1- LES FONCTIONS UTILISEES :

On a travaillé dans le mode connecté c.-à-d. le message est reçu d'un seul bloc. Ainsi en mode connecté, la fonction listen() permet de placer le socket en mode passif (à l'écoute des messages). En cas de message entrant, la connexion peut être acceptée grâce à la fonction accept(). Lorsque la connexion a été acceptée, le serveur reçoit les données grâce à la fonction recv().

Les fonctions utilisées :

2-1 les primitives des sockets:

La primitive socket : socket(AF_INET,SOCK_STREAM,0)

Point d 'ancrage qui permet à l'application d'obtenir un lien de communication vers la suite de protocoles qui servira d'échange, et il définit le mode de communication utilisé (connecté ou non-connecté)

La primitive listen : listen(s,5)

Permet à un serveur d'entrer dans un mode d'écoute de communication , – dés lors le serveur est « connectable » par un client dés lors le serveur est « connectable » par un client, – le processus est bloqué jusqu'à l'arrivée d'une communication entrante.

La primitive bind : bind(s,(struct sockaddr)&serveur *), sizeof(serveur))

Après la création du socket, la fonction bind lie le socket à l'adresse et au numéro de port spécifiés dans addr (structure de données personnalisée). Dans l'exemple de code, nous lions le serveur à l'hôte local, nous utilisons donc INADDR_ANY pour spécifier l'adresse IP.

La primitive connect : connect(int sockfd,const struct sockaddr *servaddr,socklen t addrlen)

La primitive connect permet d'établir la connexion avec une socket distante, supposée à l'écoute sur un port connu à l'avance de la partie cliente. Son usage principal est d'utiliser une socket en mode « connecté». L'usage d'une socket en mode datagramme est possible mais a un autre sens (voir plus loin) et est moins utilisé.

la primitive accept : `accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)`

Il extrait la première demande de connexion de la file d'attente des connexions en attente pour le socket d'écoute, sockfd, crée un nouveau socket connecté et renvoie un nouveau descripteur de fichier faisant référence à ce socket. À ce stade, la connexion est établie entre le client et le serveur, et ils sont prêts à transférer des données.

la primitive send : `send(int s, const void *msg, size_t len, int flags)`

send est à utiliser pour le mode connecté. Elle envoie sur le socket s, les données pointées par msg, pour une taille de len octets. Cette fonction renvoi le nombre d'octets envoyés.

la primitive recv : `int recv(int s, void *buf, int len, unsigned int flags)`

recv est à utiliser pour le mode connecté. Elle reçoit sur le socket s, des données qu'elle stockera à l'endroit pointé par buf, pour une taille maximale de len octets. Cette fonction renvoi le nombre d'octets reçus.

2-2 les fonctions des fichiers:

fopen:

Chaque fois que vous voulez ouvrir un fichier, que ce soit pour le lire ou pour y écrire, il faut :

1. Appeler la fonction d'**ouverture de fichier** fopen qui renvoie un pointeur sur le fichier.
2. **Vérifier si l'ouverture a réussi** (c'est-à-dire si le fichier existait) en testant la valeur du pointeur qu'on a reçu. Nous verrons comment faire dans un instant.

fclose:

Si l'ouverture du fichier a réussi, vous pouvez le lire et y écrire (nous allons voir sous peu comment faire).
Une fois que vous aurez fini de travailler avec le fichier, il faudra le « fermer ». On utilise pour cela la fonction fclose qui a pour rôle de libérer la mémoire, c'est-à-dire supprimer votre fichier chargé dans la mémoire vive

fgets:

Cette fonction lit une chaîne dans le fichier. Ça vous évite d'avoir à lire tous les caractères un par un. La fonction **lit au maximum une ligne** (elle s'arrête au premier \n qu'elle rencontre). Si vous voulez lire plusieurs lignes, il faudra faire une boucle.

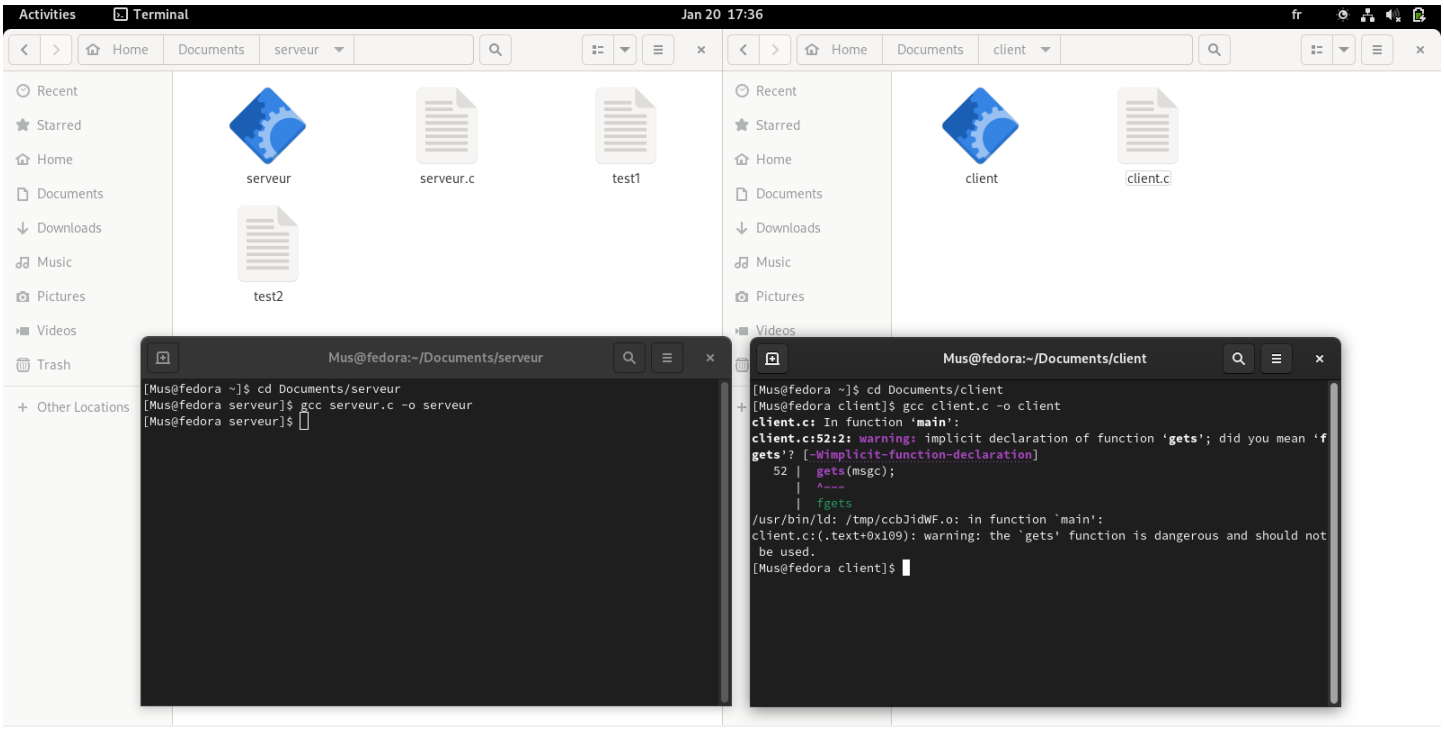
fputs:

La fonction puts en C est utilisée pour écrire une ligne ou une chaîne dans le flux de sortie (stdout) qui est jusqu'au caractère null, mais n'en inclut pas. La fonction puts ajoute également un caractère de saut de ligne à la sortie et renvoie un entier.

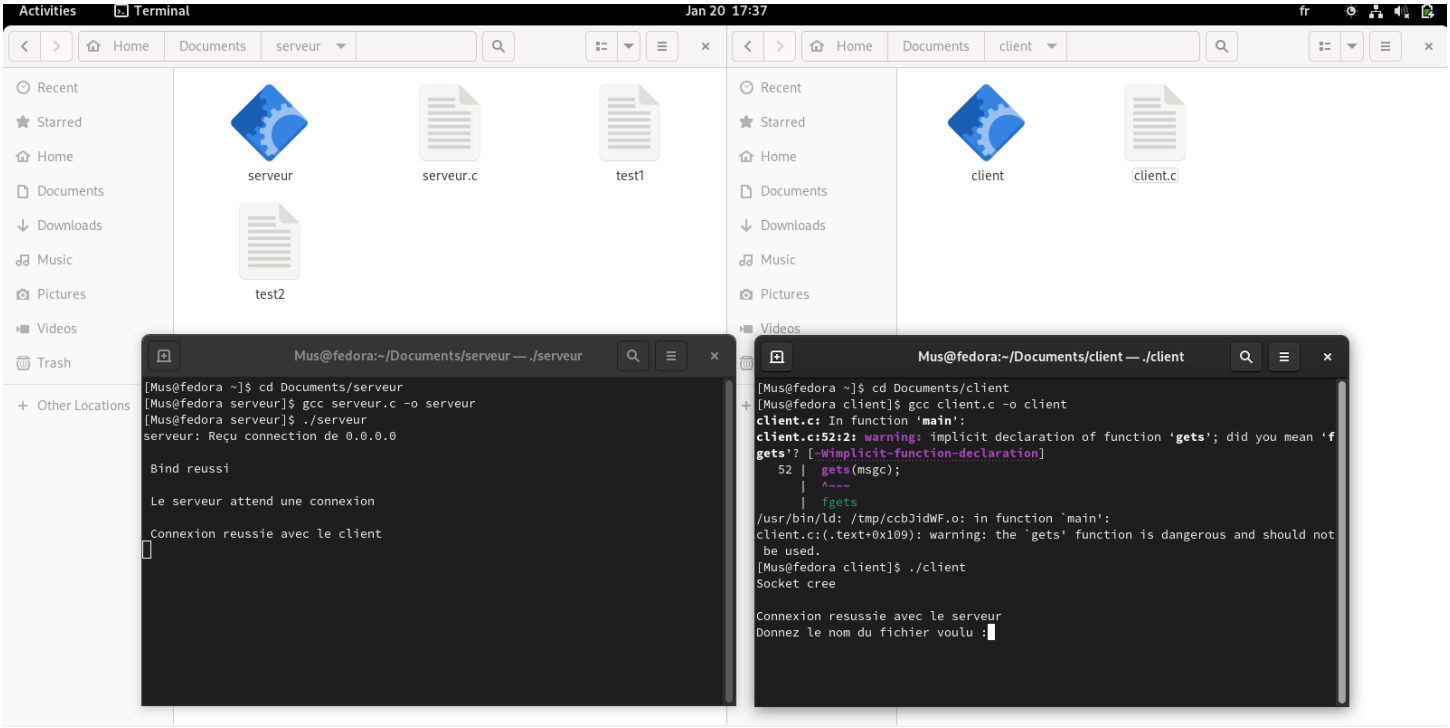
2- L'APPLICATION:

2-1 Une machine/un seul client :

On compile les deux fichiers dans deux différents répertoires

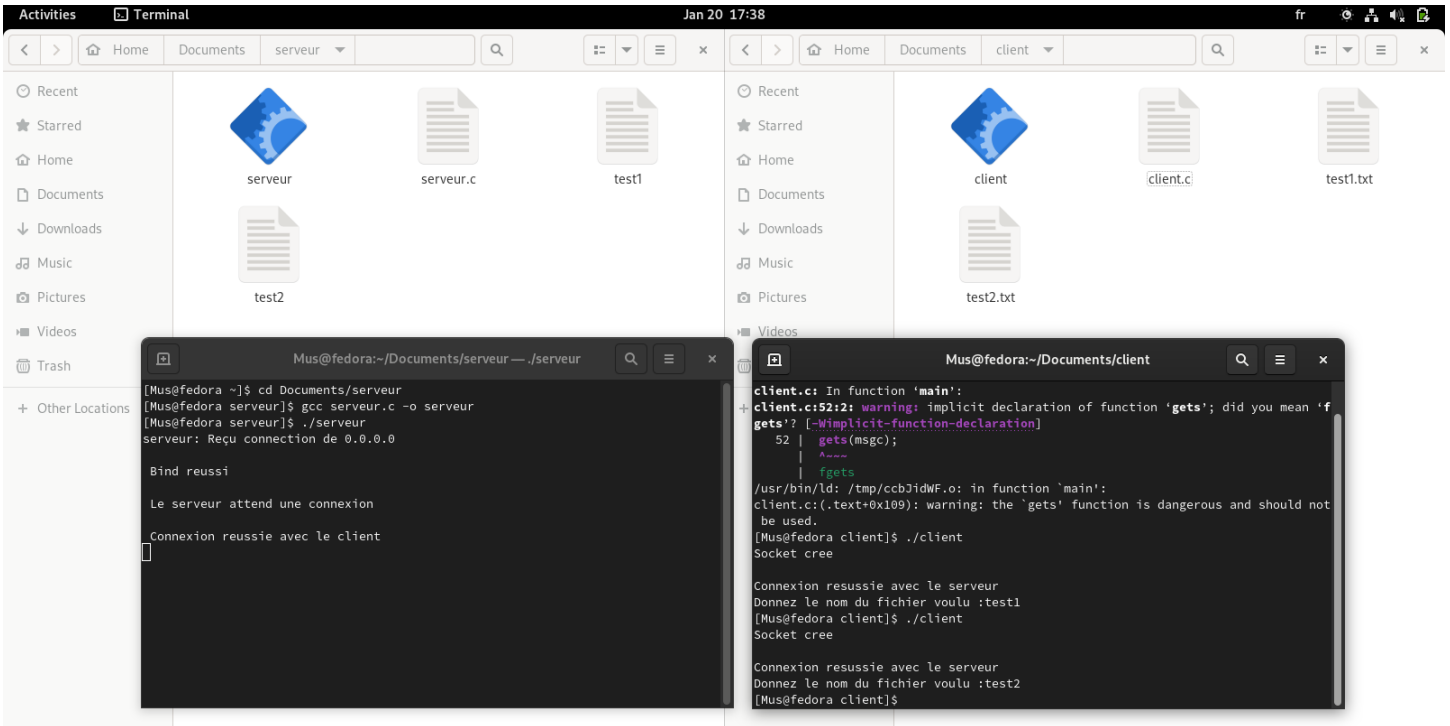


Puis on les exécute :

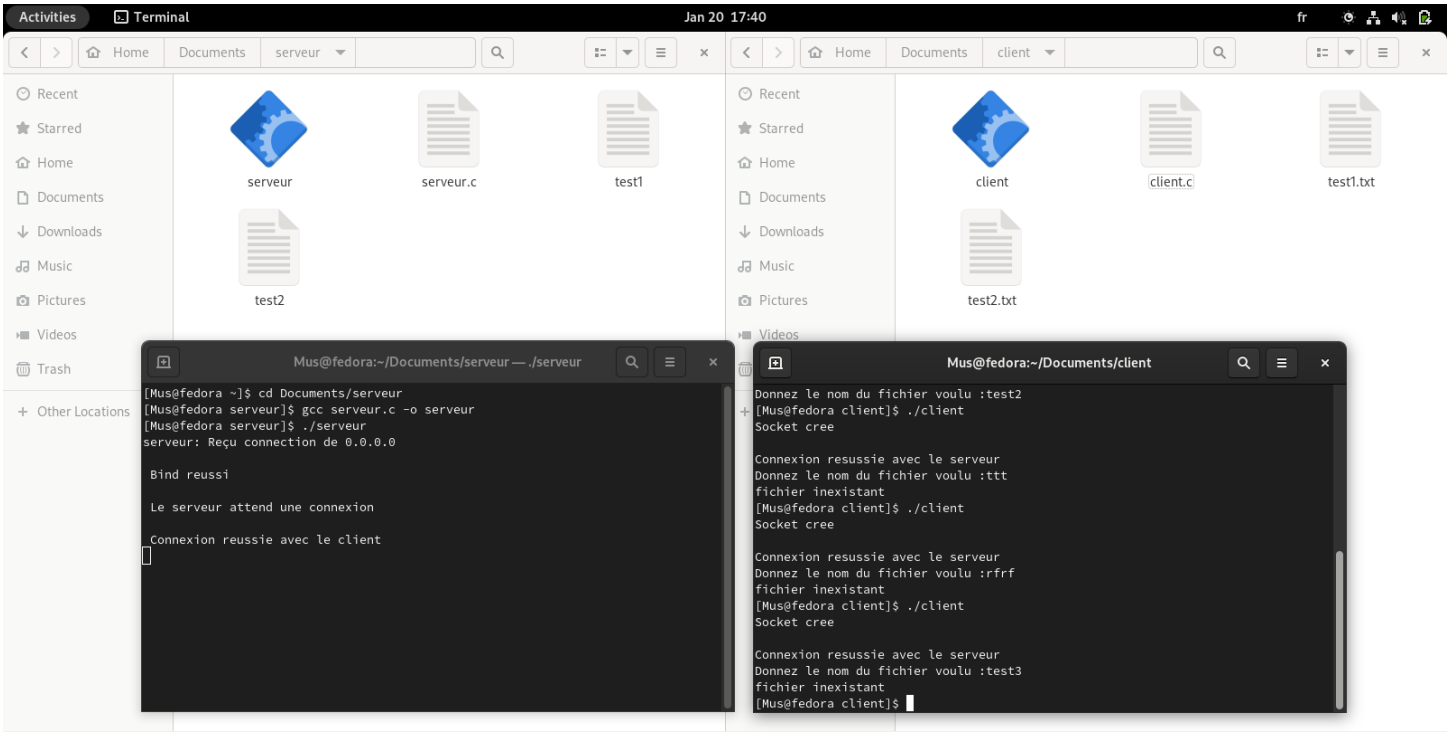


On choisit les fichiers désirés

- Cas d'un fichier existant :

