Socket Programming

Sockets or virtual ports are used by TCP or UDP for maintaining end to end connection between devices for communication. We know that the transport layer is the fourth layer from the bottom in the Open Systems Interconnection (OSI) model and responsible for the process to process delivery, congestion control, error and flow control, and data integrity. All kinds of communication include a source and a destination. It is not different for computer networks and data communication too. For TCP and UDP, there is a source port and destination port used for socket connections. Computers in the same LAN are provided by a unique IP address which is used along with the MAC address when inter LAN communication is required. The IP address and port number play important role in data delivery, IP address helps in locating the destination address from a large number of devices/host available in the world whereas after selecting the destination device port number is used to deliver data to the specific process on that particular host.

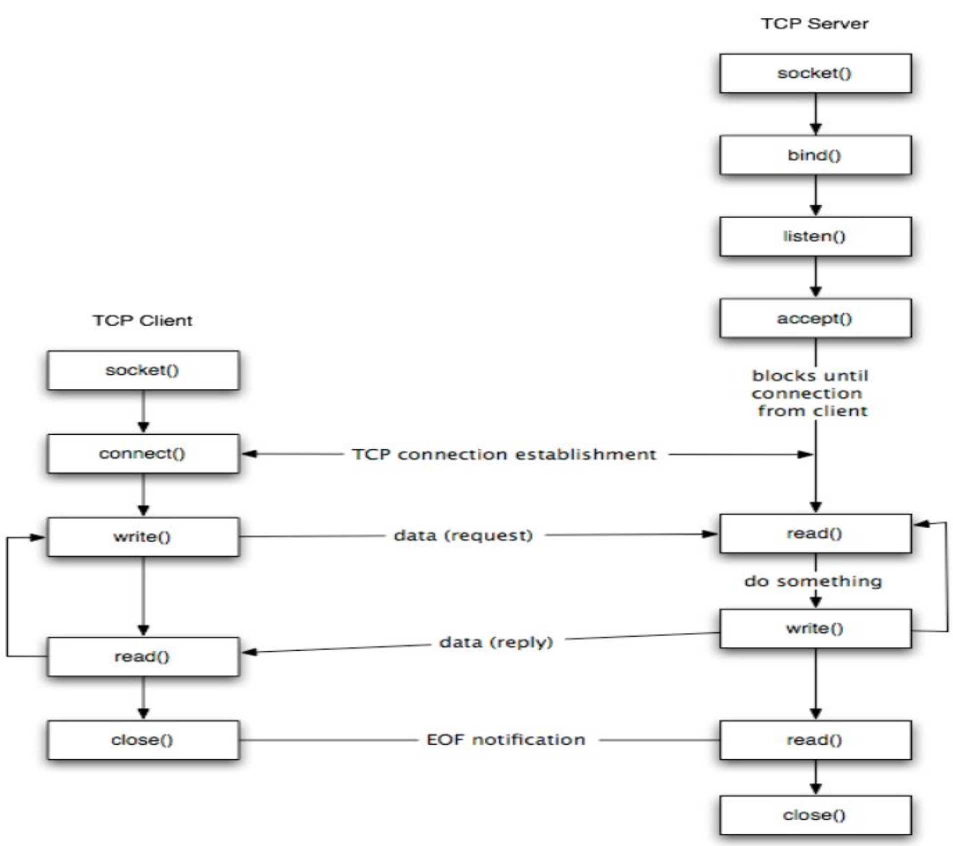


Fig 1: TCP Sockets Server - Client Implementation

The most common method to display process to process delivery is a server-client paradigm. In this socket program, we are trying to establish a connection between a server and multiple clients. Linux environment (Ubuntu) is used for working with POSIX compliant sockets and threading libraries to implement the final code. The ports ranging from 49,152 to 65,535 are neither controlled nor registered so they can be used by any process. These are called dynamic ports. We will be using port number 50000 for our purpose. There will be an authentication check for the client before connecting to the server. After the client-server connection is established the client can perform tasks like downloading the file from the server (tget function) and uploading files to the server (tput function). Error handling is done so that when an error occurs the client will know what went wrong and try again.

There are 2 types of TCP/IP sockets: Stream Sockets (uses TCP), provides reliable byte stream service, and Datagram Sockets (uses UDP), unreliable delivery of data.
In fig 1.,
bind -> attach the local address to the socket along with other parameters necessary for socket data transfer
listen -> waits for client connection by actively listening on the port
accept -> accepts the connection from another socket
connect -> establishes a connection for transfer of data
send -> sends data over the connection channel
receive -> receives data over the connection channel
close -> releases the socket connection

socket() creates a socket descriptor, an integer while functions like a file handle. If it fails it will return -1.

How concurrency is implemented?

There are two types of servers: Iterative server and Concurrent server. When the iterative server accepts connection from the client it blocks any other clients who want to connect to the server. Until the server handles the current client functions it cannot go back to listening for other clients. The iterative server is useful when the client's function takes very little time. So even though implementing iterative server is simple the performance limitation can hinder its usage.

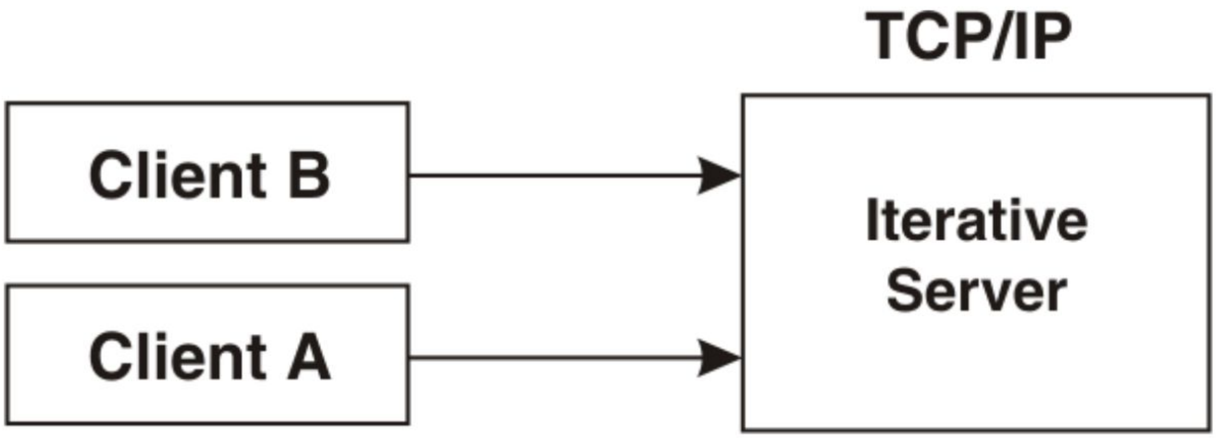


Fig 2: Iterative Server

For this reason, concurrent servers became popular. In concurrent server a separate thread is created every time a client is connected to the server, allowing it to deal with multiple clients simultaneously. A Thread Group is a set of threads all executing inside the same process. They all share the same memory and thus can access the same global variables, same heap memory, the same set of file descriptors, etc. All these threads execute in parallel. The advantage of using a thread group instead of a normal serial program is that several operations may be carried out in parallel, and thus events can be handled immediately as they arrive. For example, if we have one thread handling a user interface, and another thread handling database queries, we can execute a heavy query requested by the user, and still respond to user input while the query is executed. The advantage of using a thread group over using a process group is that context switching between threads is much faster than context switching between processes. Also, communications between two threads are usually faster and easier to implement then communications between two processes.

