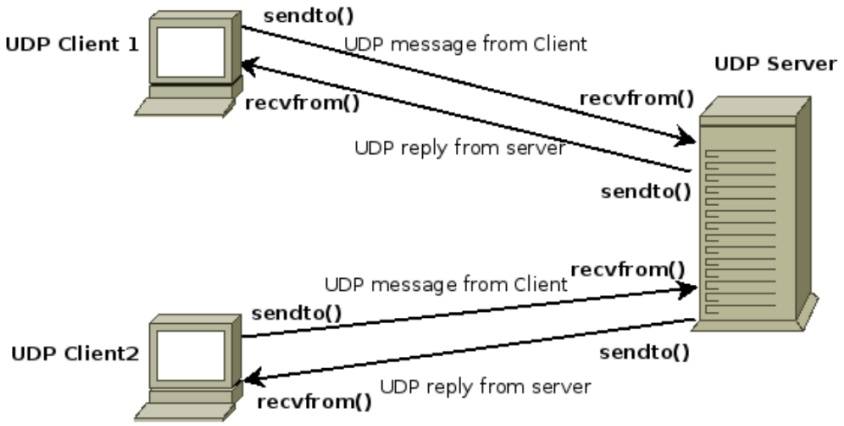
JOURDREN Vincent

TFTP Client

We will create a client program that must be able to exchange files using the Trivial File Transfer Protocol (TFTP).



1) Using the command line arguments of the gettftp and puttftp programs to obtain request information (server and file)

```
ensea@StudentLab:~/Documents/MajeurInfo/TP2$ ./gettftp 127.0.0.1 ones512
ensea@StudentLab:~/Documents/MajeurInfo/TP2$ ./puttftp 127.0.0.1 ones512
```

As we can see above, the arguments we need are the IP address of the server and the name of the file we want to download or upload.

2) Call to getaddrinfo to get the server address

The following data allows us to orientate the type of result we wish to have on the server.

```c
/* Obtenir l'adresse correspondant à l'hôte */
memset(&hints, 0, sizeof(struct addrinfo)); // Initialise la structure hints avec des 0
hints.ai_family = AF_INET;  // Autorise IPv4 ou IPv6
hints.ai_socktype = SOCK_DGRAM; // Datagram socket
hints.ai_flags = 0;
hints.ai_protocol = IPPROTO_UDP; // Protocole UDP

s = getaddrinfo(ip_adress, port, &hints, &result);
if (s != 0){ // Gestion de l'erreur getaddrinfo
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
    exit(EXIT_FAILURE);
}

/*  getaddrinfo() retourne une liste de structures d'adresse. */
```

To use the getaddrinfo() we need :

- the host address: 127.0.0.1

- the port: 1069 as we used the provided tftp server

- the hints, specifying the requirements for filtering the fetched socket structures

- pointer, where the function will dynamically allocate a linked list of addrinfo structures.

3) Reservation of a connection socket to the server

The socket allows to create a communication point, and returns a descriptor.
To create it, the following configuration must be used:

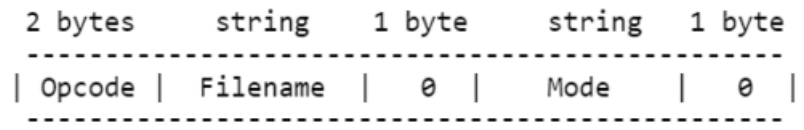
```
sfd = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
if (sfd == -1) { // Gestion de l'erreur socket
    printf("Socket error\n");
    exit(EXIT_FAILURE);
}
```

The configuration also includes socket error handling.

4) For gettftp :

a) Builds a properly formed read request (RRQ) and sends it to the server

Once the communication is established, we need to create a request to ask the server to send us a file.



The procedure for creating the RRQ can be found in RFC 1350.
We make a string composed of :

-OPCODE = 01
- The name of the file
- A 0
- The transmission mode: the byte
- And a final 0 to indicate the end of the word.

