UNIVERSITÉ PARIS SUD

IUT D'ORSAY

M3101 · Principes des systèmes d'exploitation

Sockets (fin) – Multiplexage

IUT d'Orsay · DUT Informatique · 2015 / 2016

Serveur UDP — uniprocessus

```
1  int sserveur = socket(AF_INET, SOCK_DGRAM, 0);
2  struct sockaddr_in saddr = {0}, caddr = {0};
3  saddr.sin_family = AF_INET;          // domaine IPv4
4  saddr.sin_port = htons(PORT);        // port en format réseau
5  saddr.sin_addr.s_addr = htonl(INADDR_ANY);  // toute adresse
6  bind(sserveur, (struct sockaddr*) &saddr, sizeof(saddr));
7  while (1) {                        // traiter les requêtes entrantes
8    int calen = sizeof(caddr); // préparer calen pour recvfrom()
9    int nbLus = recvfrom(sserveur, &requete, sizeof(requete),
10                         0, (struct sockaddr*) &caddr, &calen);
11   if (nbLus == 0) continue;  // un datagramme vide
12        ...                   // traiter la requête
13   sendto(sserveur, &reponse, sizeof(reponse),
14                         0, (struct sockaddr*) &caddr, calen); }
15 close(sserveur);
```

Serveur TCP — multiprocessus

```
1  int secoute = socket(AF_INET, SOCK_STREAM, 0);
2       ...              // préparer la structure d'adresse
3  bind(secoute,(struct sockaddr*) &saddr,sizeof(saddr));
4  listen(secoute,5);  // passer la socket en mode écoute
5  while (1) {         // traiter les demandes de connexion
6    int calen = sizeof(caddr); // préparer calen pour accept()
7    sservice = accept(secoute,(struct sockaddr*) &caddr,&calen);
8    if (fork() > 0) { close(sservice); continue; /* goto 5 */}
9    while (1) { // le code du processus dédié à la connexion
10     int nbLus = read(sservice, &requete, sizeof(requete));
11     if (nbLus <= 0) break;   // fin de connexion
12        ...                   // traiter la requête
13     write(sservice, &reponse, sizeof(reponse)); }
14   shutdown(sservice, SHUT_RDWR);
15   close(sservice);
16   exit(0); }
17 close(secoute);
```

Architecture du serveur

Multiprocessus est bien adaptée aux applications
où les connexions client sont indépendantes

- les fils héritent la mémoire du père mais ne la partagent pas

Si les connexions client sont interdépendantes
les processus dédiés doivent échanger des données

- tubes, sockets Unix, mémoire partagée, queues de messages...

Q : Peut-on gérer toutes les connexions dans un seul processus serveur ?

R : Oui, mais on est obligé à suivre plusieurs canaux en même temps

- socket d'écoute, sockets de service, tubes, entrée standard...

Solution : multiplexage

Ensembles de descripteurs

Type `fd_set` — ensemble de descripteurs de fichiers

Peut stocker les descripteurs entre 0 et `FD_SETSIZE` − 1

Quatre opérations :

```
void FD_ZERO(fd_set *set);           /* vide set */
void FD_SET(int fd, fd_set *set);    /* ajoute fd à set */
void FD_CLR(int fd, fd_set *set);    /* retire fd de set */
int  FD_ISSET(int fd, fd_set *set); /* vrai si fd dans set */
```

Implémentés comme tableaux de bits, une case par descripteur

Sous Linux : `FD_SETSIZE` = 1024, **`sizeof`**`(fd_set)` = 128 octets

```
int select(
  int maxfd,                  /* n° maximum de descripteur + 1 */
  fd_set *readfds,            /* descripteurs en lecture */
  fd_set *writefds,           /* descripteurs en écriture */
  fd_set *exceptfds,          /* condition exceptionnelle */
  struct timeval *delay); /* attente maximum ou NULL */
```

Se met en attente jusqu'à ce que au moins un descripteur soit prêt :

- dans `readfds` — prêt pour lecture : `read()` ne bloquera pas
- dans `writefds` — prêt pour écriture : `write()` ne bloquera pas
- dans `exceptfds` — une exception est arrivée (p.e. données OOB)

