Les prises (sockets)

Timothy Bourke Marc Pouzet

DI, École normale supérieure

Cours L3, Système et Réseaux, 2016–2017

Les prises

- Les prises (*sockets*) sont une interface de programmation pour communication entre processus, notamment au travers un réseaux TCP/IP.
- Ces diapositives sont une introduction rapide à leur utilisation en OCaml. N'oubliez pas de faire un `open Unix` avant de commencer vos expériences.
- Le programme `telnet [serveur] [port]` est parfois utile comme aide de déboguage.

- Quelques références:
  - Les pages man unix: `socket(2)`, `bind(2)`, `listen(2)`, `accept(2)`, `connect(2)`, etcetera.
  - La page manuel d'OCaml pour la bibliothèque `Unix`:

    `https://caml.inria.fr/pub/docs/manual-ocaml/libref/Unix.html#VALsocket`
  - *Programmation du système Unix en Objective Caml* par X. Leroy et D. Rémy:

    `http://gallium.inria.fr/~remy/camlunix/cours.html#htoc45`

- Admettons un ordinateur **client**, un réseau et un ordinateur **serveur**.
- Le serveur a des processus qui écoutent, chacun sur un **port**, un entier unique qui sert à identifier le service. Par ex., 22 est utilisé pour le service ssh et 80 pour le service web.
- On veut créer un nouveau service en utilisant la bibliothèque Unix.


```
let s = socket PF_INET SOCK_STREAM 0
```

- On appelle la fonction `socket` en choisissant un domaine de communication (`PF_INET`), un type de communication (`SOCK_STREAM`) et le protocole par défaut (`0`).
- Une **prise** représente un extrémité d'un canal de communication. Les processus les manipulent à travers des descripteurs de fichier.


```
let s = socket PF_INET SOCK_STREAM 0
```

↓

```
bind s addr
```

```
let ip_addr = inet_addr_any
let port = 12345
let addr = ADDR_INET (ip_addr, port)
```

- Sur le serveur, il faut connecter la prise à une adresse et un port.
- On spécifie que l'on accepte toutes les connections qui arrivent au serveur (`inet_addr_any`) sur le port '12345' — un entier arbitraire plus grand que 1024. Il est possible qu'un serveur ait plusieurs adresses et que l'on ne veut accepter des connections que sur un sous-ensemble.


```
let s = socket PF_INET SOCK_STREAM 0
        ↓
bind s addr
        ↓
listen s 20
```

- On appelle `listen` pour écouter les nouvelles connections avec une file d'attente de taille 20.
- Le nouveau service est enregistré par le système d'exploitation. (`lsof -n -i4TCP` ou `netstat` peut afficher l'état des prises ouvertes.)


```
let s = socket PF_INET SOCK_STREAM 0
        ↓
bind s addr
        ↓
listen s 20
        ↓
let s_cl, addr_cl = accept s
```

- L'appel `accept` bloque en attendant qu'un client se connecte au port 12345...


```
let s = socket PF_INET SOCK_STREAM 0
```

```
let s = socket PF_INET SOCK_STREAM 0
        ↓
bind s addr
        ↓
listen s 20
        ↓
let s_cl, addr_cl = accept s
```

- Maintenant, le client crée lui aussi une prise. . .


```
let s = socket PF_INET SOCK_STREAM 0
```

↓

```
connect s addr
```

```
let host = gethostbyname "tabac"
let ip_addr = host.h_addr_list.(0)
let port = 12345
let addr = ADDR_INET (ip_addr, port)
```

```
let s = socket PF_INET SOCK_STREAM 0
```

