

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.

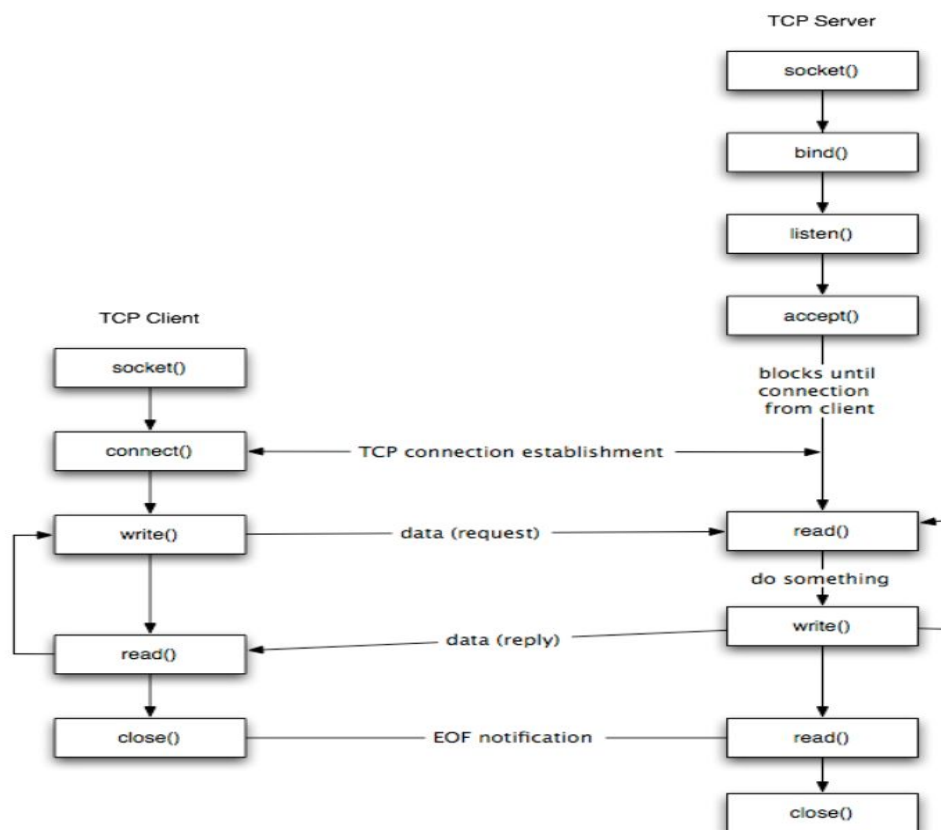


Fig 1: TCP Sockets Server - Client Implementation

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.

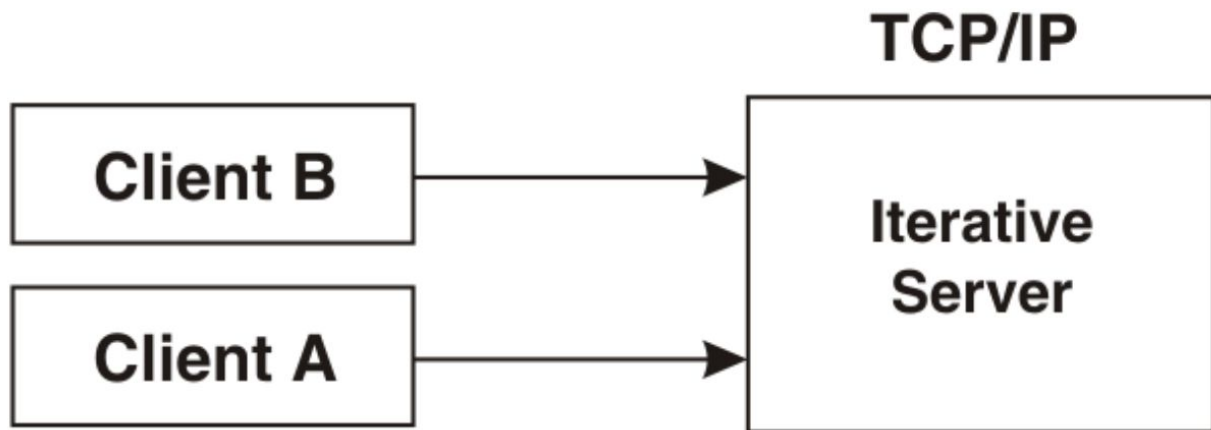


Fig 2: Iterative Server

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 than communications between two processes.