Useful functions for the building are :
- strcpy() which allows us to copy the content of a string into another string
- strlen() which allows us to get the length of a string

```
//RRQ = 01-zeros256-0-octet-0
RRQ[0] = 0;
RRQ[1] = 1;
strcpy(RRQ+2, filename);
RRQ[2+strlen(filename)] = 0;
strcpy(RRQ+3+strlen(filename), mode);
RRQ[3+strlen(filename)+strlen(mode)] = 0;
lenRRQ = 2+strlen(filename)+strlen(mode)+1+1;
```

b) Receiving a file consisting of a single data packet (DAT) and acknowledging it (ACK)

The request must be sent using the sendto() function

```
nsend = sendto(sfd, RRQ, lenRRQ, 0, result->ai_addr, result->ai_addrlen);
```

We create a buf where we will store the requested data. It is of size 516 because the maximum quantity that can be transported with the byte mode is 512 and the 4 bits for the information.

```
              2 bytes    2 bytes       n bytes
              ---------------------------------
    DATA     | 03     |   Block #  |    Data    |
              ---------------------------------
```

We also create an empty file that will receive the data received from the file we are interested in.

```
file = creat(filename, S_IRUSR | S_IWUSR | S_IRGRP);
if(file == 0){ // Gestion de l'erreur dans la création du fichier
    perror("create");
    exit(EXIT_FAILURE);
}
```

An ACK must be created, which is an acknowledgement of receipt that confirms receipt of the data. Then we send it to the server.

```
ACK[0] = 0;
ACK[1] = 4;
lenACK = 4;
```

```c
nrecv = recvfrom(sfd, buf, blocksize+4, 0, result->ai_addr, &(result->ai_addrlen));
ACK[2] = buf[2];
ACK[3] = buf[3];
nsend = sendto(sfd, ACK, lenACK, 0, result->ai_addr, result->ai_addrlen);
nbwr = write(file, buf+4, nrecv-4); // Recopie dans notre fichier vide en local, les données du fichier du serveur
printf("Nombre de bits reçus : %ld\n %d %d %d %d\n", nrecv, buf[0], buf[1], buf[2], buf[3]);

if (nbwr == -1){ // Gestion de l'erreur write
    perror("write");
    close(file);
    exit(EXIT_FAILURE);
}
```

This is the result :

```
ensea@StudentLab:~/Documents/MajeurInfo/TP2$ ./gettftp 127.0.0.1 ones512
Server IP : 127.0.0.1
Server Port : 1069
Nom du fichier : ones512
Nombre de bits envoyés : 16
Nombre de bits reçus : 516
 0 3 0 1
Nombre de bits reçus : 4
 0 3 0 2
Fin du téléchargement
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	60	46850 → 1069 Len=16
2	0.004095239	127.0.0.1	127.0.0.1	UDP	560	34855 → 46850 Len=516
3	0.004108092	127.0.0.1	127.0.0.1	UDP	48	46850 → 34855 Len=4
4	0.004252108	127.0.0.1	127.0.0.1	UDP	48	34855 → 46850 Len=4
5	0.004262364	127.0.0.1	127.0.0.1	UDP	48	46850 → 34855 Len=4

On the wireshark capture, we see the connection with the server (port 1069) with first the RRQ request which is sent with a size of 16 bits.
Then, we receive the word with a size of 516 bits as expected in UDP protocol.
Finally, we see the 4-bit ACK acknowledgement between the server and the client.

c) Receiving a file consisting of several data packets (DAT) and their respective acknowledgements (ACK)

This is the same process as the previous question, we just add a while(1) loop that stops when the data received is less than 516.

```
while(1){
    nrecv = recvfrom(sfd, buf, blocksize+4, 0, result->ai_addr, &(result->ai_addrlen));
    ACK[2] = buf[2];
    ACK[3] = buf[3];
    nsend = sendto(sfd, ACK, lenACK, 0, result->ai_addr, result->ai_addrlen);
    nbwr = write(file, buf+4, nrecv-4); // Recopie dans notre fichier vide en local, les données du fichier du serveur
    printf("Nombre de bits reçus : %ld\n %d %d %d %d\n", nrecv, buf[0], buf[1], buf[2], buf[3]);

    if (nbwr == -1){ // Gestion de l'erreur write
        perror("write");
        close(file);
        exit(EXIT_FAILURE);
    }

    if(nrecv < 516){ // Arrête le transfert quand les données reçues sont < 516
        printf("Fin du téléchargement\n");
        break;
    }
```

To do the test we need a file with a size greater than 512 bits.
The « ones1024 » file is perfectly suitable because it is supposed to be sent twice (512*2=1024).

This is the result :

