

Cours

Systeme et Réseaux

Majeure Informatique
Ecole Polytechnique

7

Applications client/serveur

François Bourdoncle

`Francois.Bourdoncle@ensmp.fr`

`http://www.ensmp.fr/~bourdonc/`

Plan

- Les sockets d'Unix BSD
- Utilisation des sockets
- Client X-Windows
- Serveur X-Window
- Serveur HTTP
- Datagrammes

Adresses réseau (sockets)

- Domaine `AF_UNIX` : nom de fichier

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

```
struct sockaddr_un {
    short sun_family;
    char  sun_path[104];
};
```

- Domaine `AF_INET` : adresse IP + numéro de port

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
```

```
struct in_addr {
    unsigned long s_addr;
};
```

```
struct sockaddr_in {
    short          sin_family;
    u_short        sin_port;
    struct in_addr sin_addr;
    char           sin_zero[8];
};
```

- Adresses de taille variable (passée en paramètre)

Les sockets d'Unix BSD

- Apparu sur Unix BSD4.2 (Leffler, etc.)
 - Transparence (local ou distant unifié)
 - Efficacité
 - Compatibilité (programmes existants)
- `socket` : crée un descripteur
 - Domaine (`AF_UNIX`, `AF_INET`)
 - Type (`SOCK_DGRAM`, `SOCK_STREAM`)
 - Protocole (0 en général)
- `bind` : associe le descripteur à un port (ports < 1024 réservés au super-utilisateur)
- `listen` : taille de la file d'attente
- `accept` : accepte les connections
- `connect` : connecte à un serveur

Utilisation des sockets (client)

```
fd = socket(domaine, type, protocole);  
  
connect(fd, adresse_serveur,  
         longueur_adresse_serveur);  
  
write(fd, commande, longueur_commande);  
longueur_resultat =  
    read(fd, resultat, max_longueur_resultat);  
  
close(fd);
```

Utilisation des sockets (serveur)

```
fds = socket(domaine, type, protocole);

bind(fds, adresse_serveur,
      longueur_adresse_serveur);

listen(fds, max_clients);

for (;;) {
    int fdc = accept(fds, adresse_client,
                    longueur_adresse_client);

    longueur_commande =
        read(fdc, commande,
            max_longueur_commande);

    resultat = Traitement(commande);

    write(fdc, resultat, longueur_resultat);
    close(fdc);
}

close(fds);
```

Client X-Windows (DISPLAY=unix :0)

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

Display* XOpenDisplay (int disp)
{
    struct sockaddr_un unaddr;
    struct sockaddr* addr;
    int addrlen;
    Display dpy = (Display*) malloc(sizeof(Display));
    ...
    unaddr.sun_family = AF_UNIX;
    sprintf(unaddr.sun_path,
            "/tmp/.X11-unix/X%d",
            disp);

    addr = (struct sockaddr*) &unaddr;
    addrlen = sizeof(unaddr);

    if ((fd = socket(((int) addr->sa_family,
                    SOCK_STREAM, 0)) < 0) {
        return -1;
    }
    if (connect(fd, addr, addrlen) < 0) {
        int olderrno = errno;
        (void) close (fd);
        errno = olderrno;
        return -1;
    }
    ...
    dpy->fd = fd;
    return dpy;
}
```

Serveur X-Windows (DISPLAY=unix :0)

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

int OpenSocket(int disp)
{
    struct sockaddr_un unaddr;
    int mask;
    int fd;

    unaddr.sun_family = AF_UNIX;
    mask = umask(0);
    sprintf(unaddr.sun_path,
            "/tmp/.X11-unix/X%d", disp);
    unlink(unaddr.sun_path);
    if ((fd = socket(AF_UNIX, SOCK_STREAM, 0)) < 0) {
        return -1;
    }
    if (bind(fd, (struct sockaddr*) &unaddr,
            sizeof(unaddr)) != 0) {
        (void) close(fd);
        return -1;
    }
    if (listen(fd, 5) != 0) {
        (void) close(fd);
        return -1;
    }
    (void) umask(mask);
    return fd;
}
```


Serveur HTTP

