Socket (C/Unix)

Pr Bouabid EL OUAHIDI

Faculté des Sciences Rabat

Université Mohammed-V Agdal

Email: **ouahidi@fsr.ac.ma**

Introduction à la communication par le mécanisme des sockets en C/Unix. Des exemples de programmes types sont à votre disposition sur simple requête par email.

Avertissement certains appels system ne sont plus d'actualité.

Strctures et fonctions à connaître

Les adresse de machines

struct in_addr
{ u_long s_addr; }

```
struct in_addr {
 union { struct u_char s1_b1, s2_b2, s3_b3, s4_b4;} S_un_b;
         struct u_char s1_w1, s2_w2, s3_w3, s4_w4;} S_un_w;
         u_long S_addr;
       } S_un;
#define s_addr S_un.S_addr;
```

Fonctions de conversions

#include<netinet/in.h>

u_short htons (u_short) passage de la forme locale à la forme réseau d ’un entier court

u_short ntohs (u_short) passage de la forme réseau à la forme locale d ’un entier court

u_long ntohl (u_long) passage de la forme réseau à la forme locale d ’un entier long

u_long htonl (u_long) passage de la forme locale à la forme réseau d ’un entier long

struct hostent

Cette structure est utilisée dans des programmes de sockets.

```
#include<netdb.h>
struct hostent
{
        char * h_name; /* nom officiel de la machine*/
        char **h_aliases;/* liste d 'alias */
         int h_addrtype ;/* type d 'adresse (PF_INET )*/
        int h_length; /* la taille d 'adresse en octets (4) */
        char ** h_addr_list; /* liste d 'adresses */
    };

# define h_addr  h_addr_list[0] /* la première adresse */
```

Ces champs seront fournies au retour d 'un appel à l 'une des fonctions:

gethostname(), gethostby name(), gethostbyaddr() ou gethostent().

gethostname()

```
#include <unistd.h>
int gethostname (char *x, size_t lg);
```

Permet de recevoir dans la variable x le nom de la machine locale. lg représente la taille de l ’espace alloué à x. Retourne 0 ou -1 suivant que l ’appel a réussi ou pas.

```
#include <unistd.h>
long gethostid()
```

! Ne pas utiliser !!

Ces deux fonctions sont utilisables uniquement en local.

gethostbyname()

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
```

```
struct hostent * gethostbyname(char *x);
```

Renvoie un pointeur sur une structure hostent en zone statique contenant les informations relatives à la machine de nom x donné en paramètre. Retourne NULL si x ne correspond pas à une machine identifiable.

Le pointeur renvoyé est en zone statique et chaque appel écrase donc le résultat du précédent: ==> copier avant de faire un nouvel appel.

```
#include<string.h>
int memcpy(void * destination, void *origine, size_t  lg);
int memmove ((void * destination, void *origine, size_t  lg);
```

gethostbyaddr()

```
#include<sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

struct hostent * gethostbyadd( char * p_adr; int lg , int type);
```

Renvoie en zone statique un pointeur sur la structure hostent contenant les informations relatives à la machine d ’adresse p_adr de longueur lg. Retourne NULL si p_adr ne correspond pas à une @IP ou lg et type ne sont pas correctes.

Le dernier paramètre vaut PF_INET. (lg =sizeof(long)).

gethostent()

(+ d ’infos chapitre DNS)

Deux manières de résolutions de nom de machine en adresse IP:
Statique: grâce au fichier /etc/hosts

Dynamique: grâce au BIND (processus serveur de noms).

```
#include <sys/types.h>
#include <sys/socket.h>
#include<netdb.h>
```

struct hostent * gethostent(); lit la ligne suivante de /etc/hosts
int sethostent (int indicateur);
int endhostent ();

struct netent

```
#include <netdb.h>

struct netent {
        char * name ; /* nom officiel du réseau*/

        char **n_aliases; /* liste aliases */

        int n_addrtype ; /* type d 'adresses (AF_INET) */

        unsigned long n_net;

        };
```

getnetbyaddr()

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

struct netent * getnetbyaddr ( u_long x, int type ; /* AF_INET */)
```

Renvoie un pointeur en zone statique sur la structure netent du réseau d ’adresse x. NULL si x n ’est pas une adresse valide.

Voir aussi :

getnetent() endnetent(), setnetent().

Permettent de manipuler le fichier /etc/network.

struct servent

```
#include<netdb.h>
struct servent
{
char * s_name; /* nom officiel de service */

char **s_aliases; /* listes alias */

char s_port; /* numéro de port */

char *s_proto; /* protocole de transport utilisé*/
}
```

getservbyname()

struct servent * getservbyname(char *x, char *p)

Permet de récupérer un pointeur en zone statique sur le service de x utilisant le protocole p.

struct servent * servbyport(int numport, char *p)

Permet de récupérer un pointeur en zone statique sur le service de utilisant le port numport et le protocole p.