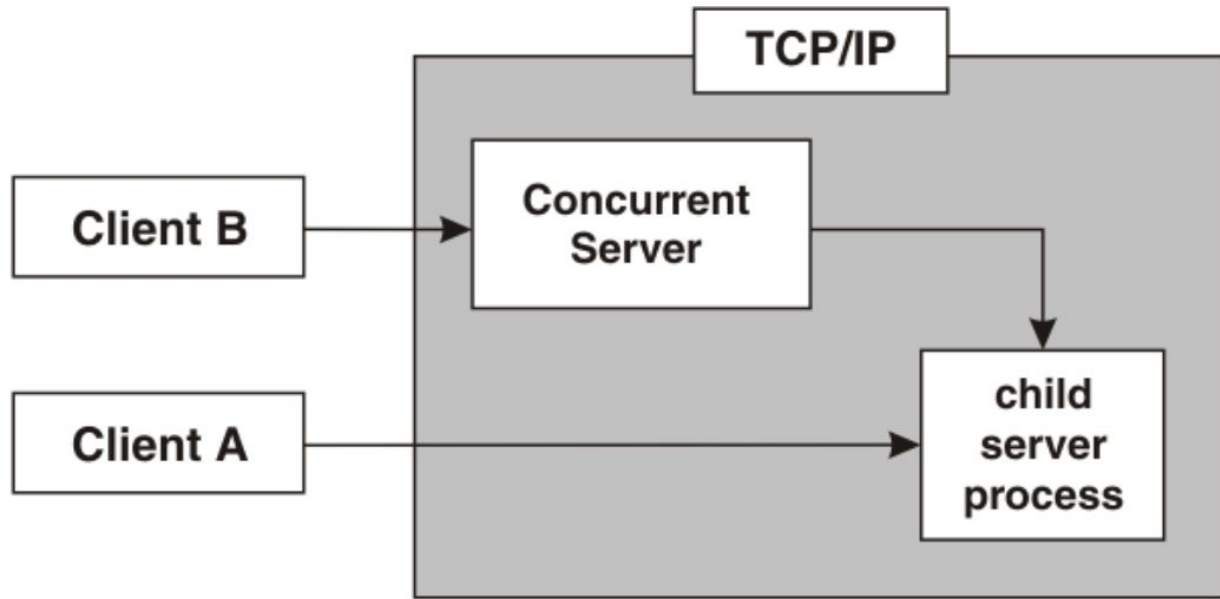


Fig 3: Concurrent Server

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.

