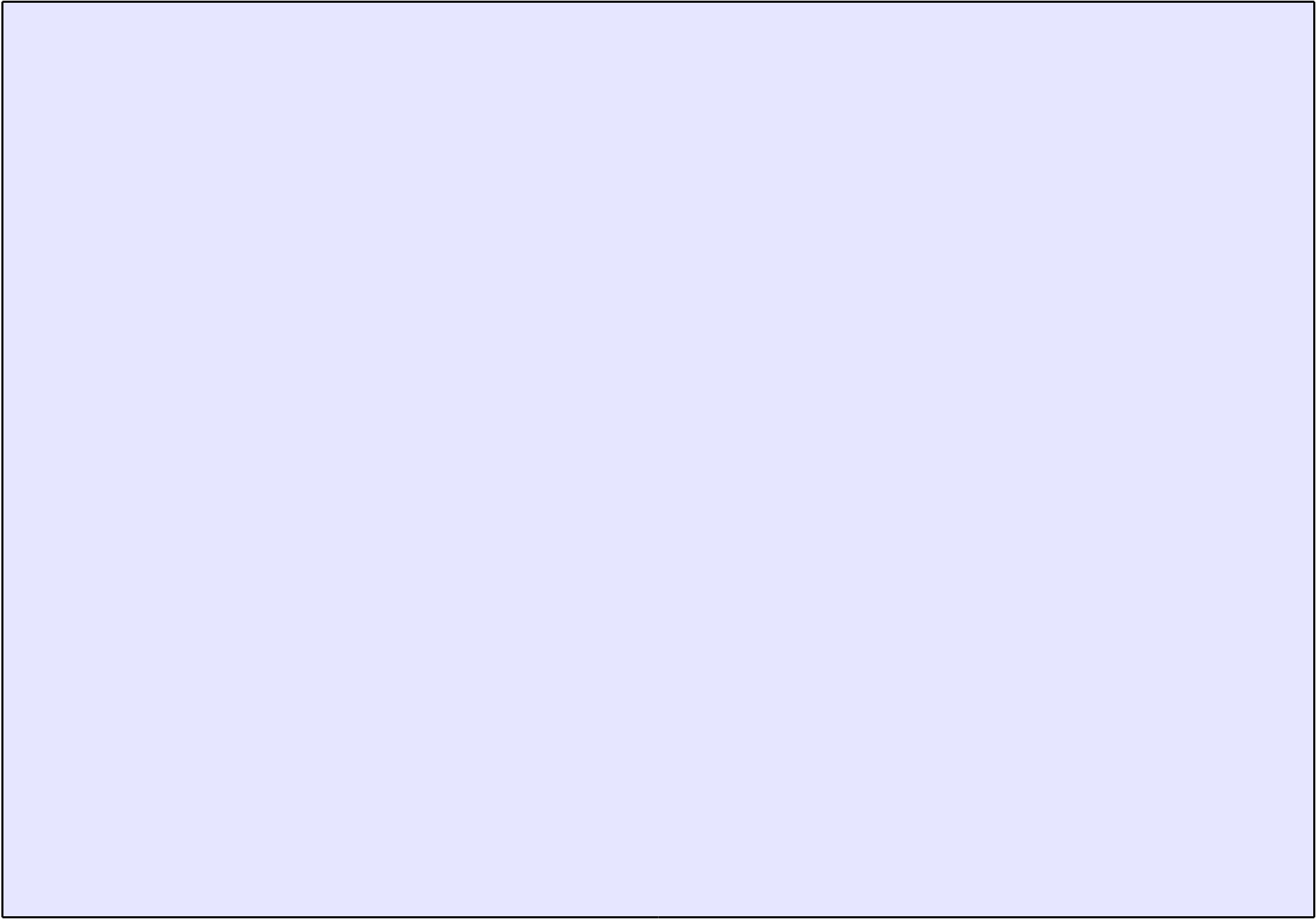
Non-bloquant – Récepteur

|  |  |
| --- | --- |
| **void \* recoit\_udp( void \*arg){** |  |
| **int sock=\*((int \*)arg);** |  |
| **char tampon[100];** |  |
| **while(1){** |  |
| **int rec=recv(sock,tampon,100,0);** |  |
| **tampon[rec]='\0';} printf("Message recu: %s\n", tampon);}** |  |
| **int main(){**  **int sock1=socket(PF\_INET,SOCK\_DGRAM,0);** |  |
| **struct sockaddr\_in address\_sock1;** |  |
| **address\_sock1.sin\_family=AF\_INET;** |  |
| **address\_sock1.sin\_port=htons(5555);** |  |
| **address\_sock1.sin\_addr.s\_addr=htonl(INADDR\_ANY);** |  |
| **int sock2=socket(PF\_INET,SOCK\_DGRAM,0);** |  |
| **struct sockaddr\_in address\_sock2;** |  |
| **address\_sock2.sin\_family=AF\_INET;** |  |
| **address\_sock2.sin\_port=htons(5556);** |  |
| **address\_sock2.sin\_addr.s\_addr=htonl(INADDR\_ANY);** |  |
| **int r=bind(sock1,(struct sockaddr \*)&address\_sock1,sizeof(struct sockaddr\_in));** |  |
| **if(r==0){** |  |
| **int r2=bind(sock2,(struct sockaddr \*)&address\_sock2,sizeof(struct sockaddr\_in));** |  |
| **if(r2==0){** |  |
| **pthread\_t th1,th2;** |  |
| **pthread\_create(&th1,NULL,recoit\_udp,&sock1);** |  |
| **pthread\_create(&th2,NULL,recoit\_udp,&sock2);** |  |
| **pthread\_join(th1,NULL);** |  |
| **pthread\_join(th2,NULL);}}** |  |
| **return 0;}** |  |
|  |  |
|  | 6 |

