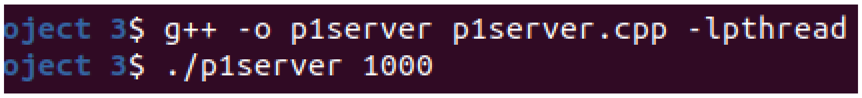
Technical Report for Project 3

By Edwin Tran
Dearo Yam
Joshua Perez
Joshua Cabrera

User Manual

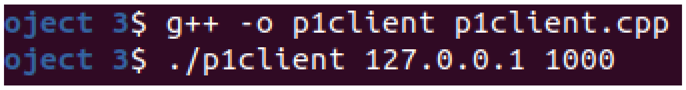
Program 1: Basic client-server:

This is a basic client-server program that allows for multiple clients to communicate with the server simultaneously. In order to run the client program the server program must be compiled and run first. To compile the server and all the other programs in the folder simply type 'make' in the command line in the respective directory of where all the files are located or you can type 'g++ -o p1server p1server.cpp -lpthread'.To run the server you must type './p1server *port#*' where the port number can be any number that you specify. Below is an example of compiling and running respectively.



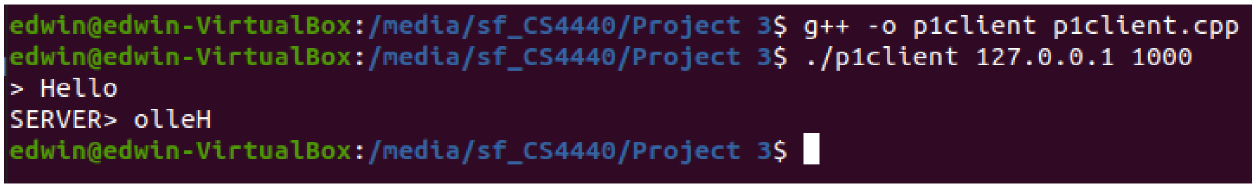
```
oject 3$ g++ -o p1server p1server.cpp -lpthread
oject 3$ ./p1server 1000
```

Now that the server is currently running the client can then be compiled and run. To compile the client you must type 'g++ -o p1client p1client.cpp'. Then to run the client you must type './p1client *ipaddress port#*' where the port # has to be the same one as the server is running on and the ip address can be any that you specify. Below is an example of compiling and running.



```
oject 3$ g++ -o p1client p1client.cpp
oject 3$ ./p1client 127.0.0.1 1000
```

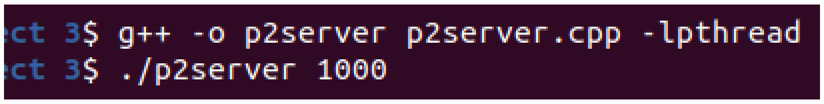
The program will then allow you to input any string where the server will reverse it and send it back to the client to print it out and will end the program. Here is an example output.



```
edwin@edwin-VirtualBox:/media/sf_CS4440/Project 3$ g++ -o p1client p1client.cpp
edwin@edwin-VirtualBox:/media/sf_CS4440/Project 3$ ./p1client 127.0.0.1 1000
> Hello
SERVER> olleH
edwin@edwin-VirtualBox:/media/sf_CS4440/Project 3$
```

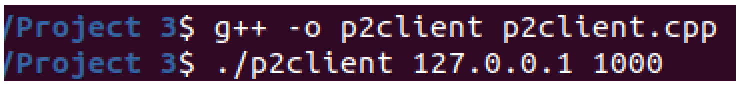
Program 2: Directory listing server:

This is a directory listing server program that allows for multiple clients to communicate with the server simultaneously. In order to run the client program the server program must be compiled and run first like the previous program. To compile the server the process is the same as before, simply type 'g++ -o p2server p2server.cpp -lpthread'. To run the server you must type './p2server *port#*'. Below is an example.