Voir aussi:
getservent(), setservent(), endservent qui manipule /etc/services

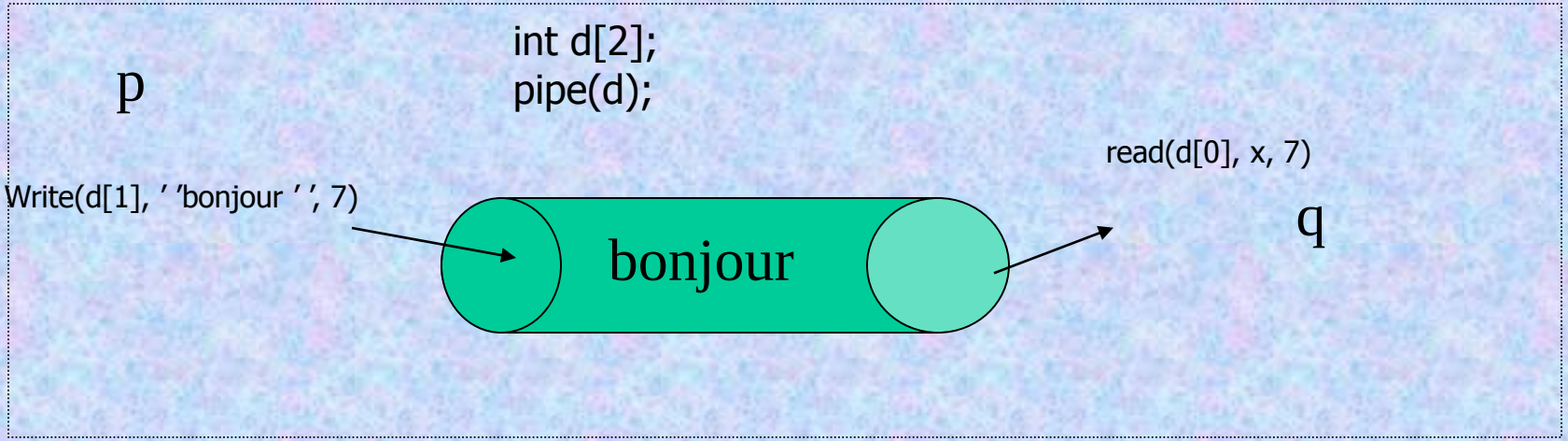
struct protoent

```
#include <netdb.h>
struct protoent {

        char * p_name ; /* nom officiel */

        char **p_aliases /* liste d 'alias*/

        int p_proto; /* numero protocole */

        }
```

struct protoent * getprotobyname(char *x);
voir aussi getprotobynumber(int numero);
getprotoent(), setprotoent(), endprotoent() permettent de manipuler
/etc/protocols

Les sockets

Le mécanisme socket (TLI) est une généralisation des communications locales (pipe() mkfifo()) entre processus.



Les deux processus p et q sont sur la même machine. En outre parentés.

Le mécanisme de socket (ou TLI) permet de généraliser la communication entre processus:

- processus sur des machines différents,
- processus n 'ayant aucun lien de parenté

socket

Une socket est un point de communication par lequel un processus peut recevoir et émettre des données.

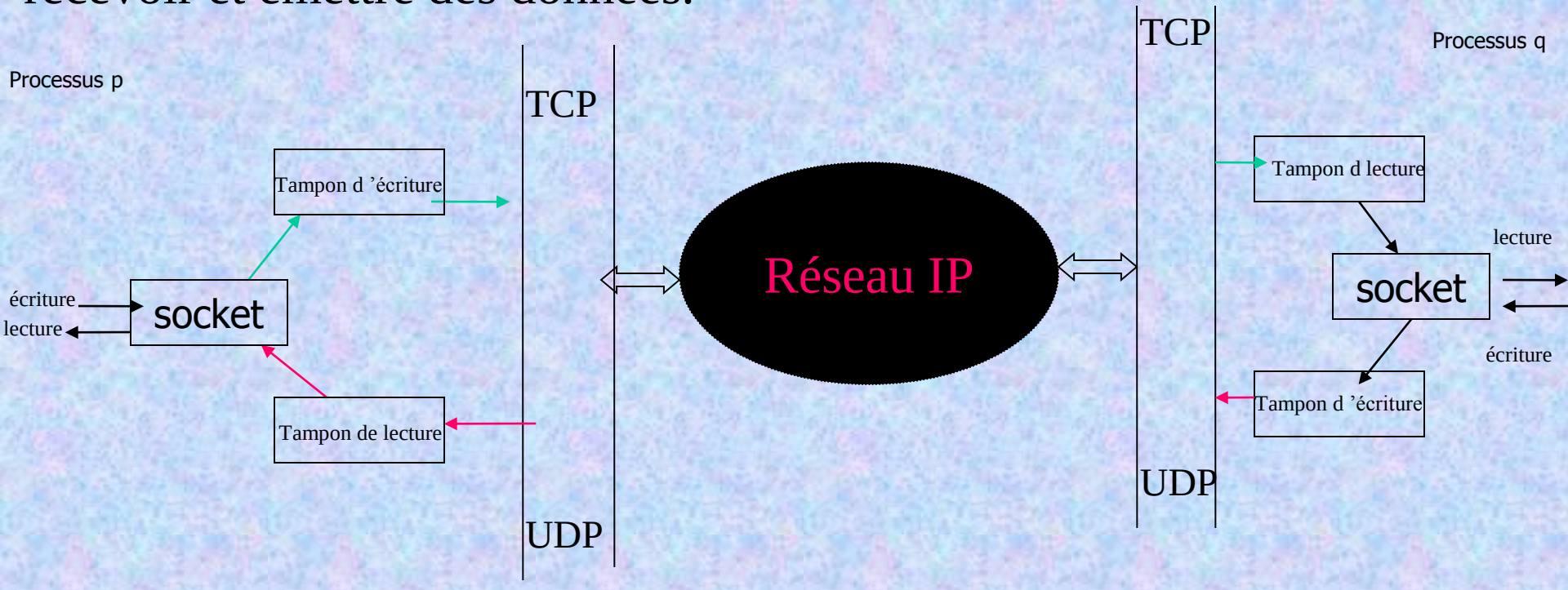


Schéma général

Client		Serveur
socket()		socket()
bind()		bind()
	Connexion ou non	
Envoie de données	→	Réception de données
Réception de données	←	Envoie de données
Close()		Close()

Le type d ’une socket

Type? Le type détermine la sémantique des communications qu ’elle permet de réaliser.