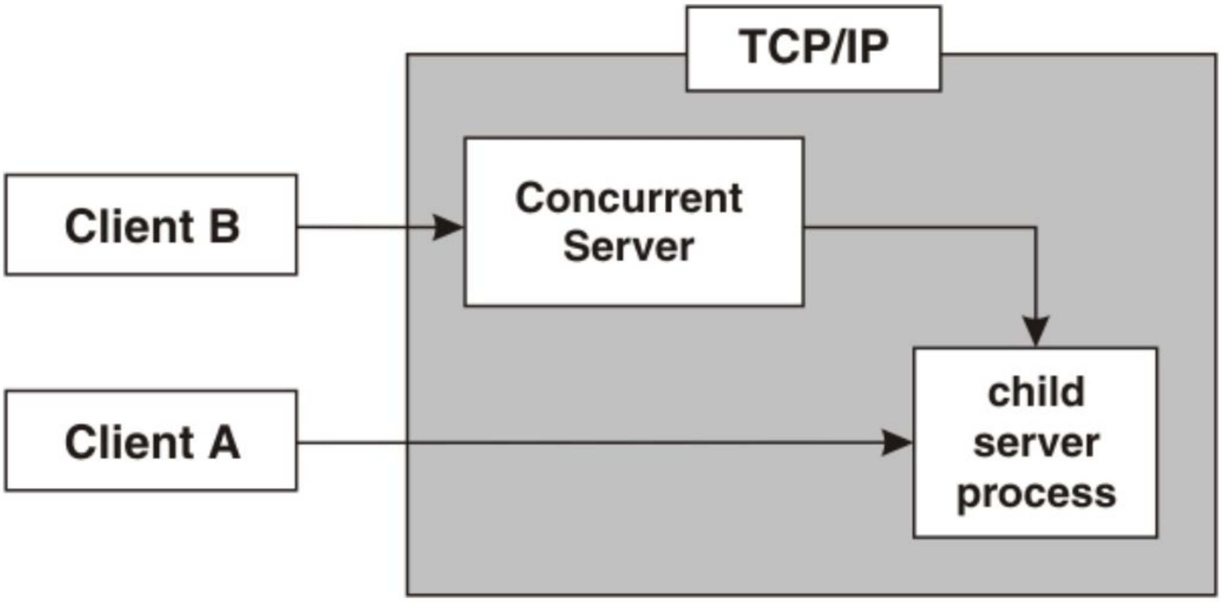


Fig 3: Concurrent Server

POSIX pthread library:
pthread_create() : To create a thread
pthread_exit(): To destroy an existing thread
pthread_mutex_init(): To initialise the MUTEX
pthread_mutex_lock(): Locks a thread to a specific process
pthread_mutex_unlock(): Unlocks the thread from the process so that other threads can use that function.

The arguments used in these functions and other functions of the pthread library can be found at IEEE Open Group Page (Check References).
If proper synchronization is not applied race condition may occur which could result in loss of data integrity with respect to different client threads. MUTEX is used to ensure synchronization so that shared resources can be used effectively.

