TP 2 – Pilote de périphérique

Les systèmes Unix disposent des pseudo-périphériques `/dev/null`, `/dev/zero`, `/dev/mem` et `/dev/kmem`. Ils sont tous les quatre gérés par un pilote unique et répondent aux spécifications suivantes :

- une lecture dans `/dev/null` retourne toujours 0 (fin de fichier) et toute écriture réussit, mais les données écrites sont perdues ;
- une lecture dans `/dev/zero` retourne le nombre demandé d'octets nuls, et toute écriture réussit, mais les données écrites sont perdues (comme pour `/dev/null`) ;
- `/dev/mem` correspond à la mémoire physique de l'ordinateur : une lecture ou écriture à l'offset o courant lit ou écrit les données à l'adresse physique o. Toute tentative d'accès à une adresse en dehors de la mémoire installée est une erreur.
- `/dev/kmem` correspond à la mémoire virtuelle du noyau : une lecture ou écriture à l'offset o courant lit ou écrit les données à l'adresse virtuelle o telle que vue par le noyau. Une tentative d'accès à une adresse ne correspondant pas à une adresse virtuelle existante est traitée comme si c'était une page vide (i.e. la lecture renvoie des octets nuls, et l'écriture est perdue).

L'objectif de ce TP est d'implémenter une version minimale de ce pilote. On ne s'intéressera ici qu'à la lecture.

Question 1

Cette question préliminaire a pour but de compléter xv6 pour réaliser ce TP.

1. Assurez-vous de repartir de vos fichiers sources modifiés après le TP précédent (implémentation de `lseek`).
2. Modifiez les fichiers `file.h` (champs `read` et `write` de la structure `devsw`), `fs.c` (appels via `devsw`) et `console.c` (fonctions `consoleread` et `consolewrite`) pour passer un paramètre supplémentaire `off` (de type `uint`) car celui-ci n'est pas fourni aux pilotes par xv6.

   Démarrez xv6 pour vérifier que l'accès à la console se déroule sans encombre.
3. Créez un nouveau programme utilisateur `mknod` qui prend 3 paramètres (un nom de fichier, un numéro de majeur et un numéro de mineur) et appelle la primitive `mknod` de xv6. N'oubliez pas de vérifier le code de retour et d'afficher un message en cas d'erreur.

   Réinstallez l'image disque (il suffit de démarrer xv6). Assurez-vous que vous pouvez bien créer un fichier spécial (par exemple `/toto`) sans erreur. Note : le programme `ls` fourni par xv6 affiche le type de l'inode, soit 3 pour les fichiers périphériques.
4. Créez un nouveau programme `testread` qui prend 3 paramètres : un nom de fichier, un offset o (en hexadécimal) et un nombre d'octets n. Il doit ouvrir le fichier en lecture, se déplacer avec `lseek` à l'offset o, lire n octets, et afficher le nombre d'octets lus et leur valeur hexadécimale. Pour simplifier, on pourra considérer que $n \leq 100$.

   Démarrez xv6 et vérifiez que votre programme fonctionne avec un fichier texte.
5. Adaptez votre implémentation de `lseek` en vous assurant, d'une part, de ne pas se limiter à la taille du fichier dans le cas d'un fichier spécial (type `T_DEV` dans l'inode), et d'autre part d'accepter des offsets compris entre à 2^{31} et $2^{32} - 1$ (utiles pour `/dev/kmem`). Pour ce dernier point, vous pouvez simplifier votre code de `lseek` pour ne considérer que le cas `SEEK_SET`.

Question 2

Cette question a pour but d'insérer le pilote minimum (`/dev/null`) dans le noyau.

1. Créez un nouveau fichier `drvmem.c` pour contenir le pilote. Dans un premier temps, on se limitera à gérer le périphérique `/dev/null` (vous pourrez l'appeler `/n` pour vous économiser de la frappe au clavier).