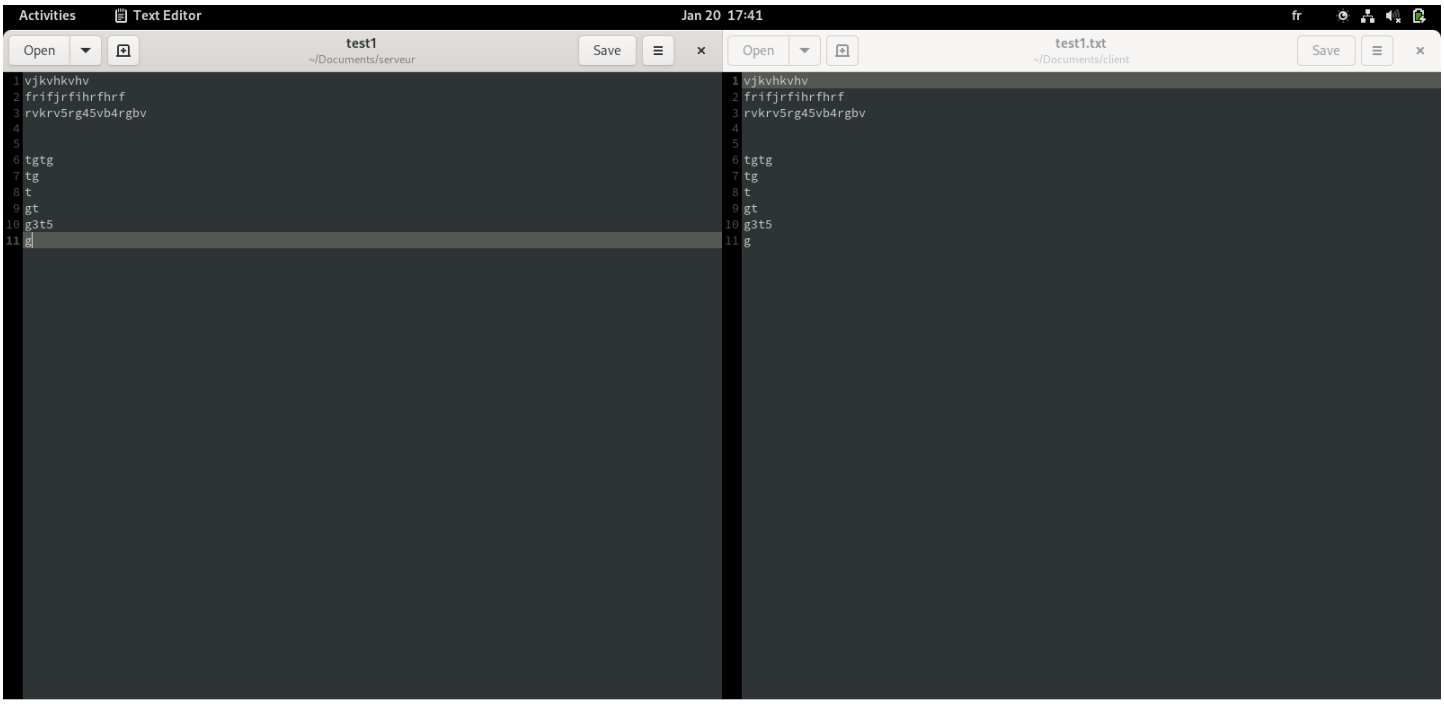


- Cas d'un fichier inexistant :

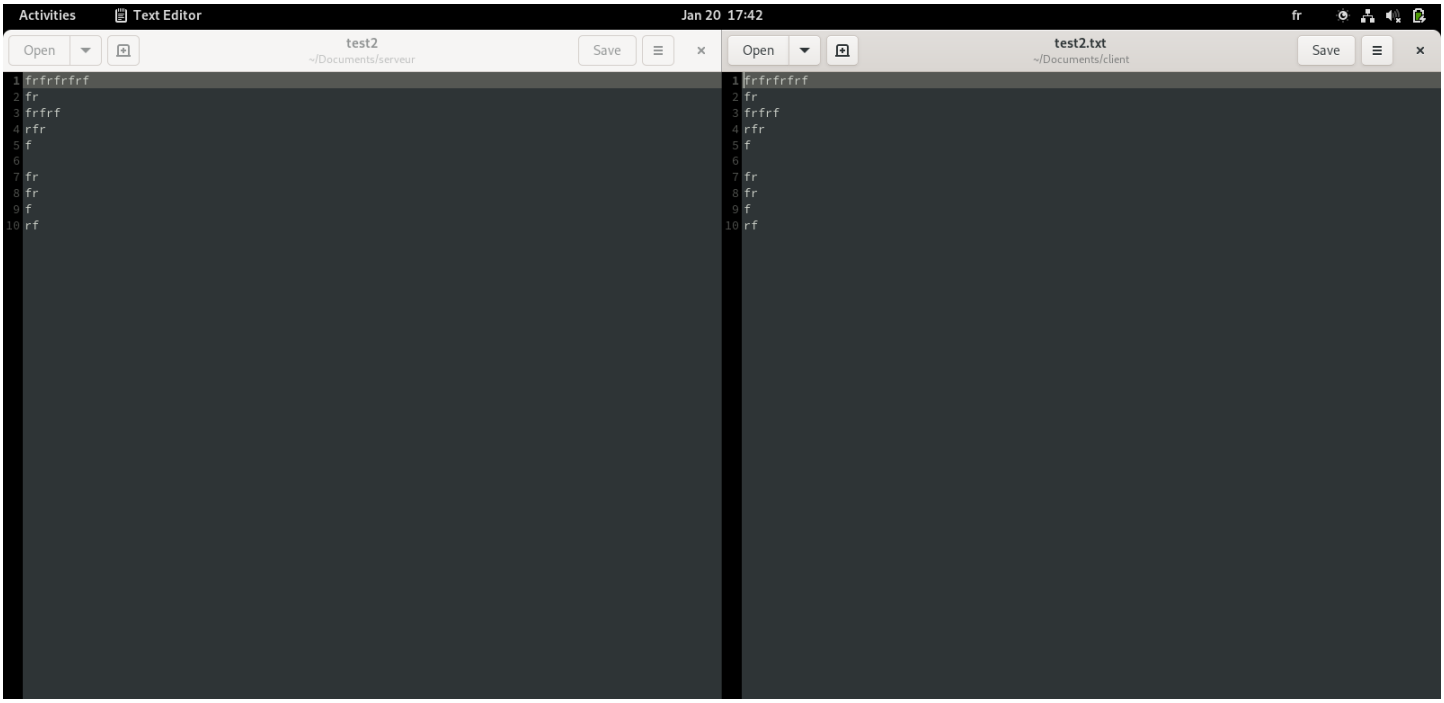


Test des fichiers crée :

- test1 :

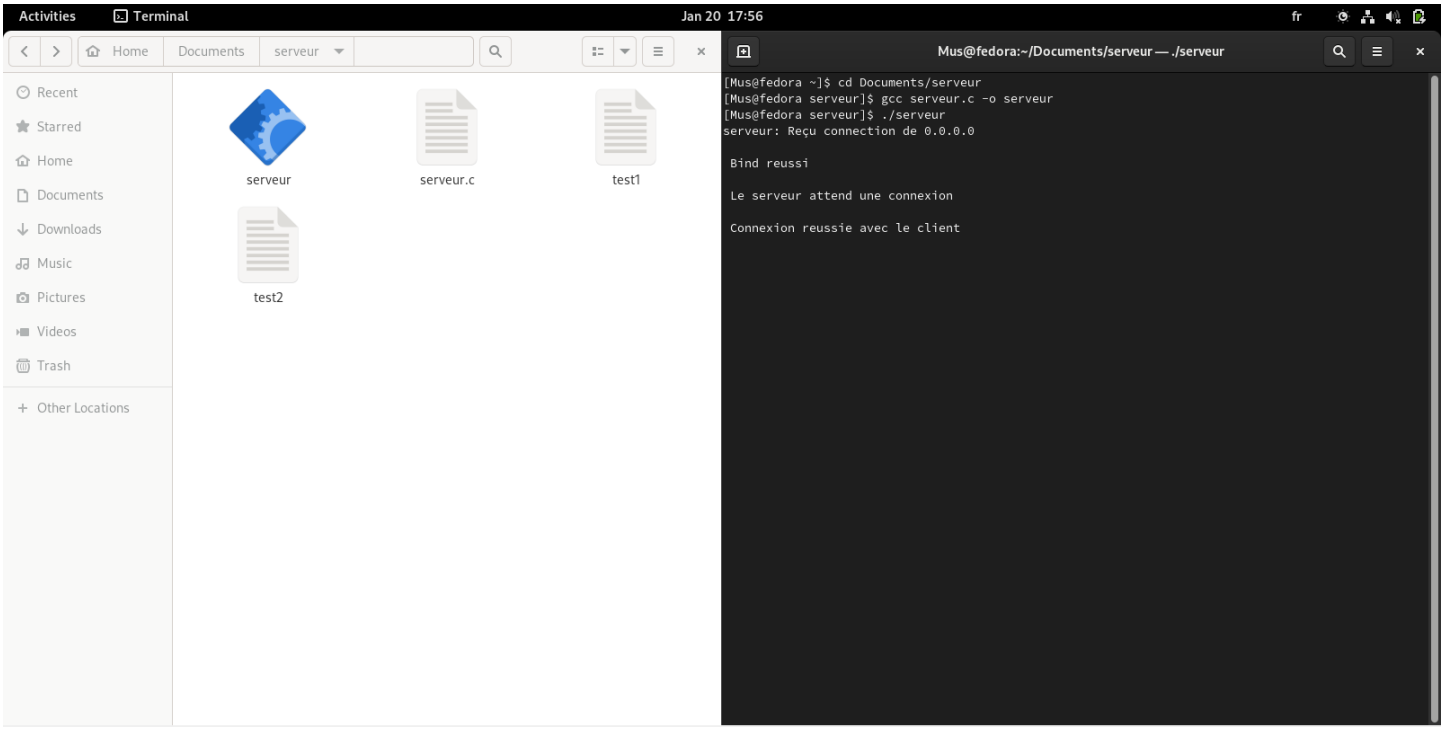


- test2 :