↓

```
bind s addr
```

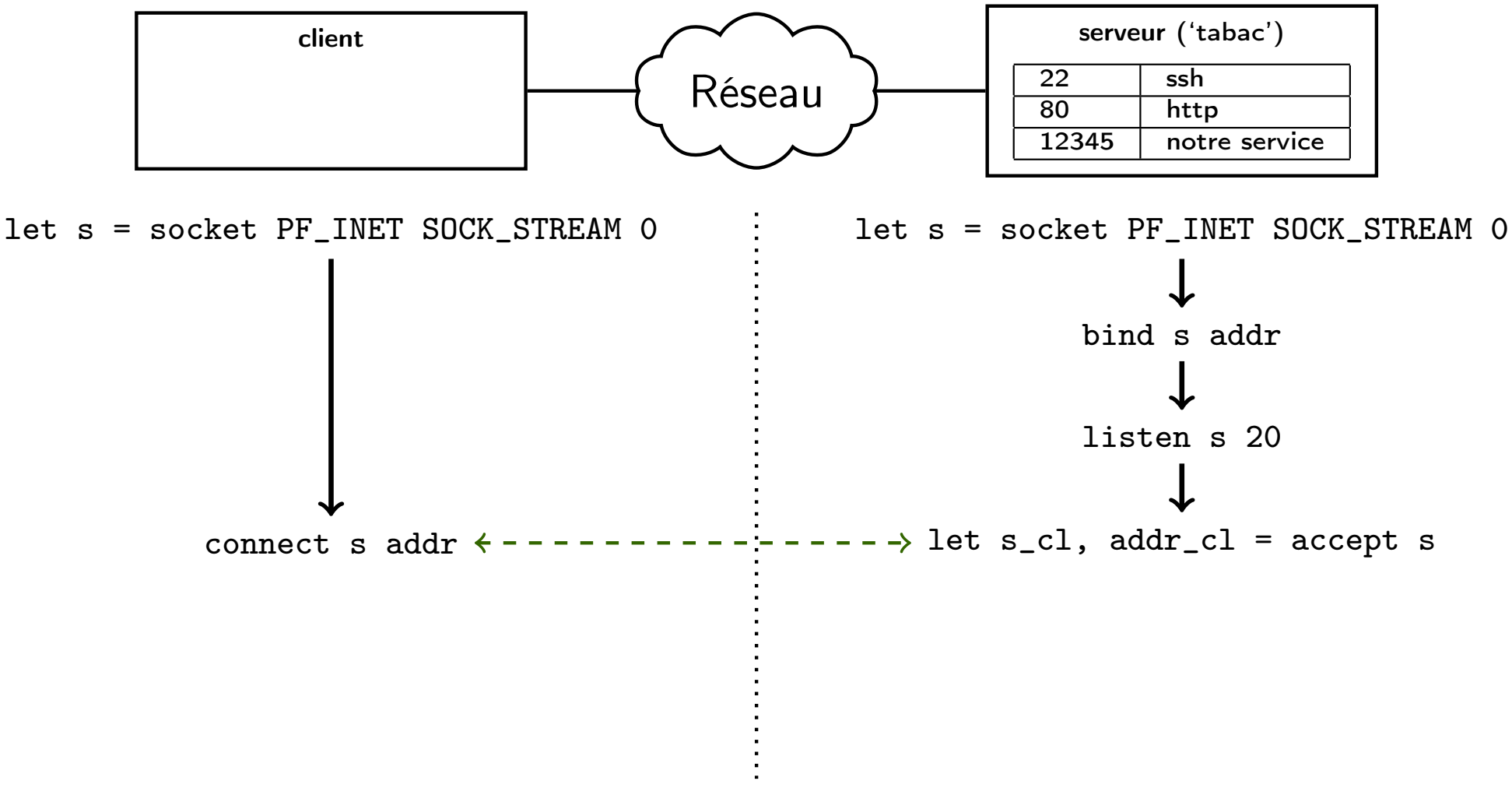
↓

```
listen s 20
```

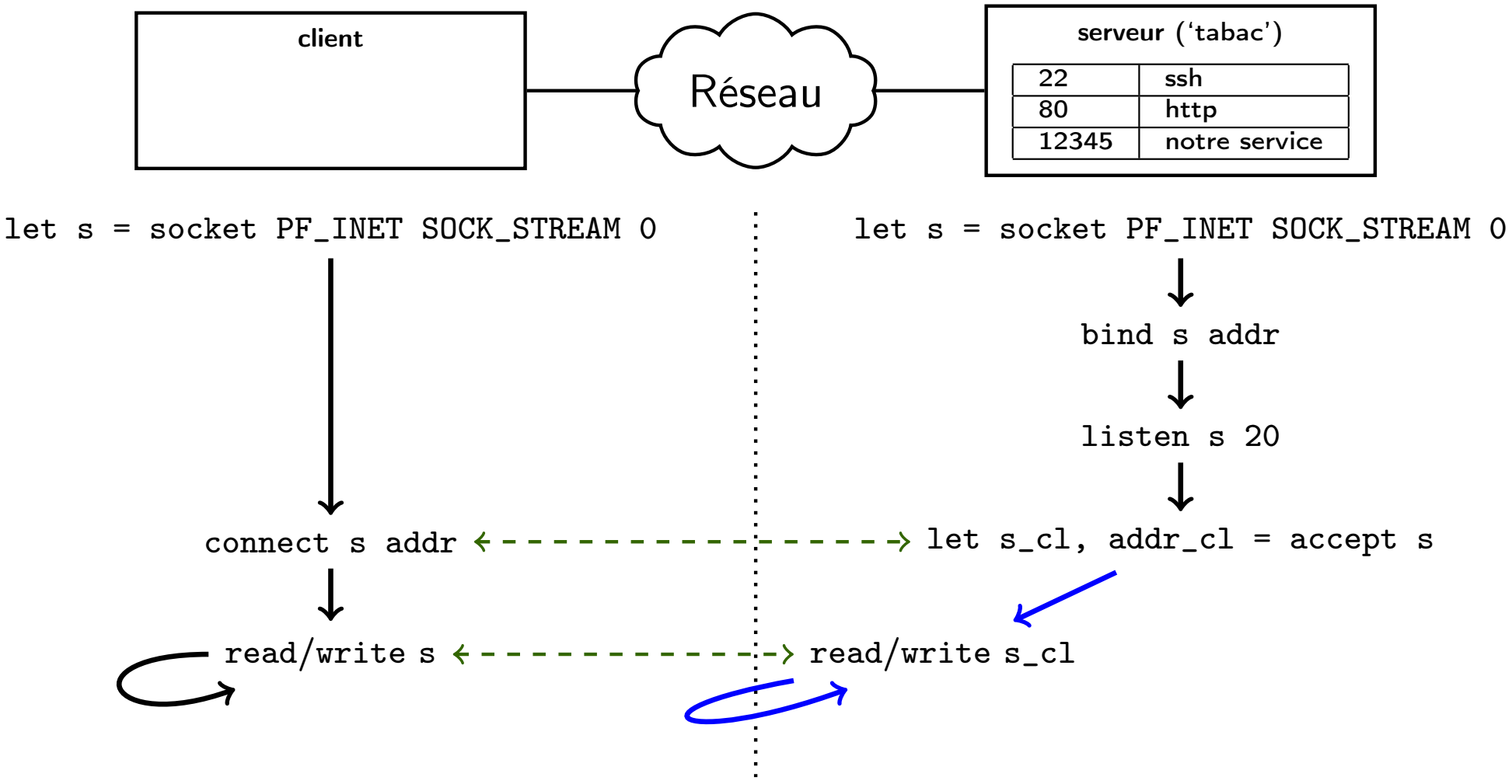
↓

```
let s_cl, addr_cl = accept s
```

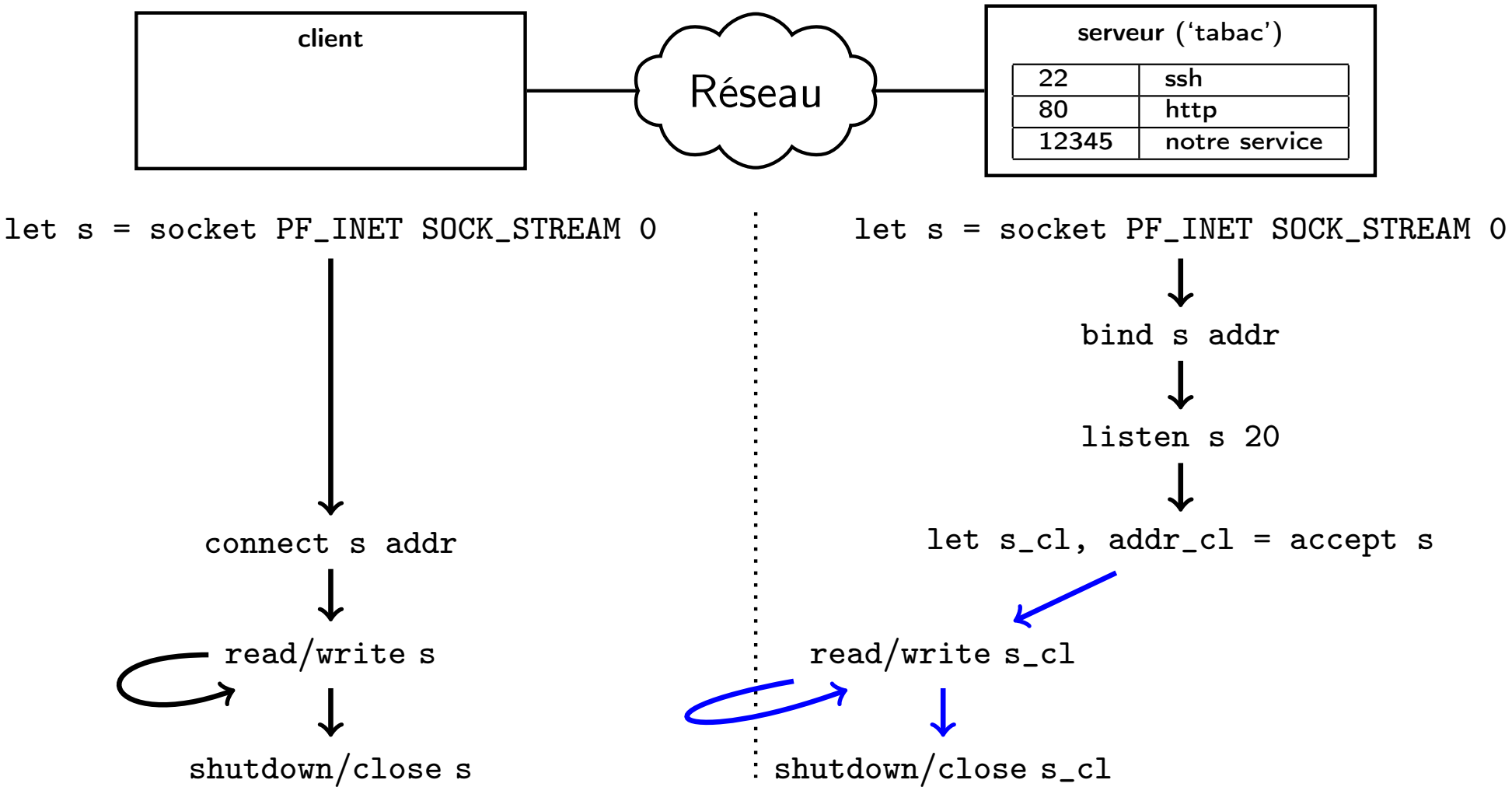
- . . . mais au lieu d'appeler la fonction `bind`, il appelle la fonction `connect` avec l'adresse du serveur et le port du service.
- On peut demander au système d'exploitation de trouver l'adresse d'un ordinateur en passant son nom. On peut aussi passer une adresse directement (par ex., `inet_addr_loopback` pour la machine locale).



