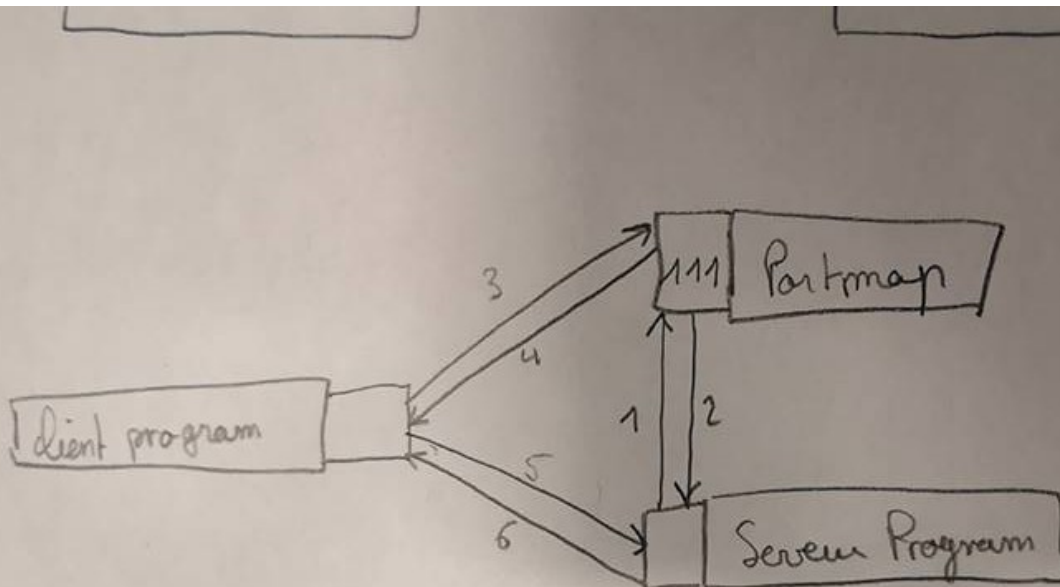


## TD10

Q1) On remarque que lorsque la connexion est rompue portmap permet de renvoyer une requête pour jusqu'à ce que le programme distant réponde



- 1: Le serveur demande à portmap de supprimer le port qu'il lui est attribué si il y en a un
- 2 : Portmap renvoie la suppression de ce port
- 3: Le serveur demande à portmap de lui attribuer un port pour le protocole UDP
- 4: portmap lui renvoie le port demandé, qu'il peut utiliser pour UDP
- 5: Le serveur demande à portmap un port pour le protocole TCP
- 6: Portmap renvoie au programme le port demandé, qu'il peut utiliser pour TCP

Q2) Le port enregistrés par le serveur auprès de portmap pour UDP est : 40978  
et 51592 pour TCP

Q3) Ici le premier paquet correspond au client qui demande à portmap sur quelle port du serveur il peut envoyer une requête UDP. Le deuxième paquet est la réponse de portmap qui lui renvoie le port en question.

Q4)

Les commandes que le clients peut utiliser sont contenus dans le fichier CodeClient.hc

