Programmation Réseau

Attente multiple en C

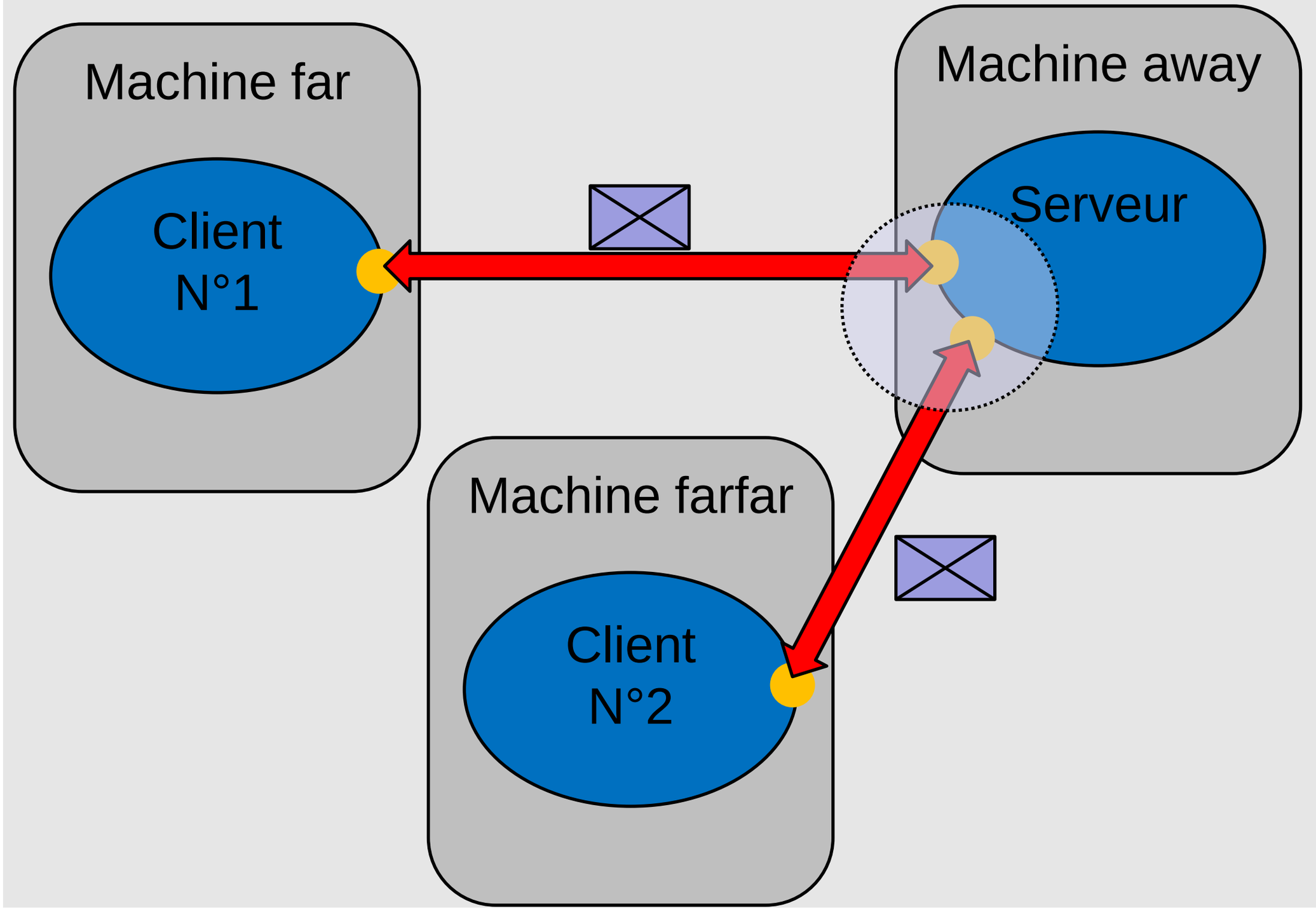
Jean-Baptiste.Yunes@liafa.jussieu.fr

Coloriages: François Armand
armand@informatique.univ-paris-diderot.fr

UFR Informatique

2011-2012

Attente multiple



Attente multiple

- La lecture (attente de données) est bloquante:
  - Le programme ressort de l'appel de lecture:
    - Quand il y a des données
    - Ou une erreur
- Comment faire si un serveur veut s'occuper de plusieurs clients simultanément?
  - A tour de rôle?... Oui mais, si un client n'envoie pas de données, le serveur va rester bloqué!