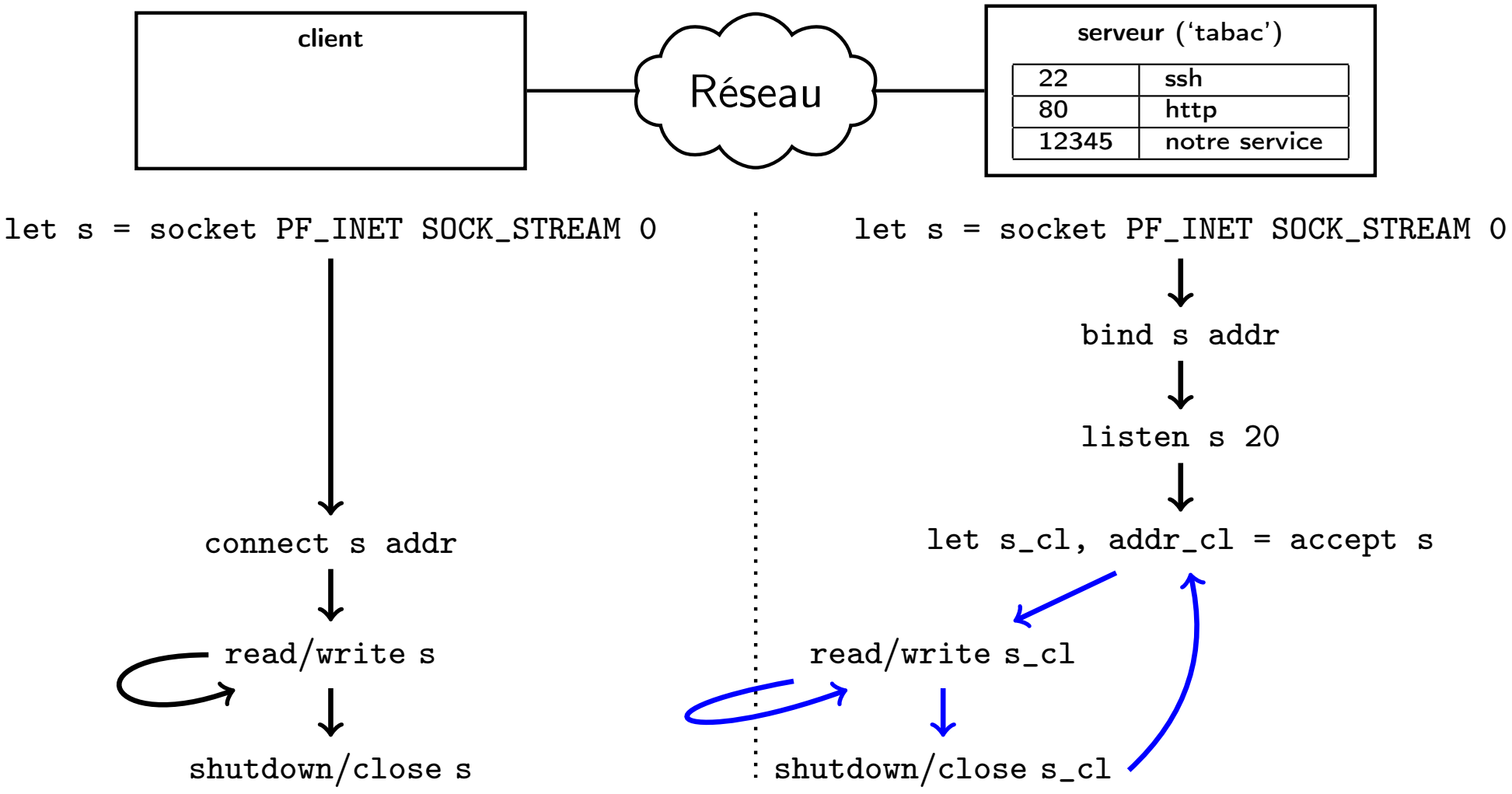
- L'appel `connect` bloque en attendant une connection avec le serveur.
- Au serveur, dès qu'une connection est établie, l'appel `accept` rend une nouvelle prise (`s_cl`) vers le client (représenté par un descripteur de fichier) et l'adresse du client.