Non-bloquant – Émetteur

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **int main(int argc,char \*argv[]) {** | | |  |  |
|  | **if(argc>1){** | |  |  |
|  |  | **int sock=socket(PF\_INET,SOCK\_DGRAM,0);** |  |  |
|  |  | **struct addrinfo \*first\_info;** |  |  |
|  |  | **struct addrinfo hints;** |  |  |
|  |  | **memset(&hints, 0, sizeof(struct addrinfo));** |  |  |
|  |  | **hints.ai\_family = AF\_INET;** |  |  |
|  |  | **hints.ai\_socktype=SOCK\_DGRAM;** |  |  |
|  |  | **int r=getaddrinfo("localhost",argv[1],&hints,&first\_info);** |  |  |
|  |  | **if(r==0){** |  |  |
|  |  | **if(first\_info!=NULL){** |  |  |
|  |  | **struct sockaddr \*saddr=first\_info->ai\_addr;** |  |  |
|  |  | **char tampon[100];** |  |  |
|  |  | **int i=0;** |  |  |
|  |  | **for(i=0;i<=3;i++){** |  |  |
|  |  | **strcpy(tampon,"MESSAGE ");** |  |  |
|  |  | **strcat(tampon,argv[1]);** |  |  |
|  |  | **strcat(tampon," ");** |  |  |
|  |  | **char entier[3];** |  |  |
|  |  | **sprintf(entier,"%d",i);** |  |  |
|  |  | **strcat(tampon,entier);** |  |  |
|  |  | **sendto(sock,tampon,strlen(tampon),0,saddr,(socklen\_t)sizeof(struct** |  |  |
|  |  | **sockaddr\_in));** |  |  |
|  |  | **} sleep(1);** |  |  |
|  |  | **} }** |  |  |
| **}** | **}** | **close(sock);** |  |  |
| **return 0;** | | 7 |  |
|  |  |  |  |



Quelles solutions ? (2)

* Une autre solution consiste à supprimer la concurrence et à rendre les deux sockets UDP sur lesquels on attend les messages non-bloquantes

