TP 1 – Primitive système lseek

Le noyau xv6 ne dispose pas de la primitive lseek, bien que les éléments nécessaires pour la réaliser soient présents (champ off de la structure file en particulier). Le but de ce TP est d'écrire cette primitive ainsi qu'un petit programme pour l'utiliser.

Note importante : le TP 2 supposera que vous avez terminé ce TP 1 et que vous avez une primitive lseek fonctionnelle. Il est donc important que vous terminiez ce TP et que vous ayez votre version disponible lors du TP 2.

Question 1

Cette question a pour but d'insérer le squelette de la primitive lseek dans le noyau.

- 1. Cherchez le source C dans lequel sont les primitives travaillant sur les fichiers. Localisez les primitives travaillant sur les fichiers ouverts et ajoutez à la suite la primitive lseek ne faisant que renvoyer la valeur 0 pour le moment.
- 2. Localisez la table permettant à xv6 d'aiguiller un appel système vers la fonction que vous venez d'écrire. Quelle est la structure de cette table ? Comment est-elle exploitée ? Comment xv6 récupère-t'il le numéro de la primitive à appeler?
 - Ajoutez une entrée dans cette table pour votre primitive.
- 3. Compilez le noyau xv6.

Question 2

On s'intéresse à présent à la manière dont un programme peut appeler une primitive système.

- 1. Pour qu'on puisse utiliser notre primitive, il faut définir une fonction lseek qu'un processus peut appeler en mode utilisateur. Cette fonction doit être définie dans le fichier usys. S.
 - Ajoutez cette fonction. Comment fonctionne-t'elle?
 - Que manque-t'il par rapport à un vrai système compatible POSIX?
- 2. Lorsqu'un source C référence la fonction <code>lseek</code>, son prototype doit être défini quelque part. Pourquoi? Pour simplifier, xv6 met tous ces prototypes dans <code>user.h</code>. Ajoutez votre prototype (on rappelle que <code>lseek</code> admet 3 arguments) à ce fichier. En l'absence de types POSIX définis par xv6, quels sont les types à utiliser dans le prototype et pourquoi?
- 3. La primitive lseek utilise les constantes SEEK_SET, SEEK_CUR et SEEK_END. Définissez-les dans le fichier fcntl.h qui définit d'autres constantes relatives aux ouvertures de fichier, et que les programmes utilisateurs peuvent inclure.
- 4. Écrivez un programme testlseek qui ne fait qu'appeler la primitive lseek, et ajoutez-le dans la liste des fichiers exécutables que make installe dans l'image disque (fs.img).
 - Vérifiez avec gdb que vous réussissez à rentrer dans la fonction que vous avez définie à la question précédente. Affichez la pile (commande where de gdb).

Question 3

Étendez le programme testlseek pour admettre 3 arguments : un nom de fichier et deux entiers o et n. Votre programme doit afficher n octets à partir de l'offset o dans le fichier. Si $o \ge 0$, l'offset est absolu (i.e. à partir du début du fichier). Si o < 0, l'offset doit être compté à partir de la fin du fichier.

En l'absence d'une implémentation fonctionnelle de lseek, vous pouvez tester votre programme sur un fichier texte existant (utilisez ls pour en trouver un). L'appel de lseek ne provoquera aucun déplacement de l'offset, votre programme affichera systématiquement le début du fichier, ce qui est suffisant pour tester.

Rappel: xv6 fournit un petit sous-ensemble de la bibliothèque standard du langage C, la variable ULIB du fichier Makefile cite les fichiers inclus avec chaque programme utilisateur. La fonction atoi du fichier ulib.c, en particulier, ne gère pas les arguments négatifs, vous aurez à l'étendre pour les gérer.

Question 4

Complétez maintenant le squelette de la primitive système lsek que vous avez rédigé à la première question. Cette primitive se contente de modifir l'offset déjà stocké dans une table du noyau (laquelle?). Pour simplifier :

- on n'acceptera les requêtes que pour les fichiers (et pas pour les tubes);
- on bornera le nouvel offset à l'intervalle [0..size] où size représente la taille du fichier.