Périphérique virtuel 'abcd'

Guillaume Chanel

1 Objectifs

Le but ce ce TP est de créer un module de périphérique virtuel. Ce module implémentera un buffer circulaire dont les pointeurs de lecture / écriture sont toujours identiques.

Les objectifs pédagogiques sont:

- développer un module linux;
- implémenter les réponses aux appels systèmes coté noyau;
- interpréter les résultats d'appels systèmes sur votre module.

2 Buffer circulaire

Le buffer sera représenté par un tableau de caractères (statique ou dynamique suivant les besoins). Le module permettra de lire et écrire dans ce tableau.

Notre buffer pourra avoir une taille spécifiée par l'utilisateur au chargement du module. Par défaut le buffer aura une taille de 26 caractères contenant l'alphabet.

Les lectures / écritures se feront de manière séquentielles, comme pour un fichier. Lorsqu'une opération (lecture / écriture) est effectuée, la prochaine opération se fera toujous en partant du caractère suivant. Autrement dit, si une lecture retourne "abc" alors une écriture de "01" effacera les caractère 'd' et 'e'. Nous avons donc un curseur partagé par les deux opérations.

Lorsqu'une opération atteinds la fin du buffer, elle ne sera pas complétée. Dans le cas d'une lecture on ne retournera que les caractères présents entre le curseur et la fin du buffer. Dans le cas d'une écriture on n'écrira que jusqu'à la fin du buffer. Il faudra cependant remettre le curseur au début du buffer pour les opérations suivantes.

Un inode devra être créé dans `/dev` avec le nom `abcd`.

Dans les versions le périphérique ne supportera pas les accès aléatoires (appel système `lseek`).

Vérifiez dans `/sys` et `/proc` le numéro majeur de votre module et les paramètres de votre module.

3 Travail à rendre

Nous allons implémenter deux version du buffer: une version globale et une version locale (voir ci-dessous). **Il faudra impérativement rendre les deux versions clairement séparées** (e.g. deux dossiers distinct dans votre zip).

Un troisième dossier contiendra les petits programmes demandés, avec un unique `makefile` pour les compiler.

Les réponses aux questions devront être founis dans un fichier `questions.md`.

4 Buffer comme variable globale

Pour commencer nous implémenteront le buffer comme une variable globale à notre module (potentiellement dans une structure globale).

Dans un premier temps, implémenter la fonctionalité de lecture. Puis:

1. Implémenter un court programme qui effectue un `read` de 50 caratères sur `/dev/abcd`. Quelle est la valeur retournée. Pourquoi ?
2. Tester la commande `cat /dev/abcd`. La lecture devrais se faire de manière continue. Pourquoi ? Quelle fonction C de lecture est utilisée par `cat` ? Que fais cette fonction ?
3. Essayer de lancer la commande `cat /dev/abcd` deux fois dans deux terminaux différents. Observez-vous des conflits de lecture ? Pourquoi ?

Dans un deuxième temps, implémenter la fonctionnalité d'écriture. Puis:

4. Implémenter un court programme qui effectue un `write` de 50 cacatères sur `/dev/abcd`. Quelle est la valeur retournée. Pourquoi ?
5. Tester la commande `echo ceci est une longue chaine de plus de 24 char > /dev/abcd`. Quel est le résultat ? Pourquoi ?
6. Tracer la commande ci-dessus avec `strace`. La commande fait-elle plusieurs appels systèmes `write` ? Pourquoi ?
7. Essayer de lancer la commande `echo` ci-dessus deux fois dans deux terminaux différents. Observez-vous des conflits d'écriture ? Pourquoi ?

5 Buffer comme variable locale dynamique

Dans cette section le buffer sera implémenté comme variable locale dynamique. L'idée est d'avoir un buffer distinct pour chaque processus utilisant notre périphérique virtuel.

Il faudra donc allouer de la mémoire sur l'appel système `open` et la désallouer lorsque il n'y a plus de référence sur le canal de communication.

Une fois le module implémenté et testé, répondre aux questions suivantes:

8. tester à nouveau la commande `echo` comme indiqué dans la question 7. Qu'observez-vous et pourquoi ?
9. implémenter un court programme qui ouvre le fichier de périphérique puis duplique le descripteur de fichier. Vérifier que: (*i*) lorsque l'on écrit sur un descripteur la modification est perçue par l'autre; (*ii*) la mémoire n'est pas libérée avant de fermer les deux descripteurs.