Exercice 1. Utilisation de write.

En vous aidant de la page de manuel de `write`, rédigez en C la fonction suivante :

```
int write_exact(int fd, char* buf, int n)
```

qui écrit sur le descripteur `fd` exactement `n` octets de la chaîne pointée par `buf`. Prenez garde que l'appel système `write` peut être interrompu avant d'avoir écrit tous les octets qu'on lui a demandé d'écrire : ceci n'est pas un cas d'erreur ; il faut simplement, réappeler `write` avec en argument les octets qui n'ont pas encore été écrits, et recommencer ainsi jusqu'à ce que tous les octets aient été écrits.

La fonction doit retourner 0 si on a réussi à écrire tous les octets, et -1 sinon (c'est à dire si une véritable erreur s'est produite).

Exercice 2. Utilisation de read.

Même chose avec `read`. Rédigez en C la fonction suivante :

```
int read_exact(int fd, char* buf, int n)
```

qui lit sur le descripteur `fd` exactement `n` octets et les place dans le buffer pointée par `buf` (on supposera que le buffer est suffisament grand pour ces `n` octets). Même remarques que précédement.

Exercice 3. Tubes nommés (explications).

Considérons les deux fonctions suivantes :

```
function parle() { echo "hello"; }
function ecoute_et_repete() {
  local buf;
  read buf;
  echo "message: $buf";
}
```

On peut les connecter par un tube de la manière suivante :

```
parle | ecoute_et_repete
```

ce qui affiche :

```
message: hello
```

On peut faire la même chose avec un tube nommé :

```
mkfifo mon-tube
parle > mon-tube &
ecoute_et_repete < mon-tube
rm mon-tube
```

En utilisant un tube normal (anonyme) et un tube nommé on peut facilement connecter le stdout d'une `commande1` au stdin d'une `commande2` et connecter en plus le stdout de `commande2` au stdin de `commande1` pour former une boucle de communication :

```
mkfifo mon-tube
commande1 < mon-tube | commande2 > mon-tube
rm mon-tube
```

Exercice 4. Boucle de communication (locale).

Vous allez écrire deux fonctions bash, `premier` et `second`, pour les connecter dans une boucle de communication comme décrit dans l'exercice précédent.

- `premier`
  - envoie le message `HELLO` sur son stdout
  - lit une réponse sur son stdin, affiche cette réponse sur son stderr, retransmet cette réponse sur son stdout
  - lit à nouveau une réponse et l'affiche sur son stderr (mais ne la retransmet pas)
  - envoie le message `QUIT` sur son stdout et s'arrête.
- `second` fait une boucle infinie qui lit un message sur son stdin et le réécrit sur son stdout mais avec des parenthèses autour sauf si le message est `QUIT` auquel cas la fonction s'arrête.

L'execution de cette boucle de communication entre `premier` et `second` affichera :

```
(HELLO)
((HELLO))
```

Exercice 5. Boucle de communication (internet).

Dans cet exercice, on veut à nouveau créer une boucle de communication entre `premier` et `second`, mais, cette fois ci, on veut que `premier` tourne sur une machine et `second` tourne sur une autre. On se servira de `netcat` pour établir la communication entre machines.

`netcat -l -p PORT` invoque la fonctionalité "listen" (option `-l`) de `netcat` qui attend qu'un client se connecte au port `PORT` sur cette machine. Une fois cette connexion établie, tout ce qui est écrit sur le stdin de `netcat`, celui-ci l'envoie au client, et tout ce que le client lui envoie est écrit sur le stdout de `netcat`.

`netcat HOTE PORT` permet de se connecter à un serveur sur le port `PORT` de la machine `HOTE`. Une fois cette connexion établie, tout ce qui est écrit sur le stdin de `netcat` est envoyé au serveur, et tout ce que le serveur lui envoie est écrit sur le stdout de `netcat`.

On voudra créer une boucle de communication correspondant au diagramme de la Figure 1.

(a) écrivez un script `premier.sh` qui utilise la fonction `premier` et `netcat -l` pour réaliser la boucle du haut du diagramme (la partie "serveur").