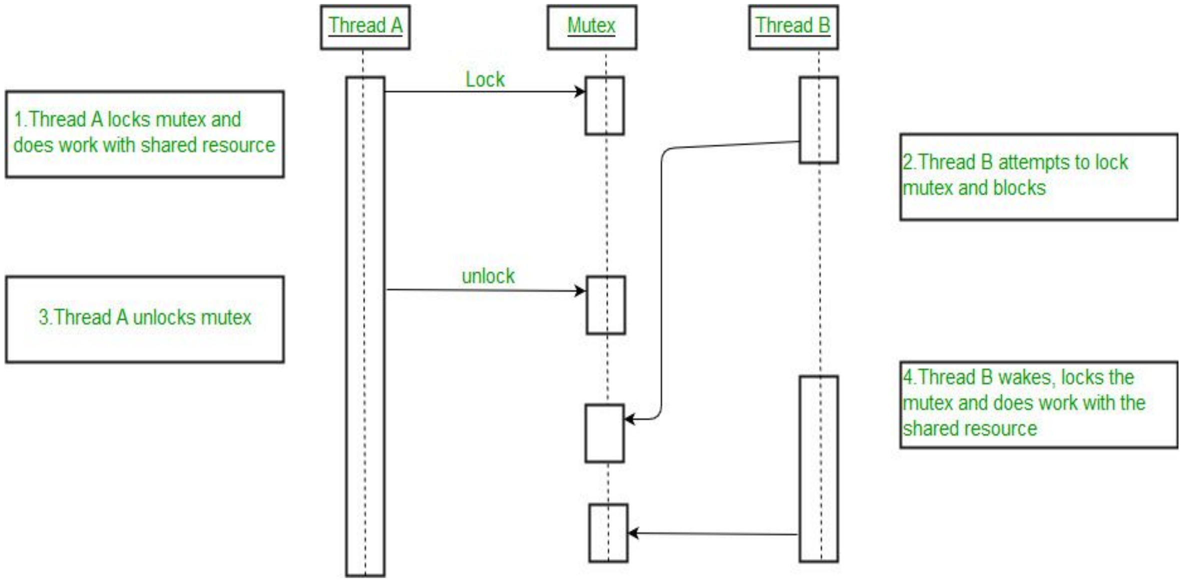


Fig 3: Mutex lock for Linux Thread Synchronization

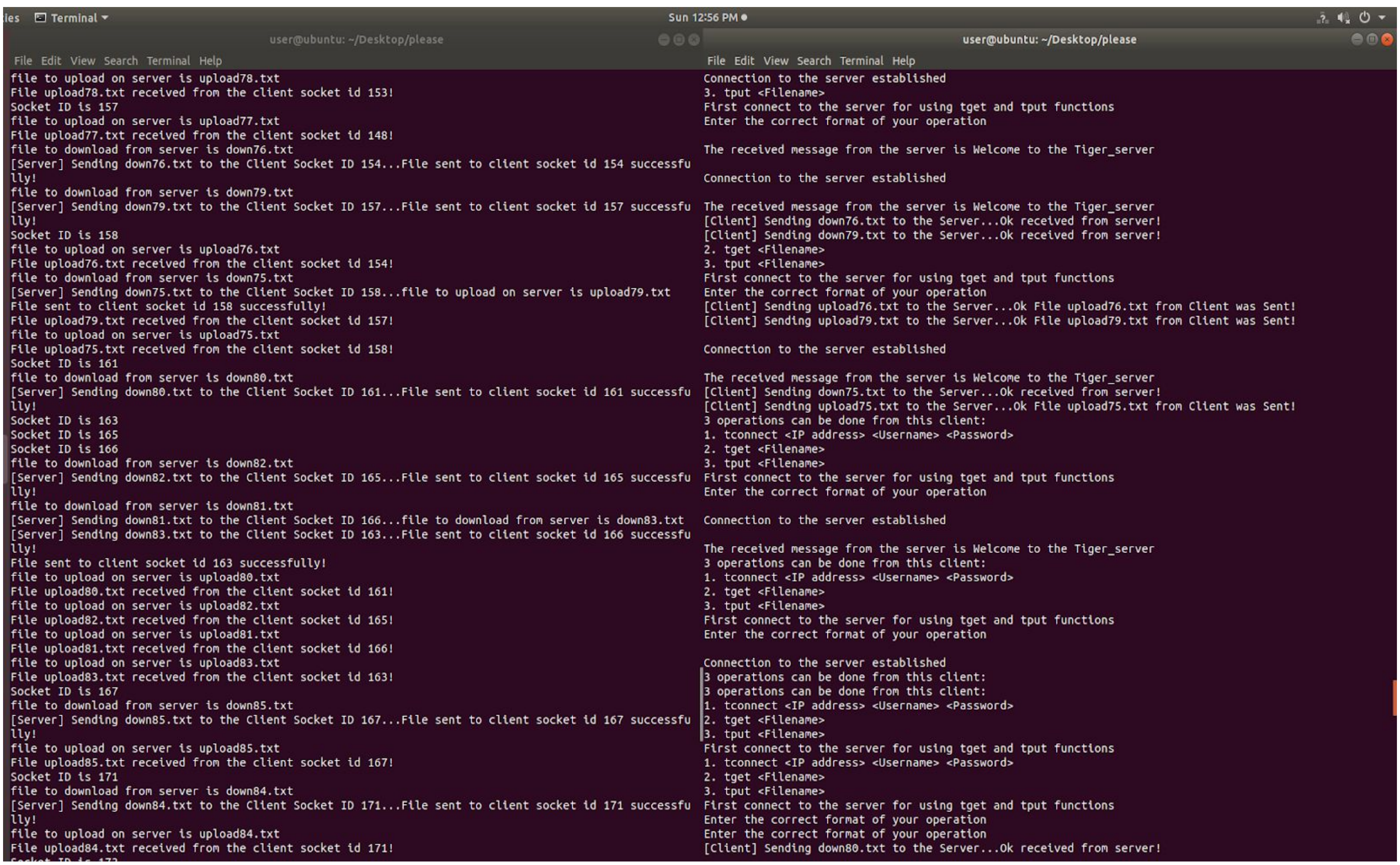


Fig 4: Concurrency using pthread library

From Fig 4 we can see that upload78.txt was uploaded to the server before down76.txt was downloaded at the client. Similarly, we can find such instances where clients with different socket ID get connected concurrently to the server and uploading-downloading of files occur simultaneously for all client threads.

Show that your server can serve at least 100 TigerC clients concurrently and provide a sample run.

2 bash shell scripts are run before executing the main program. These bash shells create 2 folders will 100 files in them on the Desktop.

serv_storage_bash.sh
~/Desktop

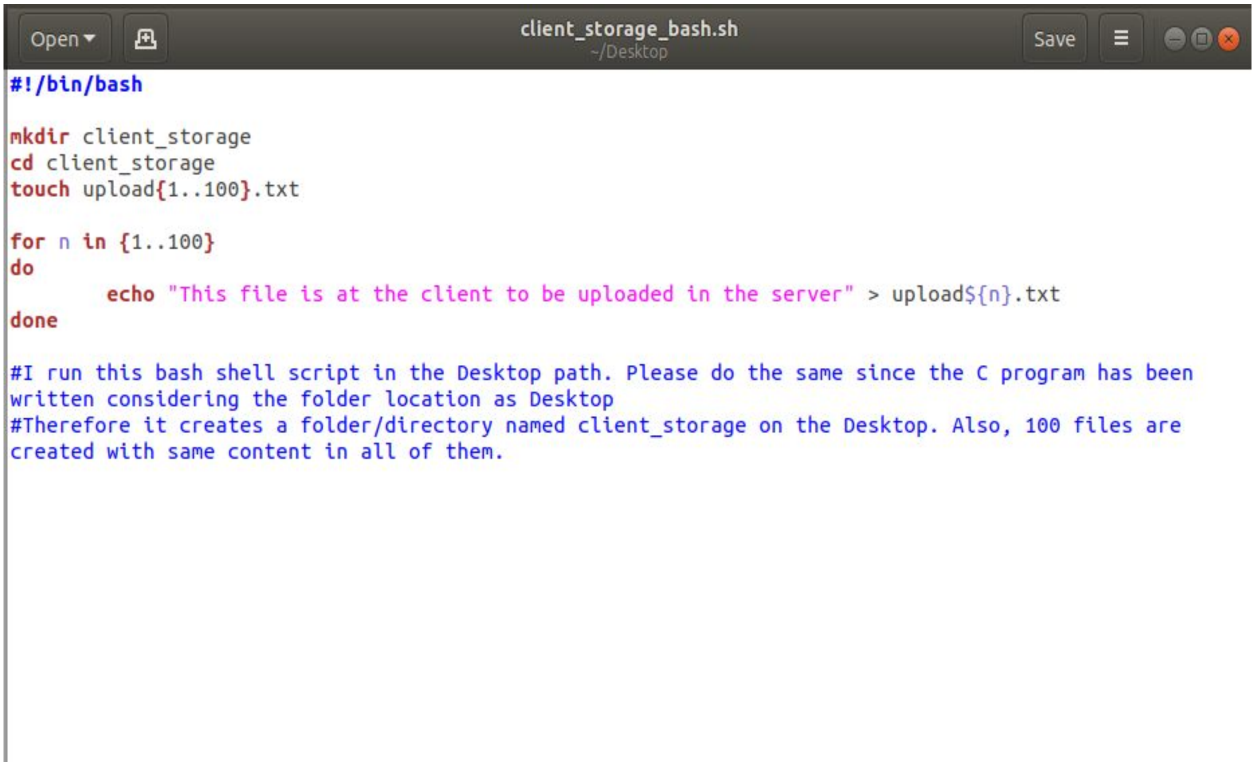
```bash
#!/bin/bash


mkdir server_storage
cd server_storage
touch down{1..100}.txt

for n in {1..100}
do
	echo "This file is at the server to be downloaded by client" > down${n}.txt
done

#I run this bash shell script in the Desktop path. Please do the same since the C program has been written considering the folder location as Desktop
#Therefore it creates a folder/directory named server_storage on the Desktop. Also, 100 files are created with same content in all of them.
```

Fig 5: Bash shell to create 100 files to download at the client




```bash
#!/bin/bash

mkdir client_storage
cd client_storage
touch upload{1..100}.txt

for n in {1..100}
do
        echo "This file is at the client to be uploaded in the server" > upload${n}.txt
done

#I run this bash shell script in the Desktop path. Please do the same since the C program has been written considering the folder location as Desktop
#Therefore it creates a folder/directory named client_storage on the Desktop. Also, 100 files are created with same content in all of them.
```