**fcntl( sock1, F\_SETFL, O\_NONBLOCK);**



**fcntl ( sock2, F\_SETFL, O\_NONBLOCK);**

**...**

**rec1=recv(sock1,tampon,100,0);**

**printf("Taille de données recues %d\n",rec1); if(rec1>=0){**

**tampon[rec1]='\0';**

**} printf("Message recu : %s\n",tampon);**

**rec2=recv(sock2,tampon,100,0);**

**printf("Taille de données recues %d\n",rec2);**

**if(rec2>=0){**

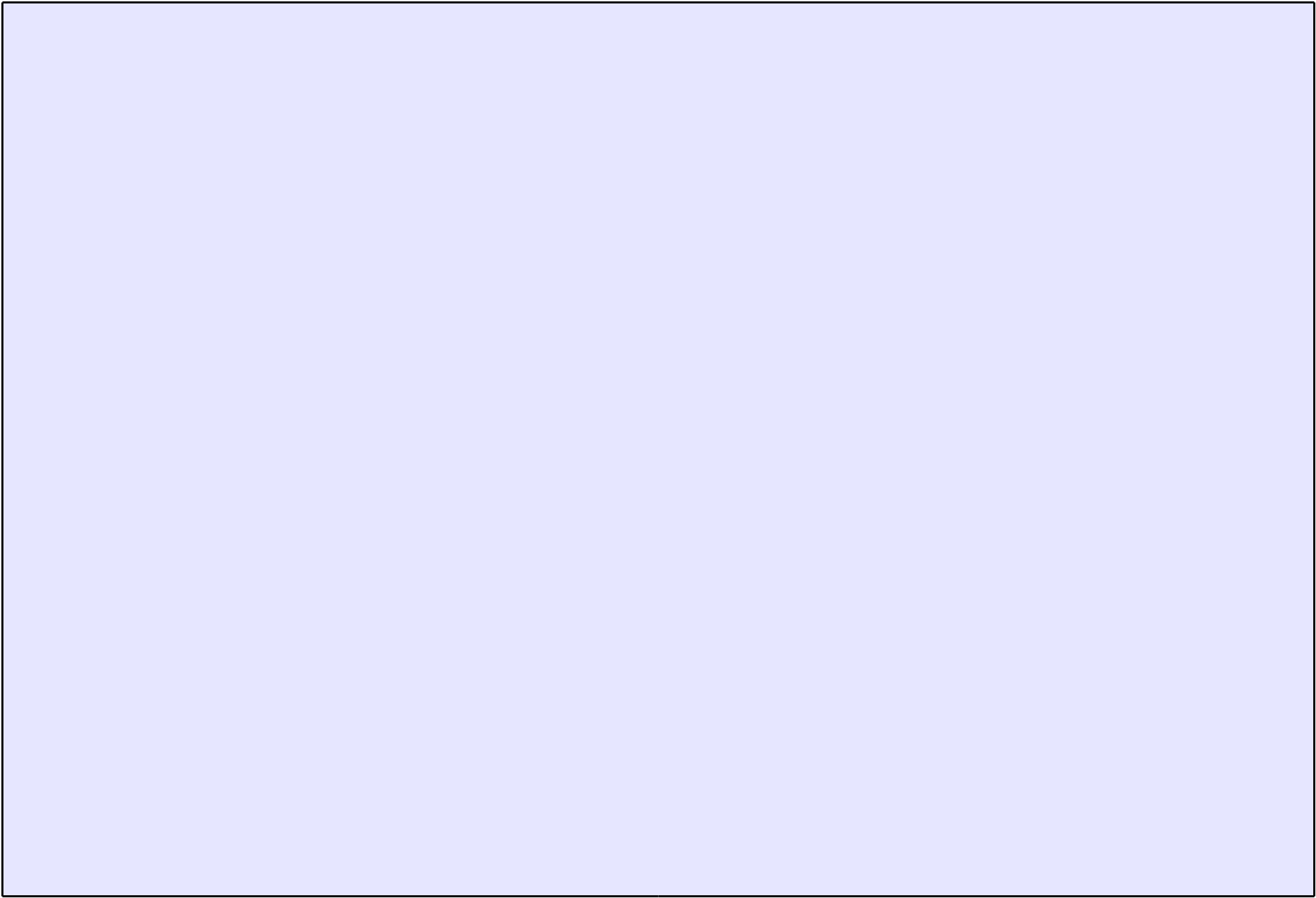
**tampon[rec2]='\0';**

**} printf("Message recu : %s\n",tampon);**

* Le problème est que lorsque l'on fait **fcntl( sock, F\_SETFL, O\_NONBLOCK)** et qu'on fait **recv** sur **sock**. si il n'y a pas de messages, le **recv** renvoie -1

PR - E/S NON BLOQUANT EN C 8

Exemple - Récepteur non satisfaisant



|  |  |  |
| --- | --- | --- |
| **int main() {** |  |  |
| **int sock1=socket(PF\_INET,SOCK\_DGRAM,0);** |  |  |
| **struct sockaddr\_in address\_sock1;** |  |  |
| **address\_sock1.sin\_family=AF\_INET;** |  |  |
| **address\_sock1.sin\_port=htons(5555);** |  |  |
| **address\_sock1.sin\_addr.s\_addr=htonl(INADDR\_ANY);** |  |  |
| **int sock2=socket(PF\_INET,SOCK\_DGRAM,0);** |  |  |
| **struct sockaddr\_in address\_sock2;** |  |  |
| **address\_sock2.sin\_family=AF\_INET;** |  |  |
| **address\_sock2.sin\_port=htons(5556);** |  |  |
| **address\_sock2.sin\_addr.s\_addr=htonl(INADDR\_ANY);** |  |  |
| **int r=bind(sock1,(struct sockaddr \*)&address\_sock1,sizeof(struct sockaddr\_in));** |  |  |
| **if(r==0){** |  |  |
| **int r2=bind(sock2,(struct sockaddr \*)&address\_sock2,sizeof(struct sockaddr\_in));** |  |  |
| **if(r2==0){** |  |  |
| **fcntl( sock1, F\_SETFL, O\_NONBLOCK);** |  |  |
| **fcntl ( sock2, F\_SETFL, O\_NONBLOCK);** |  |  |
| **char tampon[100];** |  |  |
| **int rec1=0;** |  |  |
| **int rec2=0;** |  |  |
| **while(1){** |  |  |
| **rec1=recv(sock1,tampon,100,0);** |  |  |
| **printf("Taille de données recues %d\n",rec1);** |  |  |
| **if(rec1>=0){** |  |  |
| **tampon[rec1]='\0';** |  |  |
| **} printf("Message recu : %s\n",tampon);** |  |  |
| **rec2=recv(sock2,tampon,100,0);** |  |  |
| **printf("Taille de données recues %d\n",rec2);** |  |  |
| **if(rec2>=0){** |  |  |
| **tampon[rec2]='\0';** |  |  |
| **} printf("Message recu : %s\n",tampon);** |  |  |
| **} }** |  |  |
| **}** | 9 |  |
| **return 0;** |  |
| **}** |  |  |

Comment lire au bon moment ?