```
ensea@StudentLab:~/Documents/MajeurInfo/TP2$ ./gettftp 127.0.0.1 ones1024
Server IP : 127.0.0.1
Server Port : 1069
Nom du fichier : ones1024
Nombre de bits envoyés : 17
Nombre de bits reçus : 516
 0 3 0 1
Nombre de bits reçus : 516
 0 3 0 2
Nombre de bits reçus : 4
 0 3 0 3
Fin du téléchargement
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	61	58165 → 1069 Len=17
2	0.000705012	127.0.0.1	127.0.0.1	UDP	560	47384 → 58165 Len=516
3	0.002575335	127.0.0.1	127.0.0.1	UDP	48	58165 → 47384 Len=4
4	0.002650810	127.0.0.1	127.0.0.1	UDP	560	47384 → 58165 Len=516
5	0.002682076	127.0.0.1	127.0.0.1	UDP	48	58165 → 47384 Len=4
6	0.002786649	127.0.0.1	127.0.0.1	UDP	48	47384 → 58165 Len=4
7	0.002825172	127.0.0.1	127.0.0.1	UDP	48	58165 → 47384 Len=4

On the wireshark capture, as before, we can see the link with the server as well as the reception of data and its acknowledgement.
But this time, two words of 516 bits are received, with their respective acknowledgement after each reception, corresponding as agreed to the ones1024 file of size 1024 bits.

5) For puttftp :

a) Builds a properly formed write request (WRQ) and sends it to the server

The construction of the WRQ is almost identical to that of the RRQ, except for the OPCODE which will now be 02.

```
//WRQ = 02-zeros256-0-octet-0
WRQ[0] = 0;
WRQ[1] = 1;
strcpy(WRQ+2, filename);
WRQ[2+strlen(filename)] = 0;
strcpy(WRQ+3+strlen(filename), mode);
WRQ[3+strlen(filename)+strlen(mode)] = 0;
lenWRQ = 2+strlen(filename)+strlen(mode)+1+1;
```

b) Sending a file consisting of a single data packet (DAT) and receiving its acknowledgement (ACK)

The request must be sent using the sendto() function

```
nsend = sendto(sfd, WRQ, lenWRQ, 0, result->ai_addr, result->ai_addrlen);
```

As with gettftp, a buf is created where the requested data will be stored.

We open the file we want to send to the server.

```
file = open(filename, O_RDONLY);
if (file == 0) { // Gestion de l'erreur dans l'ouverture du fichier
    perror("open");
    exit(EXIT_FAILURE);
}
```

The ACK acknowledgement is also created as before.

```
nrecv = recvfrom(sfd, ACK, lenACK, 0, result->ai_addr, &(result->ai_addrlen));
buf[2] = ACK[2];
buf[3] = ACK[3]+1;
nbrd = read(file, buf+4, nrecv-4);
nsend = sendto(sfd, buf, blocksize+4, 0, result->ai_addr, result->ai_addrlen);
printf("Nombre de bits reçus : %ld\n %d %d %d %d\n", nrecv, buf[0], buf[1], buf[2], buf[3]);

if (nbrd == -1){ // Gestion de l'erreur read
    perror("read");
    close(file);
    exit(EXIT_FAILURE);
}
```

This is the result :

```
ensea@StudentLab:~/Documents/MajeurInfo/TP2$ ./puttftp 127.0.0.1 ones2040
Server IP : 127.0.0.1
Server Port : 1069
Nom du fichier : ones2040
Nombre de bits envoyés : 17
Nombre de bits reçus : 4
 0 3 0 1
Fin du téléchargement
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	61	58405 → 1069 Len=17
2	0.000827768	127.0.0.1	127.0.0.1	UDP	48	46830 → 58405 Len=4
3	0.001346650	127.0.0.1	127.0.0.1	UDP	560	58405 → 46830 Len=516
4	0.001648231	127.0.0.1	127.0.0.1	UDP	48	46830 → 58405 Len=4

On the wireshark capture, we see the connection with the server (port 1069) with first the WRQ request which is sent with a size of 17 bits.
Then, we send the word with a size of 516 bits as expected in UDP protocol.
Finally, we see the 4-bit ACK acknowledgement between the server and the client.