```
main(int argc, argv[1])
{
    struct sockaddr_in addr;
    int lens = sizeof(addr);
    int fds;

    if (GetHostAddress(NULL, 8080, &addr) == 0 ||
        (fds = socket(AF_INET,
                     SOCK_STREAM, 0)) == -1 ||
        bind(fds, (struct sockaddr*) &addr, lens) != 0 ||
        listen(fds, 5) != 0) {
        return 1;
    }

    for (;;) {
        struct sockaddr_in addr;
        int lenc = sizeof(addr);
        int fdc =
            accept(fds, (struct sockaddr*) &addr, &lenc);

        if (fdc == -1) {
            if (errno != EINTR) {
                return 1;
            }
        } else {
            /* Traiter la commande. */
            ...
            /* Envoyer le resultat.
            ...
            /* Fermer la connection au client. */
            close(fdc);
        }
    }
}
```

Datagrammes

- Utilisé quand beaucoup de clients (`talk`)
- Pas de connection permanente
- Échanges épisodiques
 - `sendto(... adresse ... données ...)`
 - `recvfrom(... adresse ... données ...)`
- Protocole : UDP (User Datagram Protocol)

Datagrammes (serveur)

```
fds = socket(domaine, type, protocole);

bind(fds, adresse_serveur,
      longueur_adresse_serveur);

for (;;) {
    int n = recvfrom(fds,
                    requete,
                    longueur_requete,
                    options,
                    adresse_client,
                    longueur_adresse_client);
    reponse = Traitement(requete);
    sendto(fds,
           reponse,
           longueur_reponse,
           options,
           adresse_client,
           longueur_adresse_client);
}

close(fds);
```

Datagrammes (client)

```
fd = socket(domaine, type, protocole);

bind(fd, adresse_client,
      longueur_adresse_client);

sendto(fd,
        requete,
        longueur_requete,
        options,
        adresse_serveur,
        longueur_adresse_serveur);

n = recvfrom(fd,
             reponse,
             longueur_reponse,
             options,
             adresse_serveur,
             longueur_adresse_serveur);

close(fd);
```

Serveur en Java (1/2)

```
import java.io.*;
import java.net.*;
import java.util.*;

class Manager {
    ...
    static public add(Connection c) {
        ...
    }

    static public remove(Connection c) {
        ...
    }
}

public class Server {
    public static void main(String[] args)
        throws IOException {
        ServerSocket socket =
            new ServerSocket(Integer.parseInt(args[0]));

        for (;;) {
            try {
                Socket client = socket.accept();
                Connection c = new Connection(client);
                Manager.add(c);
                c.start();
            }
            catch (Exception e) {
                // ...
            }
        }
    }
}
```

Serveur en Java (2/2)

```
class Connection extends Thread {
    Socket client;

    public Connection(Socket client) {
        this.client = client;
    }

    public void run() {
        try {
            String line;
            InputStream is = client.getInputStream();
            OutputStream os = client.getOutputStream();
            BufferedReader in =
                new BufferedReader(new InputStreamReader(is));
            PrintWriter out =
                new PrintWriter(new OutputStreamWriter(os));
            out.println("HTTP/1.0 200 ");
            out.println("Content-Type: text/plain");
            out.println();

            while((line = in.readLine()) != null &&
                line.length() != 0) {
                out.println(line);
            }
            out.close();
            in.close();
        }
        catch (Exception e) {
            // ...
        }
        finally {
            Manager.remove(this);
        }
    }
}
```

Travaux dirigés

- Commande `geturl url`
(commande '`GET url`' sur le port 80 du serveur)
- Execution de processus a distance (`rsh`)
- Serveur proxy HTTP (un process par connection)
 - Serveur vis-à-vis du navigateur
 - Client vis-a-vis du serveur HTTP
- Serveur proxy HTTP multi-thread en Java