Nom symbolique	Caractéristiques principales
SOCK_DGRAM	Transmission de datagrammes (non connecté, UDP)
SOCK_STREAM	Échange de séquences continues de caractères (connecté, TCP)
SOCK_DRM	Transmission de datagrammes en mode non connecté en garantissant le maximum de fiabilité
SOCK_SEQPACKET	Transmission de messages structurés en mode connecté
SOCK_RAW	Transmission de datagrammes au niveau d ’IP

Domaine d'une socket

Domaine?

Le domaine d 'une socket détermine les autres points de communications qu 'elle permet d 'atteindre.

Il existe plusieurs domaines dont les deux principaux:

-AF_UNIX (PF_UNIX) : le domaine local.

-AF_INET (PF_UNIX): le domaine Internet qui permet d 'atteindre des processus s 'exécutant sur des machines de l 'Internet

Adresse d ’une socket (AF_UNIX)

L ’adresse dépend du domaine de la socket.

```
#include<sys/un.h>
struct sockaddr_un
        {
         short sun_family; /* AF_UNIX */

        char sun_path[108]; /* référence du fichier */
        }
```

AF_INET

```
#include<netinet/in.h>
struct sockaddr_in
{ short sin_family; /* AF_INET*/

u_short sin_port; /*numéro de port */

struct in_addr  sin_addr; /* adresse IP de la machine*/

char sin_zero[8]; /* non utilisés*/

}
```

```
struct in_addr
{
        u_long s_addr;
}
```

Adresse générique

```
#include<sys/socket.h>

struct sockaddr
        {
                u_short sa_family;

                char sa_date[4];
        }
```

Création de socket

int socket (int *domaine*, int *type*, int *protocole*);

Domaine = AF_UNIX, AF_INET,
type = SOCK_DGRAM, SOCK_STREAM,
protocole = Une constante (#include <netinet/in.h>
définissant le protocole sous_jacent:
IPPROTO_UDP, IPPROTO_TCP.
Mettre toujours *protocole*=0.

socket() retourne soit un descripteur (lecture et écriture), soit -1 si problème de création. Pour connaître l 'erreur utiliser perror().

Fermeture :

int close(int descripteur)

int shutdown(int descripteur, int sens), /* sens =0, 1 ou 2*/

sens=0 la socket n 'accepte plus d 'opérations de lecture

Sens=1 la socket n 'accepte plus les demandes d 'envoie.

Sens=2 la socket n 'accepte plus ni lecture ni écriture.

Attachement d ’une socket à une adresse

Pourquoi faut-il attacher une socket à une adresse socket?.

bind(int *descripteur,* struct sockaddr **ptr_adresse*, int *longueur*)

Réalise l ’attachement de la socket de descripteur *descripteur* donné à l ’adresse **ptr_adresse*. Retourne -1 si un problème.

Cas AF_UNIX

```
#include <sys/stat.h>
#include <sys/un.h>
struct sockaddr_un
{       short sun_family; /* AF_UNIX */
        char sun_path[108]; /* reference */
}
```

L 'attachement ne peut se faire si la référence existe:

- rm
- unlink

La primitive sockpair():

int sockpair(int *domaine*, int *type*, int *protocole*, int **ptr_descripteur)*

Permet de récupérer deux descripteurs *ptr_descripeur* de sockets.

Cas AF_INET

```
#include<netinet/in.h>
struct sockaddr_in
{ short sin_family; /* AF_INET*/
u_short sin_port; /*numéro de port */
struct in_addr  sin_addr; /* adresse IP de la machine*/
char sin_zero[8]; /* non utilisés*/
}
```

Valeurs particulières de *sin_port* et *sin_addr:*

- la valeur INADDR_ANY de sin_addr.s_addr permet d 'associer la socket à toute adresse IP de la machine. (cas des passerelles)

- *sin_port =0*, veut dire que le système choisira un numéro de port

Pour connaître l 'adresse choisie (cas INADDR_ANY) et le port (cas sin_port =0) on utilisera la primitive:

int sockname(int *descripeur*, struct sockaddr * *ptr_adresse*, int **ptr_longueur_adresse*)

socket(), bind()

création d ’une socket: socket().

Attachement d ’une socket à une adresse de socke: bind()

Maintenant, comment envoyer et recevoir des données à travers une socket?

Cas non connecté: SOCK_DGRAM:

sendto() pour l ’envoie de données,

recvfrom() pour la réception de données

Cas connecté: SOCK_STREAM

send() et write() pour l ’envoie de données

recv() et read() pour la réception de données

Envoie en mode non Connecté: SOCK_DGRAM

int sendto(int *descripteur*, void * *message,* int *longueur,*
int *option,* struct sockaddr * *p_adresse*, int *longueur_adresse)*

descripteur: le descripteur de la socket

message: le message à envoyer

longueur: longueur du message

option: 0

p_adresse: l ’adresse de la socket distante

longueur_adresse: longueur de l ’adresse

Réception en mode non Connecté: SOCK_DGRAM

recvfrom(int *descripteur*, void * *message,* int *longueur,*
int *option,* struct sockaddr * *p_adresse*, int * *longueur_adresse)*

descripteur: le descripteur de la socket

message: adresse de réception

longueur: taille réservée

option: 0 ou MSG_PEEK

p_adresse: adresse émetteur

longueur_adresse: longueur de l 'adresse de l 'émetteur

Récapitulatif mode non connecté:SOCK_DGRAM

Client		Serveur
socket()		socket()
bind()		bind()
Sendto()	→	Recvfrom()
Recvfrom()	←	Sendto()
Close()		Close()