Fig 6: Bash shell to create 100 files to upload at the server

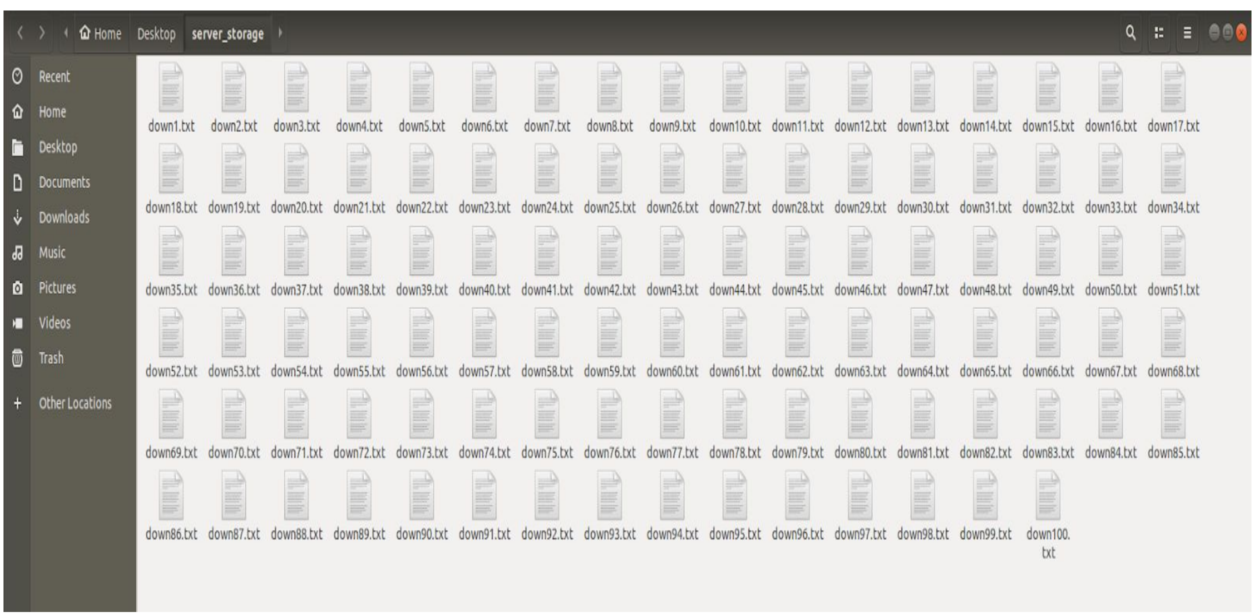


Fig 7: 100 files in the folder server_storage

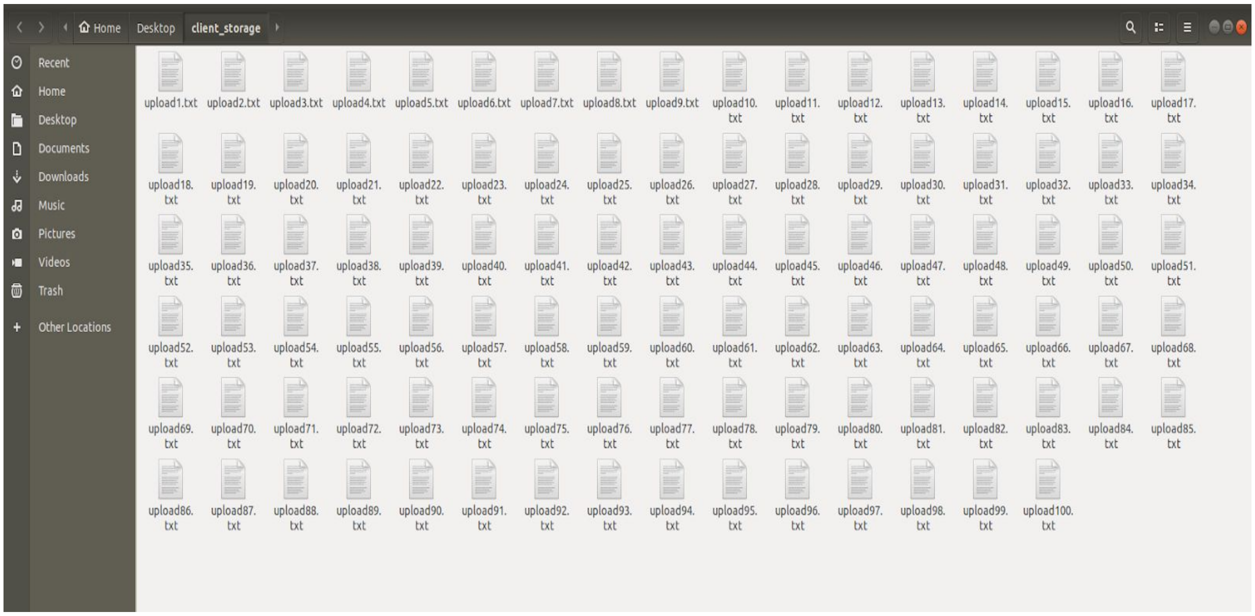


Fig 8: 100 files in the folder client_storage

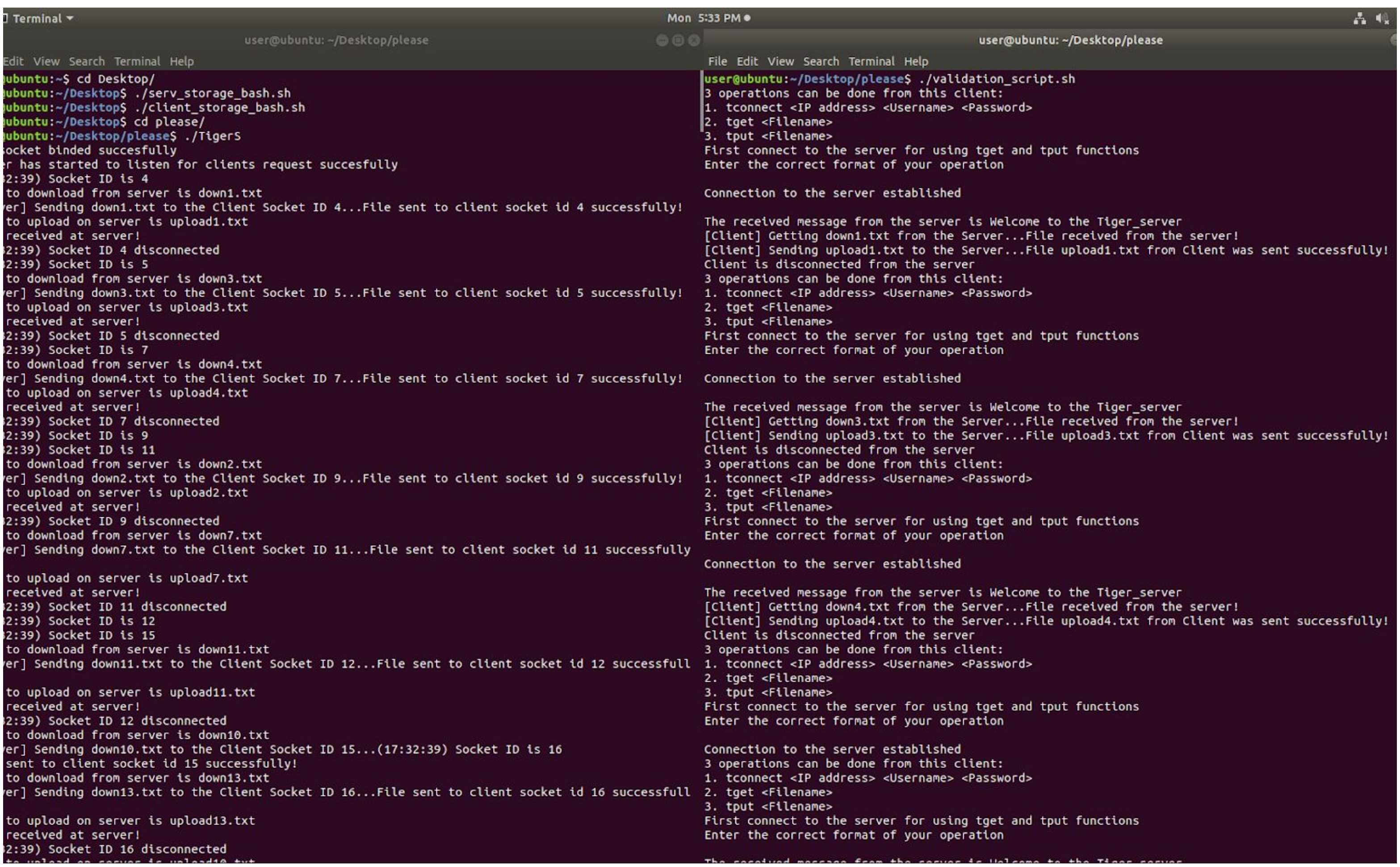


Fig 9a: 100 TigerC clients concurrently connected to the TigerS server