* Dans la solution précédente, le problème est que
  + Effectivement on ne bloque plus
  + **MAIS** on teste trop souvent si des données sont disponibles
  + On parle d' **ATTENTE ACTIVE**
* On voudrait pouvoir bloquer si il n'y a aucun message à recevoir et être débloqué dès qu'un message arrive sur l'une des deux sockets
* Comment peut-on faire cela ?
  + On va utiliser la fonction **select**
  + Cette fonction demande au système de réveiller un processus dès qu'une opération sera possible sur un descripteur parmi d'autres
  + Cette fonction permet d'attendre sur tout type de descripteur sockets, tubes, tty etc

PR - E/S NON BLOQUANT EN C 10

La fonction select

* **int select(int numfds, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, struct timeval \*timeout);**
* les **fd\_set** sont des ensembles de file descriptor
  + **numfds** est le numéro maximal du file descriptor que l'on souhaite observer plus 1
  + **readfds** est l'ensemble des descripteurs que l'on observe en lecture
  + **writefds** est l'ensemble des descripteurs sur lesquels on attend de pouvoir écrire
  + **exceptfds** est l'ensemble des descripteurs surveiller pour conditions exceptionnelles
  + **timeout** est un temps maximal d'attente, mis à NULL si on ne veut pas de timout
* Le **select** est bloquant et renvoie le nombre de descripteurs sur lequel on peut intervenir

PR - E/S NON BLOQUANT EN C 11

Exemple typique de select

* On veut écouter sur deux descripteurs d1 et d2 en lecture en boucle

●

**fd\_set initial;**



**int fd\_max=0;**

**FD\_ZERO(&initial);**

**FD\_SET(d1,&initial);**

**fd\_max=maximum(fd\_max,d1) ; //on suppose qu'il y a une fonction maximum FD\_SET(d2,&initial);**

**fd\_max=maximum(fd\_max,d2);**

**while(1){**

**fd\_set rdfs; //on initialisera a chaque tour cet ensemle FD\_COPY (&initial,&rdfs);**

**int res=select(fd\_max+1, &rdfs, NULL, NULL, NULL) ; while(res>0){**

**if(FD\_ISSET(d1,&rdfs)){ ... ; res-- ;} if(FD\_ISSET(d2,&rdfs)){ ... ; res-- ;}**

**} }**

PR - E/S NON BLOQUANT EN C 12

Que faut-il faire pour utiliser select

* Il faut se rappeler du descripteur maximal que l'on manipule
* Il faut créer les ensembles de file descriptor à donner en argument
* Comment manipuler les file descriptor :
  + **FD\_SET(int fd, fd\_set \*set);** Ajoute fd à l'ensemble
  + **FD\_CLR(int fd, fd\_set \*set);** Enlève fd de l'ensemble
  + **FD\_ISSET(int fd, fd\_set \*set);** Renvoie vraie si fd est dans l'ensemble.
  + **FD\_ZERO(fd\_set \*set) ;** Efface tous les éléments de l'ensemble
  + **FD\_COPY(fd\_set \*orig, fd\_set \*copy);** Copie orig dans copy
* Quand select termine, il renvoie le nombre de descripteurs sur lesquels les opérations ont été réalisées
* Les fd\_set ont été modifiées !!!!!! Seuls les descripteurs 'disponibles' sont restées, on peut donc tester avec **FD\_ISSET** qui est disponible
* Il faut réinitialiser les fd\_set pour un prochain appel à select

PR - E/S NON BLOQUANT EN C 13

* + - propos du timeout
* **select** n'est pas nécessairement complètement bloquant
* Comme on l'a vu on peut proposer un timeout dans le dernier argument
* Dans ce cas, soit select termine car il y a un événement soit parce que le temps d'attente à dépasser le timeout
* Le timeout est de type **struct timeval \*timeout**
  + **struct timeval {**

