Mécanismes Internes du Système d'Exploitation Unix

Communication Inter-Processus

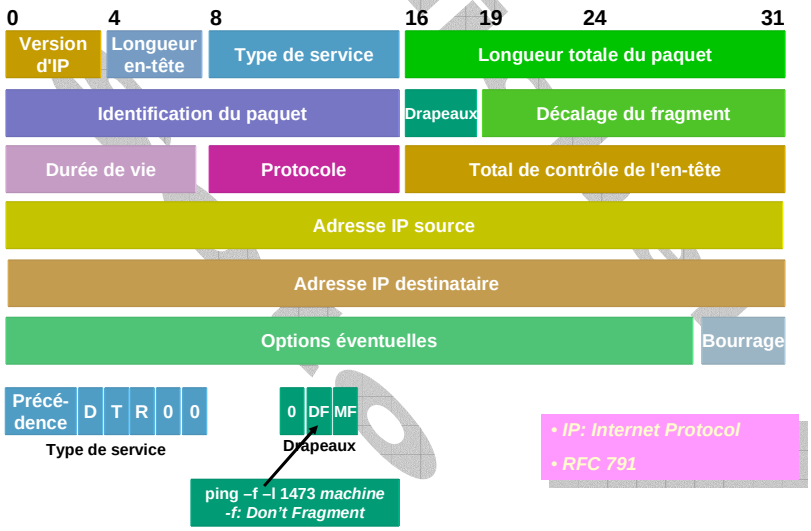
Les sockets

Caractéristiques des sockets

- Structure d'interface équivalent à un canal d'E/S (fichier)
- Type (Qualité de transmission)
  - SOCK_DGRAM - SOCK_RDM
  - SOCK_STREAM - SOCK_SEQPACKET
  - SOCK_RAW
- Domaine (Type d'adressage)
  - AF_UNIX - AF_IPX …
  - AF_INET - AF_APPLETALK …
- Associations possibles
  - DGRAM + AF_UNIX, DGRAM + AF_INET
  - STREAM + AF_UNIX, STREAM + AF_INET
  - RAW + AF_INET

1

Le paquet IPv4 (rappel)



Fichiers à inclure pour toute utilisation des sockets

- sys/types.h

```
…
typedef unsigned short u_short;
typedef unsigned long u_long;
…
```

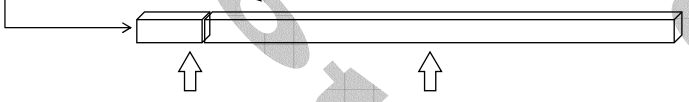
- sys/un.h

```
struct sockaddr_un {
        short sun_family;   /* AF_UNIX */
        char sun_path[108];
};
```

Fichiers à inclure pour toute utilisation des sockets (suite)

- sys/socket.h

```
struct sockaddr {
  u_short sa_family;   /* adress family: AF_xxx value */
  char sa_data [14];   /* up to 14 bytes of protocol-specific address */
};
```



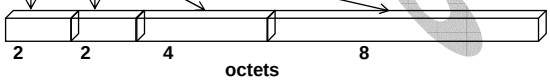
AF_INET
AF_NS
AF_X25
....

Adresse sur 14 octets maximum

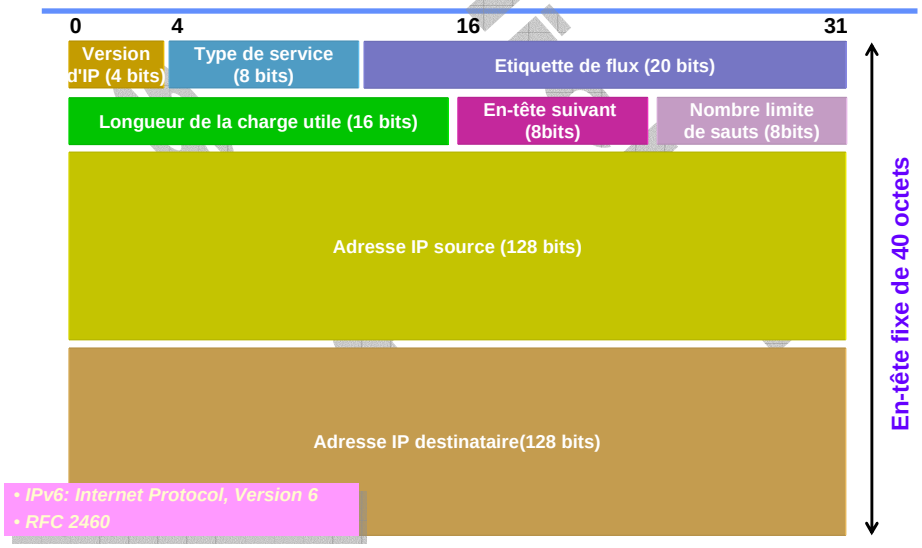
Fichiers à inclure pour toute utilisation des sockets (suite)

- netinet/in.h

```
struct in_addr {
        u_long s_addr;       /* 32-bit netid/hostid */
};

struct sockaddr_in {
        short          sin_family; /* AF_INET */
        u_short        sin_port;   /* 16-bit port number */
        struct in_addr sin_addr;    /* 32-bit netid/hostid */
        char           sin_zero[8]; /* unused */
        };
```



Le paquet IPv6 (rappel)

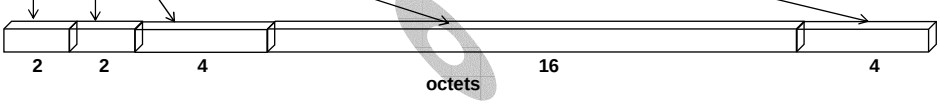


- *IPv6: Internet Protocol, Version 6*
- *RFC 2460*

6

Avec IPv6

```
struct in6_addr {
        uint8_t   s6_addr[16];      /* IPv6 address */
    };

struct sockaddr_in6 {
        sa_family_t    sin6_family;   /* AF_INET6 */
        in_port_t      sin6_port;     /* transport layer port # */
        uint32_t       sin6_flowinfo; /* IPv6 traffic class & flow info */
        struct in6_addr sin6_addr;    /* IPv6 address */
        uint32_t       sin6_scope_id; /* set of interfaces for a scope */
};
```