(b) écrivez un script `second.sh` qui utilise la fonction `second` et `netcat` pour réaliser la boucle du bas du diagramme (la partie "client").

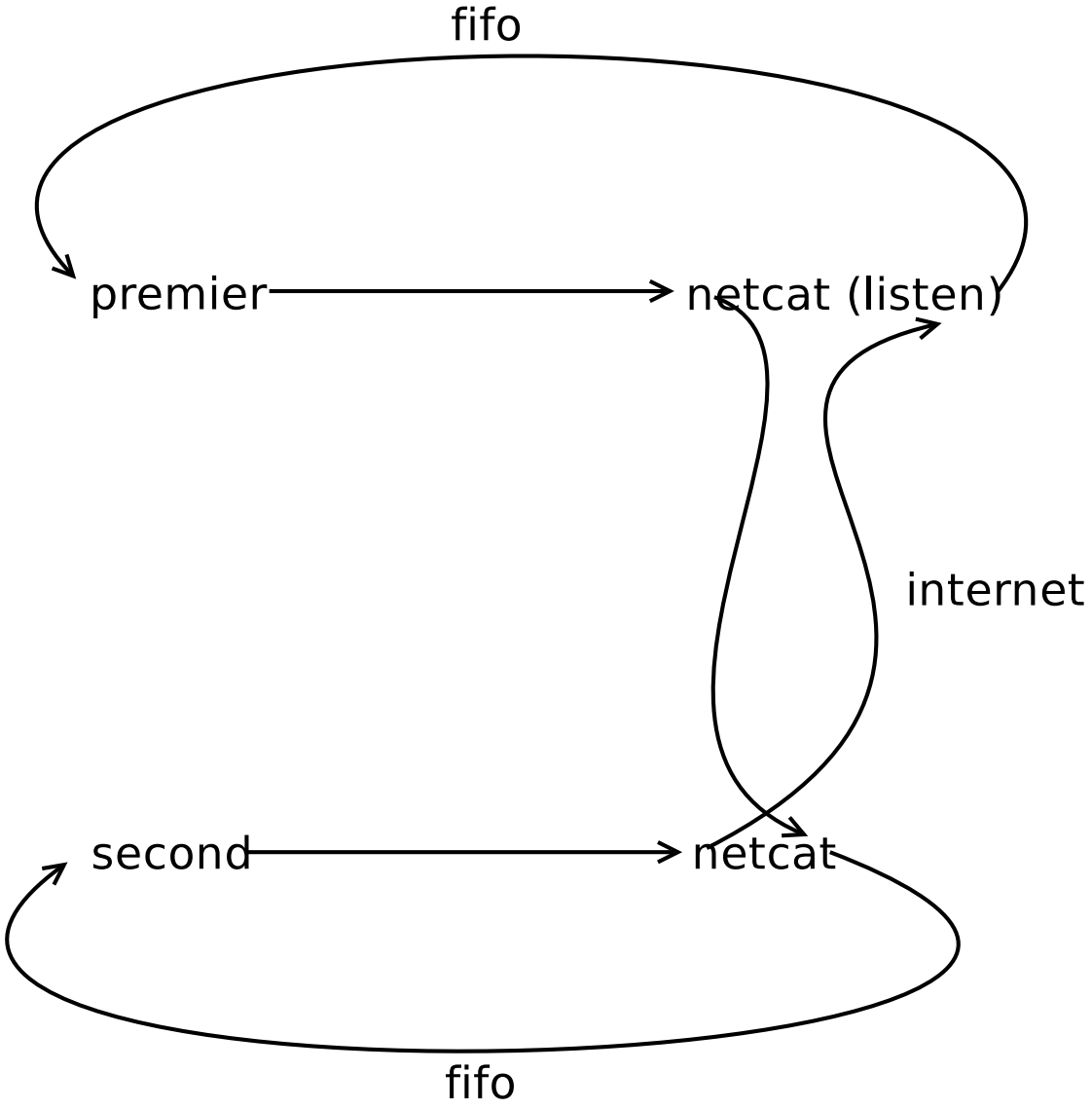

FIG. 1 – Boucle de communication passant par l'internet grâce à `netcat`