CP 2 : tubes & select

ASR2

1 Pipe

Écrivez un programme qui simule l'exécution de `top -b | grep root`
Pour faire cela, il faut :
— créer un tube `t` avec `pipe()`,
— lancer deux processus fils avec `fork()`,
— pour le premier processus, écraser la sortie standard `1` avec la sortie du tube `t[1]` (avec `dup2`, par exemple), puis faire l'`exec`,
— pour le second processus, écraser l'entrée standard `0` avec l'entrée du tube `t[0]`, puis faire l'`exec`.

2 Boucle (bonus)

Écrivez un programme qui lance deux processus : un sera `tee /dev/tty`, et l'autre sera `awk '{ print $1+1; fflush();}'`. La sortie du premier sera rédigée sur l'entrée du second, et inversement, de sorte à faire une boucle. Pour que cela donne quelque chose, le processus père devra initialiser la boucle avec un `0\n` initial.

Questions subsidiaires : que fait `tee /dev/tty` ? À quoi sert le `fflush()` ?

3 Kolakoski (super bonus)

Soit w un mot, c'est à dire une séquence lettres dans un alphabet. La *dérivée* de w est un mot sur l'alphabet $\mathbb{N}$ tel que la i-ème lettre de la dérivée est la taille du i-ème bloc dans w, où un bloc est un ensemble de même lettres consécutives. Par exemple, la dérivée du mot *abbacccdaab* est le mot 1213121.

Le *mot de Kolakoski* est le mot (infini) sur l'alphabet $\{1, 2\}$, commençant par 1, égal à sa propre dérivée. Il commence donc par 12211212212211 . . ., et la suite est uniquement définie. Ce mot est un casse-tête pour les chercheurs en combinatoire des mots : il n'y a quasiment que des conjectures, sans réussir à trouver des preuves...

Écrivez un programme qui prend un mot en entrée, et écrit en sortie sa dérivée. Puis utilisez le principe de la boucle de l'exercice précédent pour afficher le mot de Kolakoski (ou, plus simplement, une suite de préfixes de plus en plus grands du mot de Kolakoski).

Cela bloque rapidement... pourquoi ?

4 Premiers pas vers un serveur, avec select

4.1 Père et 1 fils (1/2)

Écrivez un programme qui crée un tube (`pipe`) et lance un processus fils (`fork`). Dans une boucle infinie, le fils écrira (`write`) à son père des nombres aléatoires entre 1 et 999, en caractères décimaux (`snprintf`) dans le tube. Le père lira (`read`) nombre envoyé par le fils, et l'affichera dans le terminal (`printf`).

Voyez vous un problème ? (Le problème peut dépendre, bien sûr, de votre implémentation.) Comment pourrait on y remédier ?

4.2 Père et 1 fils (2/2)

Modifiez le programme précédent pour que le fils, entre deux envois successifs, attende un temps aléatoire de quelques millisecondes (`usleep` et `rand`).

Le problème précédent est-il résolu par le seul fait de rajouter une temporisation ?

4.3 Père et 2 fils

Modifiez le programme précédent pour créer deux fils et deux tubes. Le père devra lire dans les deux tubes.

Voyez vous un problème ?

4.4 Père et 2 fils + non bloquant

Modifiez le programme précédent pour que la lecture du père soit non bloquant (`fcntl`).

Quel est le problème ?

4.5 Père et 2 fils + non bloquant + select

Modifiez le programme précédent pour que le père utilise `select` (ou `poll`) pour savoir quel tube n'est pas vide.

4.6 Notes

On verra qu'on peut faire un serveur pour un protocole réseau sur le même principe : un processus (ou thread) écoutera un ensemble de sockets réseau, non-bloquants, via select/poll.

5 gdb / valgrind

Utilisez `gdb` pour essayer de débugger le code :
`https://perso.ens-lyon.fr/michael.rao/ASR2/progs/list.c`

Ce code implémente une liste doublement chainée d'entiers. On peut ajouter un entier i à la liste en entrant "`+ i`" en entrée, et supprimer un entier avec "`- i`". Essayez de trouver des entrées qui le font planter ou faire des erreurs.

Idem pour le code suivant en C++ :
`https://perso.ens-lyon.fr/michael.rao/ASR2/progs/vector.cpp`

Idem avec `valgrind`.