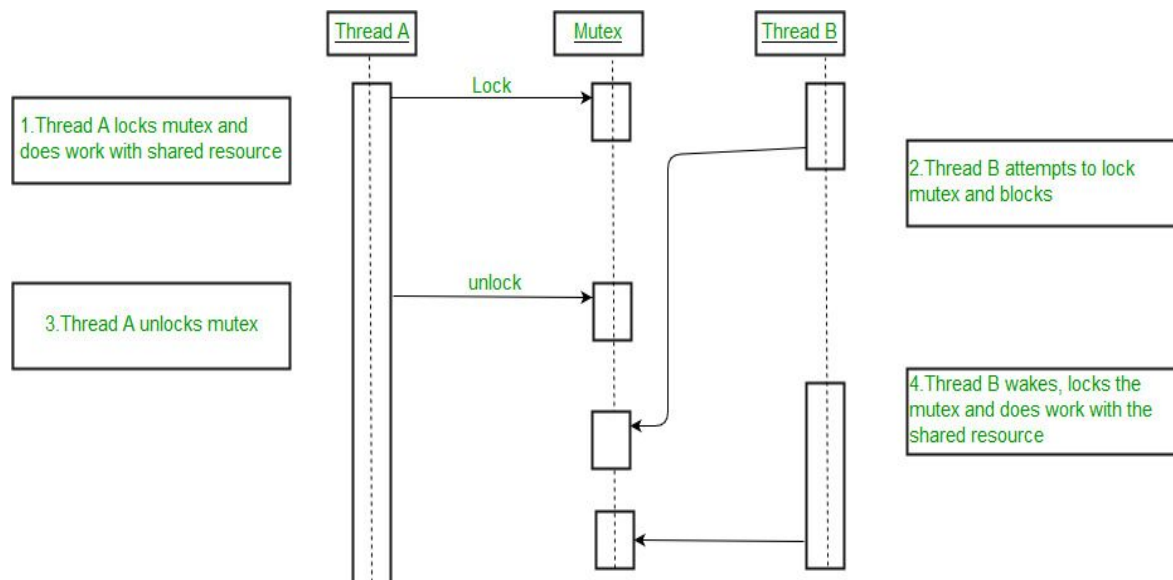


Fig 3: Mutex lock for Linux Thread Synchronization

```

File Edit View Search Terminal Help
user@ubuntu: ~/Desktop/please
File upload on server is upload78.txt
File upload78.txt received from the client socket id 153!
Socket ID is 157
File to upload on server is upload77.txt
File upload77.txt received from the client socket id 148!
File to download from server is down76.txt
[Server] Sending down76.txt to the Client Socket ID 154...File sent to client socket id 154 successfully!
File to download from server is down79.txt
[Server] Sending down79.txt to the Client Socket ID 157...File sent to client socket id 157 successfully!
Socket ID is 158
File to upload on server is upload76.txt
File upload76.txt received from the client socket id 154!
File to download from server is down75.txt
[Server] Sending down75.txt to the Client Socket ID 158...File to upload on server is upload79.txt
File sent to client socket id 158 successfully!
File upload79.txt received from the client socket id 157!
File to upload on server is upload75.txt
File upload75.txt received from the client socket id 158!
Socket ID is 161
File to download from server is down80.txt
[Server] Sending down80.txt to the Client Socket ID 161...File sent to client socket id 161 successfully!
Socket ID is 163
Socket ID is 165
Socket ID is 166
File to download from server is down82.txt
[Server] Sending down82.txt to the Client Socket ID 165...File sent to client socket id 165 successfully!
File to download from server is down81.txt
[Server] Sending down81.txt to the Client Socket ID 166...File to download from server is down83.txt
[Server] Sending down83.txt to the Client Socket ID 163...File sent to client socket id 166 successfully!
File sent to client socket id 163 successfully!
File to upload on server is upload80.txt
File upload80.txt received from the client socket id 161!
File to upload on server is upload82.txt
File upload82.txt received from the client socket id 165!
File to upload on server is upload81.txt
File upload81.txt received from the client socket id 166!
File to upload on server is upload83.txt
File upload83.txt received from the client socket id 163!
Socket ID is 167
File to download from server is down85.txt
[Server] Sending down85.txt to the Client Socket ID 167...File sent to client socket id 167 successfully!
File to upload on server is upload85.txt
File upload85.txt received from the client socket id 167!
Socket ID is 171
File to download from server is down84.txt
[Server] Sending down84.txt to the Client Socket ID 171...File sent to client socket id 171 successfully!
File to upload on server is upload84.txt
File upload84.txt received from the client socket id 171!
Socket ID is 173

File Edit View Search Terminal Help
user@ubuntu: ~/Desktop/please
Connection to the server established
3. tput <Filename>
First connect to the server for using tget and tput functions
Enter the correct format of your operation

The received message from the server is Welcome to the Tiger_server
Connection to the server established

The received message from the server is Welcome to the Tiger_server
[Client] Sending down76.txt to the Server...Ok received from server!
[Client] Sending down79.txt to the Server...Ok received from server!
2. tget <Filename>
3. tput <Filename>
First connect to the server for using tget and tput functions
Enter the correct format of your operation
[Client] Sending upload76.txt to the Server...Ok File upload76.txt from Client was Sent!
[Client] Sending upload79.txt to the Server...Ok File upload79.txt from Client was Sent!

Connection to the server established

The received message from the server is Welcome to the Tiger_server
[Client] Sending down75.txt to the Server...Ok received from server!
[Client] Sending upload75.txt to the Server...Ok File upload75.txt from Client was Sent!
3 operations can be done from this client:
1. tconnect <IP address> <Username> <Password>
2. tget <Filename>
3. tput <Filename>
First connect to the server for using tget and tput functions
Enter the correct format of your operation

Connection to the server established

The received message from the server is Welcome to the Tiger_server
3 operations can be done from this client:
1. tconnect <IP address> <Username> <Password>
2. tget <Filename>
3. tput <Filename>
First connect to the server for using tget and tput functions
Enter the correct format of your operation

Connection to the server established
3 operations can be done from this client:
3 operations can be done from this client:
1. tconnect <IP address> <Username> <Password>
2. tget <Filename>
3. tput <Filename>
First connect to the server for using tget and tput functions
Enter the correct format of your operation
Enter the correct format of your operation
[Client] Sending down80.txt to the Server...Ok received from server!

