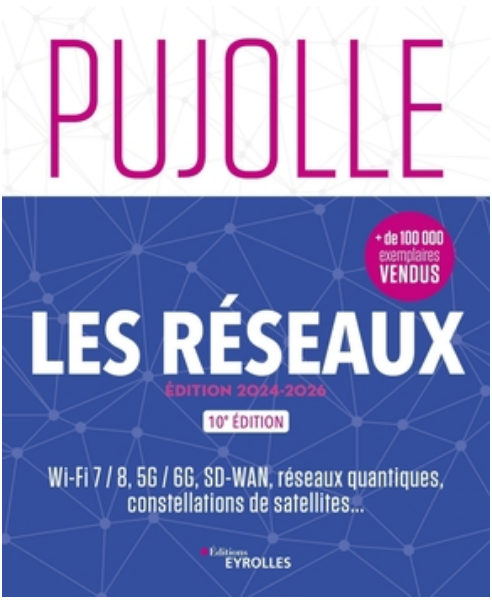
SYS1 - Network programming

Chaussette ! Chaussette !

Jules Aubert

Bibliography



HTMLography?

https://beej.us/guide/bgnet/

ipcalc

To do Network Programming, you first need to know Networking

Network address and service translation

```
man 3 getaddrinfo [!] # Manpage about the getaddrinfo function
man 3p getaddrinfo # Manpage about the POSIX getaddrinfo implementation (meh)
```

Ce n'est pas une chaussette

```
man 2 socket # Manpage about the socket syscall
man 7 socket # Manpage with miscellaneous information about the socket interface
man 3p socket # Manpage about the POSIX socket implementation
```

Going bind

This is for the server side

```
man 2 bind # Manpage about the bind syscall
man 3p bind # Manpage about the POSIX bind implementation
```

Listen here you little ****

This is for the server side

```
man 2 listen # Manpage about the listen syscall
man 3p listen # Manpage about the POSIX listen implementation
```

Accept who you are

This is for the server side

```
man 2 accept # Manpage about the accept syscall
man 3p accept # Manpage about the POSIX accept implementation
```

Signup to receive our daily newsletter

This is for the server side

```
man 2 recv [!] # Manpage about the recv syscall
man 3p recv # Manpage about the POSIX recv implementation
```

Connection? Connexion?

This is for the client side

```
man 2 connect # Manpage about the connect syscall
man 3p connect # Manpage about the POSIX connect implementation
```

Return to Sender (Elvis Presley, 1962)

This is for the client side

```
man 2 send [!] # Manpage about the send syscall
man 3p send # Manpage about the POSIX send implementation
```

Demo

Demo

Architecture

```
sh$ tree
.
+-- server
    +-- Makefile
    +-- server.c
```

Both - Headers and preprocessor

```
#include <err.h>
#include <netdb.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

CPPFLAGS = -D_POSIX_C_SOURCE=200112L
```

Server - Makefile

```
CPPFLAGS = -D_POSIX_C_SOURCE=200112L
CFLAGS = -std=c99 -pedantic -Wall -Wextra -Wvla -Werror
BIN = server

all: $(BIN)

clean:
    $(RM) $(BIN)
```

Server - Getting the network device

```c
int ret = 0;
struct addrinfo hints = { 0 };
struct addrinfo *res = NULL;
/* */

hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;

if ((ret = getaddrinfo("0.0.0.0", "27972", &hints, &res)) != 0)
    errx(EXIT_FAILURE, "getaddrinfo: %s", gai_strerror(ret));
```

Server - About getaddrinfo

Replace “0.0.0.0” with “argv[1]” and “27972” with “argv[2]”

Try to **getaddrinfo** your server somewhere he cannot and look at the error message

```
$ ./server error 27972
server: getaddrinfo: Name or service not known
$
```

Does the message seem familiar to you? Try to **ping error** and see what happens

Do you understand where the error messages from **ping(1)** come now? :)

