C

Table des matières

1 Boîte à outils

`nm file.o` : list symbols from object files => printf("%x", fonctions et variables globales) `ldd binary` : print shared library dependencies `ldconfig -p` : show cache of shared libraries used by 'ld -lname'

2 Temps-réel mou

Sous linux, 3 ordonnancements co-existent.

- OTHER : préemptif avec priorité dynamique (nice)
- RR : temps réel souple (priorité statique) mais préemptif à priorité égale
- FIFO : temps réel souple non préhemptif (comme windows 95)

S'il existe un ou plusieurs processus FIFO ou RR, ils sont séléctionnés en premier. Lors de la phase de débugage, il est important de conserver un shell s'exécutant à un niveau de priorité supérieur. Attention lors du fork, il faut placer la modification de priorité dans le code du processus père.

3 Thread

Les threads sont des processus allégés ne réclamant que peu de ressources pour les changements de contexte. Chaque thread dispose personnellement d'une pile et d'un contexte d'exécution contenant les registres du processeur et un compteur d'instruction. En revanche, ils se partagent les données statiques et dynamiques.

Sous LINUX, l'implémentation usuelle des threads est effectuée dans l'espace noyau.

Comme pour les processus, on peut modifier l'ordonnancement des threads. Les ordonnancements temps-réel nécessitent un UID effectif nul, sinon la fonction `pthread_create` échouera avec l'erreur EPERM.

4 Sockets

Il s'agit approximativement d'une extension des tubes nommés. Ici nous nous situerons dans le contexte des sockets en mode connecté.

Voici une petite application qui mesure la vitesse de connexion mettant en pratique l'ensemble des principes décrits ci-dessous.

4.1 Réseau

4.1.1 Ordre des octets du réseau

L'ordre des octets de référence sur les réseau réseaux IP est dit *Big Endian*. (l'octet de poid fort est rangé en premier : 1234 est représenté 12 34). Toute valeur numérique **entière** transmise au protocole réseau devront passer par une étape de conversion. Les 2 premières valeures concernées sont l'adresse IP et le numéreau de port.

Inclure le fichier */usr/include/netinet/in.h*.

```
/* Internet address.  */
typedef uint32_t in_addr_t;
struct in_addr
  {
    in_addr_t s_addr;
  };
```

4.1.2 Protocole

Interrogation du fichier */etc/protocoles*. Inclure le fichier */usr/include/netdb.h* :

```
/* Description of data base entry for a single service.  */
struct protoent
{
  char *p_name;                 /* Official protocol name.  */
  char **p_aliases;             /* Alias list.  */
  int p_proto;                  /* Protocol number.  */
};
```

Cf fichier */usr/include/netinet/in.h* à propos du troisième champ.

```
/* Standard well-defined IP protocols.  */
enum
  {
...
    IPPROTO_TCP = 6,       /* Transmission Control Protocol.  */
#define IPPROTO_TCP             IPPROTO_TCP
...
```

```
    IPPROTO_UDP = 17,      /* User Datagram Protocol.  */
#define IPPROTO_UDP             IPPROTO_UDP
...
  };
```

4.1.3 Service ou port

Interrogation du fichier */etc/services*.
Inclure le fichier */usr/include/netdb.h* :

```
/* Description of data base entry for a single service.  */
struct servent
{
  char *s_name;                 /* Official service name.  */
  char **s_aliases;             /* Alias list.  */
  int s_port;                   /* Port number.  */
  char *s_proto;                /* Protocol to use.  */
};
```

4.1.4 Nom d'hôte

Interrogation du serveur de nom qui supplante le fichier */etc/host*.
Inclure le fichier */usr/include/netdb.h* :

```
/* Description of data base entry for a single host.  */
struct hostent
{
  char *h_name;                 /* Official name of host.  */
  char **h_aliases;             /* Alias list.  */
  int h_addrtype;               /* Host address type.  */
  int h_length;                 /* Length of address.  */
  char **h_addr_list;           /* List of addresses from name server.  */
#define h_addr  h_addr_list[0]  /* Address, for backward compatibility.  */
};
```

4.1.5 Socket

Les 3 structures pécédentes ont pour but d'aider à remplir la structure d'adressage du socket. Cependant on peut tout aussi bien s'en passer.