7

Fichiers à inclure pour toute utilisation des sockets (suite)

- netdb.h

```
struct hostent {
   char    *h_name;       /* Nom officiel de l'hote.   */
   char    **h_aliases;   /* Liste d'alias.            */
   int     h_addrtype;    /* Type d'adresse de l'hote. */
   int     h_length;      /* Longueur de l'adresse.    */
   char    **h_addr_list; /* Liste d'adresses.         */
   };

struct servent {
   char  *s_name;          /* Nom officiel du service */
   char **s_aliases;       /* Liste d'alias           */
   int    s_port;          /* Numero de port          */
   char  *s_proto;         /* Protocole utilise       */
   };
```

8

Fichiers à inclure pour toute utilisation des sockets (suite)

```
struct netent {
   char *n_name;           /* Nom officiel  du reseau */
   char**n_aliases;        /* Liste d'alias           */
   int      n_addrtype; /* Type d'adresse reseau   */
   unsigned long int n_net;/* Adresse du reseau       */
   };

struct protoent {
   char   *p_name;         /* Nom officiel du protocole */
   char  **p_aliases;      /* Liste d'alias             */
   int     p_proto;        /* Numero du protocole       */
   };
```

9

Fichiers d'administration concernés par le réseau et les routines associées

/etc/hosts

```
  10.153.3.1      papin       s0
```

```
struct hostent *gethostent(void)
struct hostent *gethostbyname(const char *name);
struct hostent *gethostbyaddr(const char *addr,
                    int len, int type);
```

/etc/networks

```
  iup-net   10.153.3.0
```

```
struct netent *getnetent(void)
struct netent *getnetbyname(const char *name);
struct netent *getnetbyaddr(long net, int type);
```

10

Fichiers d'administration concernés par le réseau et les routines associées

(suite)

/etc/protocols

```
  ip        0     IP   # Protocole internet
  tcp       6     TCP  # Protocole de contrôle de
  transmission
```

```
struct protoent *getprotoent(void)
struct protoent *getprotobyname(const char *name);
struct protoent *getprotobynumber(int proto);
```

/etc/services

```
  smtp      25/tcp      mail
```

```
struct servent *getservent(void)
struct servent *getservbyname(const char *name,
                  const char *proto);
struct servent *getservbyport(int port, const char
  *proto);
```

11

Autres routines concernant la manipulation des adresses

- `int bcmp (const void *s1, const void *s2, int n);`
- `void bcopy (const void *src, void *dest, int n);`
- `void bzero (void *s, int n);`
- `unsigned long int htonl(unsigned long int hostlong);`
  convertit une valeur de 32 bits de l'ordre de la machine vers l'ordre du réseau.
- `unsigned short int htons(unsigned short int hostshort);`
- `unsigned long int ntohl(unsigned long int netlong);`
  convertit une valeur de 32 bits de l'ordre du réseau vers l'ordre de la machine.
- `unsigned short int ntohs(unsigned short int netshort);`

Exemple : Impression d'un numéro de port sur n'importe quelle machine, quelque soit le codage des données :

```
printf((“Numéro de port %d”, ntohs(sp->s_port);
```

12

Création d'une socket

- La primitive `socket()`

```
#include<sys/types.h>
#include<sys/socket.h>
     int socket(int domain, int type, int protocol);
```

  - retourne un descripteur de fichier,
  - si `protocol` = 0 → le système choisit le protocole
    sinon,

```
struct protoent *pp;
…
pp = getprotobyname(“tcp”);
s = socket(AF_INET, SOCK_STREAM, pp->p_proto);
```

  - à ce niveau, aucun processus d'une autre filiation ne peut atteindre la socket, il faut lui donner un nom.

13

Nommage d'une socket

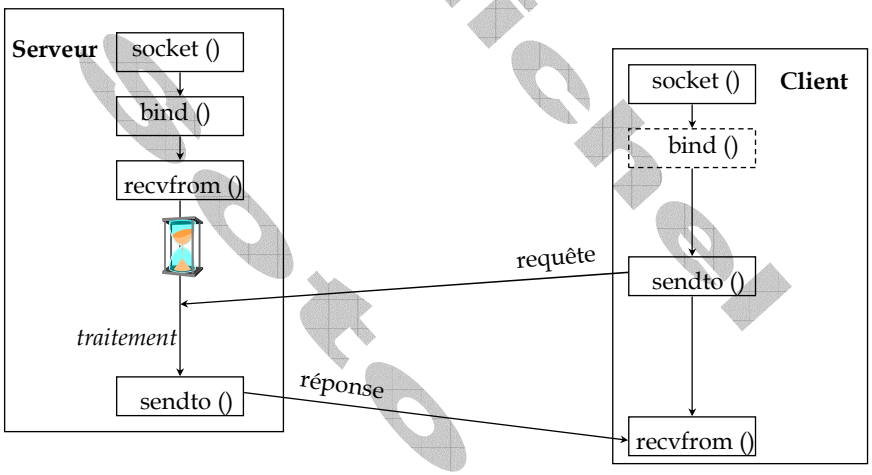
- La primitive `bind()`

```
#include<sys/types.h>
#include<sys/socket.h>
     int bind(int sockfd, struct sockaddr *my_addr,
                    int addrlen);
```

  - un nom est un pointeur sur une structure de type `sockaddr_un` ou `sockaddr_in`,
  - nom dans le domaine Unix → pathname,
    nom dans le domaine Internet → n° de port + @IP = TSAP,
  - le nommage de la socket pour l'expéditeur (mode symétrique) ou pour le client (mode asymétrique) n'est pas obligatoire,
  - retourne 0 en cas de réussite, -1 sinon,

14

Communication en mode non connecté "Canevas"



15

Communication en mode non connecté

- Type SOCK_DGRAM
- Mode non connecté, symétrique
- Emission de données

```
#include <sys/types.h
#include <sys/socket.h>
int sendto(int s, const void *msg, int len, unsigned int flags,
     const struct sockaddr *to, int tolen);
```