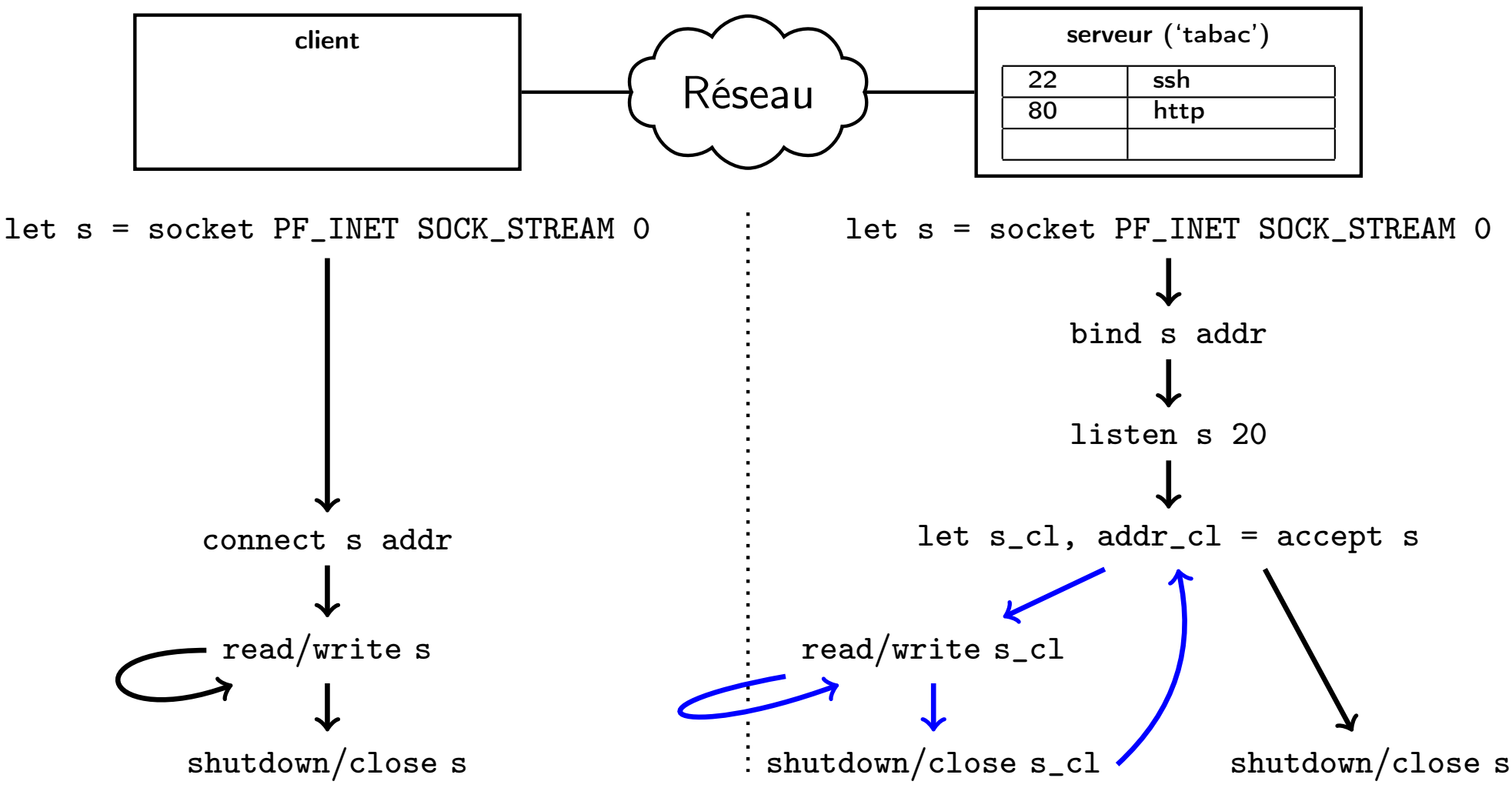
- Le client écrit sur sa prise pour envoyer des octets, le serveur lit sur la nouvelle prise pour les lire, et inversement.
- Les fonctions `out_channel_of_descr` et `in_channel_of_descr` permettent d'utiliser un flot plutôt que les fonctions de bas niveau `read` et `write`.