La structure générique suivante est inclue via le fichier */usr/include/sys/socket.h*.

```
/* POSIX.1g specifies this type name for the 'sa_family' member.  */
typedef unsigned short int sa_family_t;

/* This macro is used to declare the initial common members
   of the data types used for socket addresses, 'struct sockaddr',
   'struct sockaddr_in', 'struct sockaddr_un', etc.  */

#define __SOCKADDR_COMMON(sa_prefix) \
  sa_family_t sa_prefix##family

/* Structure describing a generic socket address.  */
struct sockaddr
  {
    __SOCKADDR_COMMON (sa_);    /* Common data: address family and length.  */
    char sa_data[14];           /* Address data.  */
```

```
};
```

La structure spécifique à la famille AF_INET est définie dans le fichier */usr/include/netinet/in.h*.

```
/* Structure describing an Internet socket address.  */
struct sockaddr_in
  {
    __SOCKADDR_COMMON (sin_);
    in_port_t sin_port;                 /* Port number.  */
    struct in_addr sin_addr;            /* Internet address.  */

    /* Pad to size of 'struct sockaddr'.  */
    unsigned char sin_zero[sizeof (struct sockaddr) -
                           __SOCKADDR_COMMON_SIZE -
                           sizeof (in_port_t) -
                           sizeof (struct in_addr)];
  };
```

4.2 Sockets côté serveur

4.2.1 Création

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

- domaine : AF_INET, AF_INETG, AF_UNIX
- type : SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
- protocole : IPROTO_TCP, IPROTO_UDP, ...
  Il s'agit du champ `p_proto` de la structure `protoent`. On peut indiquer la valeur nulle car ce champs habituellement redondant avec le champs précédent.

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

int main(){
  int sd;
  if ((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("socket");
    return -1;
  }
}
```

4.2.2 Affectation d'adresse

*int bind (int sock, struct sockaddr *adresse, socklen_t longueur) ;*

- descripteur de la socket
- adresse : pointeur `sockaddr_in*` converti en `sockaddr*`
- taille de la structure spécifique pointée ci-dessus.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h> /* pour memset */

#define _HOST "127.0.0.1"
#define _PORT 1024
```

```
int main(){
  int sd;
  struct sockaddr_in adresse;

  if ((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("socket");
    return -1;
  }

  memset(&adresse, 0, sizeof(struct sockaddr_in));
  adresse.sin_family = AF_INET;
  adresse.sin_port = htons(_PORT);
  inet_aton(_HOST, & adresse.sin_addr);

  if (bind(sd, (struct sockaddr*) &adresse, sizeof(struct sockaddr_in)) < 0) {
    perror("bind");
    return -1;
  }
}
```

4.2.3 Attente de connexion

Le principe de l'appel-système `accept()` est de prendre une demande de connexion en attente - dans la file dimensionnée avec `listen()` - puis d'ouvrir une nouvelle socket du côté serveur et d'établir la connexion sur celle-ci. La socket original reste donc intacte. On pourra invoquer `fork()` au retout de `accept()` afin de dialoguer avec plusieurs clients simultanément.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h> /* pour memset */

#define _HOST "127.0.0.1"
#define _PORT 1024

int main(){
  int sd;  /* standard */
  int sd2; /* bureau */
  struct sockaddr_in adresse;
  struct sockaddr_in adresseClient;
  socklen_t longueur = sizeof(struct sockaddr_in);
  char buff[2];

  if ((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("socket");
    return -1;
  }

  memset(&adresse, 0, sizeof(struct sockaddr_in));
  adresse.sin_family = AF_INET;
  adresse.sin_port = htons(_PORT);

  /* we decided to not use IP adress on server side (comment on of the 2 lines below) */
  /* inet_aton(_HOST, &adresse.sin_addr); */
```

```
  adresse.sin_addr.s_addr = htonl(INADDR_ANY);

  if (bind(sd, (struct sockaddr*) &adresse, longueur) < 0) {
    perror("bind");
    return -1;
  }

  /* only 1 socket should be allow to wait for server */
  if (listen(sd, 1) < 0) { /* but more can wait */
    perror("listen");
    return -1;
  }

  while(1)
    {
      if ((sd2 = accept(sd, (struct sockaddr*) &adresseClient, &longueur)) < 0) {
  perror("listen");
  return -1;
}
      write(sd2, "au revoir", 9);
      read(sd2, buff, 1);
      close(sd2);
    }
}
```

4.2.4 Finir le flux d'octets

Comme nous envoyons des flux d'octets, dont la longueur est arbitraire (donc le serveur ne peut pas determiner leur fin), une fois que toutes les informations ont été envoyées, nous fermons le côté écriture de la socket avec `shutdown()`. Sachant que toutes les informations sont arrivées, le serveur peut les traiter et nous envoyer la réponse.

4.3 Socket côté client

4.3.1 Création

Il s'agit du même paragraphe que ci-dessus, côté serveur.
int socket (int domaine, int type, int protocole) ;

- domaine : AF_INET, AF_INETG, AF_UNIX
- type : SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
- protocole : IPROTO_TCP, IPROTO_UDP, ...
  Il s'agit du champ `p_proto` de la structure `protoent`. On peut indiquer la valeur nulle car ce champs habituellement redondant avec le champs précédent.

4.3.2 Connexion

Il s'agit des même arguments que pour l'affectation d'adresse, côté serveur.
*int connect (int sockfd, struct sockaddr *serv_addr, socklen_t addrlen) ;*

- descripteur de la socket
- adresse : pointeur `sockaddr_in*` converti en `sockaddr*`
- taille de la structure spécifique pointée ci-dessus.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h> /* pour memset */
#include <stdio.h>

#define _HOST "127.0.0.1"
#define _PORT 1024

int main(){
  int sd;
  struct sockaddr_in adresse;
  char buffer[256];
  int nbLus = 0;

  if ((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("socket");
    return -1;
  }

  memset(&adresse, 0, sizeof(struct sockaddr_in));
  adresse.sin_family = AF_INET;
  adresse.sin_port = htons(_PORT);
  inet_aton(_HOST, & adresse.sin_addr);

  if (connect(sd, (struct sockaddr*) &adresse, sizeof(struct sockaddr_in)) < 0) {
    perror("connect");
    return -1;
  }

  if ((nbLus = read(sd, buffer, 256)) < 0) {
    perror("read");
    return -1;
  }

  buffer[nbLus] = '\0';
  printf("%s\n", buffer);
}
```