**int tv\_sec; // seconds**

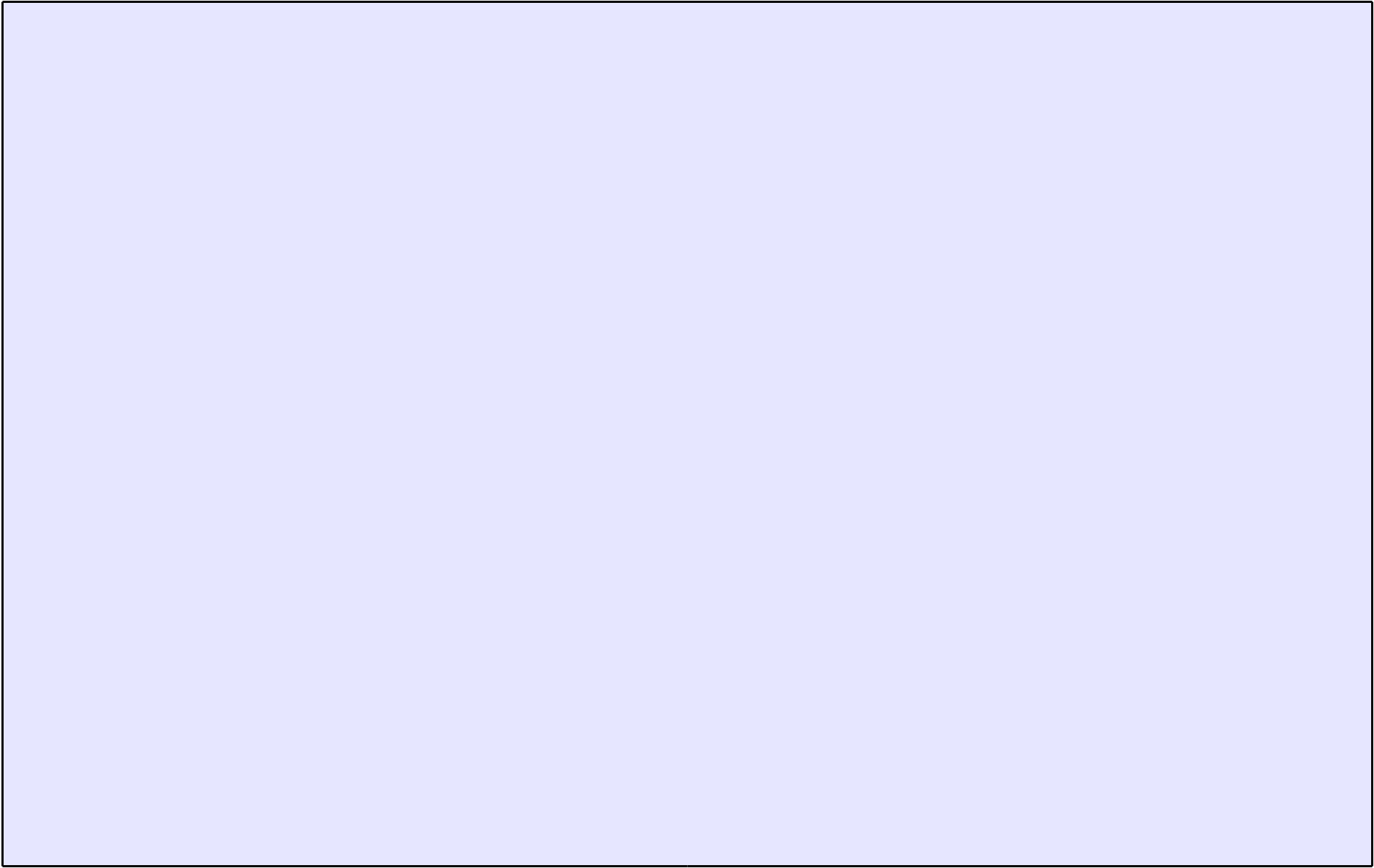
**int tv\_usec;** **// microseconds };**

* Exemple simple :
  + Programme qui attend que l'on tape un caractère au clavier et si on ne le tape au bout de 2,5 seconds s'arrête

PR - E/S NON BLOQUANT EN C 14

Exemple

|  |  |
| --- | --- |
| **int main(void)** |  |
| **{** |  |
| **struct timeval tv;** |  |
| **fd\_set initial;** |  |
| **int ret=0;** |  |
| **tv.tv\_sec = 2;** |  |
| **tv.tv\_usec = 500000;** |  |
| **FD\_ZERO(&initial);** |  |
| **FD\_SET(STDIN\_FILENO, &initial);** |  |
| **char mess[2];** |  |
| **while(1){** |  |
| **fd\_set rfds;** |  |
| **FD\_COPY(&initial,&rfds);** |  |
| **ret=select(STDIN\_FILENO+1, &rfds, NULL, NULL, &tv);** |  |
| **printf("Valeur de retour de select : %d\n",ret);** |  |
| **if (FD\_ISSET(STDIN\_FILENO, &rfds)){** |  |
| **printf("On appuie sur une touche\n");** |  |
| **read(STDIN\_FILENO,mess,1);** |  |
| **mess[1]='\0';** |  |
| **printf("Touche %s\n",mess);** |  |
| **} else {** |  |
| **printf("Timed out.\n");** |  |
| **} return 0;** |  |
| **}** |  |
| **return 0;** |  |
| **}** |  |
| PR - E/S NON BLOQUANT EN C | 15 |