  – retourne le nombre de caractères émis, -1 en cas d'échec

- Réception de données

```
     int recvfrom(int s, void *buf, int len,
            unsigned int flags, struct sockaddr *from, int
*fromlen);
```

  – retourne le nombre de caractères reçus, -1 en cas d'échec

16

Exemple de communication en mode non connecté dans le domaine AF_UNIX

```
/* Processus Emetteur */

#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <string.h>

int main()
{
   int i, sd;
   char msg[10];

   struct sockaddr serveur;

   serveur.sa_family = AF_UNIX;
   strcpy(serveur.sa_data, "serv_sock");

   sd = socket(AF_UNIX, SOCK_DGRAM, 0);

   for(;;)
   {
      printf("Entrer message \n");
      scanf("%s", msg);

      sendto(sd, msg, sizeof(msg), 0,
        &serveur, sizeof(serveur));
   }
}
```

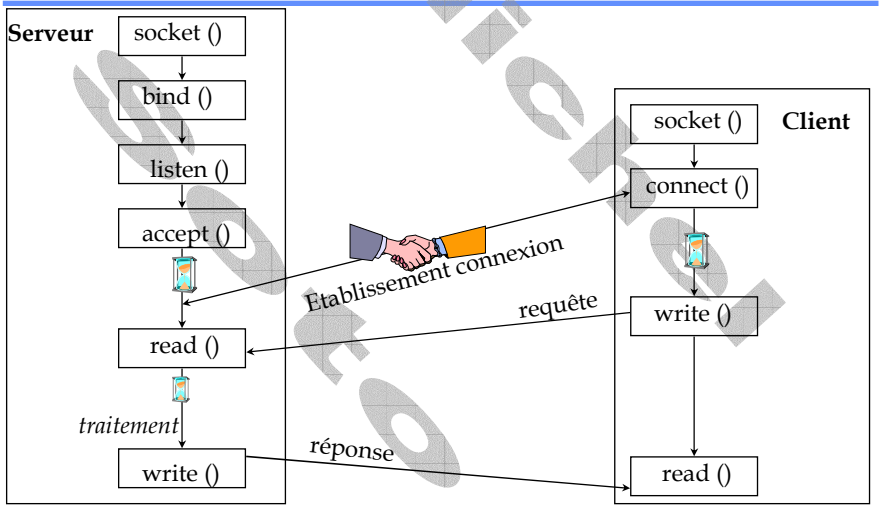
17

Exemple de communication en mode non connecté dans le domaine AF_UNIX (suite)

```
/* Processus Recepteur */

void interrupt(int signo)
{
  unlink("serv_sock");
  exit(0);
}

int main()
{
  int i, sd;
  char msg[10];
  struct sockaddr serveur, client;

  int fromlen = sizeof(client);
  signal(SIGINT, interrupt);

  serveur.sa_family = AF_UNIX;
  strcpy(serveur.sa_data, "serv_sock");

sd = socket(AF_UNIX, SOCK_DGRAM, 0);

bind(sd, &serveur, sizeof(serveur));

for(;;)
  {
   recvfrom(sd, msg, sizeof(msg), 0,
            &client, &fromlen);
   printf("Msg Recu = %s\n", msg);
  }
}
```

18

Communication en mode connecté "Canevas"



19

Communication en mode connecté

- Type `SOCK_STREAM`
- Mode client/serveur, asymétrique

- Etablissement de la connexion (côté serveur)

```
#include <sys/socket.h>
     int listen(int s, int backlog);
```

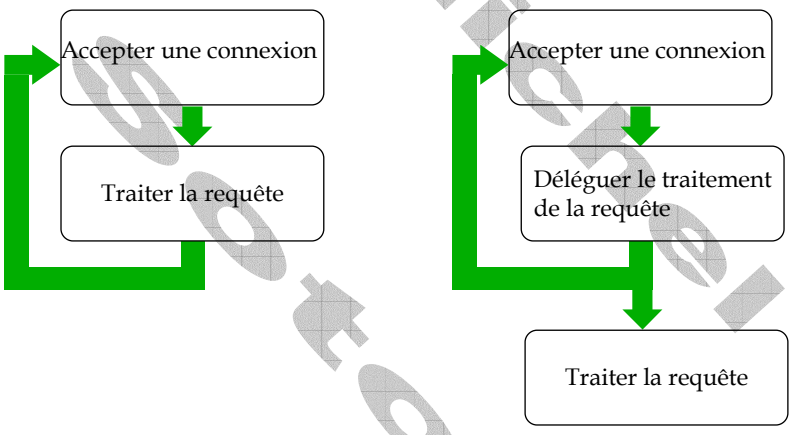
  - serveur en attente passive de connexion,
  - `backlog` correspond à la taille de la file d'attente,

```
int accept(int sock, struct sockaddr *adresse, int *longueur);
```

  - retourne le descripteur d'une **nouvelle socket**,
  - `adresse` est un paramètre résultat renseigné avec l'adresse de l'entité se connectant.

20

Serveur: séquentiel vs concurrent



21

Communication en mode connecté (suite)

- Etablissement de la connexion (côté client)

```
#include <sys/types.h>
#include <sys/socket.h>
     int connect(int sockfd, struct sockaddr *serv_addr,int
addrlen);
```

  - `sockfd` : socket locale,
  - `serv_addr` : pointe vers l'adresse de la socket distante,
  - retourne 0 en cas de réussite, -1 sinon.

22

Communication entre le serveur et le client

- Emission de données

```
#include <sys/types.h>
#include <sys/socket.h>
     int send(int s, const void *msg, int len, unsigned int flags);
```

  - retourne le nombre de caractères émis, -1 en cas d'échec

- Réception de données

```
#include <sys/types.h>
#include <sys/socket.h>
     int recv(int s, void *buf, int len, unsigned int flags);
```

  - retourne le nombre de caractères reçus, -1 en cas d'échec

23

Fin de communication

- Terminer tout ou une partie de connexion

```
#include <sys/socket.h>
    int shutdown(int s, int how);
```

  - how permet de définir la fin de connexion
    - `0` → l'utilisateur ne veut plus lire de données,
    - `1` → l'utilisateur ne veut plus écrire de données,
    - `2` → l'utilisateur ne veut plus ni lire ni écrire de données.