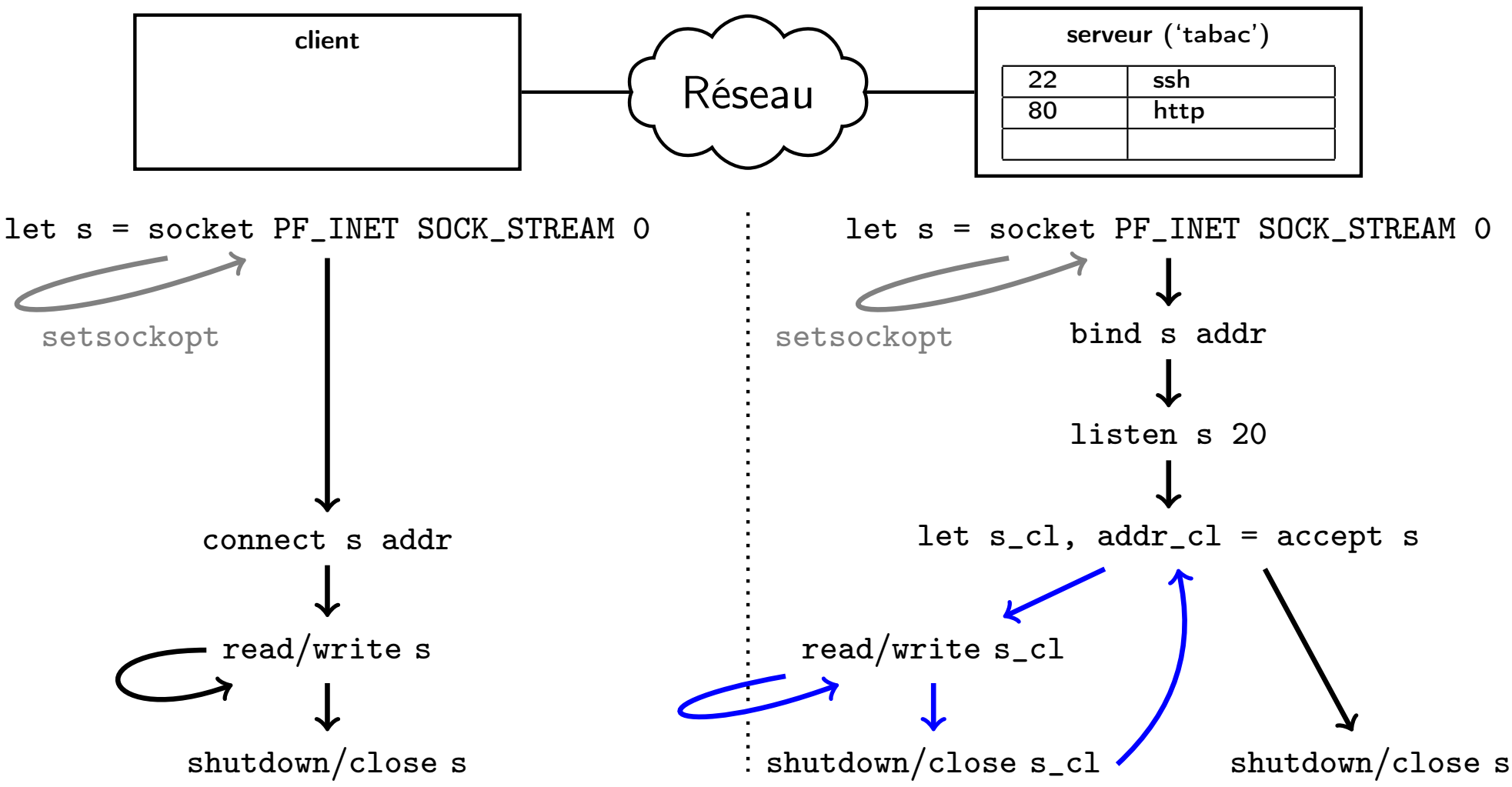
- Le client ou le serveur peut fermer la connection en appelant `close`. L'appel `shutdown` permet de fermer individuellement les deux sens d'une connection.