Retour sur notre récepteur

**int main() {**



**int sock1=socket(PF\_INET,SOCK\_DGRAM,0);**

**struct sockaddr\_in address\_sock1;**

**address\_sock1.sin\_family=AF\_INET;**

**address\_sock1.sin\_port=htons(5555);**

**address\_sock1.sin\_addr.s\_addr=htonl(INADDR\_ANY);**

**int sock2=socket(PF\_INET,SOCK\_DGRAM,0);**

**struct sockaddr\_in address\_sock2;**

**address\_sock2.sin\_family=AF\_INET;**

**address\_sock2.sin\_port=htons(5556);**

**address\_sock2.sin\_addr.s\_addr=htonl(INADDR\_ANY);**

**int r=bind(sock1,(struct sockaddr \*)&address\_sock1,sizeof(struct sockaddr\_in)); if(r==0){**

**int r2=bind(sock2,(struct sockaddr \*)&address\_sock2,sizeof(struct sockaddr \_in));**

**if(r2==0){**

**fcntl( sock1, F\_SETFL, O\_NONBLOCK);**

**fcntl( sock2, F\_SETFL, O\_NONBLOCK);**

**fd\_set initial;**

**int fd\_max=0;**

**FD\_ZERO(&initial);**

**FD\_SET(sock1,&initial);**

**if(fd\_max<sock1){fd\_max=sock1;}**

**FD\_SET(sock2,&initial);**

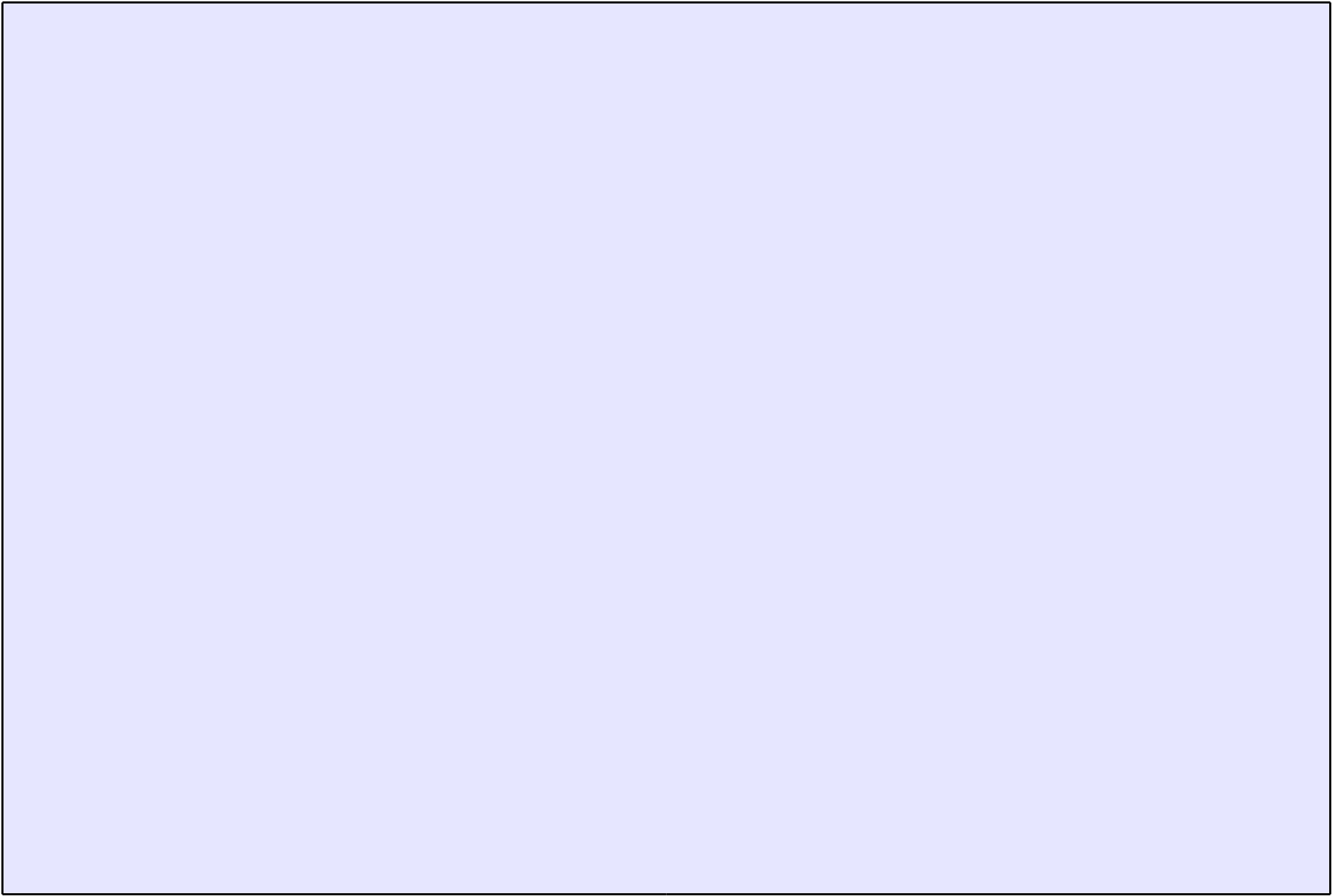
**if(fd\_max<sock2){fd\_max=sock2;}**

PR - E/S NON BLOQUANT EN C 16

**}**

Retour sur notre récepteur

**char tampon[100];**



**int rec1=0;**

**int rec2=0;**

**while(1){**

**fd\_set rdfs;**

**FD\_COPY(&initial,&rdfs);**

**int ret=select(fd\_max+1, &rdfs, NULL, NULL, NULL); while(ret>0){**

**if(FD\_ISSET(sock1,&rdfs)){**

**rec1=recv(sock1,tampon,100,0);**

**printf("Taille de données recues %d\n",rec1);**

**if(rec1>=0){**

**tampon[rec1]='\0';**

**printf("Message recu : %s\n",tampon);**

**} } ret--;**

**if(FD\_ISSET(sock2,&rdfs)){**

**rec2=recv(sock2,tampon,100,0);**

**printf("Taille de données recues %d\n",rec2); if(rec2>=0){**

**tampon[rec2]='\0';**

**printf("Message recu : %s\n",tampon);**

**} } } ret--;**

**} } }**

**return 0;**

Autre méthode que select

* Il existe une autre fonction pour mettre en attente des processus sur différents descripteurs
* La fonction **poll**
* La structure est différente du **select**
* Alors que select manipuler un champ de bits (les fd\_set) mis à 0 ou 1 selon si on voulait suivre on non le descripteur associé
* On fournit à la fonction poll un tableau de structure à suivre
* **int poll(struct pollfd \*ufds, unsigned int nfds, int timeout);**
  + **ufds** est un tableau de struct pollfd
  + **nfds** est est la taille du tableau