Garde dans chaque ensemble les descripteurs prêts, retire les autres

Renvoie le nombre de descripteurs prêts,
0 si le temps d'attente s'écoule, ou -1 en cas d'erreur

Exemple : tester sans bloquer

```
1  int is_ready(int fd) {
2    fd_set ensemble;          // déclarer l'ensemble
3    struct timeval delai;     // déclarer le délai
4
5    FD_ZERO(&ensemble);       // vider l'ensemble
6    FD_SET(fd, &ensemble);    // ajouter le descripteur
7    delai.tv_sec  = 0;        // zéro seconde...
8    delai.tv_usec = 0;        // ...et zéro micro-seconde
9
10   // Si select() a échoué, on renvoie -1.
11   if (select(fd+1, &ensemble, NULL, NULL, &delai) < 0)
12     return -1;
13
14   // Si fd est dans le premier ensemble de retour
15   // de select(), c'est qu'il est prêt en lecture.
16   return (FD_ISSET(fd, &ensemble) ? 1 : 0); }
```

Exemple : tester sans bloquer

```
1  int is_ready(int fd) {
2    fd_set ensemble;          // déclarer l'ensemble
3    struct timeval delai;     // déclarer le délai
4
5    FD_ZERO(&ensemble);       // vider l'ensemble
6    FD_SET(fd, &ensemble);    // ajouter le descripteur
7    delai.tv_sec  = 0;        // zéro seconde...
8    delai.tv_usec = 0;        // ...et zéro micro-seconde
9
10   // Si select() renvoie 1, c'est que fd est prêt.
11   return select(fd+1, &ensemble, NULL, NULL, &delai); }
```

Exemple : un simple netcat

```
 1  fd_set ensemble, temp;          // déclarer les ensembles
 2  char message[BUFFER_SIZE];      // déclarer le tampon
 3  int sclient;                    // déclarer la socket
 4       ...                        // établir une connexion
 5  FD_ZERO(&ensemble);             // vider l'ensemble principal
 6  FD_SET(0,&ensemble);            // ajouter l'entrée standard
 7  FD_SET(sclient,&ensemble);      // ajouter la socket
 8  while (1) {
 9    temp = ensemble;              // copier l'ensemble
10    select(sclient+1, &temp, NULL, NULL, NULL);
11    if (FD_ISSET(0, &temp)) {  // l'entrée standard est prête
12      lus = read(0, message, BUFFER_SIZE);
13      if (lus > 0) write(sclient, message, lus); else break; }
14    if (FD_ISSET(sclient, &temp)) {  // la socket est prête
15      lus = read(sclient, message, BUFFER_SIZE);
16      if (lus > 0) write(1, message, lus); else break; } }
17  shutdown(sclient, SHUT_RDWR);  // fermer la connexion
18  close(sclient);                  // fermer la socket
```

Exemple : un simple echo

```
1       ...                          // socket(), bind(), listen()
2   FD_ZERO(&ensemble);              // vider l'ensemble principal
3   FD_SET(secoute,&ensemble);       // ajouter la socket d'écoute
4   int max = secoute;               // n° maximum de descripteur
5   while (1) {
6     temp = ensemble;               // copier l'ensemble
7     select(max+1, &temp, NULL, NULL, NULL);
8     for (int fd = 0; fd <= max; fd++) {
9       if (!FD_ISSET(fd, &temp)) continue;  // fd pas prêt
10      if (fd == secoute) {           // demande de connexion
11        int ss = accept(secoute, NULL, NULL);
12        FD_SET(ss, &ensemble);       // ajouter socket de service
13        if (ss > max) max = ss;      // mettre max à jour
14        continue;                    /* au suivant */ }
15      if ((nbLus = read(fd, message, BUFFER_SIZE)) <= 0) {
16        FD_CLR(fd, &ensemble);       // retirer socket de service
17        shutdown(fd, SHUT_RDWR);     // fermer la connexion
18        close(fd); continue;         /* au suivant */ }
19      write(fd, message, nbLus);     /* réponse */ } }
```