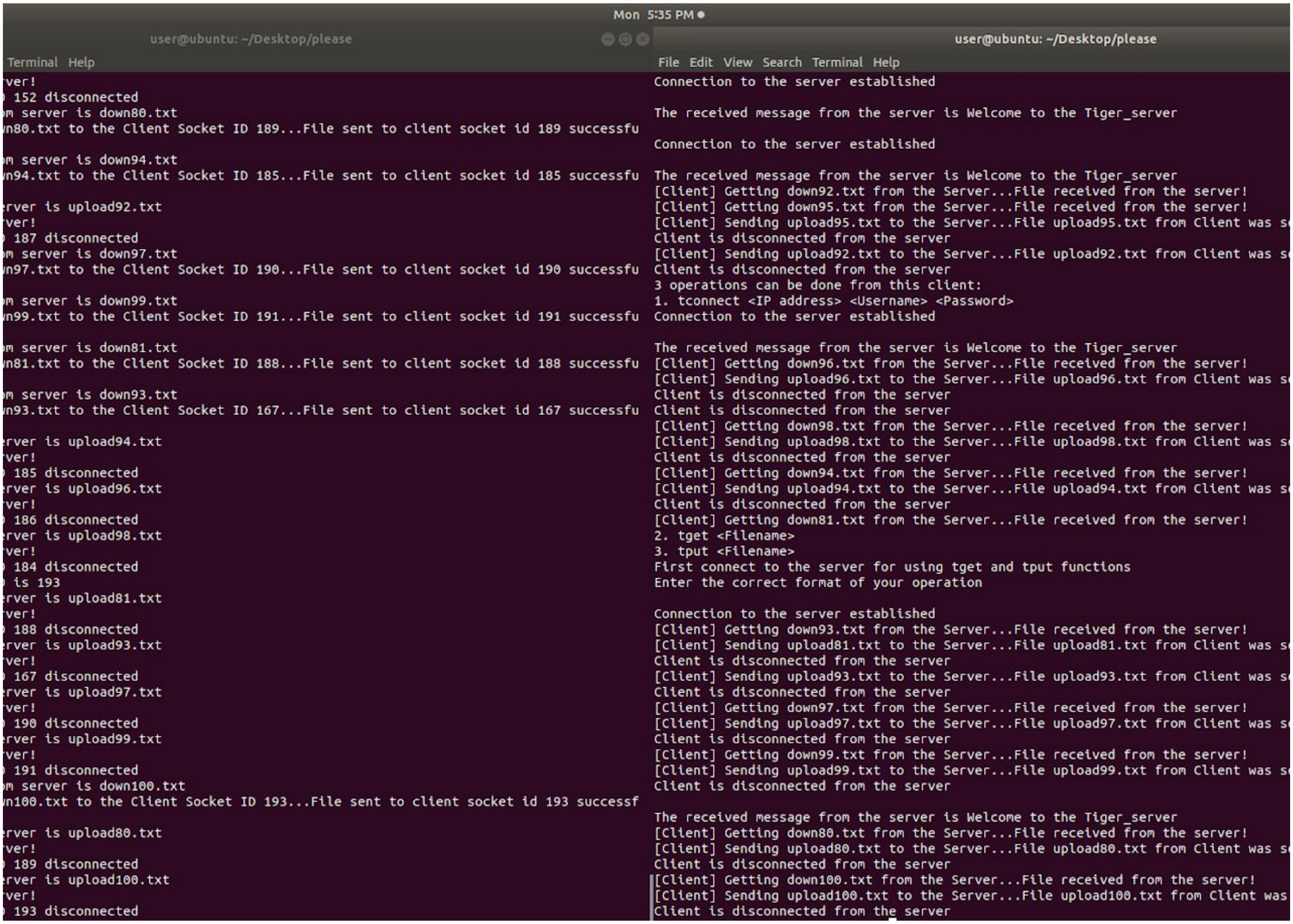


Fig 9b: 100 TigerC clients concurrently connected to the TigerS server

After executing the program the server_storage folder will have copies of 100 upload files from the client and the client_storage folder will have copies of 100 download files from the server.

