Les sockets non-bloquantes (1)

- justification : ne pas bloquer sur une socket
  - tester s'il y a des données sur une socket
  - consulter un descripteur
- fonction *int ioctl(int fd, unsigned long request, int* on)*
  - inclusion : #include <sys/ioctl.h>
  - *request : FIONBIO* (File I/O Non Blocking IO) - rend le descripteur non-bloquant
  - *on* : 0 = bloquant / ≠0 = non-bloquant
  - retourne : valeur précédente
- retour des fonctions bloquantes : code d'erreur *errno*
  - *EWOULDBLOCK* : *accept*, *recv*
  - *EINPROGRESS* : *connect*

Les sockets non-bloquantes (2)

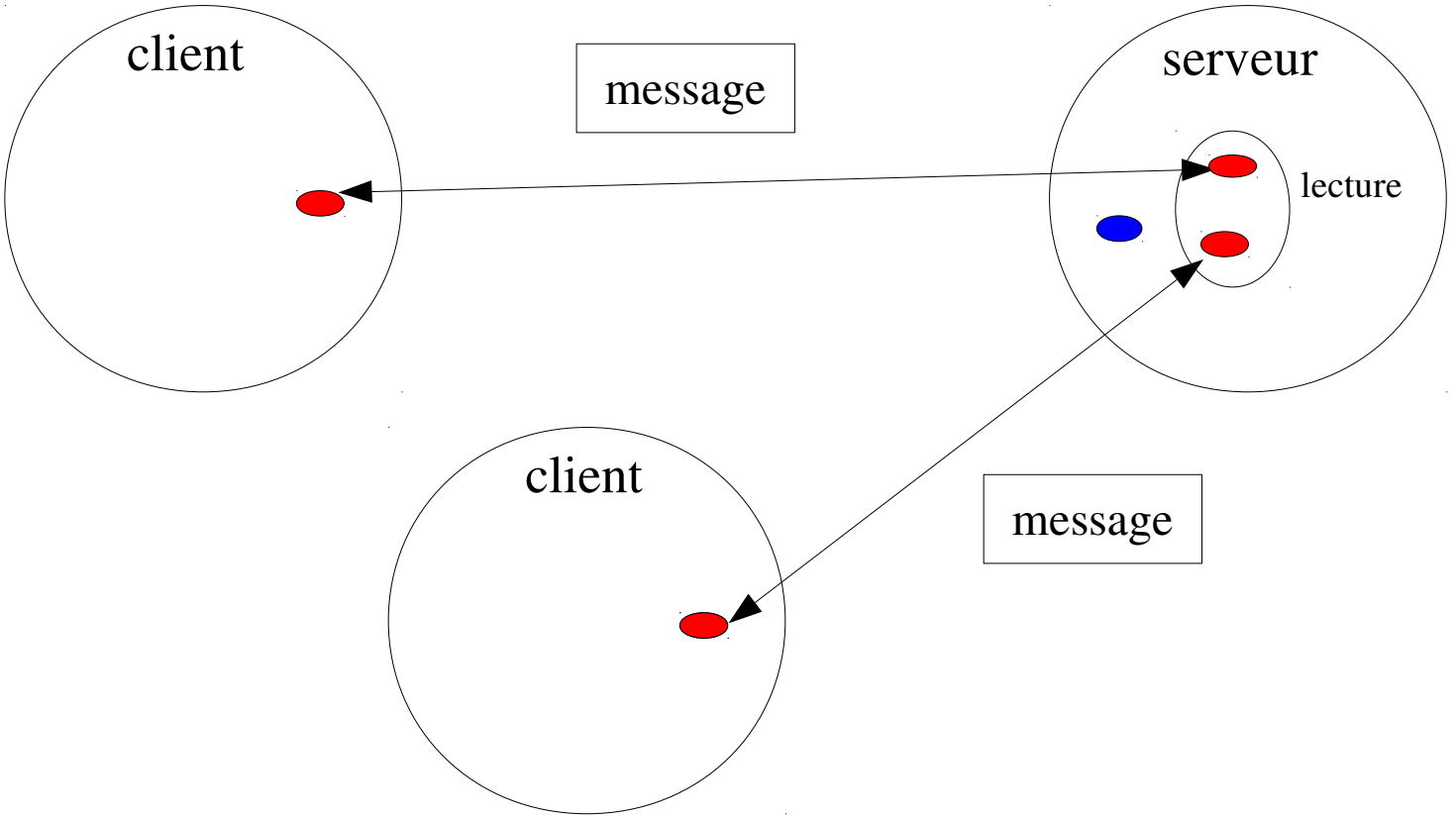
- exemple
- *recv* : 4 cas possibles
  - >0 : reçu
  - =0 : fin connexion
  - -1 + EWOULDBLOCK
    - rien à recevoir
  - -1 + ! EWOULDBLOCK
    - erreur

```
#include <errno.h>
#include <sys/ioctl.h>
…
 // la socket est rendue non bloquante
 on = 1;
 err = ioctl(sockTrans, FIONBIO, &on);
 if (err < 0) {   … // erreur    }
 // attente de la réception
 recu = 0;
 while (recu == 0) {
    // réception et affichage du message
    err = recv(sockTrans, buffer, TAIL_BUF, 0);
    if (err > 0)  recu = 1;
    else
      if (err < 0 && errno == EWOULDBLOCK)
        printf(" Serveur : pas de message sur la socket\n ");
      else {
        perror("serveur : erreur dans la reception");
        shutdown(sockTrans, 2); close(sockTrans);
        exit(7);
      }
}
```