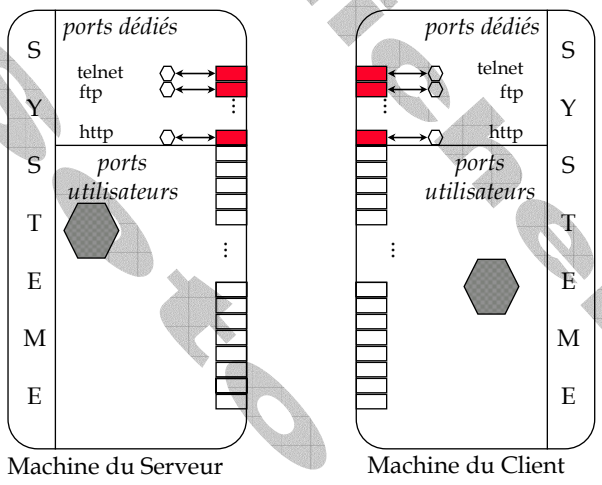
- Ferméture de descripteur de socket

```
#include <unistd.h>
    int close(int s);
```

  - libère le descripteur de fichier de la `u_ofile` du processus,
  - dans le domaine Unix, il faut supprimer le fichier créé.

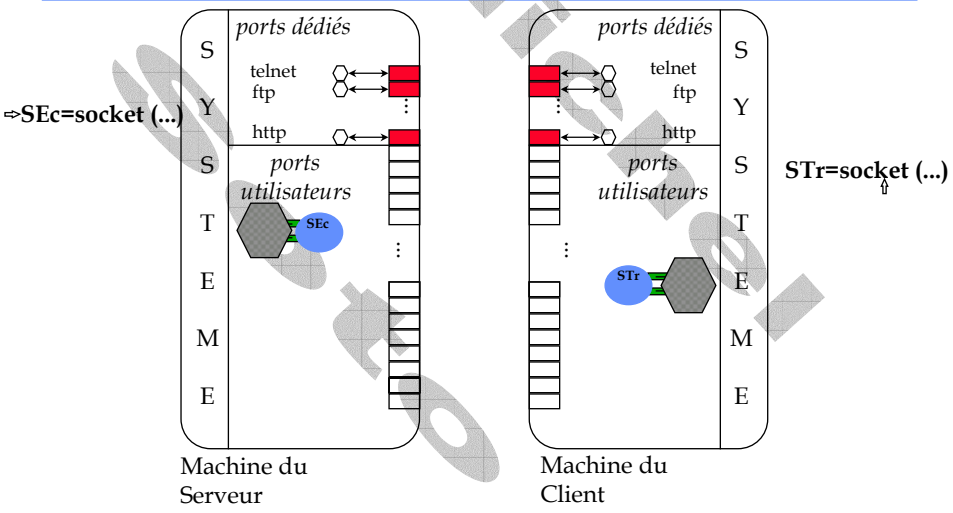
24

Les sockets vues du système

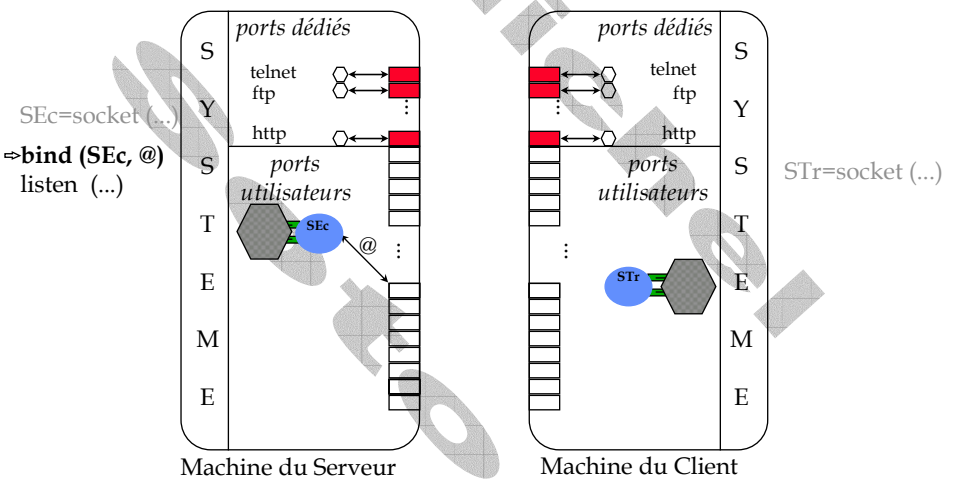


25

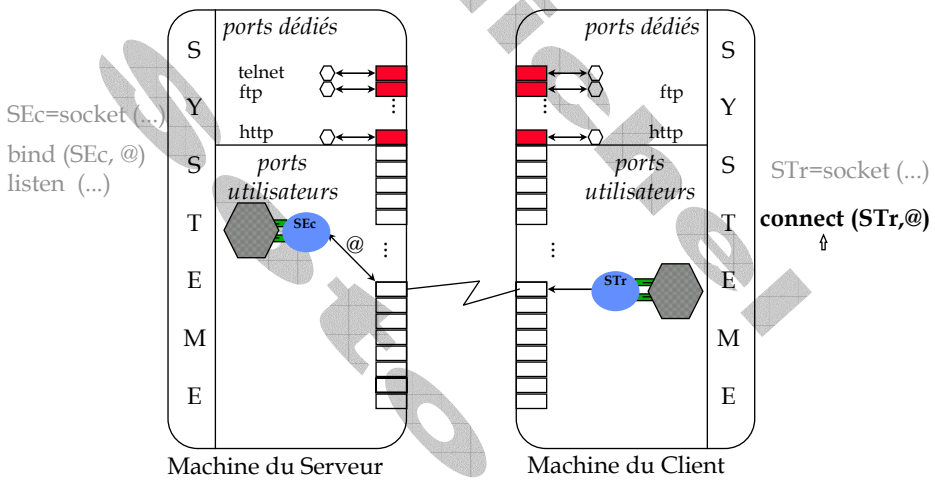
Les sockets vues du système (suite)