```

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 with 100 files in them on the Desktop.



```
#!/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


```
client_storage_bash.sh
~/Desktop
Save

#!/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

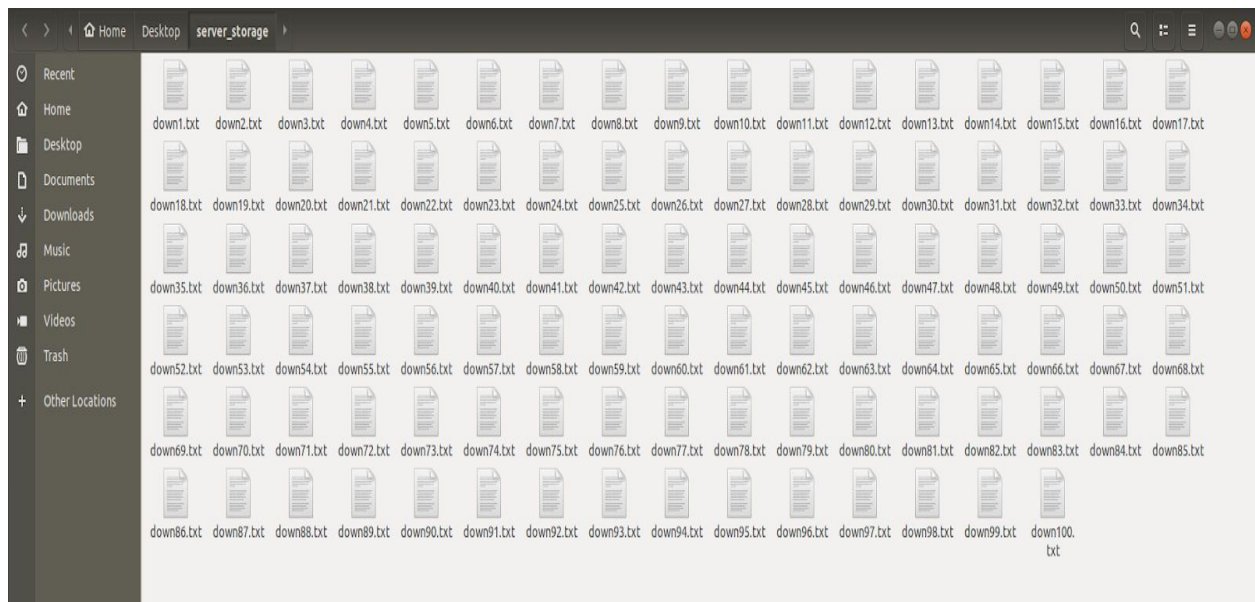


Fig 7: 100 files in the folder server_storage


```
Mon 5:35 PM •
user@ubuntu: ~/Desktop/please
Terminal Help
Server!
152 disconnected
Server is down80.txt
n80.txt to the Client Socket ID 189...File sent to client socket id 189 successfu
Server is down94.txt
n94.txt to the Client Socket ID 185...File sent to client socket id 185 successfu
Server is upload92.txt
Server!
187 disconnected
Server is down97.txt
n97.txt to the Client Socket ID 190...File sent to client socket id 190 successfu
Server is down99.txt
n99.txt to the Client Socket ID 191...File sent to client socket id 191 successfu
Server is down81.txt
n81.txt to the Client Socket ID 188...File sent to client socket id 188 successfu
Server is down93.txt
n93.txt to the Client Socket ID 167...File sent to client socket id 167 successfu
Server is upload94.txt
Server!
185 disconnected
Server is upload96.txt
Server!
186 disconnected
Server is upload98.txt
Server!
184 disconnected
183
Server is upload81.txt
Server!
188 disconnected
Server is upload93.txt
Server!
167 disconnected
Server is upload97.txt
Server!
190 disconnected
Server is upload99.txt
Server!
191 disconnected
Server is down100.txt
n100.txt to the Client Socket ID 193...File sent to client socket id 193 successf
Server is upload80.txt
Server!
189 disconnected
Server is upload100.txt
Server!
193 disconnected

File Edit View Search Terminal Help