PR - E/S NON BLOQUANT EN C 18

Structure manipulée par poll

* **struct pollfd {**

**int fd;** //le descripteur

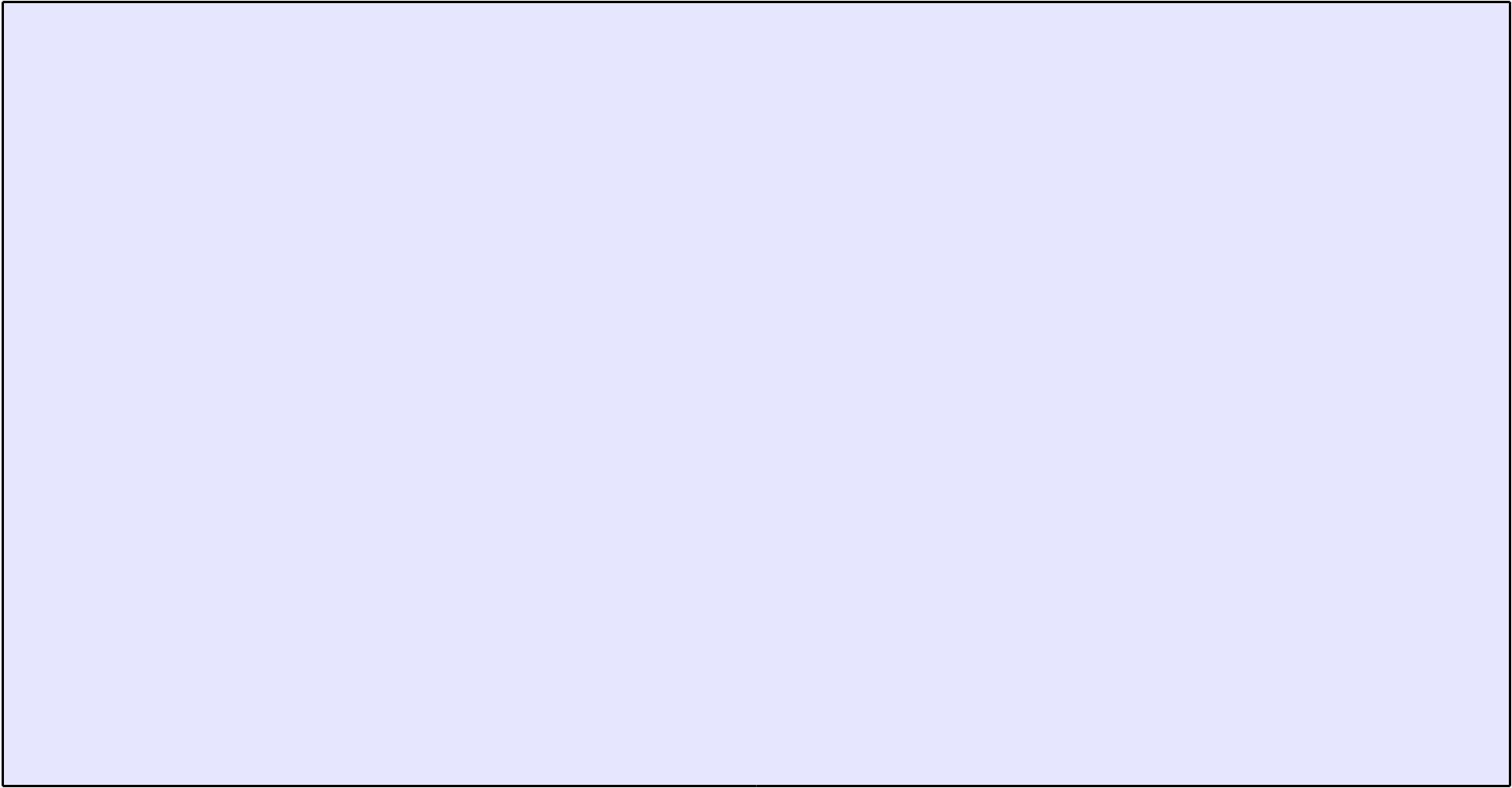
**short events;** / l'événement qui nous intéresse **short revents;** //l'événement quand poll retourne};

* Les événements possibles sont :
  + **POLLIN** : lecture ou accept
  + **POLLOUT** : écriture
  + **POLLPRI** : lecture prioritaire
  + **POLLHUP** : déconnexion
  + **POLLERR** : erreur

