

Programmation système

TP 5 – dup2() et pipe()

Exercice 1: Redirection simple

Écrivez un programme qui redirige la sortie standard vers un fichier, puis effectue un `ls` vers ce fichier. Il faut donc ouvrir ce fichier avec `open()`, puis utiliser `dup2()` afin que les accès ultérieurs à la sortie standard se fassent sur le fichier. Ensuite, on peut faire un `execvp()` pour lancer `ls -l`, et l’affichage se fera vers le fichier.

Exercice 2: Redirection après un fork()

1. Révisions sur `fork()` (car `fork()` est prérequis pour ce TP!) : écrivez un programme runner qui exécute le programme spécifié sur la ligne de commande, et en fonction de son code de retour, affiche "OK" ou "ERREUR".

Attention : il ne faut pas faire un simple `exec*()` mais un `fork() + exec*()`.

Rappel : pour récupérer le code de retour, utilisez `wait()` ou `waitpid()`; "OK" correspond à un code de retour égal à 0.

Exemple : `./runner ping -c1 tartiflette` doit afficher "ERREUR" (tartiflette ne répond pas, ce serveur n’existe pas). Par contre, `./runner ping -c1 google.com` fonctionne bien (le serveur existe), et le programme affiche "OK".

2. Modifiez le programme précédent pour qu’il n’affiche rien d’autre que "OK" ou "ERREUR", c’est-à-dire que le programme qui sert de “condition” ne doit rien afficher. Pour l’empêcher de faire de l’affichage, on propose de rediriger ses sorties vers le fichier `/dev/null`. Attention à ne détourner les sorties standard que dans l’enfant, afin que le parent puisse encore afficher "OK" ou "ERREUR" à la fin du programme.

Exercice 3: Conversion à travers un tube

En utilisant `fork()`, créez deux processus communiquant par un tube (lui-même créé avec l’appel système `pipe()`). L’enfant lira depuis l’entrée standard et recopiera dans le tube les caractères mis en majuscules avec `toupper()`. Le parent lira depuis le tube et écrira sur la sortie standard.

Exercice 4: L’enfant rationne et se suicide ; le parent en meurt de chagrin

Reprenez l’exercice précédent. Cette fois, c’est le parent qui écrit et l’enfant qui lit. L’enfant devra simplement répéter sur la sortie standard ce qu’il a lu, mais en s’arrêtant au bout de 10 caractères (comme la commande `head -c10`). Ensuite, l’enfant termine son exécution. On souhaite aussi que le parent termine lorsque l’enfant termine. Plusieurs solutions sont possibles :

- Fermer soigneusement les descripteurs de fichiers inutilisés (par exemple, après le `fork()`, chaque processus ferme la moitié du tube qu’il n’utilise pas). La prochaine écriture dans le tube provoquera un `SIGPIPE` dans le parent.

- Utiliser SIGCHLD afin que le parent constate la terminaison de l'enfant. (Les signaux n'ont pas encore été abordés en cours, mais vous pouvez prendre un peu d'avance si vous le souhaitez. Cf. `man 2 signal` et `man 7 signal`.)

S'il vous reste du temps:

Exercice bonus 1: Un pipeline simplifié

1. Écrivez un programme pipeline qui prend comme arguments deux commandes (sans options, ni arguments) et les exécute dans un pipeline. Par exemple, `./pipeline ls rev` devra fonctionner comme `ls | rev`, c'est à dire afficher toutes les lignes produites par `ls`, mais renversées. La valeur de retour de votre programme doit être celle de la dernière commande du pipeline.
2. Généralisez votre programme de façon à ce qu'il puisse connecter un nombre arbitraire de commandes.