Solutions possibles?

- fcntl(fd, F_SETFL, O_NONBLOCK)
  - Les opérations bloquantes rendront EAGAIN
- ssize_t read(int fd, void *buf, size_t count);
- Exemple:

```
while(true) {
    if (read(sock1, buf1, lg1) != -1) {
        process(buf1);
    }
    if (read(sock2, buf2, lg2) != -1) {
        process(buf2);
    }
}
```

- Que se passe-t-il si aucun client n’envoie de données?

Select

- Demander au système (Unix ou autre) de réveiller le processus (thread) dès qu'une donnée sera disponible (en lecture) sur un descripteur de fichier parmi N.

- Appel système select

- Fournir la liste des descripteurs de fichiers sur lesquels on attend une donnée.
  - Notion de "file descriptor set"
    - (en fait un "champ de bits")

Select

```
/* According to POSIX.1-2001 */
#include <sys/select.h>

/* According to earlier standards */
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *utimeout);

void FD_CLR (int fd, fd_set *set);
int  FD_ISSET (int fd, fd_set *set);
void FD_SET (int fd, fd_set *set);
void FD_ZERO (fd_set *set);
```

Select

int **select**(int *nfds*, **fd_set** **readfds*, **fd_set** **writefds*,
fd_set **exceptfds*, struct timeval **utimeout*);

- On peut attendre sut tout type de fichiers
  - Sockets, tubes, tubes nommés, tty, autres…
- Ce n’est donc pas un appel spécifique au réseau, mais en pratique il est très utilisé dans les applications réseau
  - Des données (readfds)
  - De la place pour écrire (tube, socket)
  - Des conditions « exceptionnelles »
    - En pratique: données TCP Out-Of-Band (OOB)

Select

- **nfds:**
  - nombre de file descriptor valides
- **readfds:**
  - liste de descripteurs de fichiers sur lesquels on attend des données en réception
- **writefds:**
  - liste de descripteurs de fichiers sur lesquels on attend de pouvoir écrire
- **exceptfds:**
  - liste de descripteurs de fichiers surveillés pour conditions exceptionnelles
- **Utimeout**
  - Temps maximum d'attente (où on reste bloqué dans select – Secondes + µsecondes)
    - NULL => attente infinie
    - 0 => retour immédiat

Select

- A la sortie:
  - Les fdset ont été modifiés!
    - Il faudra les réinitialiser pour un prochain appel
  - Les bits à 1 indiquent quels fichiers correspondent aux attentes
- La valeur de rétour indique le nombre de bits à 1

Structure générale simplifiée.

```
for(;;)  {
      /* initialiser les fdsets */
      FD_CLR(&rdfs);
      F_SET(fd, &rdfs)
      nfds = max (fd, nfds);
      /* Attente */
      res = select( nfds+1 , &rdfs, null, null, null);

      fd = 0;
      /* trouver les fd intéressants */
      for (; res >0; ) {

            if (FD_ISSET(fd, &rdfs)) {

                /* do work: read, accept…*/
                res--;
            }
            fd++;
       }
}
```

Select “règles”

- Réinitialiser les fdsets à chaque itération
- Eviter les “timeout”
  - Si utilisé: le réinitialiser à chaque itération
- nfds doit être bien calculé (efficacité améliorée)
- Ne mettre dans les fdsets que les fd utiles
- Tester tous les “fd” des sets au retour
- Attention
  - Read, write, send, recv
    - Peuvent retourner / envoyer moins de données que demandé
- Eviter les read/write sur des tailles petites (1 octet)
- Eviter les read/write sur des tailles nulles
- Traiter correctement les erreurs
  - EINTR / EAGAIN
- Traiter les EOF (read qui renvoie 0)

Appel poll

```
#include <poll.h>

int poll(struct pollfd *fds, nfds_t nfds, int timeout);

struct pollfd {
    int fd; /* file descriptor */
    short events; /* requested events */
    short revents; /* returned events */
};
```

POLLIN
POLLOUT
POLPRI