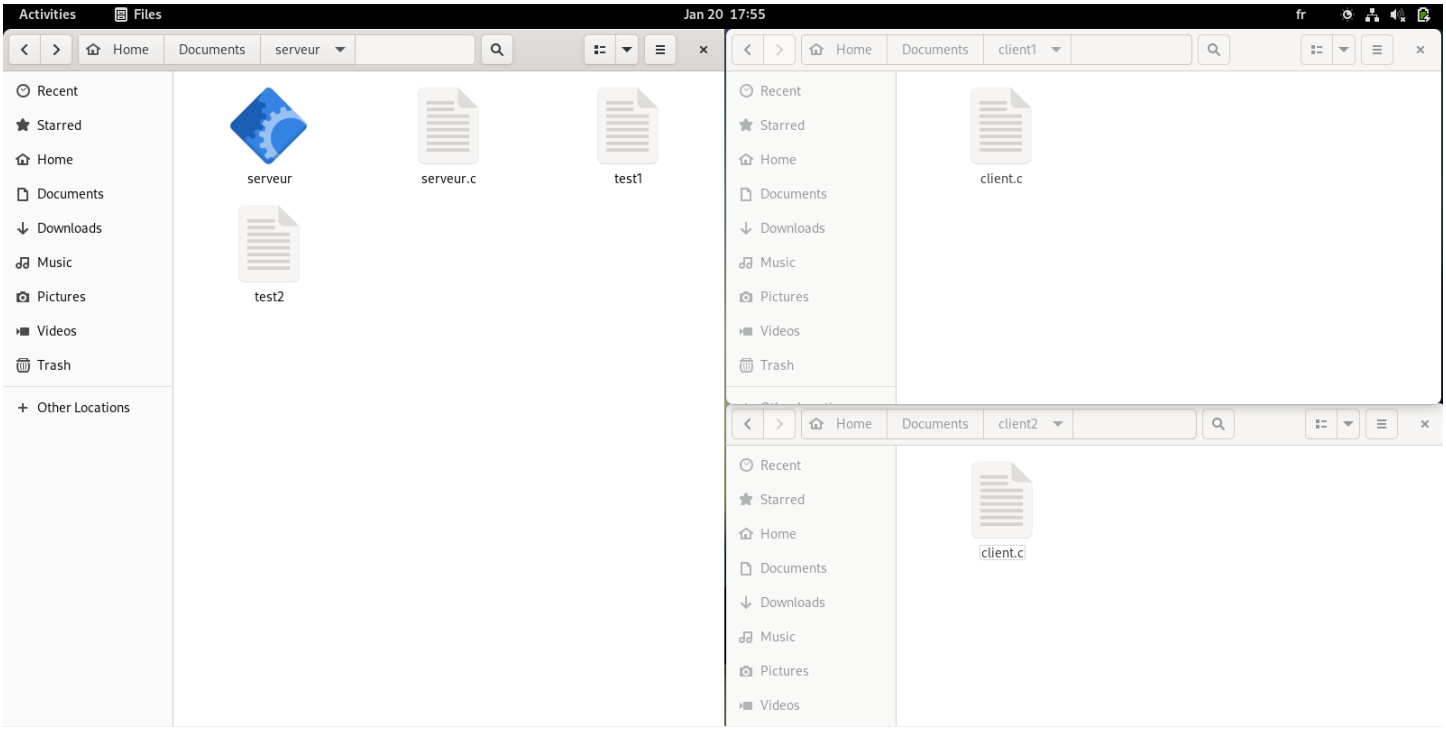


2-2 Une machine/deux client :

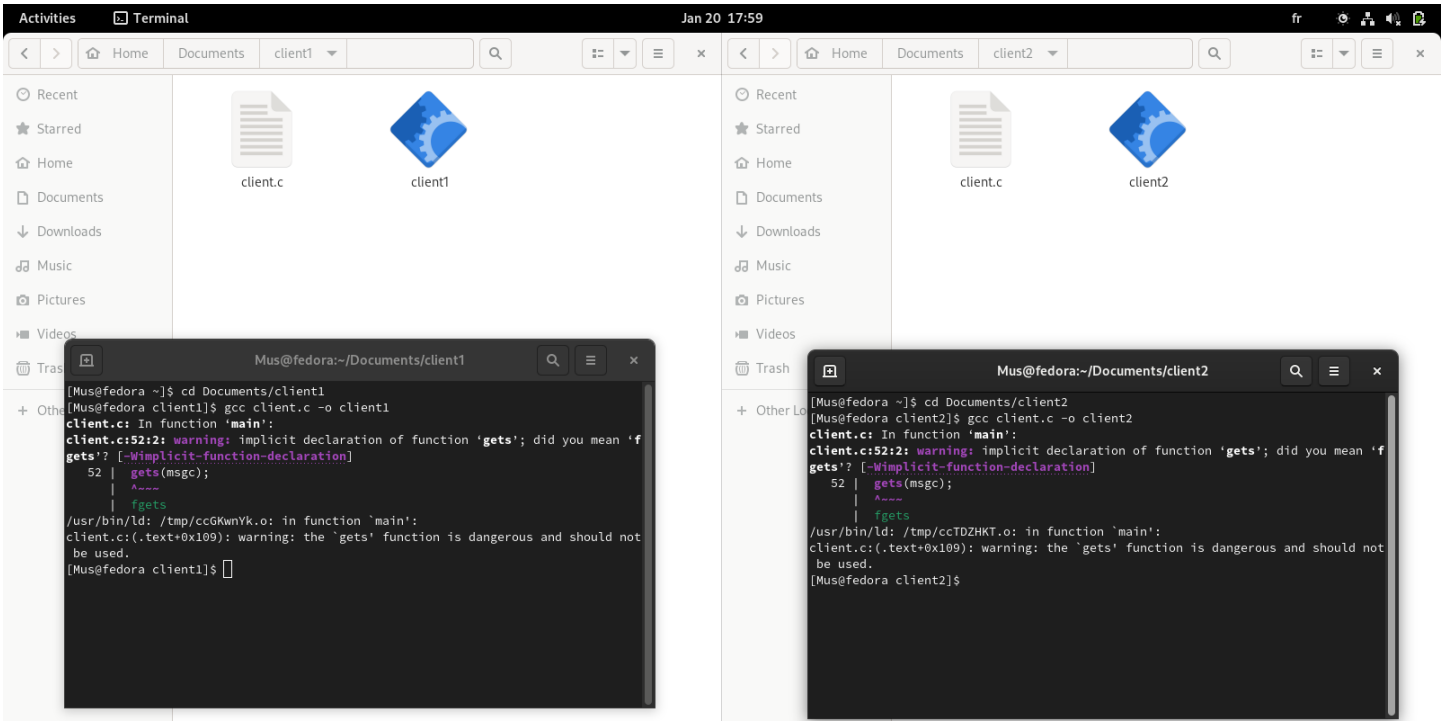
Aucun changement sur le programme serveur :



On crée deux répertoires ‘client1’ et ‘client2’ :

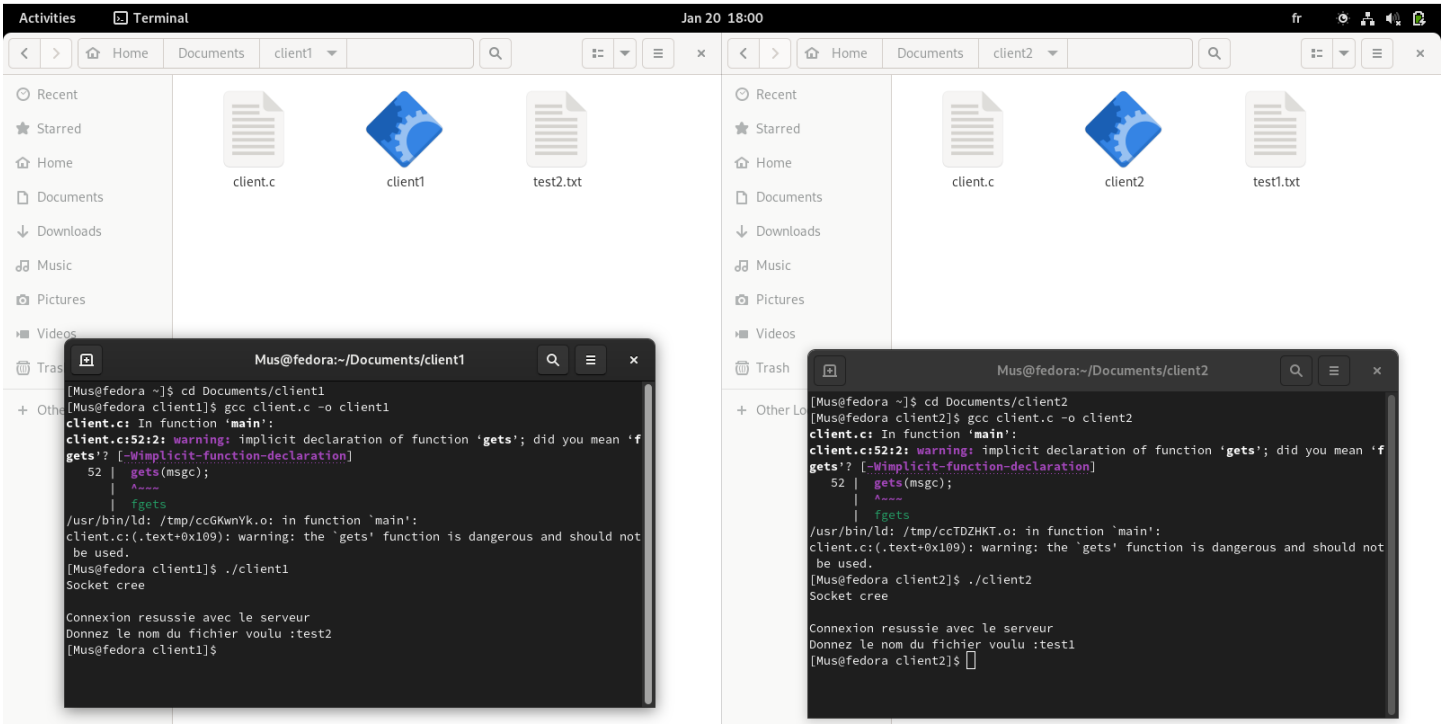


On compile les deux fichiers clients :

