slow-cat` et `ioctl`). Ces programmes ainsi que des scripts de tests sont disponibles sur le serveur à l'adresse suivante :

https://lms.univ-cotedazur.fr/2022/mod/resource/view.php?id=230709

1 Introduction

Avant de commencer, quelques remarques :

- Pour allouer dynamiquement `n` caractères vous pourrez utiliser l'appel : `kmalloc(n, GFP_KERNEL);`
- La fonction `kmalloc` est définie dans `<linux/malloc.h>` pour les versions assez anciennes du noyau et `<linux/slab.h>` pour les dernières versions.
- Lorsque l'on essaie d'écrire plus de `4ko` dans un buffer, vous devrez signaler une erreur (e.g. `ENOMEM` est pas mal pour cela).
- le programme `slow-cat` utilisé dans les exemples est une version maison de `cat` qui affiche lentement le contenu du fichier qui lui est passé en paramètre en utilisant de petits buffers. Utilisez ce programme pour vérifier votre driver (surtout la deuxième version).

2 Mise en place : création des devices

Notre pilote gérera 4 devices différents (appelés `/dev/cutbuf1`, `/dev/cutbuf2`, ...).
Créez dans le répertoire `/dev` les fichiers spéciaux correspondants. Ces fichiers spéciaux (en mode caractère) auront un numéro majeur 100 et un numéro mineur compris entre 1 et 4. Définissez aussi le fichier spécial `/dev/cbstat` de majeur 100 et de mineur 0. Ce fichier nous servira pour visualiser l'état de notre pilote.

3 Première version : Opérations de base du driver

3.1 Description du comportement souhaité du périphérique

Dans cette version, notre pilote ne permettra pas l'accès concurrent à un même périphérique. Si un périphérique est utilisé, tout autre accès se soldera par un échec (i.e. on positionne `errno` à `EBUSY`). Un premier exemple de session est donné ci-dessous :

```
$(1)> echo Hello > /dev/cutbuf1; echo world > /dev/cutbuf2
$(2)> slow-cat /dev/cutbuf2
world
$(3)> slow-cat /dev/cutbuf1
Hello
$(4)> for i in `seq 256` ; do echo -n "0123456789ABCDEF" >> todel ; done
$(5)> cat todel > /dev/cutbuf1
cat : write error : Message too long
```

Cet exemple permet de voir que :

- chaque périphérique a son propre buffer
- que la lecture sur un périphérique permet de retrouver ce qu'on y a écrit.
- que le contenu d'un périphérique n'est pas perdu lors de sa lecture.

L'exemple suivant montre le comportement en écriture.

```
$(1)> echo Un > /dev/cutbuf1
$(2)> echo Deux > /dev/cutbuf1
$(3)> slow-cat /dev/cutbuf1
Deux
```

Comme on peut le voir, l'écriture sur un périphérique efface les données qui y ont été précédemment enregistrées.
Enfin, la session suivante montre une utilisation concurrente du pilote :

```
$(1)> for i in 1 2 3; do echo $i; done > /dev/cutbuf1
$(2)> slow-cat /dev/cutbuf1
1
2
3
$(3)> for i in 1 2 3; do echo $i; sleep 2; done > /dev/cutbuf1 &
[1] 2473
$(4)> slow-cat /dev/cutbuf1
slow-cat: Device or resource busy
$(5)>
[1] + 2473 done
$(5)> for i in 1 2 3; do; echo $i; sleep 2; done > /dev/cutbuf1
$(6)> slow-cat /dev/cutbuf1
1
2
3
```

Comme on peut le voir sur cet exemple, il n'est pas possible d'accéder à un périphérique si celui-ci est déjà en cours d'utilisation.

3.2 Mise en œuvre du comportement décrit

Implémenter dans un module le comportement du driver tel que décrit précédemment

4 Deuxième version : Statistiques

On désire utiliser le fichier spécial `/dev/cbstat` pour observer notre pilote depuis l'extérieur.
Un exemple d'utilisation de ce fichier spécial est défini ci-dessous :

```
$(1)> slow-cat /dev/cbstat
cutbuf1: buffer=0xcfb1b000 6 char. in_use=0
cutbuf2: buffer=0xcfb18000 21 char. in_use=0
cutbuf3: buffer=0xcfb1c000 2 char. in_use=1
cutbuf4: buffer=0x00000000 0 char in_use=0
```

Bien-sûr vous pouvez aussi afficher toute autre information qui vous semblera utile.
Ce périphérique ne sera pas accessible en écriture. Modifier pour cela le champ `f_op` du fichier au moment de l'ouverture pour substituer les fonctions de lecture et d'écriture sur ce périphérique particulier (i.e. ce ne sont pas les fonctions d'écriture et de lecture qui font le contrôle à chaque accès).

5 Troisième version : Services « ioctl »

On veut ajouter deux services à notre pilote : le mode debug et le mode bloquant :

- en mode debug, le driver écrira une trace à chaque accès à un périphérique dans le fichier de log
- en mode bloquant, un accès concurrent à un pilote ne se soldera plus par une erreur, mais bloquera le processus appelant tant que le processus qui utilise le pilote ne l'a pas libéré.

La mise en place du mode bloquant et du mode debug se feront par l'utilisation de la primitive ioctl. On utilisera pour cela les constantes suivantes[1]:

- TST_GETDBG (valeur `0`, un paramètre) permet de stocker dans la variable passée en paramètre la valeur du champ debug associé au périphérique.
- TST_SETDBG (valeur `1`, un paramètre) permet de mettre/enlever le mode debug
- TST_GETBLOCK (valeur `3`, pas de paramètre) permet de consulter le champ indiquant si les accès sont bloquants ou non sur un périphérique donné. La valeur de ce champ est la valeur de retour de `ioctl`.
- TST_SETBLOCK (valeur `4`, un paramètre) permet de mettre/enlever le mode bloquant.

Vous trouverez aussi dans l'archive téléchargée au début du TD le un fichier permettant de changer la configuration des périphériques : `ioctl.c`.

6 Pour aller plus loin : Améliorations possibles

On peut améliorer ce driver à assez peu de frais ; par exemple :

1. Modifier le driver pour que le numéro de majeur soit passé au moment du chargement dynamique.
2. Modifier le driver pour que le numéro de majeur soit alloué dynamiquement. Pour cela, le chargement et la création des fichiers spéciaux pourront être faits par un script Shell. Le numéro de device peut être obtenu par consultation du fichier `/proc/devices`.
3. La création des fichiers spéciaux est contraignante. On pourrait l'éviter en transformant notre pilote en une nouvelle entrée dans `/proc`.
4. Étendre notre pilote pour qu'il se comporte comme des pipes nommés Unix (i.e. écriture bloquante quand le buffer est plein, lecture bloquante tant qu'il n'y a pas d'écrivain). Attention, pour cette dernière extension, c'est un peu plus compliqué.

[1] Tout cela n'est pas bien symétrique et un peu compliqué. Ceci est volontaire et devrait vous permettre de voir les différents cas de figure possibles sur un appel de type `ioctl`.