PR - E/S NON BLOQUANT EN C 19

Retour sur notre récepteur

**int main() {**



**int sock1=socket(PF\_INET,SOCK\_DGRAM,0);**

**struct sockaddr\_in address\_sock1;**

**address\_sock1.sin\_family=AF\_INET;**

**address\_sock1.sin\_port=htons(5555);**

**address\_sock1.sin\_addr.s\_addr=htonl(INADDR\_ANY);**

**int sock2=socket(PF\_INET,SOCK\_DGRAM,0);**

**struct sockaddr\_in address\_sock2;**

**address\_sock2.sin\_family=AF\_INET;**

**address\_sock2.sin\_port=htons(5556);**

**address\_sock2.sin\_addr.s\_addr=htonl(INADDR\_ANY);**

**int r=bind(sock1,(struct sockaddr \*)&address\_sock1,sizeof(struct sockaddr\_in)); if(r==0){**

**int r2=bind(sock2,(struct sockaddr \*)&address\_sock2,sizeof(struct sockaddr\_in));**

**if(r2==0){**

**fcntl( sock1, F\_SETFL, O\_NONBLOCK);**

**fcntl( sock2, F\_SETFL, O\_NONBLOCK);**

**struct pollfd p[2];**

**p[0].fd=sock1;**

**p[0].events=POLLIN;**

**p[1].fd=sock2;**

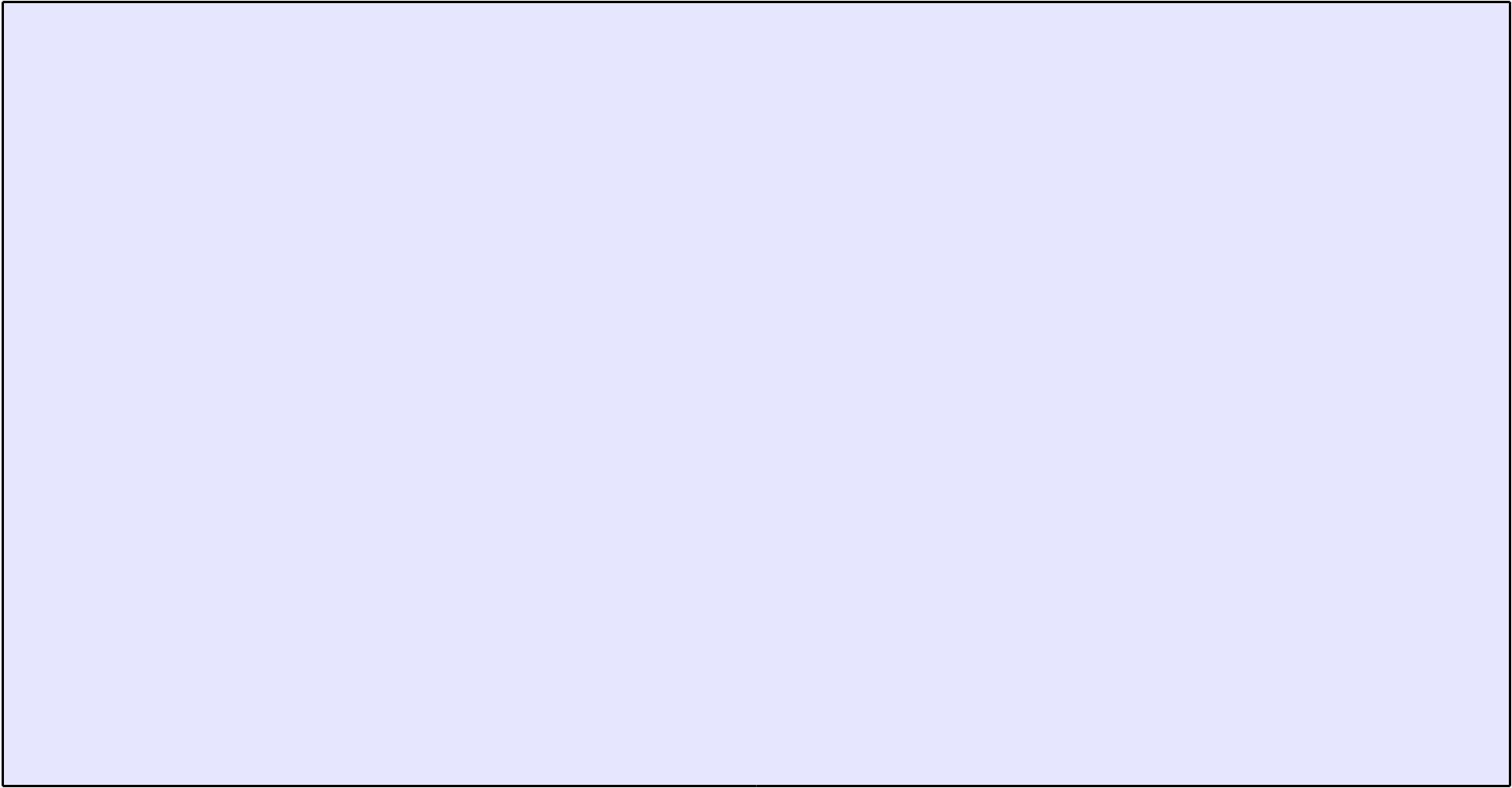
**p[1].events=POLLIN;**

PR - E/S NON BLOQUANT EN C 20

**}**

Retour sur notre récepteur

**char tampon[100];**



**int rec1=0;**

**int i;**

**while(1){**

**int ret=poll(p,2,-1);**

**if(ret>0){**

**for(i=0;i<2;i++){**

**if(p[i].revents==POLLIN){**

**rec1=recv(p[i].fd,tampon,100,0);**

**printf("Taille de données recues %d\n",rec1);**

**if(rec1>=0){**

**tampon[rec1]='\0';**

**} } printf("Message recu : %s\n",tampon);**

**}**

**} }**

**} }**

**return 0;**

PR - E/S NON BLOQUANT EN C 21