la commande I permet d'initialiser un dictionnaire

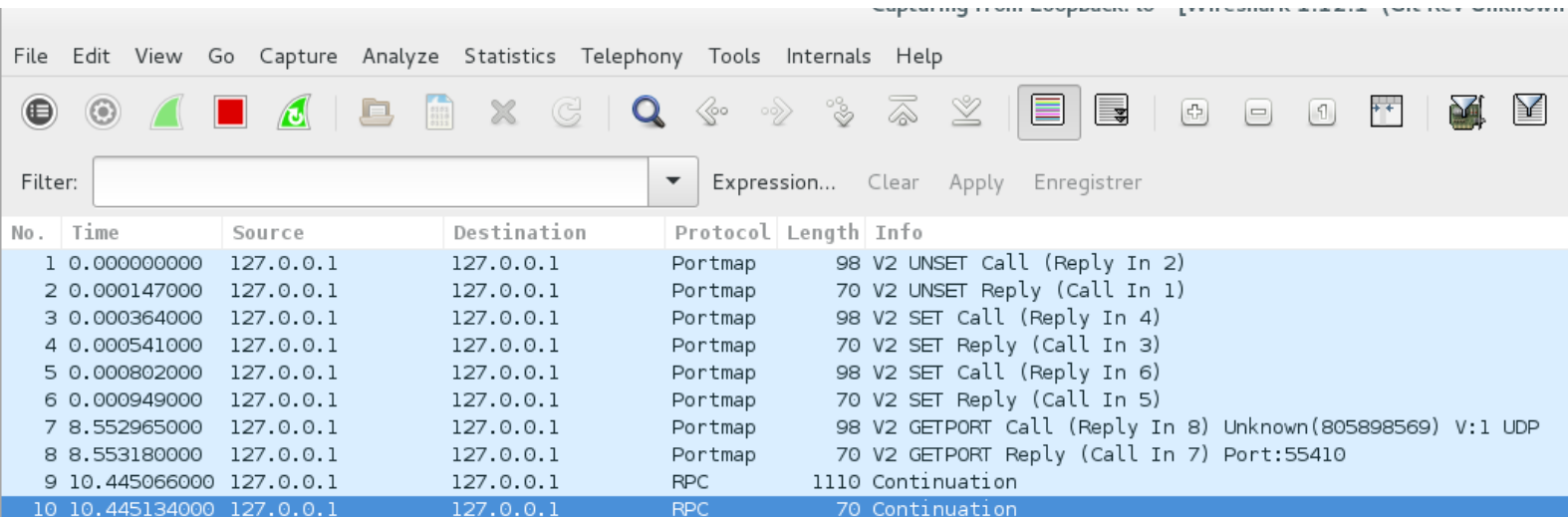
la commande i permet d'insérer des données

la commande d permet de supprimer des données

la commande l permet de rechercher des données

la commande q permet de sortir du programme

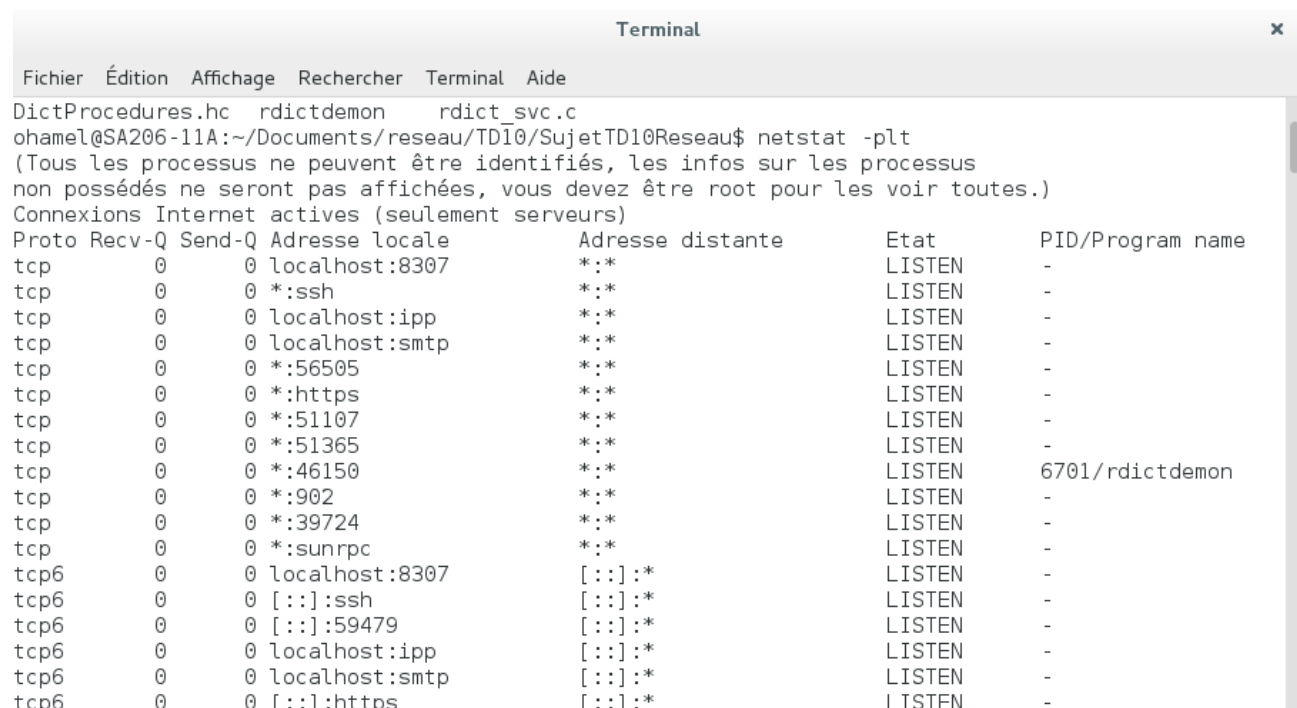
Le protocole utilisé entre le client et le serveur est UDP