4.4 Performances

- Disabling the Nagle algorithm for a TCP socket

```
#include <netinet/tcp.h> /* for TCP_NODELAY */

int flag = 1;
if (setsockopt( sd, IPPROTO_TCP, TCP_NODELAY, (char *)&flag, sizeof(flag) ) == -1)
{
  printf("Couldn't setsockopt(TCP_NODELAY)\n");
  return rc;
}
```

5 ioctl

Try `man 2 ioctl_list` and have a look here : */usr/include/linux/cdrom.h*. Above from AUTORUN project from Harald Hoyer :

```
/* eject.c
** Copyright Paul Dwerryhouse, 1997-2004
*/
#include <sys/types.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <linux/cdrom.h>
#include <stdlib.h> // exit
#include <limits.h> // INT_MAX
#include <stdio.h> // printf

#define CDDEVICE "/dev/cdrom"           /* CDROM device */

typedef enum {
  tray_is_open,
  unknown,
  unmounted,
  mounted,
  audio
} cdstatus_t;

typedef enum {
  t_tray_is_open,
  t_unknown,
  t_audio,
  t_data,
  t_mixed
} cdtype_t;

cdstatus_t cdstatus;
cdtype_t cdtype;
char* devpath;
char* mountpath;

int main(int argc,char **argv)
{
   int cdrom;                           /* CDROM device file descriptor */
   int rc;

   /* Open the CDROM device. The linux/cdrom.h header file specifies that
   ** it must be given the O_NONBLOCK flag when opening. My tests showed
   ** that if this isn't done, this program will not work.
   */
   if ((cdrom = open(CDDEVICE,O_RDONLY | O_NONBLOCK)) < 0) {
        perror("open");
        exit(1);
   }

   /* Use ioctl to send the CDROMEJECT command to the device
   if (ioctl(cdrom,CDROMEJECT,0)<0) {
        perror("ioctl CDROMEJECT");
        exit(1);
   }
   */
```

```
if ((rc = ioctl(cdrom, CDROM_DRIVE_STATUS, CDSL_CURRENT))<0) {
  perror("ioctl CDROM_DRIVE_STATUS");
  exit(1);
}

switch(rc) {
case CDS_DISC_OK:
  printf("CDS_DISC_OK\n");
  break;
case CDS_TRAY_OPEN:
  printf("CDS_TRAY_OPEN\n");
  break;
case CDS_NO_DISC:
  printf("CDS_NO_DISC\n");
  break;
case CDS_NO_INFO:
  printf("CDS_NO_INFO\n");
  break;
case CDS_DRIVE_NOT_READY:
  printf("CDS_DRIVE_NOT_READY\n");
  break;
default:
  printf("default\n");
}

if ((rc = ioctl(cdrom, CDROM_MEDIA_CHANGED, CDSL_CURRENT))<0) {
  perror("ioctl CDROM_MEDIA_CHANGED");
  exit(1);
}
printf("media change: %i\n", rc);

if ((rc = ioctl(cdrom, CDROM_DISC_STATUS, CDSL_CURRENT))<0) {
  perror("ioctl CDROM_DISC_STATUS");
  exit(1);
}

switch(rc) {
case CDS_DATA_1:
  printf("CDS_DATA_1\n");
  break;
case CDS_DATA_2:
  printf("CDS_DATA_2\n");
  break;
case CDS_AUDIO:
  printf("CDS_AUDIO\n");
  break;
case CDS_MIXED:
  printf("CDS_MIXED\n");
  break;
case CDS_NO_INFO:
  printf("CDS_NO_INFO\n");
  break;
case CDS_NO_DISC:
  printf("CDS_NO_DISC\n");
  break;
default:
  printf("default\n");
```

```
    }

    close(cdrom);
}
```