Connection to the server established
The received message from the server is Welcome to the Tiger_server
Connection to the server established
The received message from the server is Welcome to the Tiger_server
[Client] Getting down92.txt from the Server...File received from the server!
[Client] Getting down95.txt from the Server...File received from the server!
[Client] Sending upload95.txt to the Server...File upload95.txt from Client was s
Client is disconnected from the server
[Client] Sending upload92.txt to the Server...File upload92.txt from Client was s
Client is disconnected from the server
3 operations can be done from this client:
1. tconnect <IP address> <Username> <Password>
Connection to the server established
The received message from the server is Welcome to the Tiger_server
[Client] Getting down96.txt from the Server...File received from the server!
[Client] Sending upload96.txt to the Server...File upload96.txt from Client was s
Client is disconnected from the server
Client is disconnected from the server
[Client] Getting down98.txt from the Server...File received from the server!
[Client] Sending upload98.txt to the Server...File upload98.txt from Client was s
Client is disconnected from the server
[Client] Getting down94.txt from the Server...File received from the server!
[Client] Sending upload94.txt to the Server...File upload94.txt from Client was s
Client is disconnected from the server
[Client] Getting down81.txt from the Server...File received from the server!
2. tget <Filename>
3. tput <Filename>
First connect to the server for using tget and tput functions
Enter the correct format of your operation
