TD1-2 et TP 1-2 – Premiers pas avec les appels système

Objectif TD1-2 et TP 1-2 : se familiariser avec le tableau des descripteurs de fichiers ainsi que les appels systèmes open, close, read, write et lseek.

Exercice 1 : open, read, write, lseek

1- Ecrire un programme qui ouvre des fichiers sans jamais les fermer. Combien de fichiers pouvez-vous ouvrir ?
2- Utiliser l'appel système open() pour ouvrir un fichier qui sera spécifié sur la ligne de commande. Lire les 10 premiers octets du fichier avec l'appel système read(), puis les afficher sur la sortie standard en utilisant write(). Rappel : 0=stdin, 1=stdout, 2=stderr.
3- Réécrire un programme cp simple (copie d'un fichier vers un autre). Pour créer le fichier "cible", n'oubliez pas d'utiliser les flags O_CREAT, O_TRUNC, et le troisième paramètre de open(), pour positionner correctement les droits du fichier nouvellement créé (regardez ce qui se passe si vous ne mettez pas cet argument).
4- Ecrire un programme qui se comporte comme la commande cat -n. Si aucun nom de fichier n'est donné le programme affiche les caractères venant de l'entrée standard à l'écran en ajoutant un numéro de ligne. Si des noms de fichiers sont spécifiés, le programme affiche le contenu du fichier en numérotant les lignes. Que se passe-t-il si l'on fournit le nom d'un fichier qui n'existe pas ?
5- Ecrire un programme qui ouvre un fichier ./res.data en lecture et qui lit 10 caractères à partir du 30ème caractère, puis les affichent sur la sortie standard (c'est à dire l'écran, fd = 1).

Exercice 2 : la table des descripteurs de fichiers

1- Ecrire un programme en C appelé pg1.c, utilisant la primitive read, dans lequel une fonction lire() tente de lire en une seule fois 100 caractères sur l'entrée standard (c'est à dire le clavier, fd = 0). Elle affichera sur la sortie d'erreur standard (c'est à dire l'écran, fd = 2) le nombre de caractères effectivement lu, puis les affichera sur la sortie standard (c'est à dire l'écran, fd = 1).
2- Ecrire un programme en C pg2.c qui utilisera la fonction lire() sans la modifier. Ce programme devra lire à partir d'un fichier ./test.data et devra écrire le résultat dans le fichier ./res.data (sortie d'erreur et sortie standard). Il est a rappelé que lors de l'ouverture d'un fichier, son descripteur est mis dans la première case libre du tableau de descripteurs de fichiers.
3- Placer la fonction lire() dans un fichier lire.c puis la compiler avec la commande cc -c lire.c -o lire.o. On peut utiliser la fonction lire.c depuis un programme pg3.c relié au premier par la commande cc pg3.c lire.o -o pg3. Ici aussi il ne faut pas modifier le programme de la fonction lire(). Pareillement à la question précédente, pg3.c devra lire à partir d'un fichier ./test.data et devra écrire le résultat dans le fichier ./res.data (sortie d'erreur et sortie standard).

Exercice 3 : Duplication de processus et partage des pointeurs de lecture ou d'écriture.

1- Ecrire en C un programme qui commence par un appel de fork() pour dupliquer le processus initial, le père, pour obtenir un fils. Le père et le fils ont ensuite le même comportement. Ils ouvrent le fichier "./test.data" puis y lisent 10 caractères et les affichent sur la sortie standard (écran) après avoir affiché leur identité. Effectuer ensuite le fork() juste avant la lecture. Répéter l'exécution plusieurs fois et déterminer quels sont les caractères lus par le processus père et le processus fils.

2- Dans le père, positionner le pointeur de lecture sur le 30ème caractère par appel de lseek() avant le fork(). Quelle est la position des caractères lus par le fils ensuite.
3- Modifier le programme pour que le fork() ait lieu avant l'ouverture du fichier. Comparer les résultats.