Multiplexage des entrées / sorties

Application aux sockets

Hinde Bouziane

bouziane@lirmm.fr

Définitions

Multiplexage des entrées/sorties

Un moyen de scruter plusieurs descripteurs de fichier ouverts, en attendant qu'un événement se produise sur au moins l'un de ces descripteurs.

Définitions

Multiplexage des entrées/sorties

Un moyen de scruter plusieurs descripteurs de fichier ouverts, en attendant qu'un événement se produise sur au moins l'un de ces descripteurs.

Evénement

Un changement d'état à "prêt" pour effectuer une ou des opérations d'entrée/sortie. Lorsqu'un descripteur est à l'état prêt pour une opération, cette dernière peut être effectuée sans blocage.

Définitions

Multiplexage des entrées/sorties

Un moyen de scruter plusieurs descripteurs de fichier ouverts, en attendant qu'un événement se produise sur au moins l'un de ces descripteurs.

Evénement

Un changement d'état à "prêt" pour effectuer une ou des opérations d'entrée/sortie. Lorsqu'un descripteur est à l'état prêt pour une opération, cette dernière peut être effectuée sans blocage.

Eléments scrutables

Tout élément d'un système, manipulable via un descripteur de fichier : socket, tube, entrée standard, sortie standard, fichier, etc.

Objectifs

En général

- Un moyen d'éviter une situation d'interblocage en bloquant sur une opération parmi d'autres opérations possibles.
- Un moyen d'éviter de modifier le comportement bloquant d'opérations d'entrée/sortie (via les options de send/sendto/recv/recvfrom ...), imposant souvent de réaliser des boucles en attente active.

Objectifs

En général

- Un moyen d'éviter une situation d'interblocage en bloquant sur une opération parmi d'autres opérations possibles.
- Un moyen d'éviter de modifier le comportement bloquant d'opérations d'entrée/sortie (via les options de send/sendto/recv/recvfrom ...), imposant souvent de réaliser des boucles en attente active.

Dans ce cours

Développer un serveur capable de gérer plusieurs clients simultanément sans devoir créer plusieurs processus.

Objectifs

En général

- Un moyen d'éviter une situation d'interblocage en bloquant sur une opération parmi d'autres opérations possibles.
- Un moyen d'éviter de modifier le comportement bloquant d'opérations d'entrée/sortie (via les options de send/sendto/recv/recvfrom ...), imposant souvent de réaliser des boucles en attente active.

Dans ce cours

Développer un serveur capable de gérer plusieurs clients simultanément sans devoir créer plusieurs processus.

En particulier

- Scruter les arrivées de messages sur des sockets créées par un serveur.
- Vous pouvez ensuite le faire au besoin dans un programme client et/ou pour scruter d'autres éléments que les sockets.

En pratique

1. Définir l'ensemble des sockets à scruter en précisant :
   - quelles sockets sont à scruter en entrée (prêt en lecture suite à l'arrivée d'un message)
   - quelles sockets sont à scruter en écriture (prêt en écriture - quand ?)
2. Scruter les événement définis (opération bloquante).
3. Au réveil, vérifier pour chaque socket si un événement s'est produit. Si oui, appeler l'opération associée (exemple : recv pour lire un message reçu).

Ensemble de descripteurs de fichier à scruter

Le type fd_set

fd_set définit un ensemble de descripteurs de fichiers. Il s'agit d'un tableau de booléens où l'indice est la valeur du descripteur. Un élément du tableau qui est à *vrai*, signifie que le descripteur associé (valeur de son indice) est à prendre en compte dans la scrutation.

Ensemble de descripteurs de fichier à scruter

Le type fd_set

fd_set définit un ensemble de descripteurs de fichiers. Il s'agit d'un tableau de booléens où l'indice est la valeur du descripteur. Un élément du tableau qui est à *vrai*, signifie que le descripteur associé (valeur de son indice) est à prendre en compte dans la scrutation.

Exemple

Supposons qu'un programme serveur a créé 4 sockets dont les descripteurs de fichier sont 5, 8, 10 et 11. L'ensemble *fd_set* suivant :

0	0	0	0	0	0	0	0	1	0	1	0	...
0	1	2	3	4	5	6	7	8	9	10	11	...

indique que seules les sockets avec les descripteurs 8 et 10 sont à scruter

Ensemble de descripteurs de fichier à scruter

Le type fd_set

fd_set définit un ensemble de descripteurs de fichiers. Il s'agit d'un tableau de booléens où l'indice est la valeur du descripteur. Un élément du tableau qui est à *vrai*, signifie que le descripteur associé (valeur de son indice) est à prendre en compte dans la scrutation.

Exemple

Supposons qu'un programme serveur a créé 4 sockets dont les descripteurs de fichier sont 5, 8, 10 et 11. L'ensemble *fd_set* suivant :

0	0	0	0	0	0	0	0	1	0	1	0	...
0	1	2	3	4	5	6	7	8	9	10	11	...

indique que seules les sockets avec les descripteurs 8 et 10 sont à scruter

Note

La taille d'un ensemble *fd_set* est défini par la constante FD_SETSIZE.