Connection to the server established
[Client] Getting down93.txt from the Server...File received from the server!
[Client] Sending upload81.txt to the Server...File upload81.txt from Client was s
Client is disconnected from the server
[Client] Sending upload93.txt to the Server...File upload93.txt from Client was s
Client is disconnected from the server
[Client] Getting down97.txt from the Server...File received from the server!
[Client] Sending upload97.txt to the Server...File upload97.txt from Client was s
Client is disconnected from the server
[Client] Getting down99.txt from the Server...File received from the server!
[Client] Sending upload99.txt to the Server...File upload99.txt from Client was s
Client is disconnected from the server
The received message from the server is Welcome to the Tiger_server
[Client] Getting down80.txt from the Server...File received from the server!
[Client] Sending upload80.txt to the Server...File upload80.txt from Client was s
Client is disconnected from the server
[Client] Getting down100.txt from the Server...File received from the server!
[Client] Sending upload100.txt to the Server...File upload100.txt from Client was s
Client is disconnected from the 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.

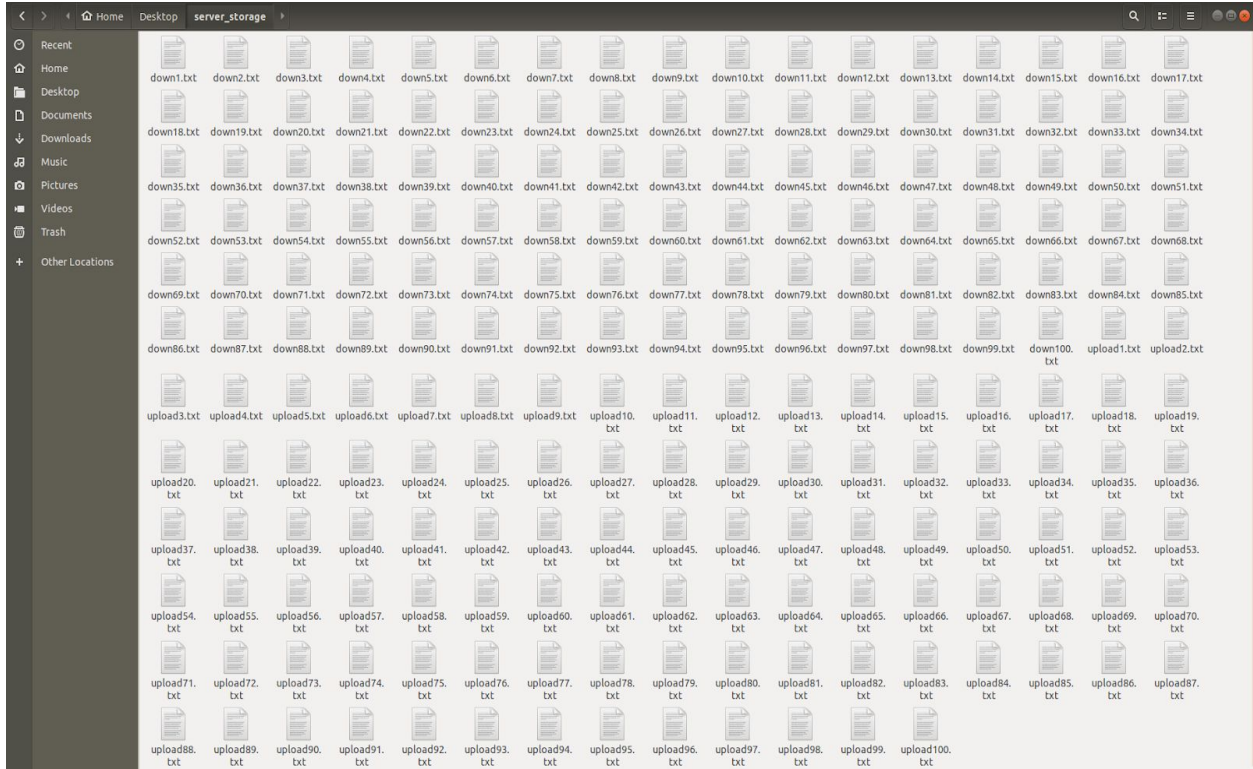


Fig 10: Copies of 100 upload#.txt uploaded on the server

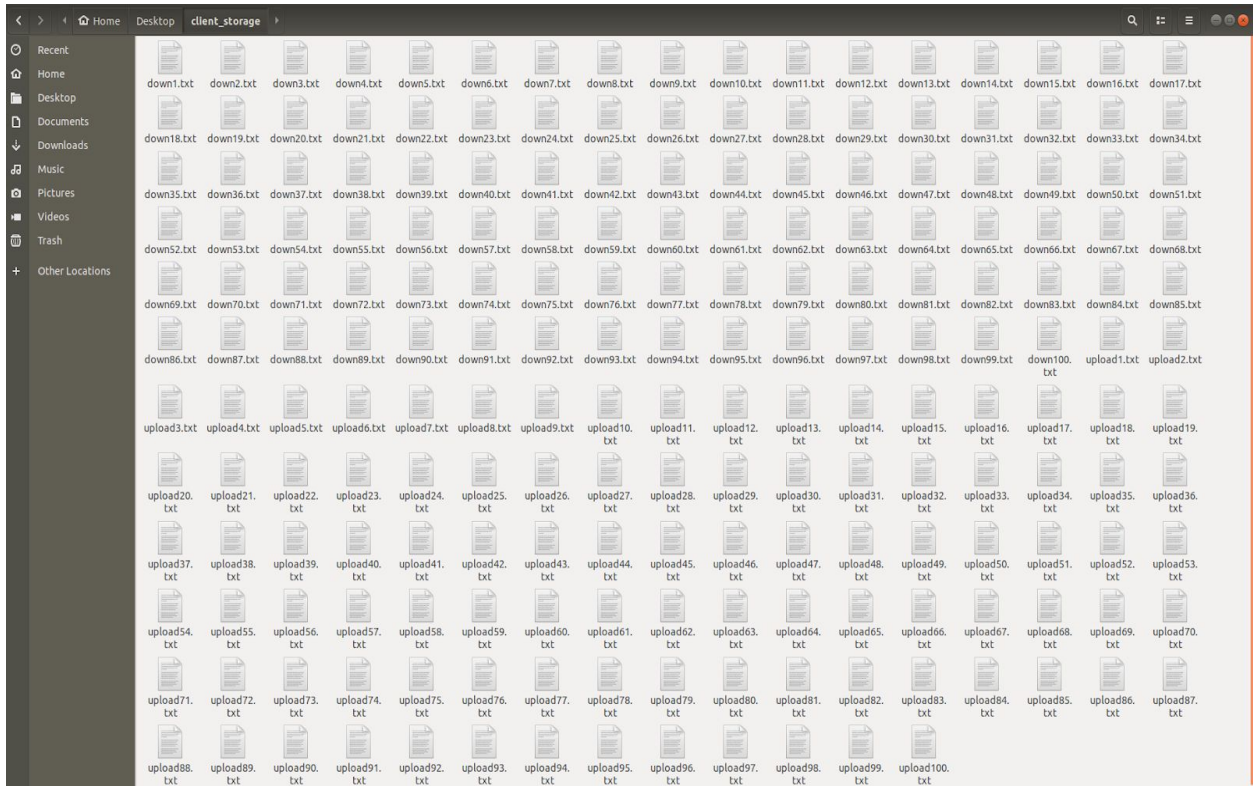


Fig 11: Copies of 100 down#.txt files downloaded at the client

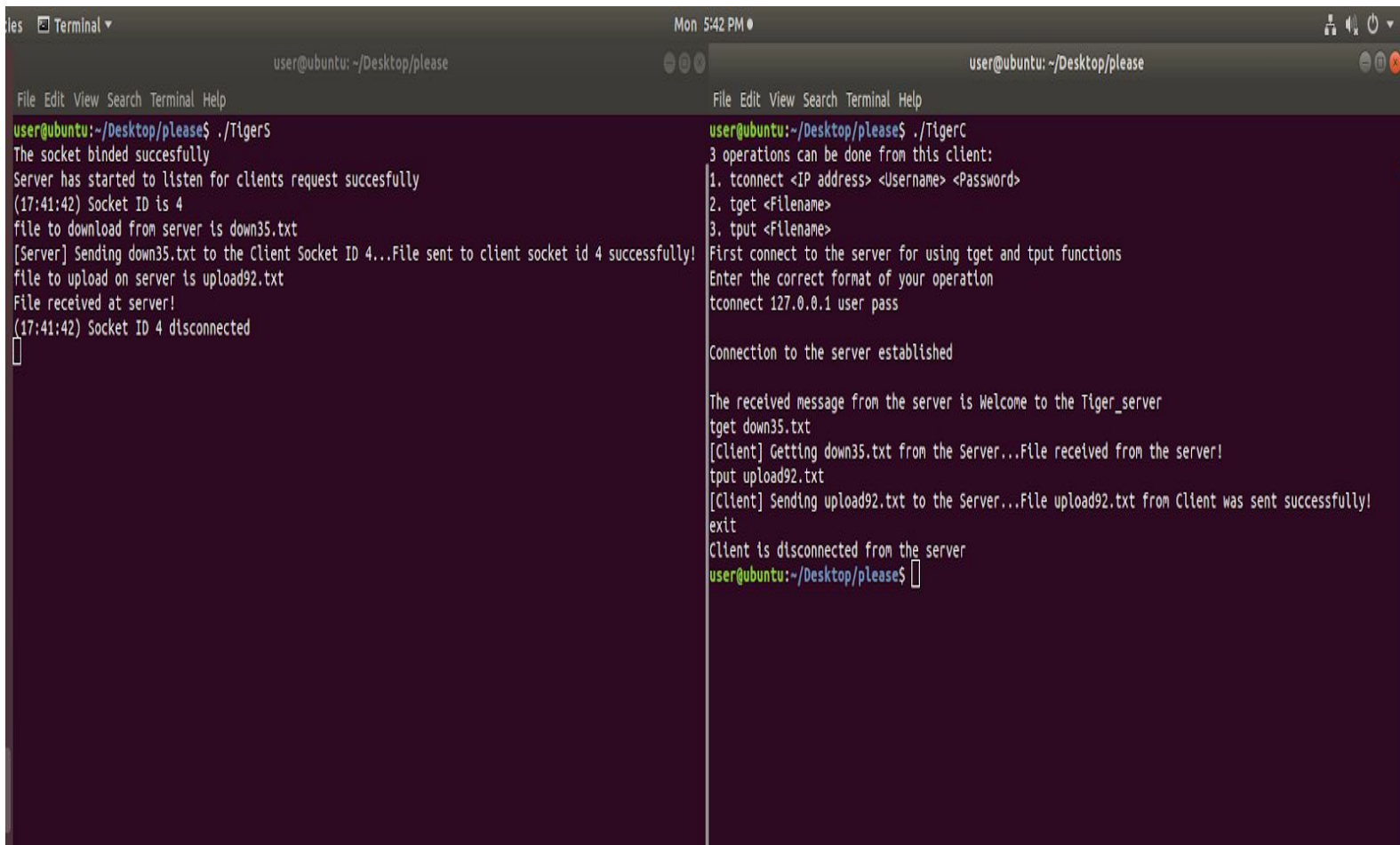
What is the maximum number of clients that your server can serve concurrently when both TigerS and TigerC is run on your laptop or PC? Justify your answer.