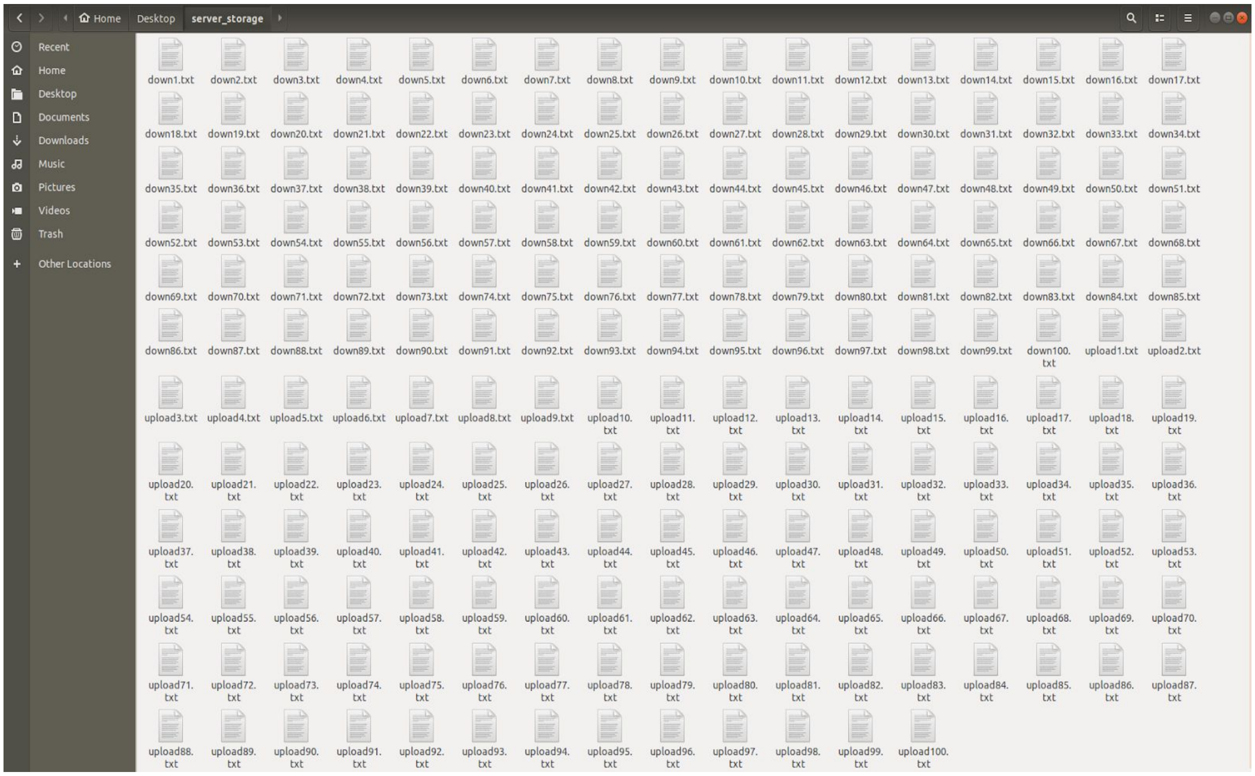


Fig 10: Copies of 100 upload#.txt uploaded on the server

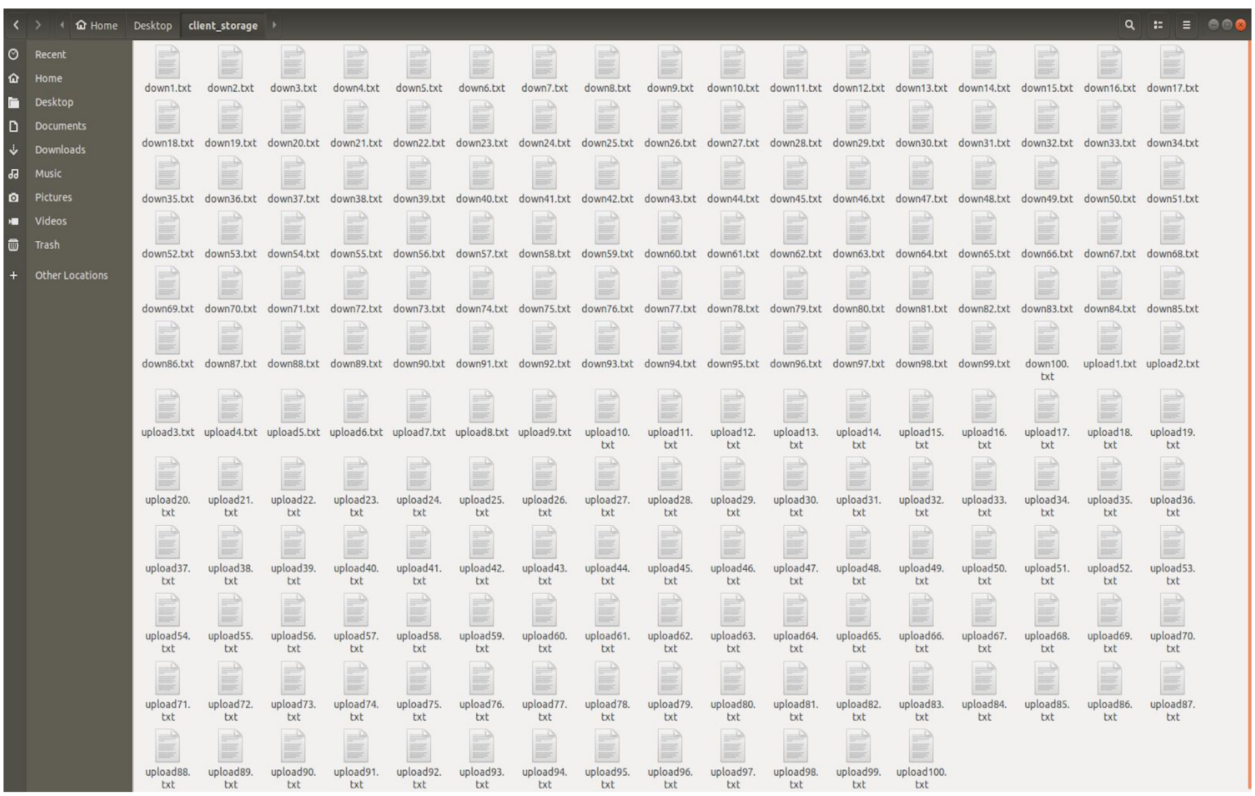


Fig 11: Copies of 100 down#.txt files downloaded at the client

What is the maximum number of clients that your server can serve concurrently when both TigerS and TigerC is run on your laptop or PC? Justify your answer.