Server - Getting the results

```c
int sockfd = 0;
struct addrinfo *r = NULL;
/* */

for (r = res; r != NULL; r = r->ai_next)
{
    if ((sockfd  = socket(r->ai_family, r->ai_socktype, r->ai_protocol)) == -1)
        continue;

    if (bind(sockfd, r->ai_addr, r->ai_addrlen) == 0)
        break;

    close(sockfd);
}
if (r == NULL)
    errx(EXIT_FAILURE, "Cannot bind to the service.");

freeaddrinfo(res);
```

Server - Listening the call

```
char buff[4096] = { 0 };
ssize_t len = 0;
int clientfd = 0;
/* */

listen(sockfd, SOMAXCONN);
clientfd = accept(sockfd, NULL, NULL);

while (true)
{
    len = recv(clientfd, buff, sizeof (buff) - 1, 0);
    if (len <= 0)
        break;

     buff[len] = '\0';
     printf("%s", buff);
}
```

Server - Gracefully exiting

```
close(sockfd);
close(clientfd);

return 0;
```

Server - Execution

```
sh$ ./server 0.0.0.0 27972
```

Server - Testing with netcat

```
sh$ sudo pacman -Syu openbsd-netcat
...
sh$ nc 127.0.0.1 27972
coucou les apping
```

On the server side:

```
sh$ ./server 0.0.0.0 27972
coucou les apping
```

Client - Architecture

```
sh$ cp -r server client
```

Client - Makefile

```
CPPFLAGS= -D_POSIX_C_SOURCE=200112L
CFLAGS = -std=c99 -pedantic -Wall -Wextra -Wvla -Werror
BIN = client

all: $(BIN)

clean:
    $(RM) $(BIN)
```

Client - Getting the network device

```c
int ret = 0;
struct addrinfo hints = { 0 };
struct addrinfo *res = NULL;
/* */

hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;

if ((ret = getaddrinfo("127.0.0.1", "27972", &hints, &res)) != 0)
    errx(EXIT_FAILURE, "getaddrinfo: %s", gai_strerror(ret));
```

Client - About getaddrinfo

Replace “127.0.0.1” with “argv[1]” and “27972” with “argv[2]”

Client - Getting the results

```c
int sockfd = 0;
struct addrinfo *r = NULL;
/* */

for (r = res; r != NULL; r = r->ai_next)
{
    if ((sockfd  = socket(r->ai_family, r->ai_socktype, r->ai_protocol)) == -1)
        continue;

    if (connect(sockfd, r->ai_addr, r->ai_addrlen) == 0)
        break;

    close(sockfd);
}
if (r == NULL)
    errx(EXIT_FAILURE, "Cannot connect to the service.");

freeaddrinfo(res);
```

Client - Sending data

```
send(sockfd, argv[3], strlen(argv[3]), 0);
```

Client - Gracefully exiting

```
close(sockfd);
return 0;
```

Client - Execution

```
sh$ nc -lp 27972
```

On the client side:

```
sh$ ./client 127.0.0.1 27972 "Coucou les apping"
sh$
```

Multiplexing

https://en.wikipedia.org/wiki/C10k_problem

It's not about the CPU but the time.

You juggle between the awaiting connections, the actions to do with the clients, etc.

First there was select(2) but, meh.

Then the standardized poll(2), but, meh.

Come Together (ePoll McCartney, 1969)

This is for the server side

```
man 7 epoll [!] # Manpage with miscellaneous information about the epoll interfac
```

You can also look the deprecated implementations

```
man 2 poll # Manpage about the deprecated poll syscall
```

select is a deprecated syscall. poll is newer than accept, but both are meh.

Please use epoll.

Even newer

```
man 7 io_uring [!] # Manpage with misc. info about an async I/O facility
```

Kernel? Where we're going we don't need kernel

Give a look at DPDK. It's a framework to bypass the kernel and directly comunicate with the network drivers.

DPDK: Data Plane Development Kit

?

Questions?