```
user@ubuntu: ~/Desktop/please
File Edit View Search Terminal Help
(23:53:33) Socket ID is 1006
file to upload on server is upload509.txt
Ok received from server!
(23:53:33) Socket ID 1003 disconnected
file to download from server is down513.txt
[Server] Sending down513.txt to the client Socket ID 1006...File sent to client socket id 1006 succes
sfully!
file to upload on server is upload513.txt
Ok received from server!
(23:53:33) Socket ID 1006 disconnected
(23:53:33) Socket ID is 1009
(23:53:33) Socket ID is 1011
file to download from server is down515.txt
[Server] Sending down515.txt to the client Socket ID 1011...File sent to client socket id 1011 succes
sfully!
(23:53:33) Socket ID is 1012
file to upload on server is upload515.txt
Ok received from server!
(23:53:33) Socket ID 1011 disconnected
file to download from server is down516.txt
[Server] Sending down516.txt to the client Socket ID 1009...File sent to client socket id 1009 succes
sfully!
file to upload on server is upload516.txt
Ok received from server!
(23:53:33) Socket ID 1009 disconnected
file to download from server is down514.txt
[Server] Sending down514.txt to the client Socket ID 1012...File sent to client socket id 1012 succes
sfully!
file to upload on server is upload514.txt
Ok received from server!
(23:53:33) Socket ID 1012 disconnected
(23:53:33) Socket ID is 1014
(23:53:33) Socket ID is 1017
file to download from server is down518.txt
[Server] Sending down518.txt to the client Socket ID 1014...File sent to client socket id 1014 succes
sfully!
file to download from server is down521.txt
[Server] Sending down521.txt to the client Socket ID 1017...File sent to client socket id 1017 succes
sfully!
file to upload on server is upload521.txt
Ok received from server!
(23:53:33) Socket ID 1017 disconnected
file to upload on server is upload518.txt
Ok received from server!
(23:53:33) Socket ID 1014 disconnected
(23:53:33) Socket ID is 1018
file to download from server is down517.txt
[Server] Sending down517.txt to the client Socket ID 1018...File sent to client socket id 1018 succes
sfully!
(23:53:33) Socket ID is 1021
file to upload on server is upload517.txt
File upload517.txt Cannot be opened.
(23:53:33) Socket ID 1018 disconnected
No client to connect to the Server or Error occured while connecting a client
user@ubuntu:~/Desktop/please$
```

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



The image shows two terminal windows side-by-side. The left window, titled 'Terminal', shows the execution of the TigerS program. The right window, also titled 'Terminal', shows the execution of the TigerC program. Both windows show the user's prompt as 'user@ubuntu: ~/Desktop/please'.