Ajoutez dans ce fichier `drvmem.c` trois fonctions `drvmemread`, `drvmemwrite` et `drvmeminit`. Cette dernière initialise la table `devsw` pour le numéro de majeur que vous aurez choisi, et que vous pourrez ajouter comme constante dans `file.h` (par analogie avec le majeur de la console). Appelez `drvmeminit` dans `main.c` (juste après l'initialisation de la console, par exemple). Complétez `defs.h`

Quelle valeur doit renvoyer votre fonction `drvmemread` ?

2. Démarrez xv6, posez un point d'arrêt dans `drvmemread`, créez un fichier spécial `n` avec le mineur 0, et utilisez votre programme `testread` pour vérifier que vous rentrez bien dans la fonction `drvmemread` et que `testread` lit bien 0 octet, quel que soit le nombre demandé.

Question 3

Cette question a pour but d'utiliser le numéro de mineur et d'implémenter `/dev/zero`.

1. Adaptez votre fonction `drvmemread` pour agir en fonction du numéro de mineur (localisé dans l'inode) : 0 pour `/dev/null` et 1 pour `/dev/zero`.

   Vous pouvez utiliser la fonction `memset` (définie dans `string.c`) pour implémenter le pilote.
2. Testez votre pilote en créant un fichier spécial `/z` avec le mineur 1.

Question 4

Il s'agit maintenant d'implémenter `/dev/mem`. On rappelle que xv6 considère que la mémoire physique est comprise entre les adresses `EXTMEM` et `PHYSTOP` (voir `memlayout.h`) soit 224 Mo et que cette mémoire est entièrement accessible par le noyau dans son espace d'adressage.

1. Ajoutez à votre pilote la gestion de `/dev/mem` avec le numéro de mineur 2. Vérifiez bien que vous n'accédez qu'aux adresses correspondant à la mémoire physique.

   Petit coup de pouce : vous pouvez utiliser la fonction `memmove` pour recopier une zone mémoire dans une autre, ainsi que les macros définies dans `memlayout.h`.
2. Testez votre pilote en créant un fichier spécial `/m` avec le mineur 2.

   Pour vérifier le bon fonctionnement, posez un point d'arrêt dans `drvmemread` et appelez `testread` avec la première adresse de la mémoire physique (offset = 0x10 0000). Une fois arrêté, utilisez la commande « `x /20bx 0x80100000` » de `gdb` pour afficher (commande `x`) 20 octets (format `b`) en hexadécimal (format `x`). Continuez l'exécution : votre programme `testread` doit afficher exactement les mêmes valeurs.

Question 5

Il s'agit maintenant d'implémenter `/dev/kmem`. Pour cela, il faut vérifier la présence des pages concernées, et si ce n'est pas le cas renvoyer des octets nuls sans risquer un défaut de page.

1. Ajoutez dans le fichier `vm.c` (et dans `defs.h`) la fonction :

   ```
   int kmemread (char *dst, uint off, int n)
   ```

   Cette fonction réalise le travail effectif du pilote pour ce mineur. Elle récupère l'offset et le nombre d'octets, et vérifie pour chaque page avec la fonction `walkpgdir` (interne à `vm.c`, ce qui explique pourquoi il faut ajouter votre fonction à ce fichier) que l'entrée est bien présente dans la table des pages. Si c'est le cas, son contenu est recopié à partir de l'adresse de destination. Sinon, l'adresse de destination est initialisée avec des zéros.
2. Ajoutez à votre pilote la gestion de `/dev/kem` avec le numéro de mineur 3. Il vous suffit d'appeler votre fonction.
3. Pour tester votre pilote, créez le programme `ps.c`. Celui-ci attend 2 arguments : le nom du fichier spécial et l'adresse de la variable `ptable` (que vous obtiendrez en consultant le fichier `kernel.sym` après compilation). Votre programme doit lire toute la variable `ptable` en mémoire, puis doit explorer cette table pour afficher les indications lisibles (essentiellement `pid`, `state` et `name`) des processus.