TP 7
Tuyaux

Durée prévue : 1 séance (1h30)

EXERCICE 1 ▶ Blocage

Dans le CM, nous avons insisté sur la nécessité de *fermer dès que possible les descripteurs de fichiers qui ne sont pas utilisés par le processus courant*. Dans cet exercice, nous allons expérimenter les risques de blocage engendrés quand les extrémités d'un pipe `p` ne sont pas fermées en suivant cette règle.

On rappelle que si `p` est un tube bien initialisé, `p[0]` sert pour la lecture, et `p[1]` sert pour l'écriture.

Nous vous fournissons un programme `desc_blocage.cpp` qui crée deux processus : le processus père écrit dans un tube pour envoyer des données à son fils ; et le fils lit à partir du tube et affiche ces données. On rappelle que *tant qu'il reste un écrivain sur le tube, même s'il est vide, l'appel à read est bloquant.*

1) Décommentez le « `close(p[1]);` » du père, en laissant la même ligne commentée pour le fils, compilez, puis testez : expliquez le comportement du programme (regarder du côté du fils).

2) Re-commentez le « `close(p[1]);` » du processus père, et décommentez la même ligne du côté du fils ; compilez, puis testez : expliquez le comportement du programme.

3) Décommentez « `close(p[1]);` » pour le père et pour le fils, et vérifiez que les deux processus se terminent normalement.

EXERCICE 2 ▶ Lecteur-rédacteur dans un tuyau

Vous devez écrire un programme dans lequel le processus principal crée un processus fils. Le père envoie au processus fils les caractères de `'0'` à `'9'` au travers d'un tube anonyme `p`. Le fils devra afficher sur sa sortie standard les caractères reçus sur le pipe ; lorsqu'il n'y aura plus de caractère à lire, il affichera le nombre de caractères lus au total, puis se terminera. Vous ferez en sorte que le père attende la terminaison de son fils avant de se terminer lui-même. Vous veillerez également à fermer les descripteurs de fichier dès qu'ils ne sont plus utilisés.

EXERCICE 3 ▶ Tuyaux et signal `SIGPIPE`

Dans la page `man 7 pipe`, on peut lire :

```
If all file descriptors referring to the write end of a pipe have been closed, then an attempt to read from the pipe
will see end-of-file (read will return 0). If all file descriptors referring to the read end of a pipe have been closed,
then a write will cause a SIGPIPE signal to be generated for the calling process.
```

Dans l'exercice 1, nous avons utilisé la valeur de retour 0 pour arrêter le travail du processus fils lorsque le processus père ferme son descripteur de fichier en écriture sur le pipe. Maintenant, nous allons tester le deuxième comportement indiqué par la page du manuel : s'il n'y a plus de lecteur potentiel sur le pipe, le processus qui tente d'écrire dessus reçoit le signal SIGPIPE. Nous allons utiliser un gestionnaire de signal pour cela et l'appel `sigaction()` pour l'installer. Cette fois on utilise le fichier `desc_sigpipe.cpp`.

1) Quelle est la valeur entière du signal `SIGPIPE` (commande `kill -l`) ?

2) Observez le code source :
- quel est le nom du gestionnaire de signal pour `SIGPIPE` que l'on installe dans le programme ?
- par quel message doit se terminer le processus père ?
- par quel message doit se terminer le processus fils ?

3) Lancez plusieurs fois le programme `desc_sigpipe` :
- comment se termine le processus fils ?
- comment se termine le processus père ?
- la suite des entiers affichés est-elle toujours la même ? Pourquoi ?
- le numéro du signal affiché est-il bien celui de la question précédente ?
- quel processus reçoit le signal `SIGPIPE` ?

4) Observez à nouveau le code source :
- expliquez plus en détail comment se termine le programme ?
- pour quels processus le nouveau gestionnaire de signal est-il installé ?

5) Modifier le programme de façon à ce que tous les descripteurs de fichiers soient fermés proprement lorsqu'on reçoit le signal `SIGPIPE`.