Opérations sur un ensemble fd_set

```
void FD_ZERO(fd_set *set);
```

Description

Initialise à faux les éléments de l'ensemble pointé par *set*.

Opérations sur un ensemble fd_set

```
void FD_ZERO(fd_set *set);
```

Description

Initialise à faux les éléments de l'ensemble pointé par *set*.

```
void FD_SET(int desc, fd_set *set);
```

Description

Ajoute le descripteur *desc* à la liste des descripteurs de **set* à scruter, i.e. positionne l'élément à l'indice *desc* à *vrai*.

Opérations sur un ensemble fd_set -suite

```
int FD_ISSET(int desc, fd_set *set);
```

Description

Teste si le descripteur *desc* est dans la liste des descripteurs de **set* à scruter ou scrutés, i.e. si l'élément à l'indice *desc* est à *vrai*.

Opérations sur un ensemble fd_set -suite

```
int FD_ISSET(int desc, fd_set *set);
```

Description

Teste si le descripteur *desc* est dans la liste des descripteurs de **set* à scruter ou scrutés, i.e. si l'élément à l'indice *desc* est à *vrai*.

```
void FD_CLR(int desc, fd_set *set);
```

Description

Supprime le descripteur *desc* de la liste des descripteurs de **set* à scruter, i.e. positionne l'élément à l'indice *desc* à *faux*.

Scruter des événements

```
int select(int nb_desc,                  // descripteur max à scruter +1
        fd_set *readDescs,               // descripteurs à scruter en entrée
        fd_set *writeDescs,              // descripteurs à scruter en sortie
        fd_set *exceptDescs,             // pour scruter des exceptions
        struct timeval *timeout)         // délais d'attente ou NULL
```

Scruter des événements

```
int select(int nb_desc,                 // descripteur max à scruter +1
           fd_set *readDescs,           // descripteurs à scruter en entrée
           fd_set *writeDescs,          // descripteurs à scruter en sortie
           fd_set *exceptDescs,         // pour scruter des exceptions
           struct timeval *timeout)     // délais d'attente ou NULL
```

Description

La fonction se met en attente jusqu'à ce qu'un événement se produise sur au moins un descripteur scruté. Dans ce cas, elle modifie les ensembles passés en paramètres pour ne garder que les descripteurs passés à l'état "prêt" et retirer les autres. Il devient possible d'appeler sans blocage

- des fonctions de lecture sur les descripteurs prêts dans *readDescs*,
- des fonctions d'écriture sur les descripteurs prêts dans *writeDescs*,
- des fonctions de capture d'exceptions sur les descripteurs prêts dans *exceptDescs*.

Scruter des événements - suite

```
int select(int nb_desc,                  // descripteur max à scruter +1
           fd_set *readDescs,            // descripteurs à scruter en entrée
           fd_set *writeDescs,           // descripteurs à scruter en sortie
           fd_set *exceptDescs,          // pour scruter des exceptions
           struct timeval *timeout)      // délais d'attente ou NULL
```

Délais d'attente

Si le paramètre *timeout* est NULL, la fonction attend jusqu'à l'occurrence d'un événement, sinon, un délais doit être défini :

struct timeval{ long tv_sec ; // secondes
long tv_usec ; } // microsecondes

Scruter des événements - suite

```
int select(int nb_desc,                 // descripteur max à scruter +1
           fd_set *readDescs,           // descripteurs à scruter en entrée
           fd_set *writeDescs,          // descripteurs à scruter en sortie
           fd_set *exceptDescs,         // pour scruter des exceptions
           struct timeval *timeout)     // délais d'attente ou NULL
```

Délais d'attente

Si le paramètre *timeout* est NULL, la fonction attend jusqu'à l'occurrence d'un événement, sinon, un délais doit être défini :

struct timeval{ long tv_sec ; // secondes
long tv_usec ; } // microsecondes

Valeur de retour

Le nombre de descripteurs scrutés et passés à l'état "prêt", 0 si le temps d'attente s'est écoulé sans occurrence d'un événement, -1 en cas d'erreur.

On reprend tout : exemple d'un serveur

Un serveur qui reçoit, en boucle, des requêtes de clients et retourne des réponses.