Mode connecté: SOCK_STREAM

Client

Création de la socket
socket()

Opération optionnelle

Attachement sur un port quelconque
bind()

Construction de l ’adresse du serveur

Demande de connexion
connect()

réussite

Echec

Dialogue
send()/write()
recv()/read(

Serveur

Création de la socket
socket()

Attachement de la socket sur un port connu
bind()

Ouverture du service
listen()

Acceptation d ’une connexion pendante
accept()

Création dun procéssus fil ou d ’une nouvelle activité (fork()/pthread_creat() gérant le dialogue avec le client

connect()

connect(int *descripetur*, struct sockaddr * *ptr_adresse*, int *lg_adresse*)

descripteur: descripteur de socket

ptr_adresse: adresse de la socket distante

lg_adresse: longueur de l ’adresse

listen() et accept()

int listen (int *descripteur*, int *nb_pendantes*)

descripteur: descripteur de la socket d 'écoute
nb_pendantes: nombre maximale de connexions

accept(int *descripteur*, struct sockaddr * *ptr_adresse*, int **ptr_lg_adresse)*

descripteur: descripteur de la socket
ptr_adresse: mémoire pour recevoir l 'adresse du client
ptr_lg_adresse: longeur réservée et en retour la longueur effective

Envoie de données: SOCK_STREAM

t write (int *descripteur*, void * *message*, int *longueur_message*)

send (int *descripteur*, void * *message*, int *longueur_message, int option*)

option = 0 ou MSG_OOB (AF_INET)

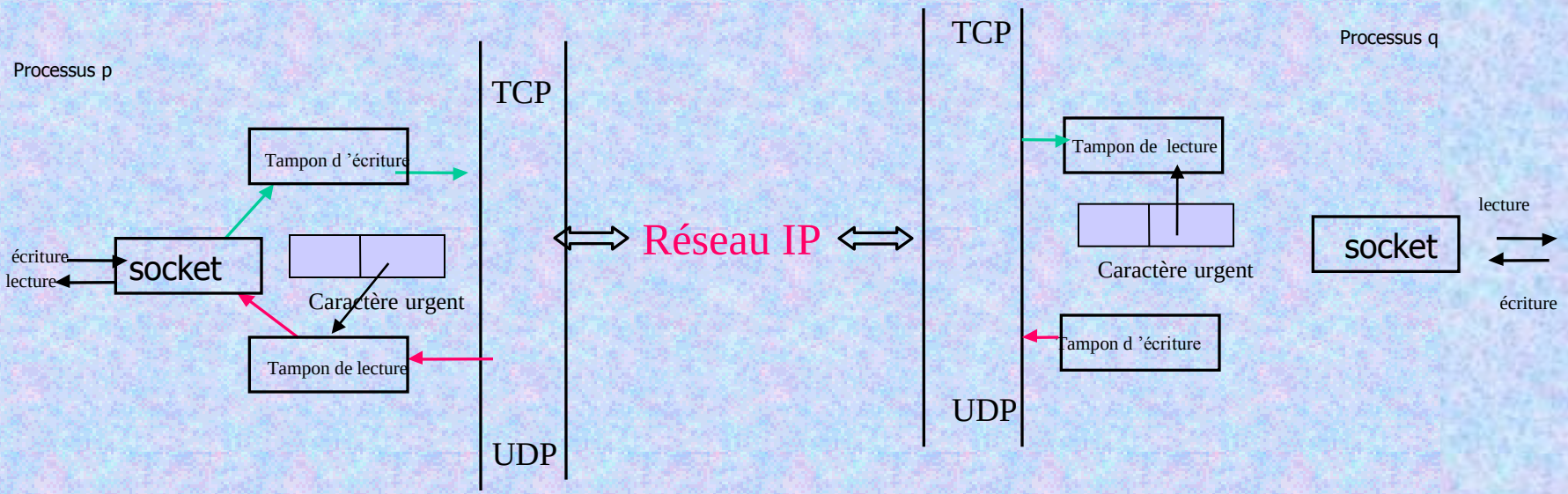
Réception de données:SOCK_STREAM

t read (int *descripteur*, void * *message*, int *longueur_message*)

recv (int *descripteur*, void * *message*, int *longueur_message, int option*)

option = 0 ou MSG_OOB (AF_INET)

Envoi de caractères urgents: SOCK_STREAM



Caractère urgent

La prise en compte du char urgent: MSG_OOB

Le signal SIGURG est envoyé dès l ’arrivé d ’un char urgent

Une lecture sans l ’option MSG_OOB bute sur le char urgent

Attention à l ’envoie successifs de char urgent

Le char urgent n ’est pas intégré au tampon d ’émission seule sa position est repérée.

ntégration des char urgents dans le tampon de lecture:

```
t on=1,
t desc_socket;
etsockpot(desc_socket, SOL_SOCKET, SO_OOBINLINE, &on, sizeof(on))
```

La lecture se fera sans l ’option MSG_OOB

Comment repérer le char urgent?

```
/* #incude<sys/ioctl.h>   : version BSD */
* #include <sys/sockio.h>: version system V */
int reponse;
int desc_socket;
….
ictl(desc_socket, SIOCATMARK, &reponse);

if (reponse==1) { /* le char est urgent */
                    …
        }
else { /* le char n ’est pas urgent */
…
}
```

Avis d ’arrivé de caractère urgent

Mise en œuvre du mécanisme d ’émission du signal SIGURG:

1) installer un handler pour le signal SIGURG, en faisant la Lecture avec l’option MSG_OOB.