Les sockets vues du système (suite)

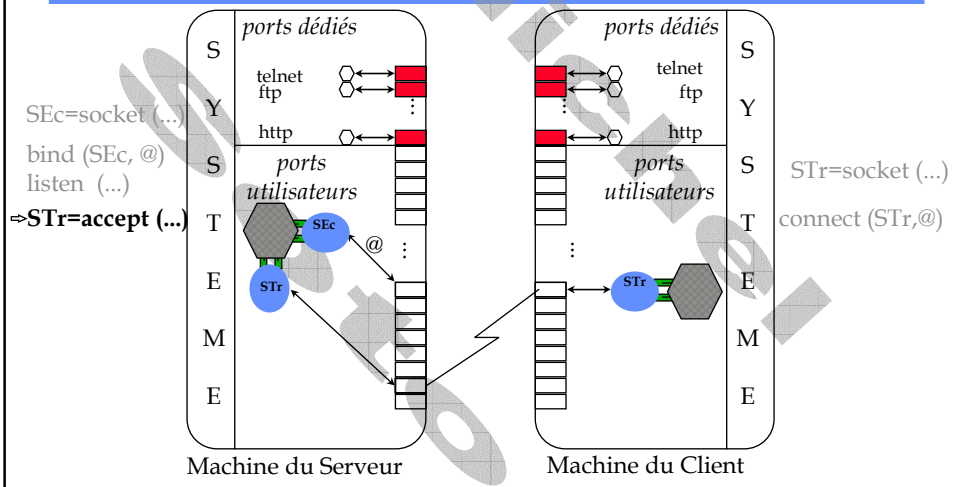


Les sockets vues du système (suite)



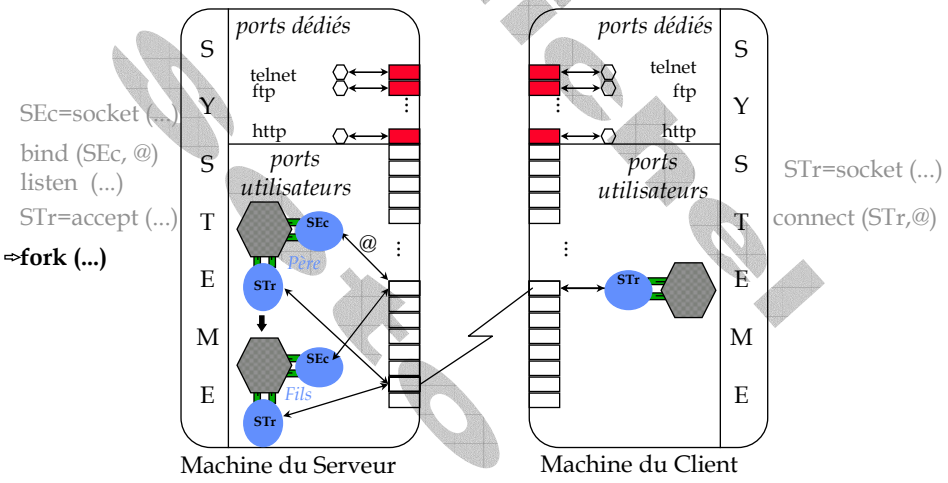
28

Les sockets vues du système (suite)



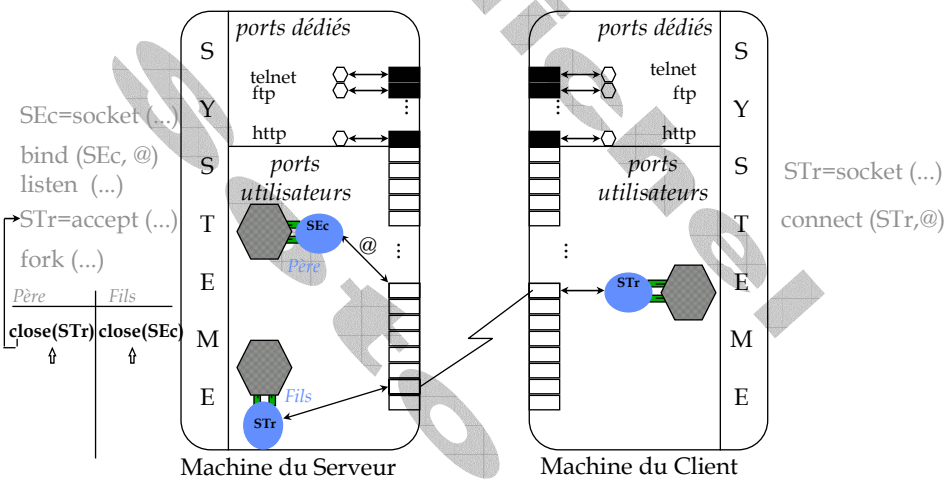
29

Les sockets vues du système (suite)



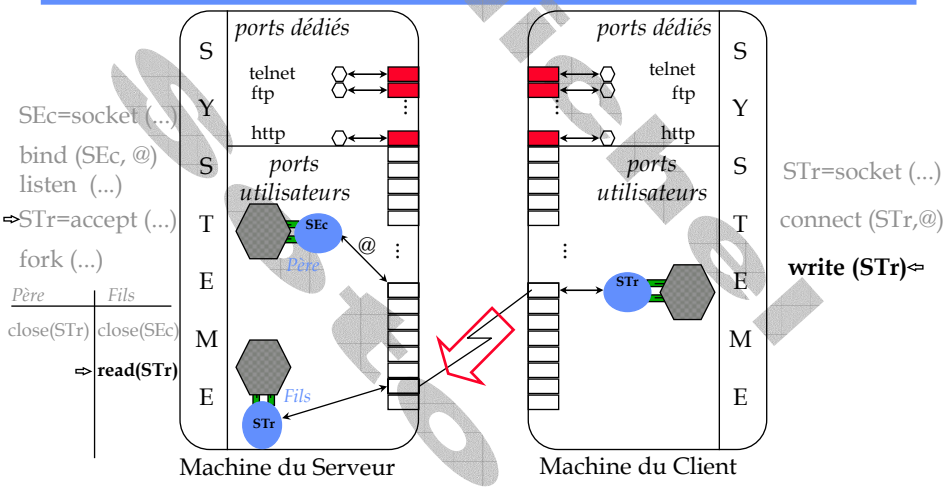
30

Les sockets vues du système (suite)