```
user@ubuntu:~/Desktop/please$ ./TigerS
The socket binded succesfully
Server has started to listen for clients request succesfully
(17:41:42) Socket ID is 4
file to download from server is down35.txt
[Server] Sending down35.txt to the Client Socket ID 4...File sent to client socket id 4 successfully!
file to upload on server is upload92.txt
File received at server!
(17:41:42) Socket ID 4 disconnected
█

user@ubuntu:~/Desktop/please$ ./TigerC
3 operations can be done from this client:
1. tconnect <IP address> <Username> <Password>
2. tget <Filename>
3. tput <Filename>
First connect to the server for using tget and tput functions
Enter the correct format of your operation
tconnect 127.0.0.1 user pass

Connection to the server established

The received message from the server is Welcome to the Tiger_server
tget down35.txt
[Client] Getting down35.txt from the Server...File received from the server!
tput upload92.txt
[Client] Sending upload92.txt to the Server...File upload92.txt from Client was sent successfully!
exit
Client is disconnected from the server
user@ubuntu:~/Desktop/please$ █
```

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

```
File Edit View Search Terminal Help
user@ubuntu:~/Desktop/please$ ./TigerS
The socket binded succesfully
Server has started to listen for clients request succesfully
[]

File Edit View Search Terminal Help
user@ubuntu:~/Desktop/please$ ./TigerC
3 operations can be done from this client:
1. tconnect <IP address> <Username> <Password>
2. tget <Filename>
3. tput <Filename>
First connect to the server for using tget and tput functions
Enter the correct format of your operation
tconnect 127.0.0.1 user pass
First connect to the server. Please enter correct operation format
user@ubuntu:~/Desktop/please$ ./TigerC
3 operations can be done from this client:
1. tconnect <IP address> <Username> <Password>
2. tget <Filename>
3. tput <Filename>
First connect to the server for using tget and tput functions
Enter the correct format of your operation
tconnect 127.0.0.1 usr pass
Incorrect username entered
user@ubuntu:~/Desktop/please$ ./TigerC
3 operations can be done from this client:
1. tconnect <IP address> <Username> <Password>
2. tget <Filename>
3. tput <Filename>
First connect to the server for using tget and tput functions
Enter the correct format of your operation
tconnect 127.0.0.1 user psas
Password incorrect. Try again with the correct password
(errno = 0)
user@ubuntu:~/Desktop/please$ []
```

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

```
les Terminal
user@ubuntu: ~/Desktop/please
File Edit View Search Terminal Help
user@ubuntu:~/Desktop/please$ ./TigerS
The socket binded succesfully
Server has started to listen for clients request succesfully
(00:40:09) Socket ID is 4
file to download from server is 3
ERROR: File 3 not found on server. (errno = 2)
[Server] Sending 3 to the Client Socket ID 4...user@ubuntu:~/Desktop/please$ []

Mon 12:41 AM
user@ubuntu: ~/Desktop/please
File Edit View Search Terminal Help
user@ubuntu:~/Desktop/please$ ./TigerC
3 operations can be done from this client:
1. tconnect <IP address> <Username> <Password>
2. tget <Filename>
3. tput <Filename>
First connect to the server for using tget and tput functions
Enter the correct format of your operation
tconnect 127.0.0.1 user pass
Connection to the server established
The received message from the server is Welcome to the Tiger_server
tgt down2.txt
Incorrect operation format entered. Please enter correct format tget<filename.txt> (errno = 0)
Now please enter for tput <filename.txt> or start a new connection session with server
tut upload3.txt
Incorrect operation format entered. Please enter correct format tput<filename.txt> (errno = 0)
Please start new connection session for using tput <filename.txt>
exit
Client is disconnected from the server
user@ubuntu:~/Desktop/please$ []
```

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