2) Se rendre propriétaire de la socket:

```
pid=getpid();

ioctcl(sock, SIOCSPGRP, &pid);

ou fntcl(sock, F_SETOWN, &pid);
```

Scrutation de plusieurs sockets

But développement d'un multi-serveur:

Lecture sur plusieurs descripteurs:
Le processus est susceptible de recevoir des données sur différents
Descripteurs:
- Comment choisir alors le descripteur sur lequel des données sont arrivées?
- Càd sur quel sur descripteur on peut faire un read() sans être bloqué.

Écriture sur plusieurs descripteurs:
De même un write sur un tampon plein provoque l'attente.
Comment choisir le descripteur sur lequel on peut écrire sans être bloqué.

Une solution triviale consiste à rendre ces primitives non bloquantes,
Mais n'est pas satisfaisante car provoque attente active.

Un autre solution: select(), poll() combiné avec le signal SIGIO.

Select()

Cette primitive permet de scruter plusieurs descripteurs (fichiers, tubes, sockets, ….), en se mettant en attente d'événements. Trois événements peuvent être examinés par select():

- la possibilité de lire sur un descripteur
- La possibilité d'écrire sur un descripteur
- l'existence d'un condition exceptionnelle sur un descripteur.

La primitive select()

```
#include <sys/types.h>
#include <sys/times.h>
nt select(      int nb_desc,
        fd_set * prt_lecture,
        fd_set * ptr_ecriture,
        fd_set * ptr_exception,
        const struct timeval * ptr_temporisation);
```

Le type fd_set est prédéfini dans le fichier sys/types.h

Différentes macros sont définies:

Prototype de la fonction	Effet
FD_ZERO(fd_set *ptr_set)	*ptr_set=0
FD_CLR(int desc, fd_set *ptr_set)	*ptr_set= *ptr_set-{desc}
FD_SET(int desc, fd_set *ptr_set)	*ptr_set= *ptr_set U {desc}
FD_ISSET(int desc, fd_set *ptr_set)	Valeur non nulle $\Leftrightarrow$ desc $\in$ *ptr_set

struct timeval

```
#include <sys/times.h>

struct timeval {
        time_t  tv_sec ; /* seconds */
        suseconds_t tv_usec; /*and microseconds */
      }
```

select()

prt_lecture : *l'ensemble des descripteurs sur lequel on souhaite pouvoir réaliser la lecture*

r_ecriture: *l'ensemble des descripteurs sur lequel on souhaite pouvoir réaliser une écriture*

ptr_exception : *l'ensemble des des descripteurs sur lequel on souhaite la réalisation d'un test de condition exceptionnelle. Par exemple, l'arrivée d'un caractère urgent.*

nb_desc: *doit être au moins égal à la valeur du plus grand descripteur appartenant à l'un des trois ensembles augmentée de 1. Il indique que l'opération porte sur les nb_des premiers descripteurs (0, 1, …, nb-desc).*

ptr_temporisation: *pointe sur une valeur qui définit un temps maximal d'attente avant que l'une des événement souhaitée se produit. NULL correspond à une attente infinie.*

Le processus appelant select() est bloqué jusqu'à ce que l'une des condition soit réalisée:

- l'un des événement attendus sur un descripteur de l'un de ces ensembles s'est produit. **prt_lecture*, **ptr_ecriture*, **ptr_exception* contiennent alors les descripteurs sur lesquels l'opération Correspondante est possible.
- le temps maximal est écoulé. La valeur de retour est 0, les trois ensembles sont modifiés.
- un signal capté par le processus est survenu: la valeur de retour est alors –1 et la variable errno=EINTR.

Asynchronisme: SIGIO

Toutes les opérations de lecture (sauf demande contraire) sont bloquantes.
Une des solution était select() (ou poll), mais select() est aussi bloquante.

Activation du mécanisme d'asynchronisme)

- ✓ Construire un handler pour le signal SIGIO. SIGIO reçu impliquera Un message reçu est donc le lire dans le handler. Lire tous les messages Arrivés (! Rendre la socket non bloquante)

- ✓ Attribuer la socket au procéssus:

```
pid =getpid();
fnctl(desc, F-SETOWN, pid) /* SYSTEM V*/
        iocltl(desc, SIOCSPGRP, &pid)  /* BSD */
```

- ✓ Enclencher effectivement le mécanisme:

```
fnctl(desc, F-SETFL, FASYNC);
int on =1; ioctl(desc, FIOASYNC, &on);
```

Contrôle et paramétrages de socket

Ce contrôle nécessite trois primitives: ioctl(), fcntl(), setsockpot()

Exemples de paramétrage:

- Rendre une socket BROADCAST (diffusion),
- Activer le mécanisme d'envoie de char urgent (SIGURG)
- Connaître/changer la taille de segment TCP,
- Asynchronisme (SIGIO)

ioctl()

La primitive utilisable uniquement sur les systèmes BSD. Mais, sur un système V, ioctl() peut être utilisé à condition de compiler le programme avec l'option –D BSD_COMP: gcc –D BSD_COMP …