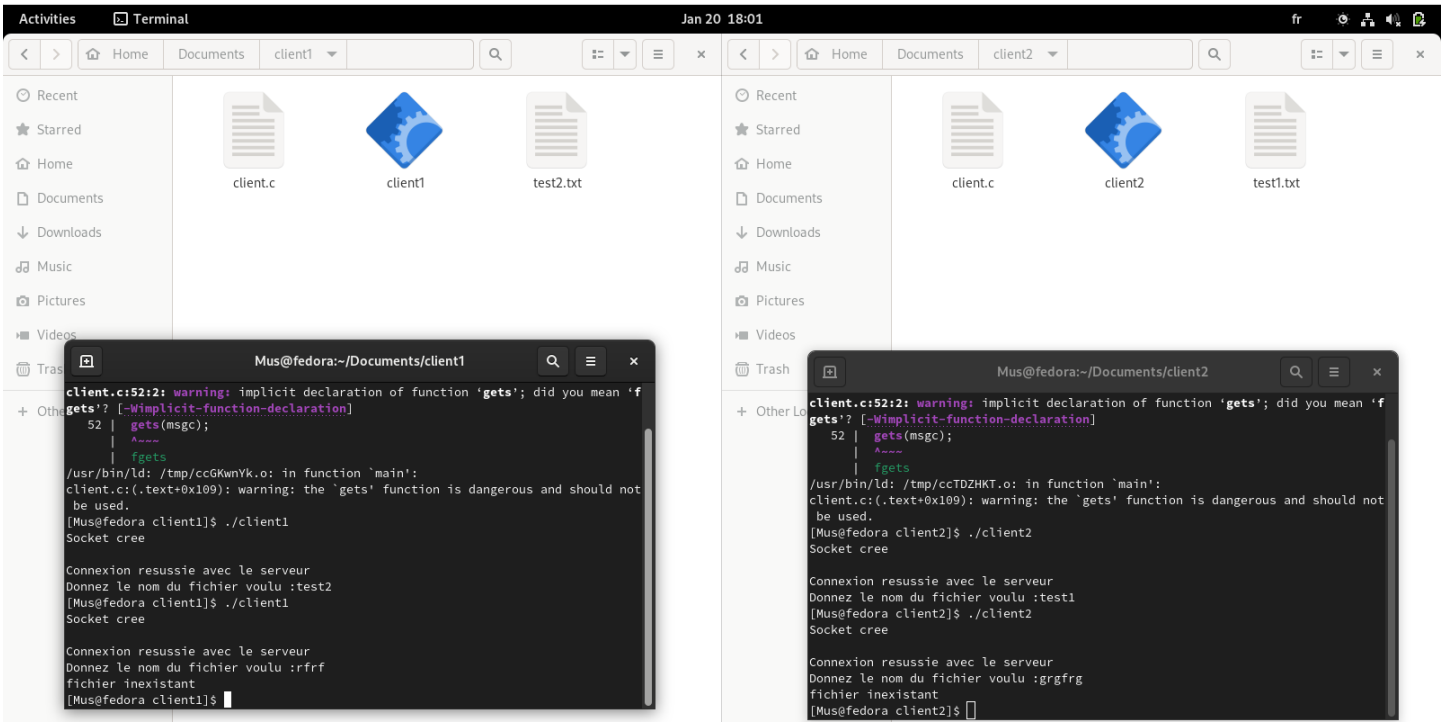


Puis on se connecte au serveur

Exemple 1 :



Exemple 2 :

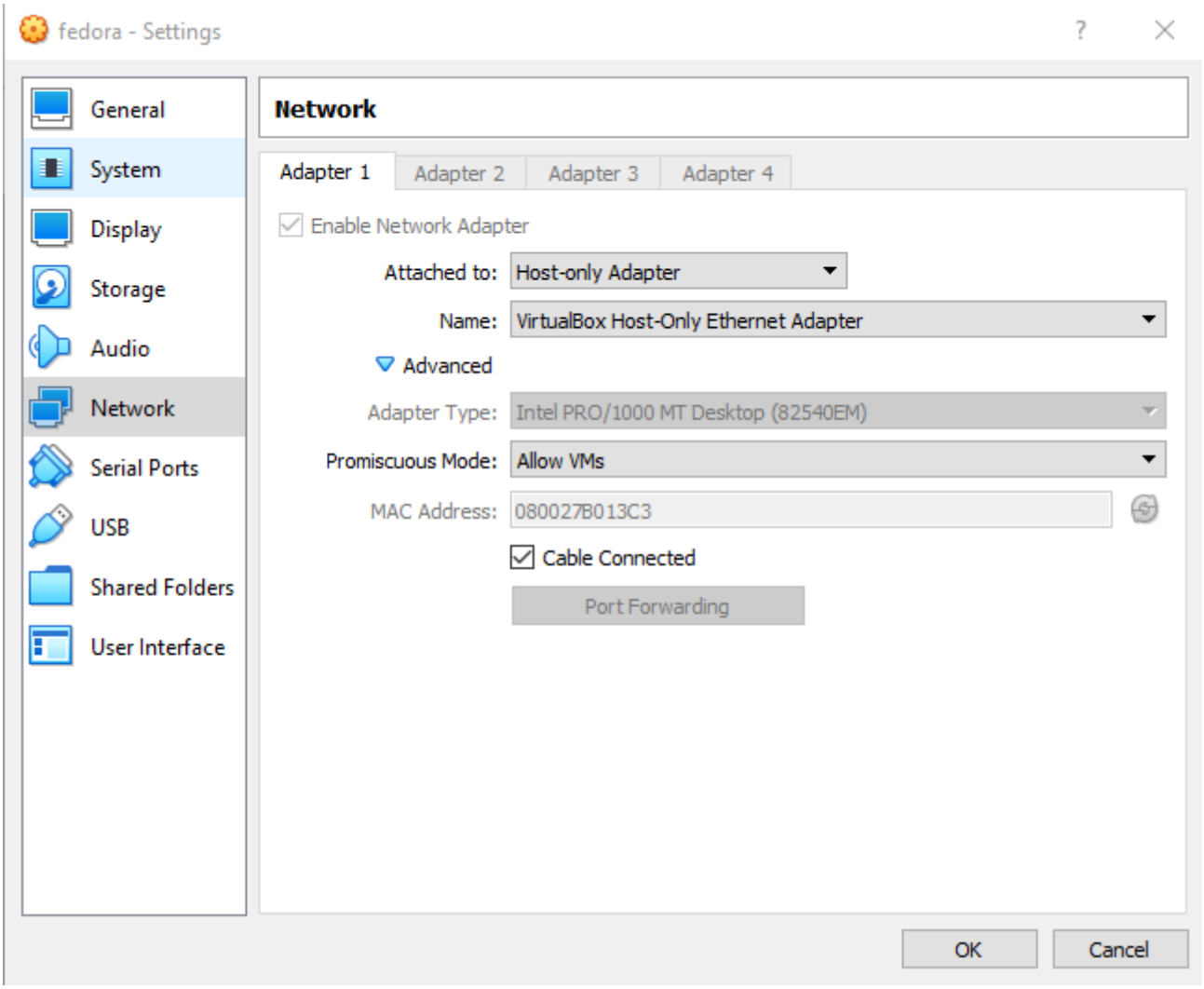


Remarque : Si un client se connecte au serveur et ne demande aucun fichier (n'écris rien sur la prompt « Donnez le nom du fichier voulu ») alors l'autre client même s il demande un fichier sa demande ne sera pas exécuter que lorsque le premier client finit sans exécution.