31

Les sockets vues du système (fin)



Exemple de communication en mode non connecté dans le domaine AF_INET

```
/* Processus Serveur */
int main()
{
int i, sd; char msg[10], name[64];
struct sockaddr_in serv, cli;
struct hostent *hp;
int fromlen = sizeof(cli);

gethostname(name, 64) != 0 );
hp = gethostbyname(name);

bzero( (char *)&serv, sizeof(serv) );
bcopy(hp->h_addr,(char*)&serv.sin_addr,
       hp->h_length);
serv.sin_family = hp->h_addrtype;
serv.sin_port = 2000;

sd = socket(AF_INET, SOCK_DGRAM, 0);
```

```
bind(sd, (struct sockaddr *)&serv,
     sizeof(serv));

for(;;)
{
   i = recvfrom(sd, msg, sizeof(msg),
            0, (struct sockaddr *)
            &cli, &fromlen);

   msg[i] = '\0';
   printf("Msg Recu = %d\n", i, msg);
}
}
```

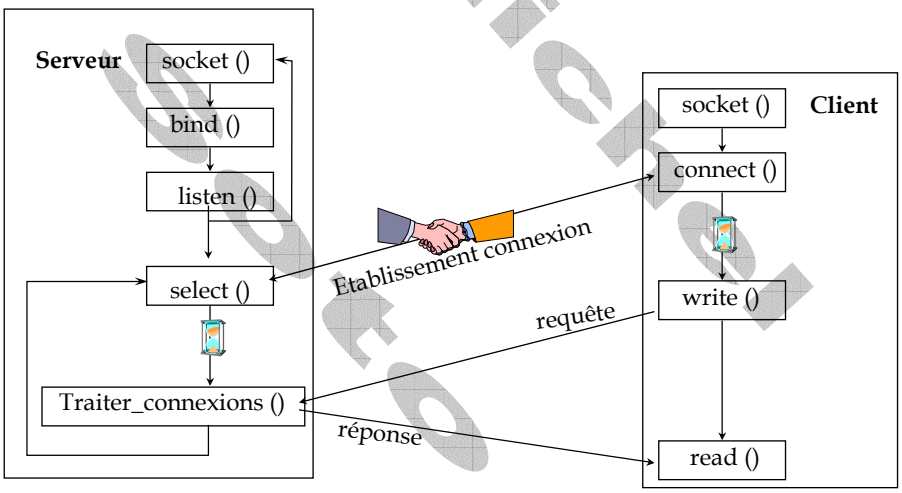
Exemple de communication en mode non connecté dans le domaine AF_INET (suite)

```
/* Processus Client */
int main(int argc, char **argv)
{
int sd;
char msg[20];
struct sockaddr_in serv;
struct hostent *hp;

printf("argv[1] = %s\n", argv[1]);
hp = gethostbyname(argv[1]);

bzero( (char *)&serv, sizeof(serv) );
bcopy(hp->h_addr,(char*)&serv.sin_addr,
       hp->h_length);
serv.sin_family = hp->h_addrtype;;
serv.sin_port = 2000;

sd = socket(AF_INET, SOCK_DGRAM, 0);
```

```
for(;;)
{
   printf("Entrer message : ");
   scanf("%s", msg);

   sendto(sd, msg, strlen(msg), 0,
     (struct sockaddr *)&serv,
     sizeof(serv));
}
}
```


34

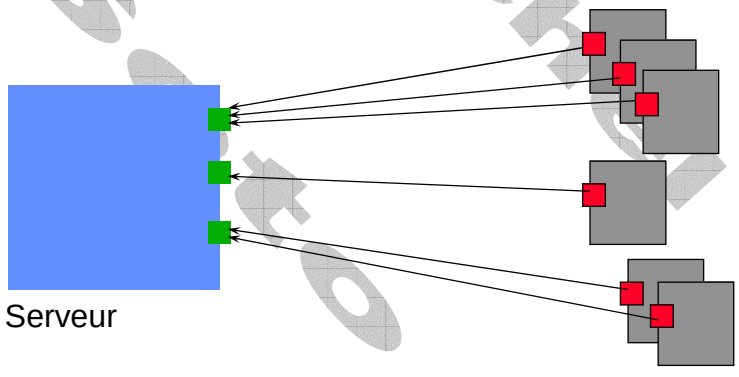
Serveur multi guichets



35

Serveur multi guichets

- Plusieurs sockets d'écoute sont utilisées par le serveur
  - Une socket correspond à un service
  - Un client se connecte sur la socket d'écoute qui correspond au service qu'il souhaite obtenir du serveur



Multiplexage des Entrées/Sorties

- La primitive `select()`

```
#include<sys/time.h>
#include<sys/types.h>
#include<unistd.h>
int select(int n, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

- Manipulation des ensembles de descripteurs

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

Serveur multi guichets: mise en oeuvre

```
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/time.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SOCKET_ECOUTE 2 /* nombre de socket d ecoute du serveur */
#define MAX_CONNEXION 2     /* nombre max de connexion simultanees */
#define MAX_LIGNE 80
#define MAX_NOM 40
#define VRAI 1
#define FAUX 0

struct sockaddr_in InfoSockClient[MAX_CONNEXION+1]; /* tableau des adresse+n°port des clients */
struct sockaddr_in InfoSockEcoute;

int SockEcoute[MAX_SOCKET_ECOUTE+1]; /* +1 pour toujours avoir un tableau... */
int SockService[MAX_CONNEXION+1];    /* +1 pour accepter une connexion de trop et la fermer */
char NomMachineServeur[MAX_NOM],
     ligne [MAX_LIGNE];
int PortServeur,
    LgInfoSockClient[MAX_CONNEXION+1],
    max_sd;

fd_set SetSocketIn; /* ensemble des descripteurs de sockets d ecoute ET de service */

struct hostent *InfoMachineServeur;
struct linger op_linger; /* pour fermer brutalement les connexions en trop */
```