Le multiplexage des appels (1)

- justification
  - attente passive
  - attente multiple
- mode opératoire : fonction *select*
  - constitution de 3 ensembles de descripteurs
    - lecture
    - écriture
    - exception
  - attente sur ces ensembles
  - en sortie, seuls les descripteurs « prêts » restent dans les ensembles

Le multiplexage des appels (2)



Le multiplexage des appels (3)

- macros de constitution des ensembles de descripteurs
  - type des ensembles : *fd_set* (chaque bit représente l'état d'un descripteur)
  - *#include <sys/select.h>*
  - *void FD_SET(int fd, fd_set *set)* - ajout d'un descripteur
  - *void FD_CLR(int fd, fd_set *set)* - retrait d'un descripteur
  - *int FD_ISSET(int fd, fd_set *set)* - test de présence
  - *void FD_ZERO(fd_set *set)* - remise à zéro de l'ensemble
- principe
  - *fd_set, FD_ZERO, FD_SET, select, FD_ISSET*

Le multiplexage des appels (4)

- exemples : utilisation des ensembles

```
main(int argc, char** argv) {
  …
 fd_set  readSet;          // déclaration ensemble
  …
  sockTrans1 = accept(sockCont, NULL, NULL);
  if (sockTrans1 < 0) { // erreur … }
  sockTrans2 = accept(sockCont, NULL, NULL);
  if (sockTrans2 < 0) { // erreur … }
  …
  FD_ZERO(&readSet) ;  // initialisation à zéro
  FD_SET(sockTrans1, &readSet);
  FD_SET(sockTrans2, &readSet);
```

```
  …
  sockTrans3 = accept(sockCont, NULL, NULL);
  if (sockTrans3 < 0) { // erreur … }
  ...
  FD_CLR(sockTrans1,&readSet);
  FD_SET(sockTrans3, &readSet);
  …
  FD_ZERO(&readSet);
  FD_SET(sockTrans1, &readSet);
  FD_SET(sockTrans3, &readSet) ;
  …
  if (FD_ISSET(sockTrans2, &readSet)) {… }
```

met le descripteur
sock_trans1
dans l'ensemble read_set

Le multiplexage des appels (5)

- fonction *select*

*int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, NULL)*

*int select(int nfds, fd_set *readfds, NULL, NULL, NULL)*

valeur maximale parmi les descripteurs

- *nfds* : ~~nombre max de descripteurs~~ + 1
- *readfds, writefds, exceptfds* : ensembles
- *int* : retour
  - nombre de descripteurs prêts
  - 0 si timeout
  - -1 si erreur

Le multiplexage des appels (6)

- fonction *select*, avec timeout
  - *int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)*
  - *timeout* : temps maximal d'attente
- structure de timeout
  - *#include <sys/time.h>*

```
struct timeval {
  long tv_sec;  /* seconds */
  long tv_usec; /* microseconds */
};
```

  - Linux : structure modifiée par l'appel à *select* → la réinitialiser pour pouvoir la réutiliser
  - NULL dans select → attente infinie

*int select(int nfds, fd_set *readfds, NULL, NULL, NULL)*

Le multiplexage des appels (7)

- exemple serveur

```
main(int argc, char** argv) {
 …
 fd_set  readSet;          // ensemble de descripteurs
                           // pour select
 …
 sockTrans1 = accept(sockCont, NULL, NULL);
 if (sockTrans1 < 0) { // erreur … }
 sockTrans2 = accept(sockCont, NULL, NULL);
 if (sockTrans2 < 0) { // erreur … }
 …
 FD_ZERO(&readSet);
 FD_SET(sockTrans1, &readSet);
 FD_SET(sockTrans2, &readSet);

 nfsd = 6;                 → FD_SETSIZE
 err = select(nfsd, &readSet, NULL, NULL, NULL);
 if (err < 0) {
    perror("serveurSelect : erreur dans select");
    // shutdown + close
    exit(5);
 }
```

```
if (FD_ISSET(sockTrans1, &readSet) != 0) {
  /* réception du message sur sockTrans1 */
 err = recv(sockTrans1, buffer, 100, 0);
 if (err < 0) {
      printf("serveur : erreur %d dans le recv1\n", errno);
      // shutdown + close
      exit(6);
 }

 if (FD_ISSET(sockTrans2, &readSet) != 0) {
      /* réception du message sur sock_trans2 */
      err = recv(sockTrans2, buffer, 100, 0);
      if (err < 0) {
        printf("serveur : erreur %d dans le recv2\n", errno);
        // shutdown + close
        exit(7);
      }
 }
```

Le multiplexage des appels (8)

- remarque
  - peut aussi être utilisé avec autres descripteurs
    - terminal : lecture
    - fichiers