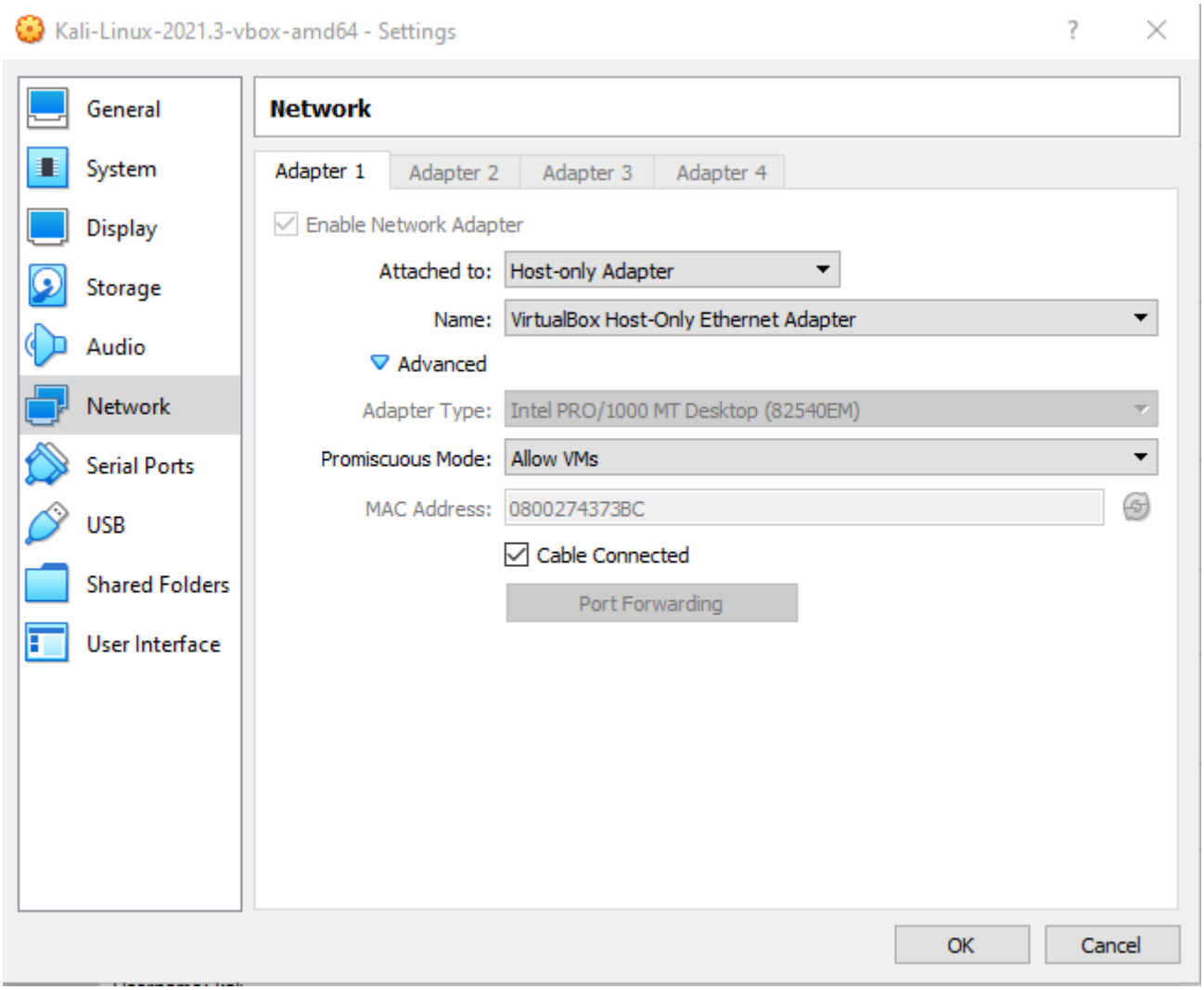
2-3 Deux machine/un seul client :

Pour cela on va utiliser la machine Fedora et la machine Kali linux, par défaut les deux seront connecté à l'internet mais pas entre eux donc on doit changer le mode de connecte :

