SR01 – TD8

Entrées/Sorties

Rappel de cours: Les E/S bas niveau

- **Primitives succeptibles d'être utilisées dans ce TD** (non exhaustif):
  open() read() write() close() dup() dup2() fcntl() flock() lseek() pipe() getdtablesize()
- **Les descripteurs** :
  A chaque canal d'E/S (fichier, /dev/tty, STDIN, STDOUT, ...) est associé un descripteur. **Chaque process** a une table de descripteurs.
  Les tois premières entrées de la table des descripteurs correspondent aux descripteurs :
  - de l'entrée standard (par défaut le clavier) : Entrée de rang STDIN_FILENO (0) de la table des descripteurs (Voir "/usr/include/unistd.h") : on parlera par abus de langage du descripteur 0.S
  - de la sortie standard (par défaut l'écran) : Entrée de rang STDOUT_FILENO (1) de la table des descripteurs : on parlera du descripteur 1.
  - de la sortie des messages d'erreur : Entrée de rang STDERR_FILENO (2). Elle contient généralement le même descripteur que l'entrée STDOUT_FILENO.

- Pour connaître la **taille de la table des descripteurs**, utiliser la fonction **getdtablesize()**"

- Un process "fils" **hérite** de la table des descripteurs de son process "père" lors d'un **fork()**, y compris de son contenu: après le fork, le fils a donc les mêmes fichiers ouverts que le père.

- **Description sommaire des primitives (fonctions) systèmes** :

  - **dup(fd)** : duplique le descripteur de rang "fd" et affecte la copie dans la **première entrée libre** de la table des descripteurs. Retourne le rang de cette entrée.

  - **close(fd)** : Détruit le descripteur de rang "fd" de la table : libère l'entrée. Pour un fichier, le fichier est fermé lors du "close" du dernier descripteur associé à ce fichier.

- **open(path,flags[,mode])** : Ouvre un fichier et retourne le rang du descripteur associé à ce fichier.

- **pipe(fd[2])** : Crée un "pipe" et retourne le rang des deux descripteurs associés à ce pipe. Sur la plupart des systèmes, ce qui est écrit en fd[1] est recupéré en fd[0] (Read-only) en mode FIFO (First In, First Out). Ce mécanisme peut être utilisé pour échanger des données entre process.
  Remarque: Sur le système que vous utilisez, un pipe est bi-directionnel; Ce qui est écrit d'"un coté" du pipe (quelqu'il soit) est récupéré de l'autre.

Exercice1: E/S bas niveau *[open() read() write() close()]*
> **cp ~sr01/td/ecrific.c .**
Examiner le source, le compiler et l'exécuter.
Remplacer la lecture de 10 entiers par 10 lectures d'un entier -> **ecrific10.c**

Exercice2: fork( ) , wait( ) et waitpid( )

1) créer un programme "**f1.c**" qui appelle fork() pour créer un process fils.
On fera écrire au process parent :

- ici le parent, mon pid est nnn, le pid de mon fils est nnn

Et au process fils :

- ici le fils, mon pid est nnn, le pid de mon père nnn.

2) Créer un programme "**f2.c**" qui crée deux fils (tous deux fils du même père). On fera écrire à chaque process les mêmes informations que **f1.c**.

3) Faites **cp f1.c f3.c**
puis, modifiez **f3.c** en ajoutant dans le père un appel à wait() pour attendre la fin de son fils, et dans le fils un appel à exit() en passant à exit
un argument (status de fin) qui sera récupéré dans le wait().

4) Faites **cp f2.c f4.c**
puis, modifiez **f4.c** en ajoutant dans le père un appel à waitpid() pour attendre la fin de **l'un de** ses fils. On fera autant d'appels à waitpid() qu'il y a de fils.

Exercice3: E/S bas niveau *[pipe(), héritage père->fils de la table des descripteurs après un fork()]*

1. Re-écrire **ecrific.c**. On ne passe plus par un fichier temporaire mais par un **pipe** -> **pipe1.c**

2. Ecrire un programme o🔲 un process **fils** exécute **ls -la** (en utilisant la fonction **execlp()**) et communique le résultat de cette commande par pipe au process **père** qui exécute la commande **wc-l** (ls -l donne le nomre des lignes) -> **pipe2.c**

Exercice4: E/S haut niveau *[fopen() fread() fwrite() fclose()]*

Les fonctions open(), read(), write(), close(), ... sont des appels systèmes qui accèdent au noyau du système. Elles utilisent la notion de **descripteur** de fichier.
Il existe une interface d'accès aux fichiers (**les flux**) qui sont des fonctions de la **bibliothèque C** : les fonctions **fopen(), fread(), fwrite(), fclose(), fflush(),...**.
Elles utilisent la notion de **flux (type FILE *)**.
Ces fonctions appartenant 🔲 la bibliothèque C sont définies dans le standard ANSI C et sont **plus portables** que les appels systèmes unix.
Il est possible d'obtenir le numéro de descripteur associé 🔲 un flux comme il est possible d'ouvrir un nouveau flux "autour" d'un descripteur donné.
cp ~sr01/td/fop.c .
Examiner le source, le compiler et l'exécuter.
Écrire un programme **foprw.c** qui va lire un fichier **fop.in**, ajouter **"xxx"** 🔲 la fin de chaque ligne et écrire le résultat dans **fop.out**.

On aura :

fop.in	**fop.out**
ligne_1 aaa bbb	ligne_1 aaa bbbxxx
ligne_2 dd ee	ligne_2 dd eexxx

Exercice5: E/S haut niveau *[freopen()]*

En utilisant **freopen()**, écrire un programme **freop.c** qui va successivement :

1. écrire des lignes sur l'écran avec un printf(),
2. rediriger stdout,
3. écrire les lignes suivantes 🔲crites avec des printf() dans le fichier **freop.log**