```
/*------------------------------------------------------------------------*/
  void maz_sock_service (int *TabSocket)
/*------------------------------------------------------------------------*/
{int s;

 for (s=0; s<MAX_CONNEXION; s++)
      TabSocket[s]=0;

}/* maz_sock_service */
/*------------------------------------------------------------------------*/
  void print_connexion_active (int *TabSocket)
/*------------------------------------------------------------------------*/
{int s,NbActive;

 for (s=0, NbActive=0; s<MAX_CONNEXION; s++)
      if (TabSocket[s]>0) NbActive++;

 printf("     -> %d CONNEXION(S) ACTIVE(S)\n",NbActive);

}/* print_connexion_active */

/*------------------------------------------------------------------------*/
  void init_set_socket_ecoute (int *TabSocket, fd_set *SetSocket)
/*------------------------------------------------------------------------*/
/* Ajoute les descripteurs de socket d ecoute dans l'ensemble des       */
/* descripteurs surveilles par le select                                */
/*------------------------------------------------------------------------*/
{int i, max_socket;

 max_socket=MAX_SOCKET_ECOUTE;

 for (i=0; i<max_socket; i++)
      FD_SET(TabSocket[i], SetSocket);

 printf ("\n     -> %d SOCKET(S) D ECOUTE AJOUTEE(S) DANS L ENSEMBLE\n", i);
}/* init_set_socket_ecoute */
```

```
   /*-------------------------------------------------------------------------*/
     void init_set_socket_service (int *TabSocket, fd_set *SetSocket)
   /*-------------------------------------------------------------------------*/
   /* Ajoute les descripteurs de socket de service dans l'ensemble des     */
   /* descripteurs surveilles par le select                                */
   /*-------------------------------------------------------------------------*/
   {int i, s, max_socket;

    max_socket=MAX_CONNEXION;
    for (i=0, s=0; i<max_socket; i++)
         if (TabSocket[i]>0){
           FD_SET(TabSocket[i], SetSocket);
           s++;
         }
    printf ("     -> %d SOCKET(S) DE SERVICE AJOUTEE(S) DANS L ENSEMBLE\n", s);

   }/* init_set_socket_service */

   /*-------------------------------------------------------------------------*/
     void etat_connexion (int *SockEcoute)
   /*-------------------------------------------------------------------------*/
   /* Affiche les socket d ecoute qui font l objet d une connexion         */
   /*-------------------------------------------------------------------------*/
   {int e;

    for (e=0; e<MAX_SOCKET_ECOUTE; e++){

       if (FD_ISSET(SockEcoute[e], &SetSocketIn))
          printf ("\n     -> ARRIVEE D UNE CONNEXION, SOCKET ECOUTE e%d\n", e);
    }
   }/* etat_connexion */
```

40

```
/*-------------------------------------------------------------------------*/
  void accepter_connexion (int *SockEcoute)
/*-------------------------------------------------------------------------*/
/* Effectue un accept sur chaque socket d ecoute faisant l objet d une  */
/* connexion                                                             */
/*-------------------------------------------------------------------------*/
{int s, e, trouve;
  for (e=0; e<MAX_SOCKET_ECOUTE; e++){
    if (FD_ISSET(SockEcoute[e], &SetSocketIn)){/* il y une connexion entrante */
       for (s=0, trouve=FAUX; s<=MAX_CONNEXION&&trouve==FAUX; s++) {/* recheche d'une entree libre
                                                                    /* dans le tableau des sockets
                                                                    /* de service

         if (SockService[s]==0){ /* l' entree du tableau et libre */
             trouve=VRAI;

             LgInfoSockClient[s]=sizeof(InfoSockClient[s]);                  */

             /* Accept de la connexion entrante */
             SockService[s]=accept(SockEcoute[e], (struct sockaddr *)&InfoSockClient[s],
                                                                          &LgInfoSockClient[s
             if (SockService[s]<0){
                 perror("\nERREUR accept");
                 exit(1);
             }
             if (s<MAX_CONNEXION)
                 printf ("\n     -> J'ACCEPTE UNE CONNEXION AVEC LA SOCKET DE SERVICE s%d\n", s);
             else { /* Le nombre max de connexion a ete atteint */
                   printf ("\n     -> MAXIMUM DE CONNEXION ATTEINT: REFUS DE LA CONNEXION\n");
                   if (close(SockService[s]<0))perror ("\nERREUR close");

             } /* if (s<MAX_CONNEXION) */
         } /* if (SockService[s]==0) */
       } /* for (s=0; s<MAX_CONNEXION; s++) */
    } /* if (FD_ISSET(SockEcoute[e], ... */
 } /* for (e=0; e<MAX_SOCKET_ECOUTE; e++) */
} /* accepter_connexion */
```