#include <sys/ioctl.h>int ioctl(int *descripteur*, int *requête*, int * *arg)*

Permet de réaliser sur la socket de descripteur *descripteur* la requête *requête*. Le troisième permet de fournir à certaines requêtes un argument

requête	Paramètre *arg*	effet
FIOSNBIO	int *	Passage en mode non-bloquant si **arg* est non nul
FIONREAD		Renvoie le nombre de caractères lisibles
FIOASYNC	int *	Si **arg* est non nul, le mode asynchrone est activé (signal SIGIO)
SIOCSPGRP Ou FIOSETOWN	int *	*arg désigne un numéro de procéssus ou de groupe De procéssus (si négatif) qui devient propriétaire de la socket (relation avec SIGURG et SIGIO)
SIOCSPGRP Ou FIOGETOWN	int *	**arg* prend comme valeur le numéro de processus Ou de groupe de processus propriétaire de la socket
SIOCATMARK		Renvoie 1 si le pointeur de la lecture pointe sur un Caractère urgent.

fnctl()

include <fnctl.h>

int fnctl(int *descripteur*, int *requête*, int *arg)*

Permet de réaliser sur le socket de descripteur *descripteur* la requête indiquée.

requête	Paramètre *arg*	effet
F_SETOW	int	*arg* est un numéro de processus ou de groupe de de processus (si négatif) qui devient propriétaire de la socket (relation avec SIGIO et SIGUR)
F_GETOWN		Renvoie 1 si le pointeur de la lecture pointe sur un Caractère urgent.
F_SETFL	FASYNC	Mode asynchrone (SIGIO)
F_SETFL	FNDELAY	Mode non-bloquant (style BSD)
F_SETFL	O_NDELAY	Mode non-bloquant (style System V)
F_SETFL	O_NONBLOCK	Mode non bloquant (style POSIX)

Mode non-bloquant

Ce mode non bloquant est réalisé comme suit:

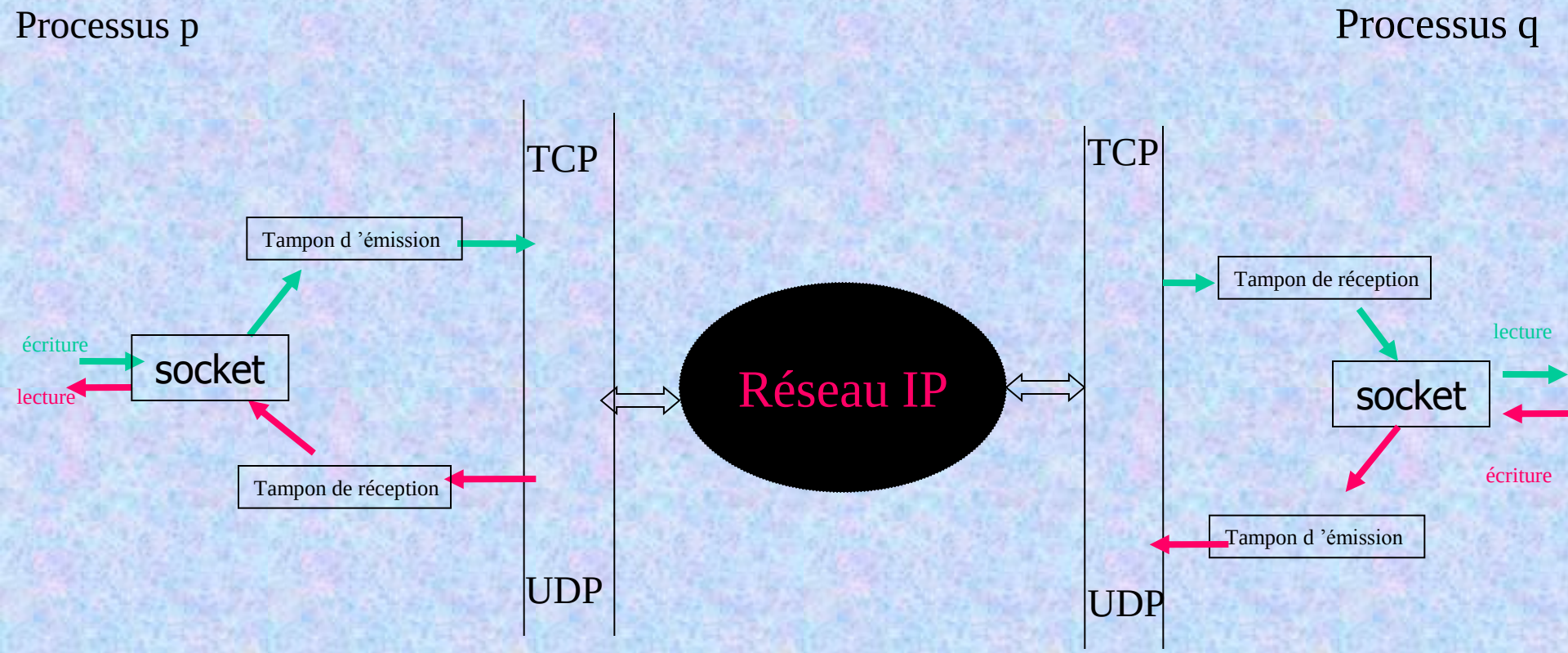
```
int on =1; …… ioctl(desc_sock,  FIOSNBIO, &on);
int on =1; …… ioctl(desc_sock,  FIONBIO, &on);
```

Ou

```
fnctl(desc_sock, F_SETFL, FNDELAY); /* BSD*/
fnctl(desc_sock, F_SETFL, O_NDELAY); /* SYSTEM V*/
fnctl(desc_sock, F_SETFL, O_NONBLOck); /*POSIX*/
```

Le mode non-bloquant affecte le comportement des primitives: send/sendto/read, recv/recvfrom/read, accept() et connect().

socket



-S le tampon de réception est vide, la lecture (read/recv, recvfrom) est bloquée

-Si le tampon d'émission est plein, l'écriture (write/send, sendto) est bloquée

-Si TCP, si tampon de réception est plein, l'entité TCP homologue cesse de transmettre. (attention le tampon d'émission de l'autre socket risque de déborder)

Effets du mode non-bloquants sur l'écriture

- Le problème se pose si le tampon d'émission est plein:

En mode non-bloquant, tout ce qui peut être écrit l'est,le reste est perdu et les primitives retournent le nombre de char effectivement écris:

- Dans le cas ou aucun char ne peut être écrit, l'effet (write, send, sendto) est le suivant:
  - FIOSNBIO: retour –1 et errno=EWOULDBLOCK,
  - O_NONBLOCK: retour –1 et errno=EAGAIN
  - O_NDELAY: retour 0.