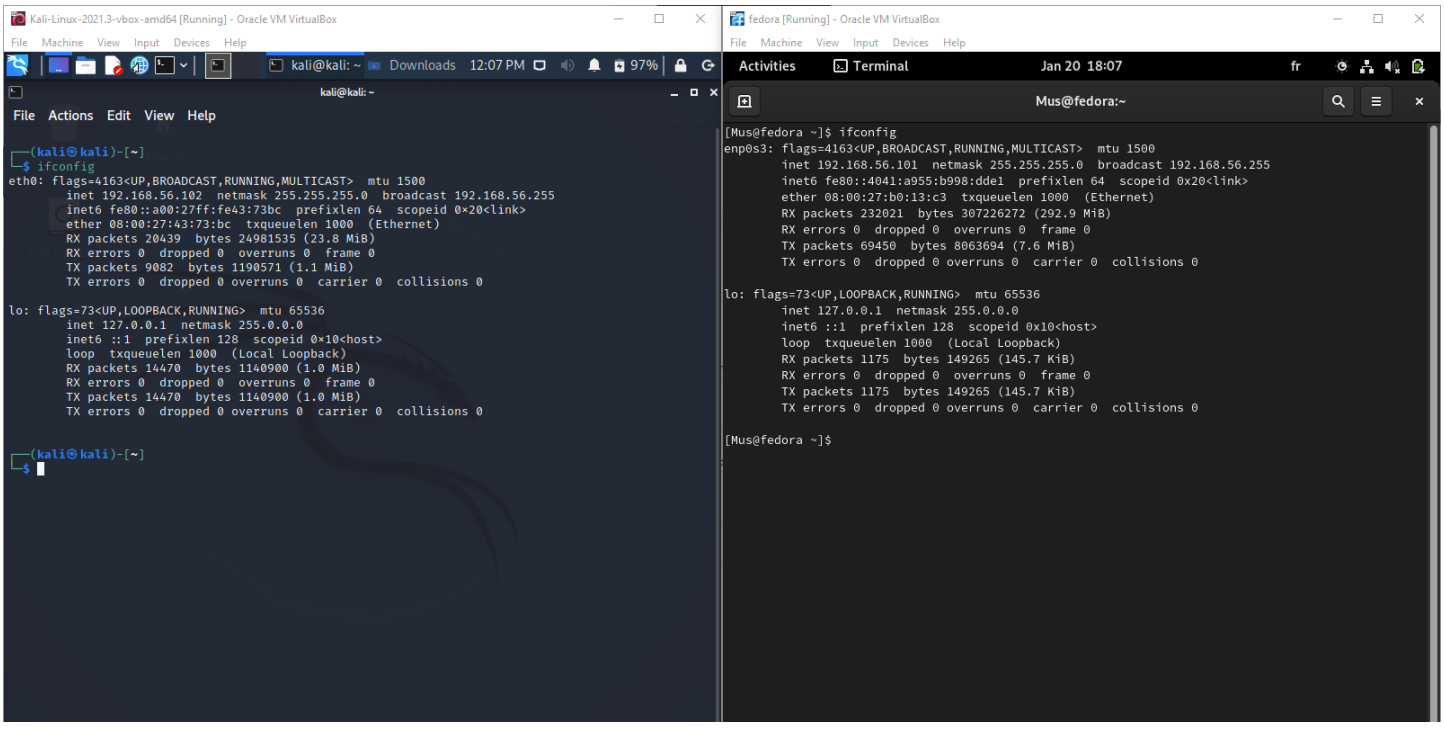
- Pour Fedora



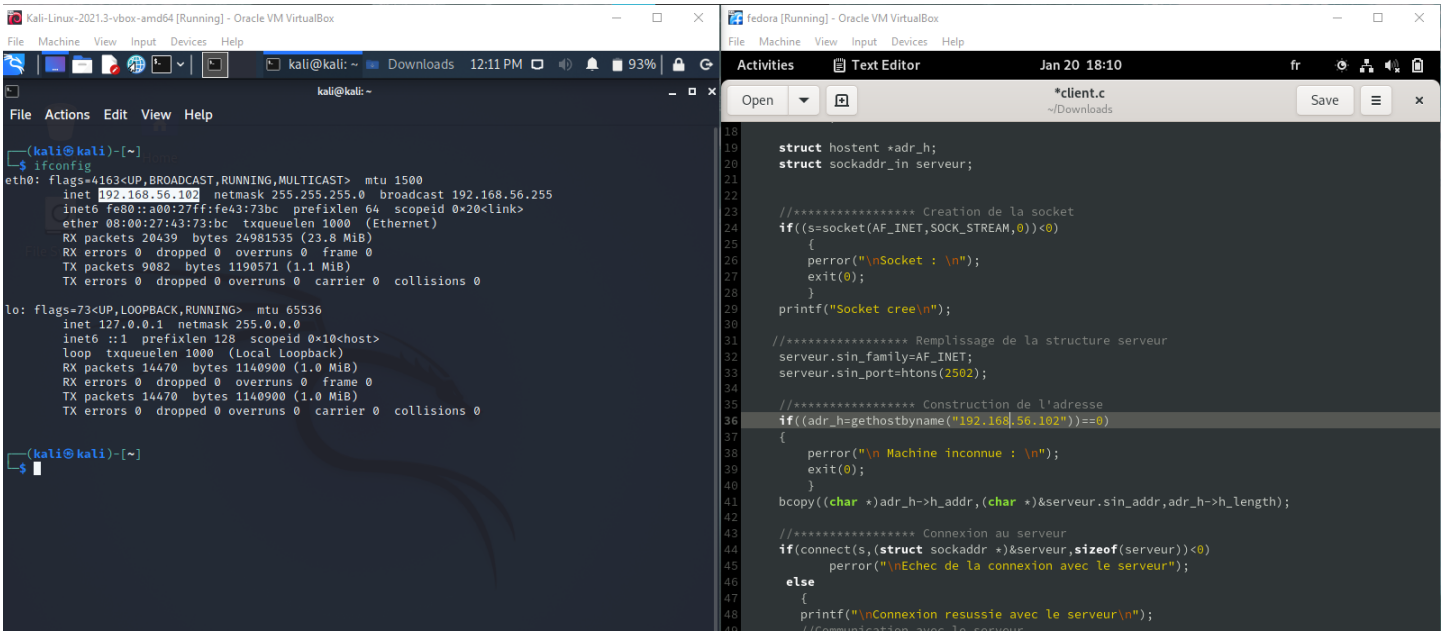
- Pour Kali linux



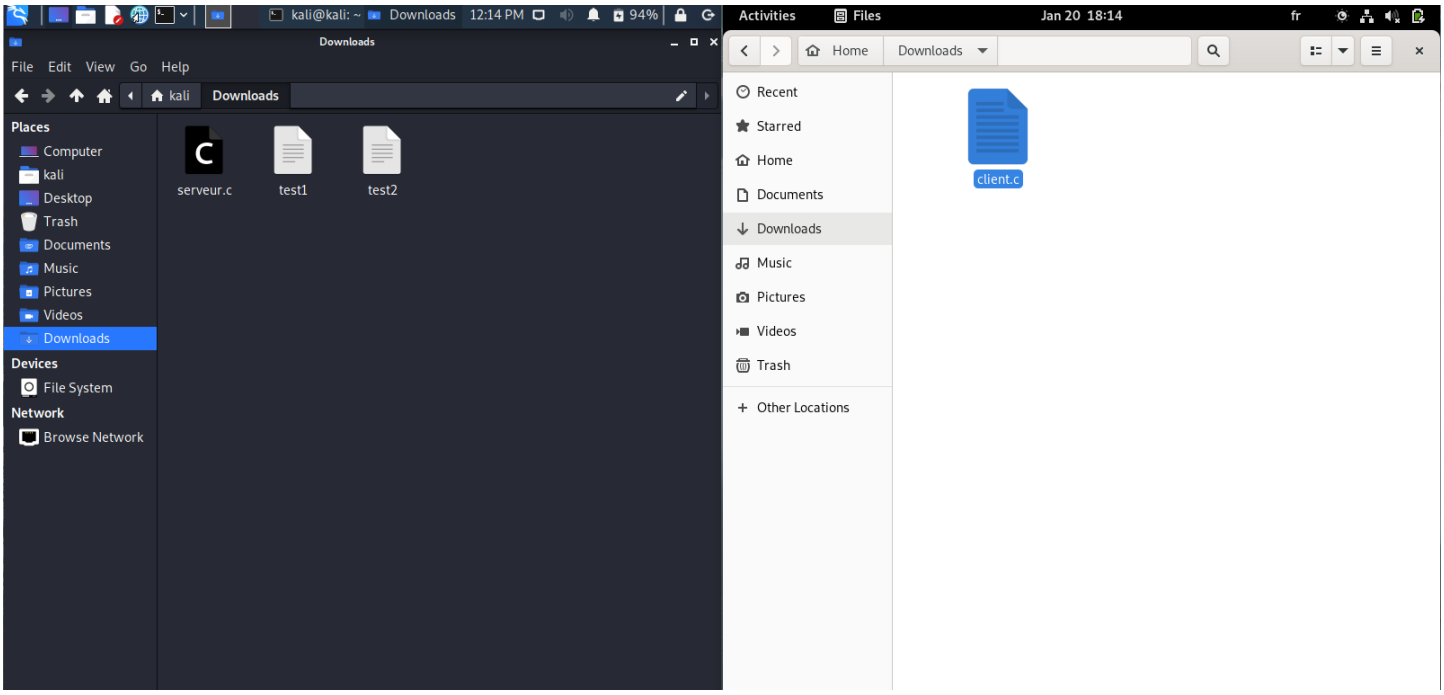
Maintenant on cherche leur adresse IP avec la commande ‘ifconfig’ :



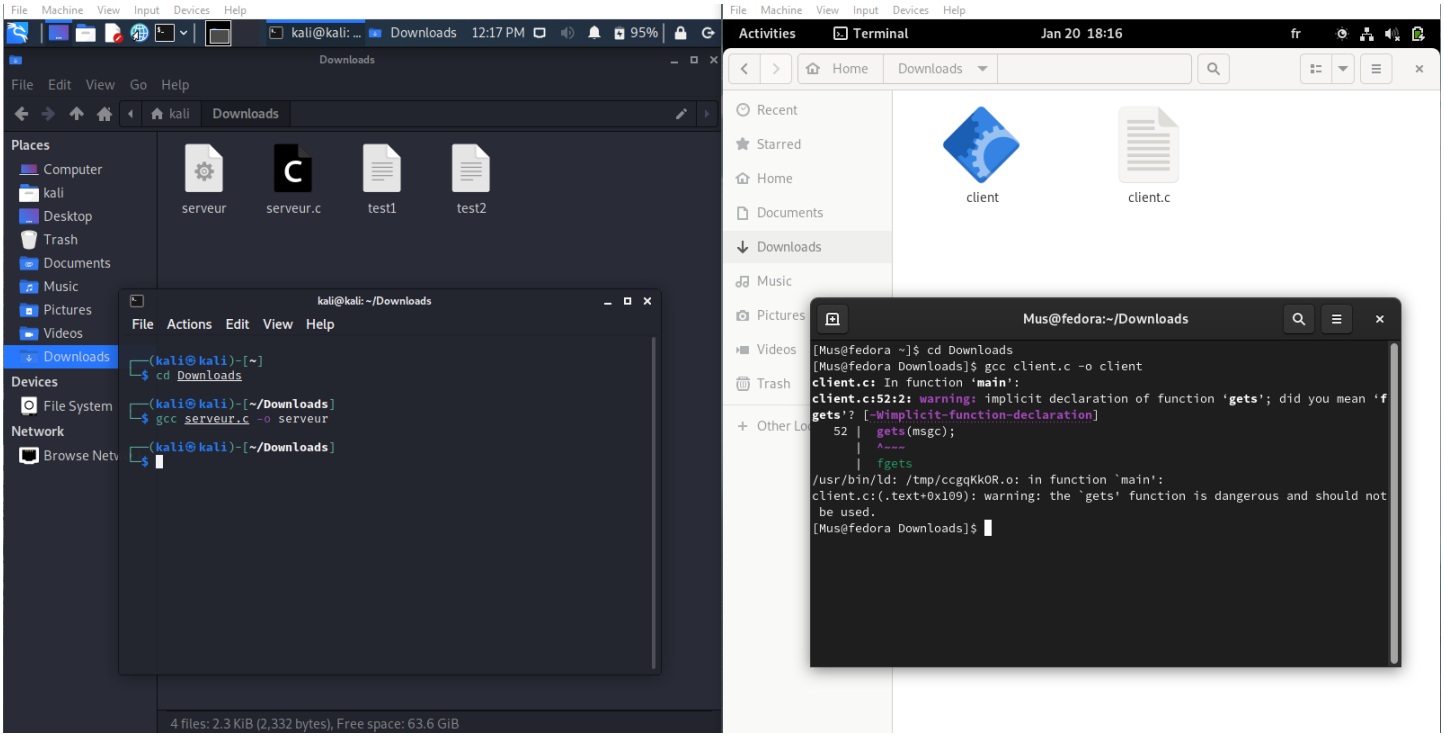
On va utiliser la machine Kali linux comme un serveur donc on doit écrire son adresse IP dans le programme ‘client’ de la machine Fedora :



