Redirections et tubes

Thèmes traités

— redirections : principe, API Unix ;
— processus communiquant par tubes : architecture, protocole d'usage, API Unix ;

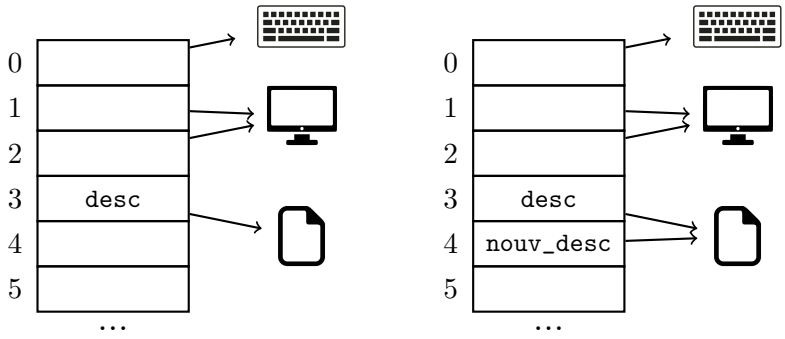
1 Duplication de descripteurs de fichiers

```
int dup(int desc);
```

La primitive `dup` fournit un nouveau descripteur ayant exactement les mêmes caractéristiques que le descripteur `desc` (qui doit nécessairement exister). Exemple d'utilisation :

```
int desc= open(...);
int nouv_desc= dup(desc);
```

Le numéro de descripteur retourné correspond au plus petit indice des entrées libres dans la table des fichiers ouverts. Sinon, la primitive retourne `-1`. La figure 1 illustre ce qui se passe lorsque la primitive `dup` est exécutée.



(a) `int desc= open(...);` (b) `int nouv_desc= dup(desc);`

Figure 1 – Illustration de la primitive `dup`

Question. *Que se passe-t-il si on exécute `close(1)` avant `dup(desc)` ?*

```
int dup2(int desc, int nouv_desc);
```

La primitive `dup2` duplique l'entrée `desc` à l'indice `nouv_desc` de la table des descripteurs ouverts. Si le descripteur `nouv_desc` était ouvert, il est préalablement fermé. La primitive retourne `-1` en cas d'erreur.

La figure 2 illustre l'exécution de la commande `dup2(desc, 1)`. Le descripteur de fichier 1 est d'abord fermé (n'est plus associé à l'écran) puis associé au fichier de descripteur de fichier `desc`. Toute exécution de `write(1,...)` se fera maintenant dans le fichier.

Exercice Écrire un programme qui réalise la commande UNIX `cp` (`cp source destination`) en utilisant la commande `cat`. On souhaite de plus que le ficher destinataire de la copie soit uniquement accessible en lecture pour tous les utilisateurs (groupe et autres).

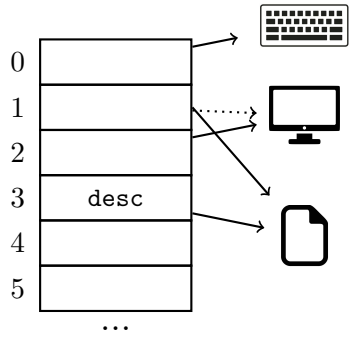


Figure 2 – Illustration de `dup2(desc, 1)`

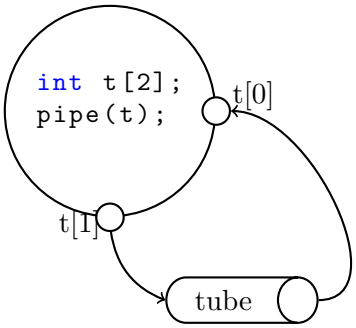
2 Tubes

Les tubes de communication permettent l'échange d'un flot de données entre des processus. Le tube se comporte comme une file de type Premier Arrivé Premier Servis (FIFO) où les données sont introduites à une extrémité et sont extraites à une autre.

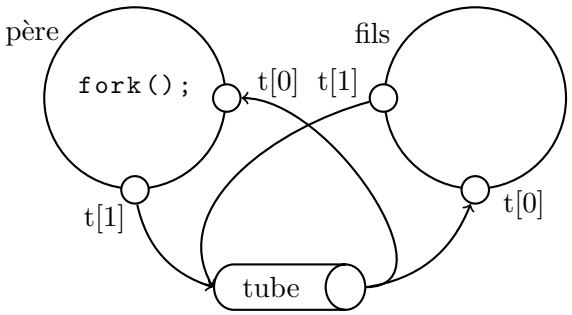
```
int pipe(int tube[2]);
```

La primitive `pipe` permet la création du tube. Le schéma de construction de l'échange d'un flot de données entre 2 processus est donné à la figure 3. Le tube est vu comme un tableau de 2 descripteurs de fichiers :

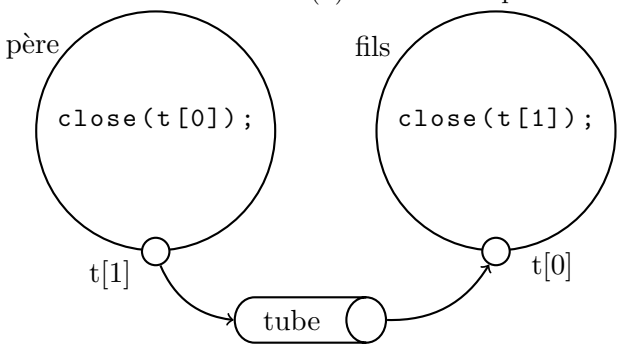
- `tube[0]` est la sortie du tube, on peut y lire des données (`read(tube[0],...);`) ;
- `tube[1]` est l'entrée du tube, on peut y écrire des données (`write(tube[1],...);`).



(a) Création du tube



(b) Création du processus fils



(c) Fermeture des descripteurs inutiles

Figure 3 – Communication entre 2 processus via un tube

La capacité du tube est limitée

— la lecture d'un tube vide est bloquante, si au moins un processus rédacteur est présent (`tube[1]` est ouvert pour au moins un des processus) ;
— l'écriture dans un tube plein est bloquante.

Cas particuliers

— La lecture dans un tube fermé en écriture (plus de rédacteur) renvoie 0. Cela permet de déterminer quand la lecture doit se terminer.
— L'écriture dans un tube fermé en lecture (plus de lecteur) provoque l'envoi du signal `SIGPIPE`.

Exercices

1. Définir deux processus devant respectivement :
   (a) écrire dans un tube les nombres 1 à N *(processus 1, producteur)* ;
   (b) lire les nombres écrits dans ce tube, en faire la somme puis afficher sur la sortie standard le résultat *(processus 2, consommateur)*. Le consommateur ne connait pas la valeur de N.
2. Écrire un programme qui utilise le mécanisme de pipes pour réaliser la commande :
   `who | grep nom_utilisateur`
3. Modifiez le programme précédent pour réaliser :
   `who | grep nom_utilisateur | wc -l`