Effets du mode non-bloquant sur la lecture

S'il n'y a rien à lire sur la socket, l'effet (read/recv, recvfrom) est le suivant:

- FIOSNBIO: retour –1 et errno=EWOULDBLOCK
- O_NONBLOCK: retour –1 et errno =EGAIN,
- O_NDELAY: retour 0.

Effet sur l'acceptation de la connexion

Mode bloquant(par défaut):
accept() bloque le processus appelant s'il n'y pas de connexions pendantes.

En mode bloquant, l'effet est le suivant:

- FIOSNBIO ou O_NDELAY: retour –1 et errno=EWOULDBLOCK
- O_NONBLOCK: retour –1 et errno=EAGAIN.

Effet sur la demande de connexion

Mode bloquant(par défaut):
connect() bloque le processus appelant jusqu'à ce que la connexion puisse être réalisée.

En mode non-bloquant, le retour de l'appel est immédiat, et la demande de connexion n'est pas abandonnée:

L'effet est le suivant:

- retour –1, errno=EINPROGRESS, la tentative de connexion est itérée.
- les appels ultérieurs à connect avec la même adresse de destination

donnent des informations importantes:
Retour 0 (connexion a pu être établie),
retour –1 et errno = ETIMEOUT(connexion a échouée),
Retour –1 et errno=EALREADY tentative en cours.

Paramétrage des sockets

Les options applicables aux opération relatives à une socket peuvent être interprétées à différents niveaux.

Deux niveaux importants dans le domaine AF_INET:

- le niveau SOL_SOCKET: les options s'appliquent directement sur la socket.
- niveaux protocoles: IPPROTO_IP, IPPROTO_TCP, IPPROTO_UDP.

Les options sont de deux types:

- Les options booléennes: à un instant donnée les options de ce types s'appliquent ou non à la socket concernée.
- Les options non booléennes: correspondent à la valeur d'un pour la communication (type de la socket ou taille des tampons)

getsockopt()

```
#include <sys/socket.h>

Int getsockopt( int descripteur, /* descripteur de la socket*/

        int niveau, /* niveau d'application de l'option */
        int option, /* option concernée */

        void *p_valeur_option, /* pointeur sur valeur */

        int *p_longueur_option, /* longueur*/
        )
```

Si option est booléenne, le retour 0 indique que l'option est positionnée et si le retour = -1 avec errno=ENOPROOOPT, alors l'option ne s'applique pas.

Dans le cas où l'option n'est pas supportée au niveau *niveau* le retour est –1 et errno = EINVAL *(si niveau=*SOL_SOCKET) et errno=EOPNOTSUPP *(*si un autre niveau).

setsockopt()

```
#include <sys/socket.h>

Int setsockopt( int descripteur, /* descripteur de la socket*/

        int niveau, /* niveau d'application de l'option */

        int option, /* option concernée */

        void *p_valeur_option, /* pointeur sur valeur */

        int  longueur_option, /* longueur de valeur*/
        )
```

Si *option* est booléenne, *p_valeur_option* pointe sur un entier, et l'option est désactivée ou activée selon que **p_valeur_option* =0 ou <>0.
Pour les options non booléennes, *p_valeur_option* doit pointer sur la valeur qu'on souhaite Donnée à l'option.

Les principales options booléennes

Elles s'appliquent au niveau SOL_SOCKET (sauf l'option TCP_NODELAY):

- SO_BROADCAST : s'applique si SOCK_DGRAM et AF_INET.

  - connaître l'adresse de diffusion de l'interface physique:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/ioctl.h>
/* #include <sys/seockio.h> : SYSTEM V */
# include<netinet/in.h>
#include<net/if.h>

long in adr_diffusion;
adr_diffusion=ioctl(desc, SIOCGIFBRDADDR, NULL);
```

Mettre cette adresse dans sin_addr.s_addr de l'adresse de destination.

  - Autoriser la diffusion

```
int on =1;
setsockopt(desc, SOL_SOCKET, SO_BROADCAST, &on, sizeof(on));
```

- SO_DONTROUTE: s'applique si SOCK_STREAM, les données émises court-circuitent la procédure standard de routage: elles sont dirigés sur l'interface déduite de la partie réseau de leur adresse destination.

Les principales options booléennes

SO_KEEPALIVE: s'applique si SOCK_STREAM et AF_INET et permet de tester si
ne connexion sur laquelle aucun transfert n'a été réalisée depuis un laps de temps est encore
e et éventuellement de la fermer.

- SO_LINGER: SOCK_STREAM et AF_INET. Cette option est utilisée avec une valeur ayant la structure linger

