Exercice 1 :

Commande utilisé ::%s/Foo/Bar Foo= ce que l'on cherche, Bar=ce par quoi on veut le remplacer

Ces noms sont plus appropriés car cela est plus représentatif.

Exercice 2 :

Le premier close sert a fermé le numOfDemandDescriptor car nous n'en n'avons plus besoin maintenant que la connexion est établie

Le deuxième ferme le numOfConnectedDescriptor si il y a une erreur

Le troisième ferme le numOfConnectedDescriptor lorsque l'on a fini de communiquer

Le quatrième ferme le numOfConnectedDescriptor lorsque l'on est dans son parent

Exercice 4 :

On écrit dans le numéro de descripteur J car il correspond à la sortie standard (l'écran)
, c'est la fonction dup qui copie ce numéro dans J.

L'appel système write va écrire sur le numéro de descriptor J (ici la sortie standard, donc l'écran), ce que contient le buffer, et la fonction le nombre de bit à afficher.

On peut subsituer cette fonction par un print.

Exercice 5 :

Premièrement on lance notre serveur qui sera donc en attente d'une connection, ensuite avec un deuxième terminal on utilise la commande telnet pour se connecter à son serveur, une fois connecté on peut envoyé des messages et on voit bien que le serveur affiche ce qu'il reçoit et renvoie au client ce qu'il a envoyé, le client affiche bien ce que le serveur lui renvoie aussi.

Les deux interfaces réseau utilisé sont l'adresse IP 127.0.0.1 et le port de connection.

Exercice 6 :

La table des descripteur contient les adresses des descripteurs donc lorsque l'on utilise close on met cette valeur a NULL

Exercice 7 :

Client.c prend en premier paramètre le port de connexion et en deuxième argument l'adresse IP de connexion donc pour ce connecter au serveur il suffit de lancer le programme client avec les bons paramètres.

Exercice 8 :

L'appel système **connect**() connecte la socket référencée par le descripteur de fichier *sockfd* à l'adresse indiquée par *serv_addr*. L'argument *addrlen* indique la taille de *serv_addr*.

Exercice 9 :

Ce système s'apparente à une application de tchat, chaque personnes doit avoir un serveur de reception ouvert et un client pour émettre.

Test :

Envoyer un message à un autre ordianteur, après que chaque personne est ouvert un serveur et un client qui seront donc interconnecté, on peut donc bien tchater avec son voisin, le serveur affiche ce que le client de l'autre personne lui envoie.

CODE :

```
#include <stdio.h>
#include <errno.h>
#include <sys/socket.h>
#include <resolv.h>
#include <arpa/inet.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>


#define MY_PORT 9998
#define MAXBUF  1024
#define NUMBER_OF_CLIENTS 2048


int main(int argc, const char *argv[])
{
	int numOfDemandDescriptor;
	if((numOfDemandDescriptor = socket(AF_INET, SOCK_STREAM, 0)) < 0)
	{
		perror("socket()");
		exit(errno);
	}

	struct sockaddr_in self;
	bzero(&self, sizeof(self));
	self.sin_family      = AF_INET;
	self.sin_port        = htons(MY_PORT);
	self.sin_addr.s_addr = INADDR_ANY;

	if(bind(numOfDemandDescriptor, (struct sockaddr*)&self, sizeof(self)) != 0)
	{
		perror("bind()");
		exit(errno);
```

```
	}

	if(listen(numOfDemandDescriptor, 20) != 0)
	{
		perror("listen()");
		exit(errno);
	}

	printf("Father pid [%d]\n", getpid());

	char buffer[MAXBUF];
	pid_t clients[NUMBER_OF_CLIENTS];
	int nbclients = 0;

				while(1)
				{
					int sizereceived = write(numOfConnectedDescriptor, buffer,
read(numOfConnectedDescriptor, buffer, MAXBUF));
					printf("Received from child [%d] : \n", getpid());
					int j = dup(1);
					write(j, buffer,sizereceived); /* Why write on descriptor number j ? (Is it
because it is the best one ?)*/

					if(sizereceived == 0)
					{
						close(numOfConnectedDescriptor);
						printf("The end of [%d]: \n", getpid());
						exit(1);
					}

					if(strstr(buffer,"/exit")==buffer) // Take care that it is not fully satisfying,
but sufficient for this directed work (TD)
					{
						printf("Je meurs [%d]\n",getpid());
						close(numOfConnectedDescriptor);
						exit(0);
					}
				}

			default:
				nbclients++;
				close(numOfConnectedDescriptor);
				break;
		}
	}


}
```