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	Portmap	98	V2 UNSET Call (Reply In 2)
2	0.000147000	127.0.0.1	127.0.0.1	Portmap	70	V2 UNSET Reply (Call In 1)
3	0.000364000	127.0.0.1	127.0.0.1	Portmap	98	V2 SET Call (Reply In 4)
4	0.000541000	127.0.0.1	127.0.0.1	Portmap	70	V2 SET Reply (Call In 3)
5	0.000802000	127.0.0.1	127.0.0.1	Portmap	98	V2 SET Call (Reply In 6)
6	0.000949000	127.0.0.1	127.0.0.1	Portmap	70	V2 SET Reply (Call In 5)
7	8.552965000	127.0.0.1	127.0.0.1	Portmap	98	V2 GETPORT Call (Reply In 8) Unknown(805898569) V:1 UDP
8	8.553180000	127.0.0.1	127.0.0.1	Portmap	70	V2 GETPORT Reply (Call In 7) Port:55410
9	10.445066000	127.0.0.1	127.0.0.1	RPC	1110	Continuation
10	10.445134000	127.0.0.1	127.0.0.1	RPC	70	Continuation

Q5) Wireshark ne capture aucun paquet car le serveur n'a pas besoin de demander un port à portmap puisqu'il n'utilise plus ce protocole.

Q6)



Proto	Recv-Q	Send-Q	Adresse locale	Adresse distante	Etat	PID/Program name
tcp	0	0	localhost:8307	*:*	LISTEN	-
tcp	0	0	*:ssh	*:*	LISTEN	-
tcp	0	0	localhost:ipp	*:*	LISTEN	-
tcp	0	0	localhost:smtp	*:*	LISTEN	-
tcp	0	0	*:56505	*:*	LISTEN	-
tcp	0	0	*:https	*:*	LISTEN	-
tcp	0	0	*:51107	*:*	LISTEN	-
tcp	0	0	*:51365	*:*	LISTEN	-
tcp	0	0	*:46150	*:*	LISTEN	6701/rdictdemon
tcp	0	0	*:902	*:*	LISTEN	-
tcp	0	0	*:39724	*:*	LISTEN	-
tcp	0	0	*:sunrpc	*:*	LISTEN	-
tcp6	0	0	localhost:8307	:::*	LISTEN	-
tcp6	0	0	:::ssh	:::*	LISTEN	-
tcp6	0	0	:::59479	:::*	LISTEN	-
tcp6	0	0	localhost:ipp	:::*	LISTEN	-
tcp6	0	0	localhost:smtp	:::*	LISTEN	-
tcp6	0	0	:::https	:::*	LISTEN	-

On trouve que le numéro de port pour le serveur est 46150

Q7) D'après `rdict.c` le premier argument est l'adresse ip du serveur donc ici « localhost » et le deuxième est le port, donc celui trouvé à la question 6 avec `netstat`

explication de `clnttcp_create` :

Cette routine crée un client RPC pour un programme distant. Le client utilise TCP/IP pour le transport. Le programme distant se trouve à l'adresse Internet *\*addr*. L'argument *sockp* est une socket. Comme les RPC basées sur TCP utilisent des entrées-sorties avec tampons, l'utilisateur peut spécifier la taille des tampons d'entrée et de sortie avec les paramètres *sendsz* et *recvsz*. Des valeurs nulles réclament l'utilisation de tampons de tailles optimales. Cette routine renvoie NULL si elle échoue.

Q8) En utilisant des commandes avec le client on peut remarquer que le client et le serveur communiquent le protocole TCP