```
struct linger {
        int l_onoff; /* valeur 0 ou 1 */
        int l_linger ; / durée en secondes */
```

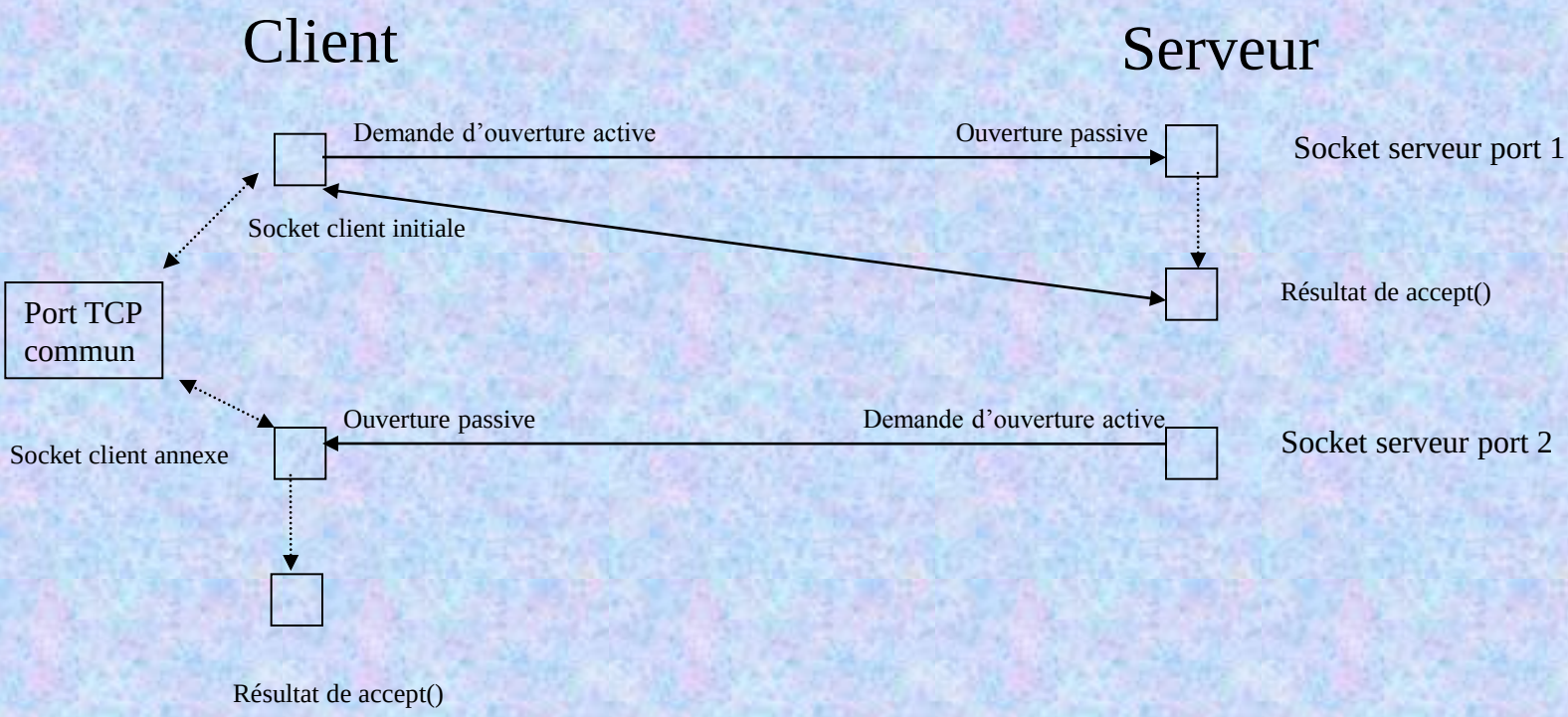
Si **l_onoff** =1, **l_linger** spécifie le temps qui s'écoule entre l'appel close() et la fermeture réelle.

Par défaut le temps est illimité. Donc aussi longtemps qu'il y des données à transmettre, elles le seront malgré la demande close()

- SO_OOBINLINE: SOCK_STREAM et AF_INET, spécifie que les données MSG_OOB sont placées dans le tampon de réception et par suite elles peuvent être lues sans MSG_OOB.

Les principales options booléennes

SO_REUSEADDR: AF_INET, permet la réutilisation d'une adresse locale pour réaliser une Connexion. Intérêt:



P_NODELAY: s'applique au niveau IPPROTO-TCP, SOCK_STREAM, AF_INET
P_NODELAY est spécifiée dans <netinet/tcp.h>). Cette option force l'envoie immédiat
segments TCP. Par défaut, les segments de petite taille ne sont pas envoyés tout de suite
s après un laps de temps espérant que des données arrivent.

Les principales booléennes

Les options suivantes s'appliquent au niveau SOL_SOCKET (sauf TCP_MAX)

SO_TYPE: option de type *int* correspondant au type de la socket.

SO_RCVBUF: concerne la taille du tampon de réception de la socket.

SO_SNDBUF: concerne la taille du tampon d'émission de la socket

SO_MAXSEG: s'applique au niveau IPPROT_TCP des sockets SOCK_STREAM et AF_INET. Permet de connaître la taille maximale d'un segment TCP.

```
int taille_option =sizeof(int);
int taille_segment;
getsockpot(desc, IPPROT_TCP, TCP_MAXSEG, &taille_segment, &taille_option);
```

Prise en compte d'un serveur par inetd

Même si les serveurs ne consomment pas de temps CPU, puisqu'ils sont en attente passive (recvfrom() avec SOCK_DGRAM et accept() cas SOCK_STREAM) ils continuent d'exister dans le système et monopolisent des ressources.

- le service doit être répertorié dans /etc/inetd.conf

- Modifier le code du serveur de tel sorte que les opérations sur la socket doivent utilisées le descripteur STDIN_FILENO