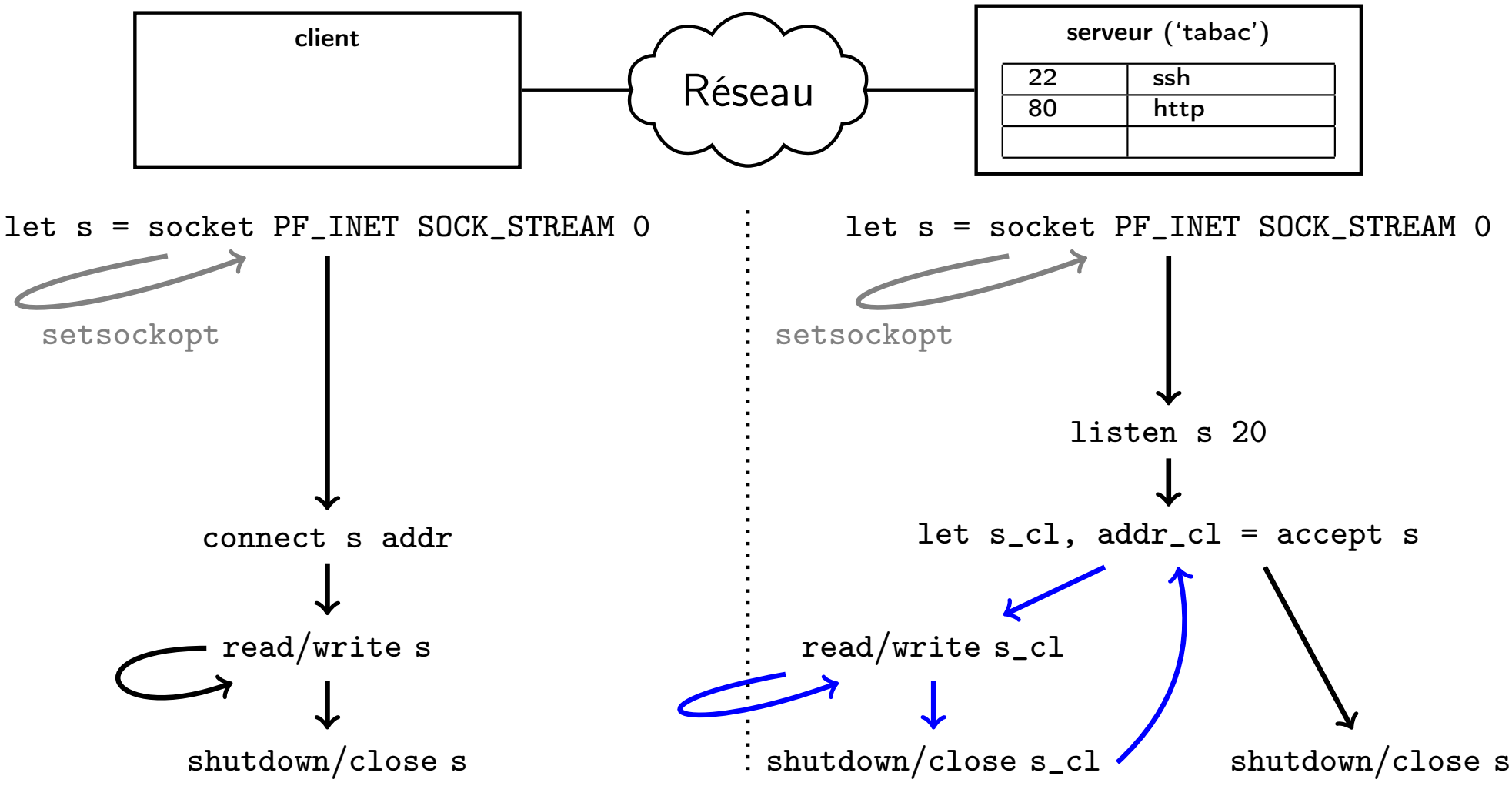
- Le serveur peut boucler sur `accept` en utilisant la prise original (`s`) pour traiter d'autres clients, qui sinon attendent dans la file d'attente.
- Il est fréquent de sous-traiter des connections dans un autre processus, par ex., créé avec `fork`, ou thread.