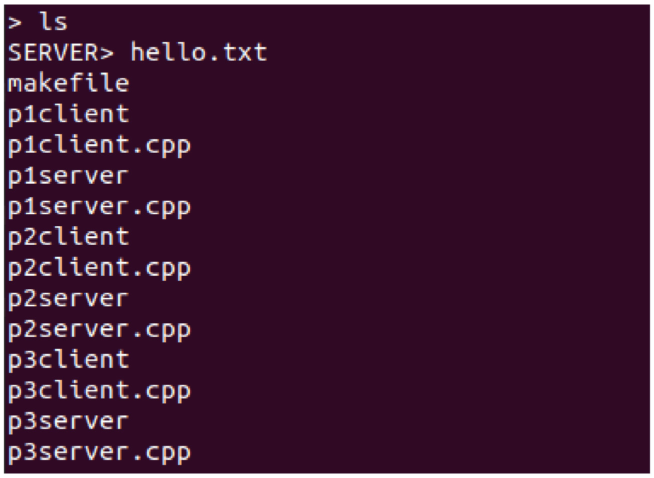
```
ct 3$ g++ -o p2server p2server.cpp -lpthread
ct 3$ ./p2server 1000
```

Now that the server is running you can run the client. To compile the client type 'g++ -o p2client p2client.cpp'. To run the client you must type './p2client *ipaddress port#*' where the port # has to be the same as the server's port number. Below is an example.



```
/Project 3$ g++ -o p2client p2client.cpp
/Project 3$ ./p2client 127.0.0.1 1000
```

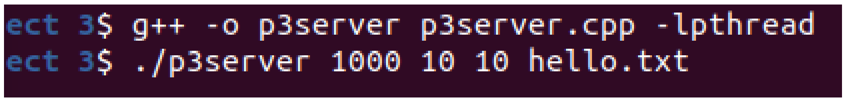
Now that everything is running the user is then allowed to use the ‘ls’ command to list all the files in the directory. Below is an example of the output.



```
> ls
SERVER> hello.txt
makefile
p1client
p1client.cpp
p1server
p1server.cpp
p2client
p2client.cpp
p2server
p2server.cpp
p3client
p3client.cpp
p3server
p3server.cpp
```

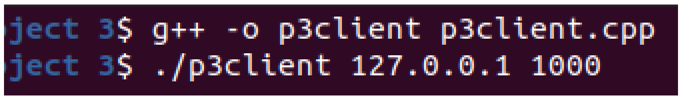
Program 3: Basic disk-storage system:

This is a basic disk-storage system that allows for multiple clients to communicate with the server simultaneously and will allow the user to simulate a physical disk where it is organized by cylinders and sectors. The server program must be compiled and run first before the client program. To compile the server the process is the same like the previous two programs. To compile type ‘g++ -o p3server p3server.cpp -lpthread’. To run the program type ‘./p3server *port# cylinders sectors filename’* where the number of cylinders and sectors are any that you specify. Below is an example of compiling and running the client.



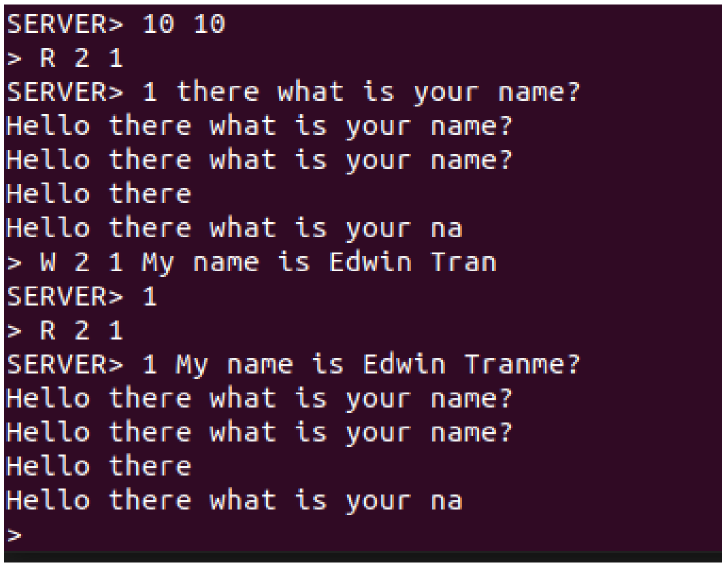
```
ect 3$ g++ -o p3server p3server.cpp -lpthread
ect 3$ ./p3server 1000 10 10 hello.txt
```

Now that the server is running you can run the client. The process is the same as the previous two. Below is an example of compiling and running the client.



```
ject 3$ g++ -o p3client p3client.cpp
ject 3$ ./p3client 127.0.0.1 1000
```

The program allows for you to see the disk geometry, read 128 bytes from a specified cylinder and sector from the file, and write 128 bytes from a specified cylinder and sector. To get the disk geometry type ‘I’. To read 128 bytes from a specified cylinder and sector type ‘R *cylinder sector’* where cylinder and sector are specified by the user. To write 128 bytes from a specified cylinder and sector type ‘W *cylinder sector*’ where cylinder and sector are specified by the user. Below is an example of the output.