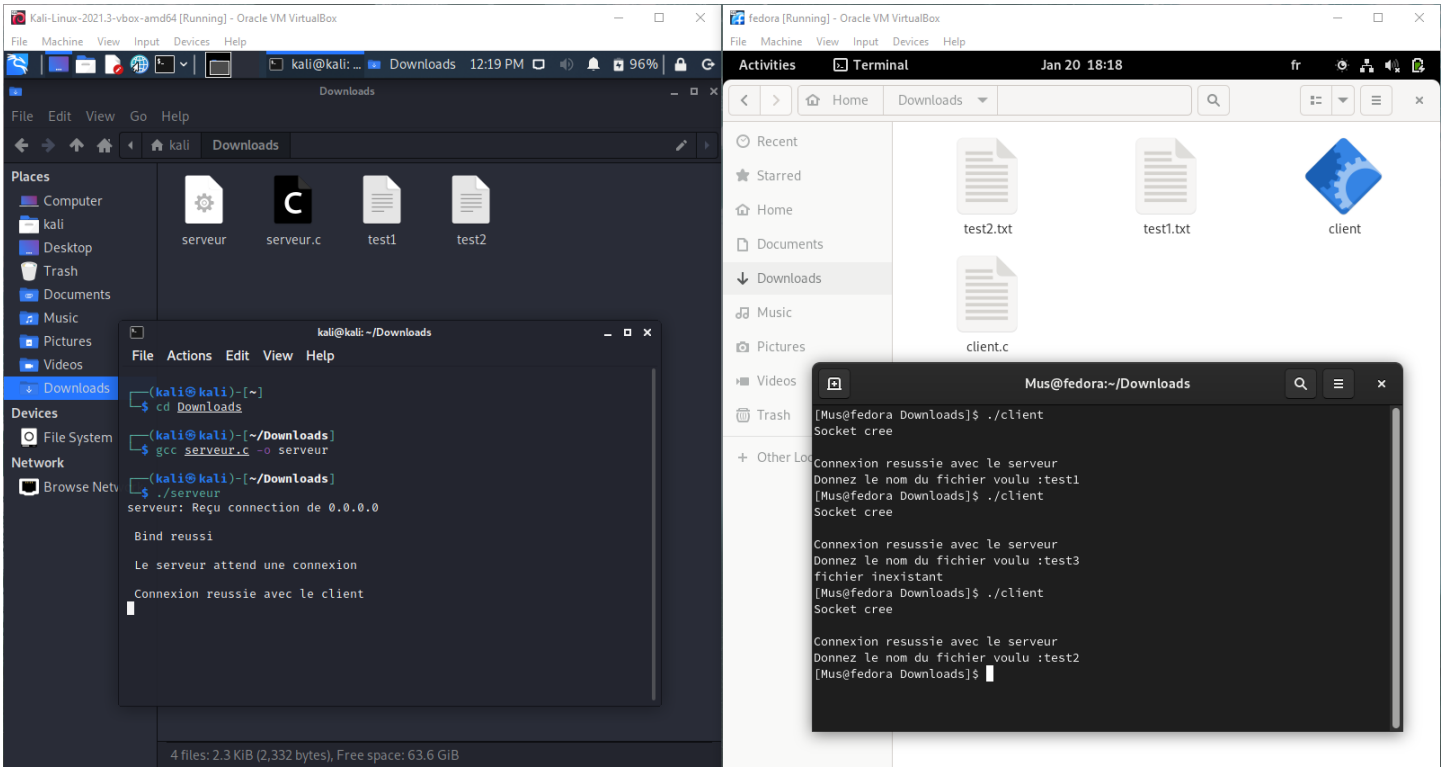
Puis on crée nos répertoires :



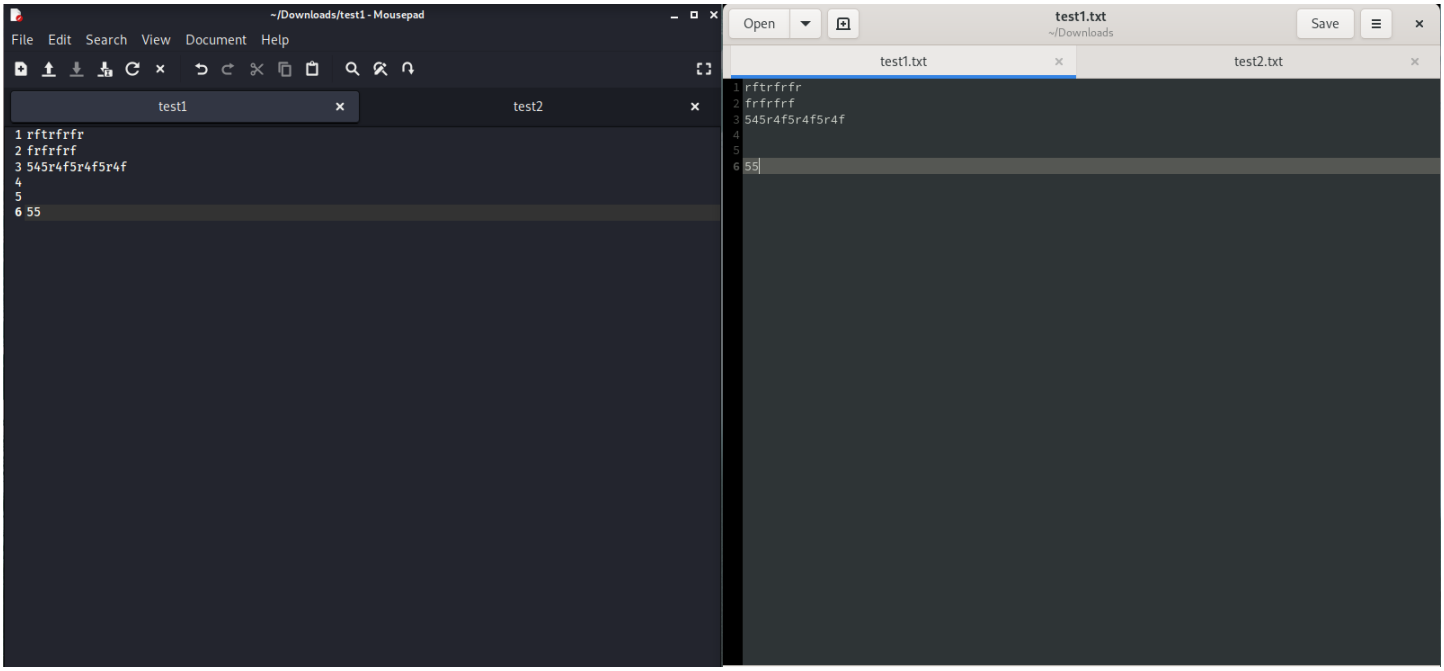
On compile les deux programmes :



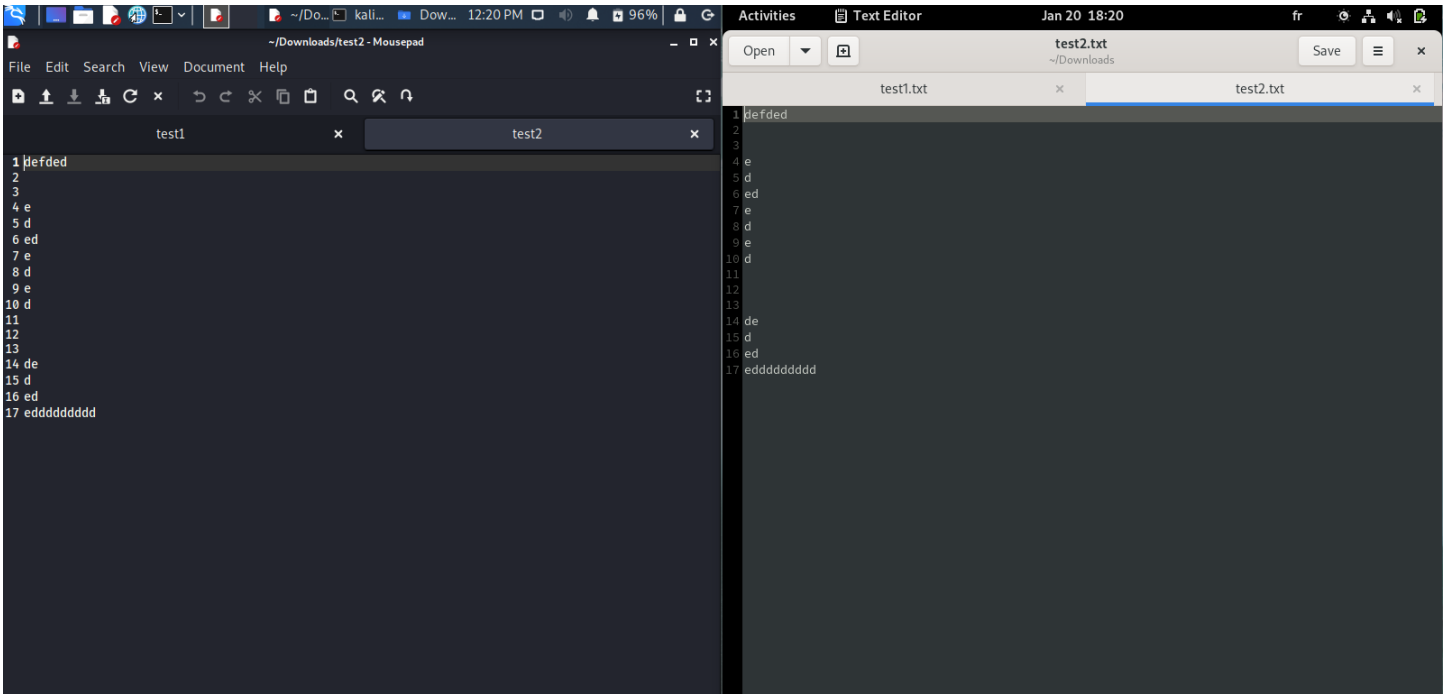
Et puis on les exécute :



On test les fichiers crée :
Test1 :


~/Downloads/test1 - Mousepad
File Edit Search View Document Help
test1
test2
1 rftrfrfr
2 frfrfrf
3 545r4f5r4f5r4f
4
5
6 55
Open
test1.txt
~/Downloads
Save
test1.txt
test2.txt
1 rftrfrfr
2 frfrfrf
3 545r4f5r4f5r4f
4
5
6 55

Test2 :


~/Downloads/test2 - Mousepad
File Edit Search View Document Help
test1
test2
1 defded
2
3
4 e
5 d
6 ed
7 e
8 d
9 e
10 d
11
12
13
14 de
15 d
16 ed
17 eddddddddd
Activities
Text Editor
Jan 20 18:20
fr
Open
test2.txt
~/Downloads
Save
test1.txt
test2.txt
1 defded
2
3
4 e
5 d
6 ed
7 e
8 d
9 e
10 d
11
12
13
14 de
15 d
16 ed
17 eddddddddd