Serveur d'origine - iteratif (extrait)

```
dS= socket(PF_INET, SOCK_STREAM, 0) ;
struct sockaddr_in ad ;
ad.sin_family = AF_INET ;
ad.sin_addr.s_addr = INADDR_ANY ;
ad.sin_port = htons(atoi(argv[1])) ;
bind(dS, (struct sockaddr*)&ad, sizeof(ad)) ;
listen(dS, 7) ;
while(1){
    dSC= accept(dS, NULL, NULL) ;
    struct requete req ; // supposée fournie
    recv(dSC, &req, sizeof(req), 0) ;
    int rep = taiter(req) ; // supposée fournie
    send(dSC, &rep, sizeof(int), 0) ;
    close (dSC) ;
}
close (dS) ;
```

On reprend tout : exemple d'un serveur - multiplexage

```
... // socket(...), bind(...), listen(...)
fd_set set, settmp ;
FD_ZERO(&set) ; FD_SET(dS, &set) ;
int max = dS ;
while(1){
    settmp = set ;
    select(max+1, &settmp, NULL, NULL, NULL) ;
    for(int df= 2 ; df <= max ; df++){
        if ( !FD_ISSET(df, &settmp)) continue ;
        if (df == dS) {
            dSC= accept(dS, NULL, NULL) ;
            FD_SET(dSC, &set) ;
            if (max < dSC) max = dSC ;
            continue ;
        }
        struct requete req ;
        if (recv(df, &req, sizeof(req), 0) <= 0) {
            FD_CLR(df, &set) ; close(df) ; continue ;
        }
        int rep = taiter(req) ;
        send(df, &rep, sizeof(int), 0) ;
    }
} ...
```

Remarques sur l'exemple

- Toutes les sockets ont été prises en compte, y compris la socket d'écoute des demandes de connexion.

Remarques sur l'exemple

- Toutes les sockets ont été prises en compte, y compris la socket d'écoute des demandes de connexion.
- L'exemple est simplifié, il n'est qu'illustratif de l'utilisation des concepts vu.

Remarques sur l'exemple

- Toutes les sockets ont été prises en compte, y compris la socket d'écoute des demandes de connexion.
- L'exemple est simplifié, il n'est qu'illustratif de l'utilisation des concepts vu.
- Il manque principalement :

Remarques sur l'exemple

- Toutes les sockets ont été prises en compte, y compris la socket d'écoute des demandes de connexion.
- L'exemple est simplifié, il n'est qu'illustratif de l'utilisation des concepts vu.
- Il manque principalement :
  - le traitement des valeurs de retours des fonctions, au cas par cas,
  - la gestion des erreurs,
  - la prise en compte des écritures dans la scrutation (send),
  - éventuellement, la mise à jour de la valeur max des descripteurs (à quoi cela servirait ?)

Pour finir

- Le multiplexage peut se faire en utilisant aussi la fonction : poll(...).

Pour finir

- Le multiplexage peut se faire en utilisant aussi la fonction : poll(...).
- Penser à utiliser le multiplexage en incluant l'entrée standard dans les descripteurs à scruter, en particulier si votre programme peut réaliser des saisies et des réceptions en parallèle.

Pour finir

- Le multiplexage peut se faire en utilisant aussi la fonction : poll(...).
- Penser à utiliser le multiplexage en incluant l'entrée standard dans les descripteurs à scruter, en particulier si votre programme peut réaliser des saisies et des réceptions en parallèle.
- Toujours tester les valeurs de retour et quitter proprement vos programmes (libérations des espaces alloués dynamiquement, fermer les sockets, etc).

Pour finir

- Le multiplexage peut se faire en utilisant aussi la fonction : poll(...).
- Penser à utiliser le multiplexage en incluant l'entrée standard dans les descripteurs à scruter, en particulier si votre programme peut réaliser des saisies et des réceptions en parallèle.
- Toujours tester les valeurs de retour et quitter proprement vos programmes (libérations des espaces alloués dynamiquement, fermer les sockets, etc).
- Favoriser le passage de paramètres à vos programmes (ou la saisie) et non le codage en dur des données supposées être modifiables.

Pour finir

- Le multiplexage peut se faire en utilisant aussi la fonction : poll(...).
- Penser à utiliser le multiplexage en incluant l'entrée standard dans les descripteurs à scruter, en particulier si votre programme peut réaliser des saisies et des réceptions en parallèle.
- Toujours tester les valeurs de retour et quitter proprement vos programmes (libérations des espaces alloués dynamiquement, fermer les sockets, etc).
- Favoriser le passage de paramètres à vos programmes (ou la saisie) et non le codage en dur des données supposées être modifiables.
- Favoriser l'allocation d'un numéro de port par le système et utiliser les fonctions getnameinfo(...) pour connaitre le numéro alloué.

Pour finir

- Le multiplexage peut se faire en utilisant aussi la fonction : poll(...).
- Penser à utiliser le multiplexage en incluant l'entrée standard dans les descripteurs à scruter, en particulier si votre programme peut réaliser des saisies et des réceptions en parallèle.
- Toujours tester les valeurs de retour et quitter proprement vos programmes (libérations des espaces alloués dynamiquement, fermer les sockets, etc).
- Favoriser le passage de paramètres à vos programmes (ou la saisie) et non le codage en dur des données supposées être modifiables.
- Favoriser l'allocation d'un numéro de port par le système et utiliser les fonctions getnameinfo(...) pour connaitre le numéro alloué.
- Tester vos applications en réseau (exécuter les clients et les serveurs sur des machines différentes).