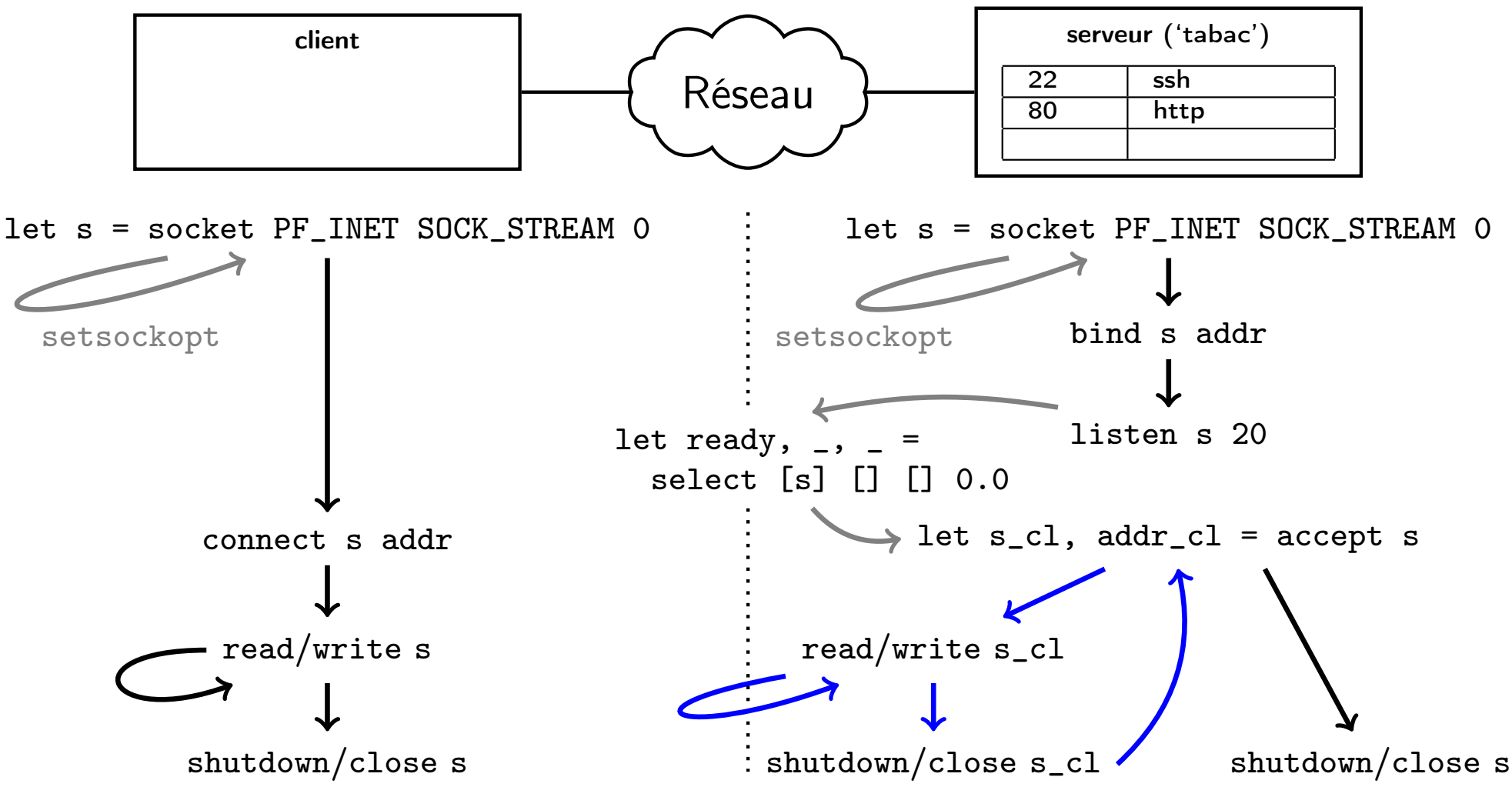
- Le serveur ferme sa prise d'écoute avant de terminer et il n'est alors plus possible pour des clients de se connecter au port.



- Entre la création d'une prise et sa connection à une adresse, il est possible de la configurer avec l'appel `setsockopt`.
- Par ex., `setsockopt s SO_REUSEADDR true` permet de réutiliser la paire d'une adresse et d'un port, notamment pour éviter d'attendre l'expiration d'un délai en déboguant un serveur programme.



- Il est possible d'appeler `listen` sans avoir appelé `bind`. Dans ce cas, le système d'exploitation alloue un port implicitement.



- Entre la création d'une file d'attente (`listen`) et l'acceptation d'une connection (`accept`), il est possible de bloquer avec l'appel `select` qui permet d'attendre sur plusieurs descripteurs de fichiers à la fois.
- En utilisant `select` un programme peut fournir plusieurs services sur plusieurs ports différents.

Code exemple

client.ml

```
open Unix

let s = socket PF_INET SOCK_STREAM 0

let host = gethostbyname "tabac"
let ip_addr = host.h_addr_list.(0)
let port = 12345
let addr = ADDR_INET (ip_addr, port)

let () = connect s addr

let out_ch = out_channel_of_descr s
let () = output_string out_ch "coucou"
let () = flush out_ch

let () = close_out out_ch
```

server.ml

```
open Unix

let s = socket PF_INET SOCK_STREAM 0

let server_name = gethostname ()
let () = print_endline server_name
let ip_addr = inet_addr_any
let port = 12345
let addr = ADDR_INET (ip_addr, port)

let () = bind s addr
let () = listen s 20
let fd_client, addr_client = accept s

let buffer = String.make 140 '\000'
let n = read fd_client buffer 0 140
let () = print_endline (String.sub buffer 0 n)
let () = close fd_client
```