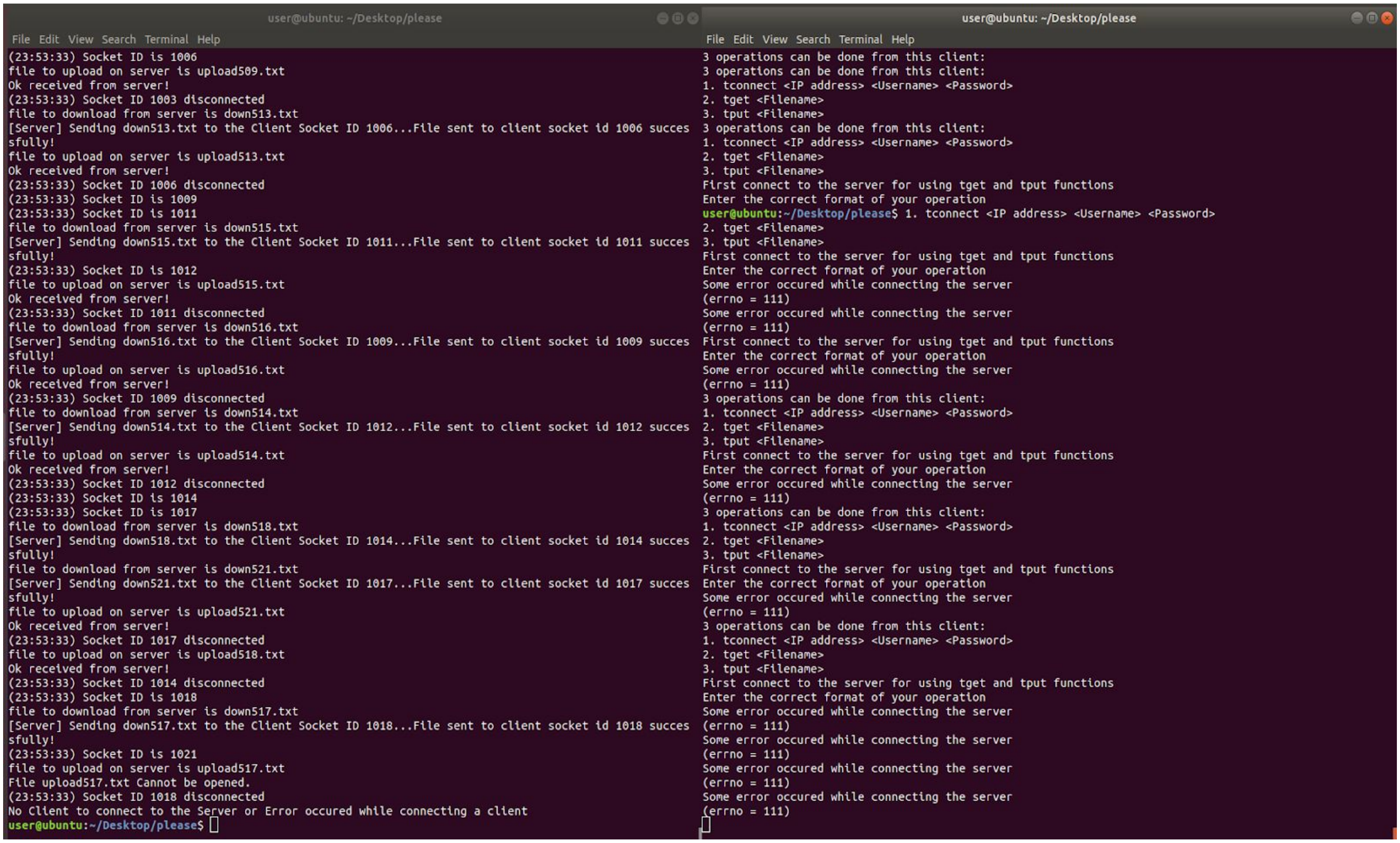


Fig 12: Trying to connect 1000 clients to the server

Using the bash shell script provided for validation I tried to connect 1000 clients concurrently but only 510 clients got connected to the TigerS server. The number of clients connecting to the TCP server theoretically should be 65535 since that is the maximum number of ports available. But individual system hardware and processing capabilities can't be ignored as this program is being executed on a personal laptop with virtual OS with minimum memory and limited speed.

Provide a sample run to demonstrate the execution of each command listed above

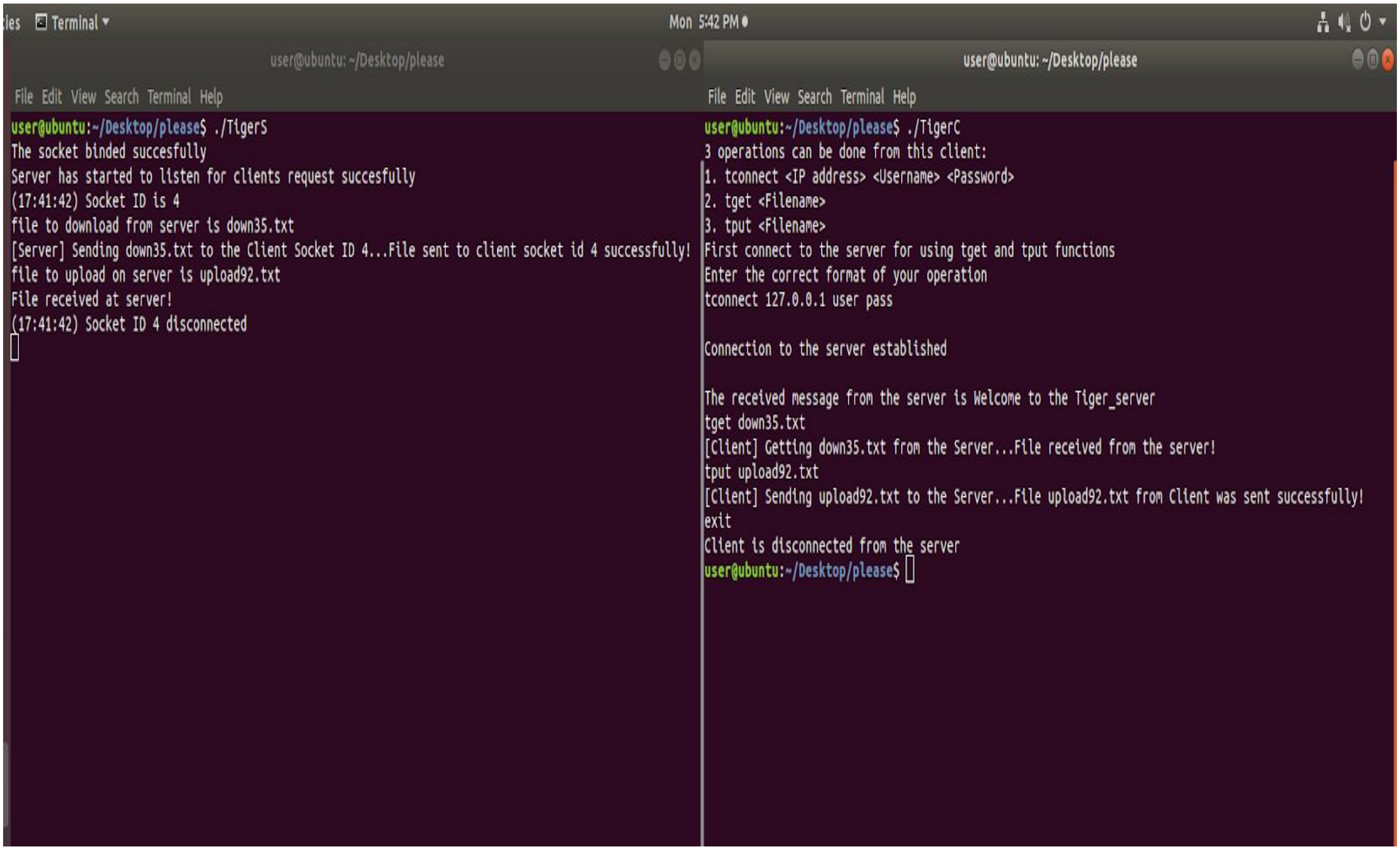


Fig 13: Working of Individual Commands (tconnect, tget,tput)