```
-------------------------------------------------------------------------*/
void traiter_connexion (int *SockService)
-------------------------------------------------------------------------*/
 Effectue un read sur chaque socket de service recevant des donnees   */
-------------------------------------------------------------------------*/
nt NbSelect, nr, s;
or (s=0; s<MAX_CONNEXION; s++){
  if (FD_ISSET(SockService[s], &SetSocketIn)){/* il y a des données à recevoir */
     /*-----------------------------------------------------------*/
     /*  LECTURE DES DONNEES DU CLIENT SUR LA SOCKET DE SERVICE  */
     /*  ET TRAITEMENT DE LA REQUETE                              */
     /*-----------------------------------------------------------*/
       nr=read (SockService[s], ligne, MAX_LIGNE);
       if (nr<0) perror("\nERREUR read");
       else {if (nr==0){ /* le client a fermé sa socket: on ferme la notre ! */
                /*-------------------------------------------------*/
                /*  FERMETURE DE LA SOCKET DE SERVICE               */
                /*-------------------------------------------------*/
                  printf ("\n  -> FIN DE LA CONNEXION EN COURS SUR LA SOCKET DE SERVICE s%d\n", s);
                  printf ("    .... FERMETURE DE LA SOCKET DE SERVICE s%d\n", s);
                  close (SockService[s]);
                  SockService[s]=0; /* l entree s est a nouveau libre */
            }
            else if (nr<(MAX_LIGNE-1)){ /* reception de donnees */
                   printf("\n     -> JE RECOIS DES DONNEES SUR LA CONNEXION AVEC: %s / %d \n",
                          inet_ntoa (InfoSockClient[s].sin_addr),
                          ntohs(InfoSockClient[s].sin_port));
                   printf("     -> JE LIS CES DONNEES SUR LA SOCKET DE SERVICE s%d\n",s);
                   ligne[nr+1]='\0';
                 } else ligne[MAX_LIGNE-1]='\0';
            if (ligne[0]!='.' && nr>0 ){printf ("     -> %d caractere(s) reçu(s):\n",nr);
                                printf ("     -> %s\n", ligne);
            }
       }/* if (nr<0) */
  } /* if (FD_ISSET(SockService[s], &SetSocketService)) */
/* for (s=0; s<MAX_CONNEXION; s++) */
* traiter_connexion */
```

```
/*-------------------------------------------------------------------------*/
    void fermer_socket (int *SockEcoute, int *SockService)
/*-------------------------------------------------------------------------*/
/* ferme toutes les sockets d ecoute et de service                         */
/*-------------------------------------------------------------------------*/
{int s;
  for (s=0; s<MAX_SOCKET_ECOUTE; s++){
    printf ("     ->FERMETURE DE LA SOCKET D ECOUTE e%d\n",s);
    close (SockEcoute[s]);
 }
 for (s=0; s<MAX_CONNEXION; s++){
    if (SockService[s]>0){
        printf ("     ->FERMETURE DE LA SOCKET DE SERVICE s%d\n",s);
        close (SockService[s]);
    }
 }
}/* fermer_socket */

/*=========================================================================*/
  main(int argc, char *argv[]){
/*=========================================================================*/
int BindOK, ListenOK, NbSelect;
int i,e;

if (argc <2){
    printf("\n\n  Usage: %s Numero_port\n\n", argv[0]);
    exit(1);}
/*=========================================================================*/
/*   RECHERCHE DE L ADRESSE DE LA MACHINE DU SERVEUR                       */
/*=========================================================================*/
    printf("\n  => RECHERCHE DE L ADRESSE DE LA MACHINE DU SERVEUR");

    gethostname(NomMachineServeur, MAX_NOM);

    InfoMachineServeur=gethostbyname (NomMachineServeur);
    if (InfoMachineServeur==NULL){
       perror("\nERREUR machine serveur");
       exit(1);}

     printf("\n  => DEMARRAGE DU SERVEUR....\n");
```

```
    /*=========================================================================*/
    /*  CREATION DES SOCKETS D ECOUTE                                          */
    /*=========================================================================*/
        for (e=0; e<MAX_SOCKET_ECOUTE; e++){
           printf("\n     -> CREATION DE LA SOCKET D ECOUTE              e%d\n", e);
           SockEcoute[e]=socket(AF_INET, SOCK_STREAM, 0);
           if (SockEcoute[e]<0){
              perror("\nERREUR creation socket");
              exit(1);
           }
    /*=========================================================================*/
    /*   NOMMAGE DE LA SOCKET D ECOUTE                                         */
    /*=========================================================================*/
           printf("     -> NOMMAGE DE LA SOCKET D ECOUTE               e%d\n", e);

           BindOK=bind (SockEcoute[e], (struct sockaddr *)&InfoSockEcoute,
                                                 sizeof(InfoSockEcoute));
           if (BindOK<0) {
              perror("\nERREUR bind serveur socket");
              close(SockEcoute[e]);
              exit(1);
           }
    /*=========================================================================*/
    /*   DIMENSIONNEMENT LA FILE D'ATTENTE DE LA SOCKET D ECOUTE               */
    /*=========================================================================*/
           printf("     -> DIMENSIONNEMENT DE LA FILE D ATTENTE, SOCKET e%d\n",e);

           ListenOK=listen (SockEcoute[e], MAX_CONNEXION);
           if (ListenOK<0){
              perror("\nERREUR listen socket");
              exit(1);
           }
        } /* for (e=0; e<MAX_SOCKET_ECOUTE; e++) */
```

44

```
/*=========================================================================*/
/*   ECOUTE                                                                */
/*=========================================================================*/
   maz_sock_service(SockService);

   while (ligne[0]!='#') {
         for (i=0; i<MAX_LIGNE; i++)ligne[i]=' ';

          FD_ZERO(&SetSocketIn);
          init_set_socket_ecoute (SockEcoute,&SetSocketIn);
          init_set_socket_service (SockService,&SetSocketIn);

         /*=======================================================================*/
         /*   ATTENTE DE CONNEXION SUR LES SOCKETS                                */
         /*=======================================================================*/

          print_connexion_active (SockService);
          printf("     -> ATTENTE .....\n");
          printf("\n=================================================================\n");

          NbSelect = select (FD_SETSIZE, &SetSocketIn,NULL, NULL, NULL);
          printf("     -> il se passe quelque chose....\n");
          if (NbSelect<0){perror("\nERREUR select"); exit(1); }
          printf("\n     -> le select a detecte %d evenement(s)\n", NbSelect);

          etat_connexion (SockEcoute);
          accepter_connexion (SockEcoute);
          traiter_connexion (SockService);

    }/* while (ligne[0]!='#') */

    printf("\n=============================================================\n");
    fermer_socket (SockEcoute, SockService);

    printf ("\n\n  =>FIN DU SERVICE\n\n");
} /* main */
```

45