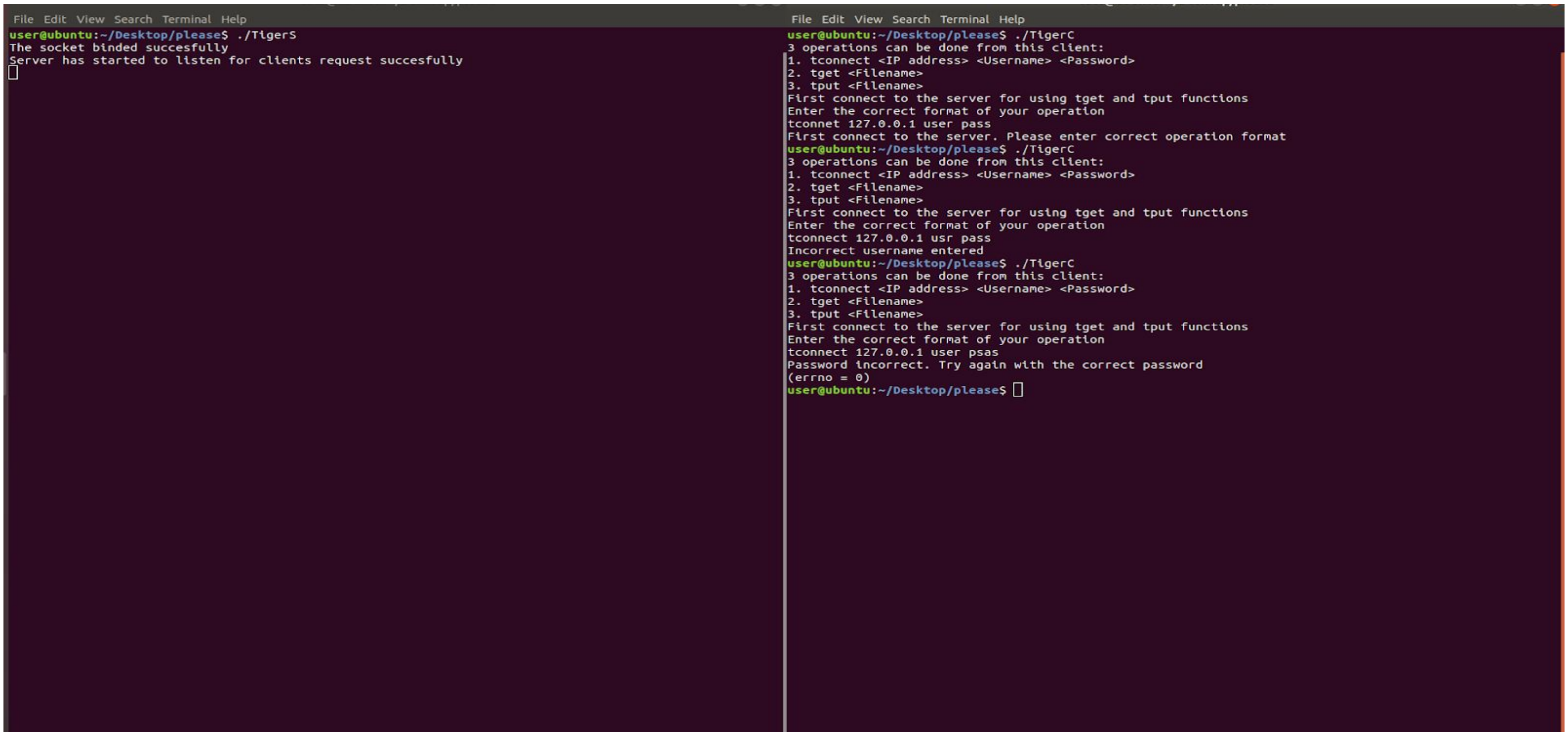


Fig 14a: Few of the error messages for the client

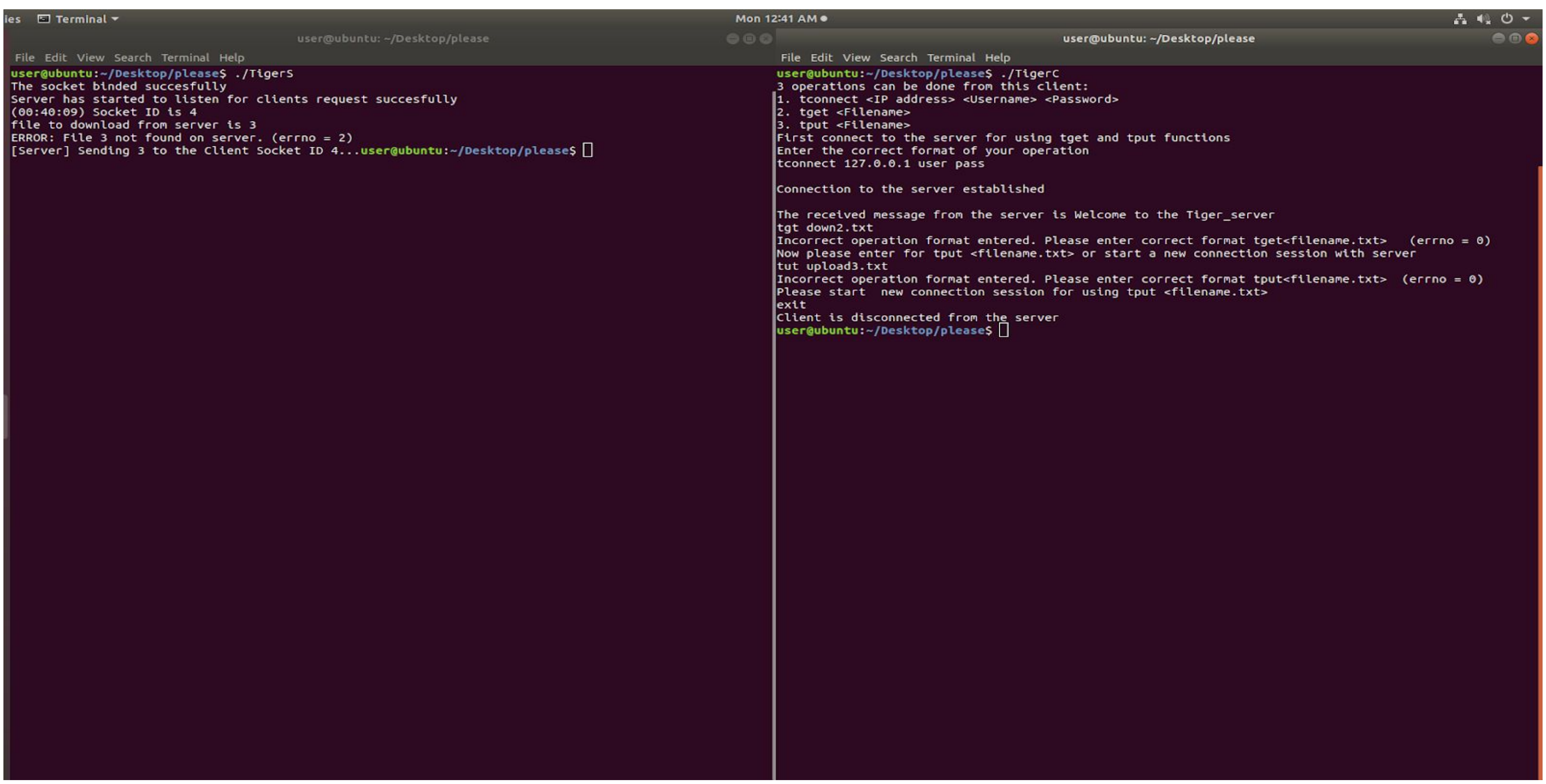


Fig 14b: Few of the error messages for the client

Reference:

1. Microsoft Core Documentation https://docs.microsoft.com/en-us/host-integration-server/core/iterative-vs-concurrent-tcp-ip-models1
2. Cisco Community https://community.cisco.com/
3. RFC 793 Transmission Control Protocol (TCP)
4. RFC 791 Internet Protocol (IP)
5. IBM Knowledge Center https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.2.0/com.ibm.zos.v2r2.hali001/concurrentanditerativeservers.html
6. Socket Programming https://www.cs.dartmouth.edu/~campbell/cs50/socketprogramming.html
7. Bash Shell https://www.linux.com/tutorials/writing-simple-bash-script/
8. Socket Programming https://pubs.opengroup.org/onlinepubs/009695399/functions/socket.html
   15-441 Computer Networks, Spring 2008, Xi Liu
   SAS Institute INC, Cary, NC
9. Concurrent Programming in UNIX - Giuseppe Lipari