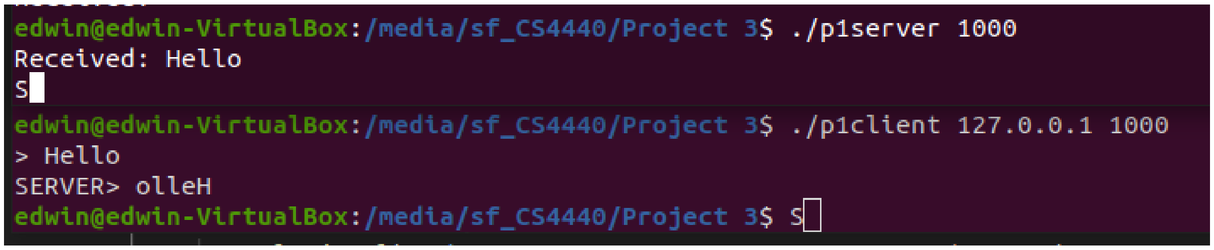


```
SERVER> 10 10
> R 2 1
SERVER> 1 there what is your name?
Hello there what is your name?
Hello there what is your name?
Hello there
Hello there what is your na
> W 2 1 My name is Edwin Tran
SERVER> 1
> R 2 1
SERVER> 1 My name is Edwin Tranme?
Hello there what is your name?
Hello there what is your name?
Hello there
Hello there what is your na
>
```

Technical Manual

Program 1: Basic client-server:

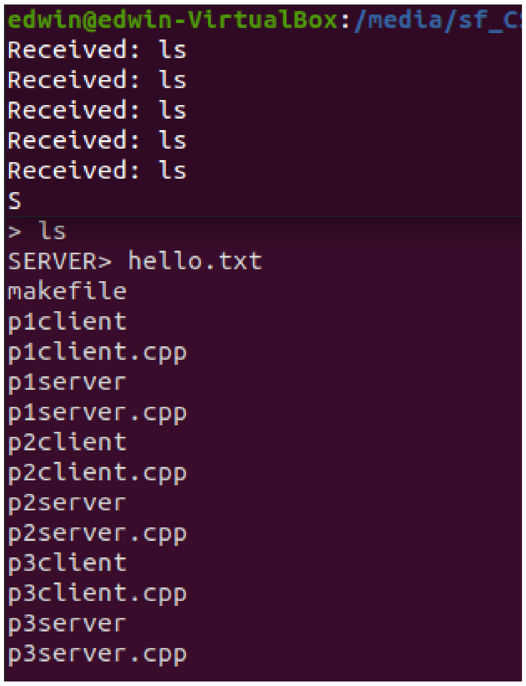
This program allows for the user to run a server and have the client connect to it. The server creates a socket and binds the server address and port number to it. The server then waits for any connections that will come and connect to the server. The client creates a socket and binds the ip address and port number to the socket where the port number has to be the same as the server allowing the server and client to communicate. In our situation the client is communicating by sending a request for the string to be reversed and in response the server is sending the string back but in reverse. Below is an example of the output.



```
edwin@edwin-VirtualBox:/media/sf_CS4440/Project 3$ ./p1server 1000
Received: Hello
S
edwin@edwin-VirtualBox:/media/sf_CS4440/Project 3$ ./p1client 127.0.0.1 1000
> Hello
SERVER> olleH
edwin@edwin-VirtualBox:/media/sf_CS4440/Project 3$ S
```

Program 2: Directory Listing Server:

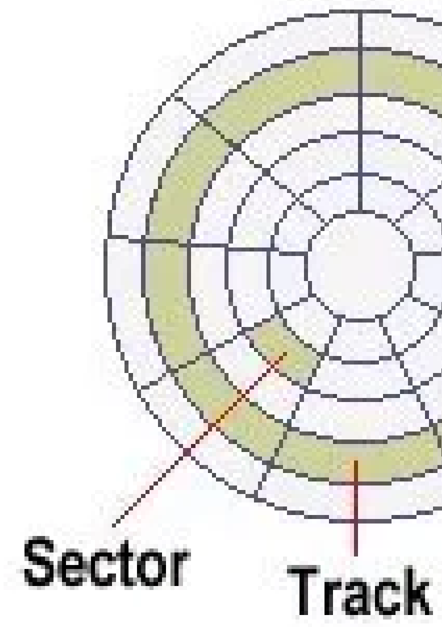
This program has a similar implementation of communication between the server and the client. The server will create a socket and bind it with a ip address and port number. The server will then wait for all the connections that will come and connect to that server. The client creates a socket and binds the ip address and port number to the socket. In program two the client is communicating by sending a request of listing the files in the directory where the server in response is the list of all the files in that current directory. The top terminal is the server and the bottom terminal is the client.



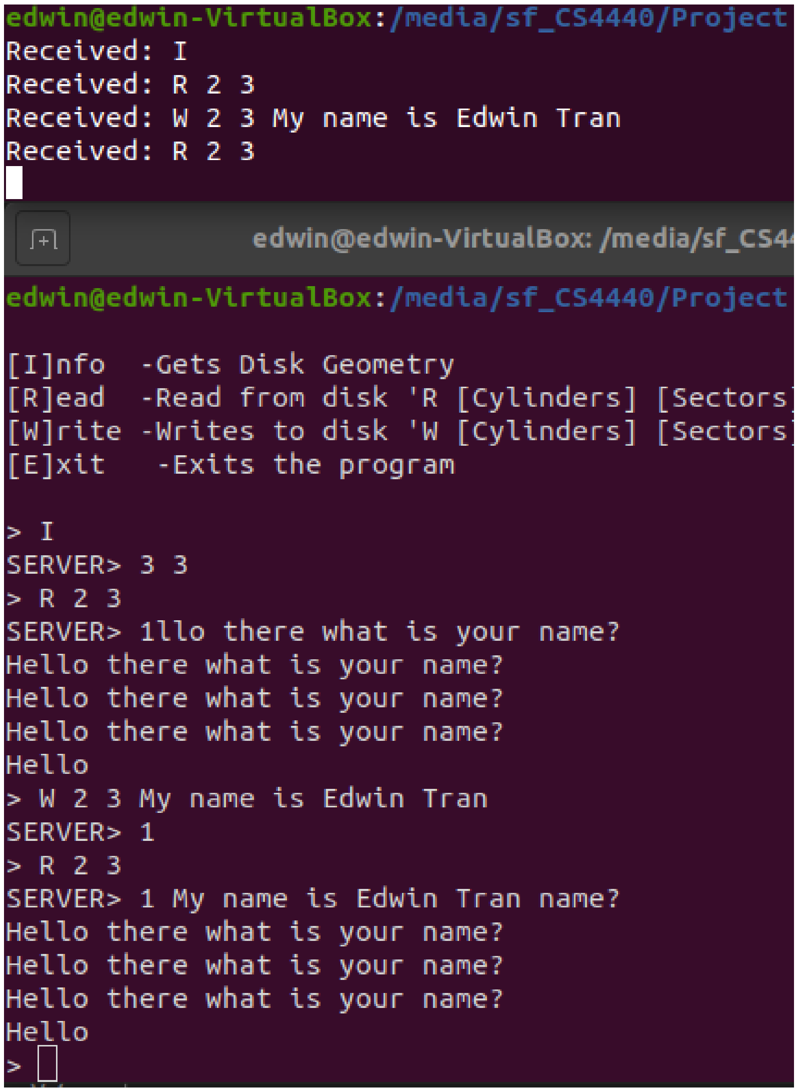
```
edwin@edwin-VirtualBox:/media/sf_C
Received: ls
Received: ls
Received: ls
Received: ls
Received: ls
S
> ls
SERVER> hello.txt
makefile
p1client
p1client.cpp
p1server
p1server.cpp
p2client
p2client.cpp
p2server
p2server.cpp
p3client
p3client.cpp
p3server
p3server.cpp
```

Program 3: Basic Disk-storage system:

This program has the same implementation of communication between the server and the client. The server will create a socket and bind an ip address and the server port number to it. The server then waits for any incoming connections coming from any client. The client creates a socket and binds an ip address and the server port number to the socket. This program simulates a physical disk where it is organized by cylinder and sectors. Below is an a visualization of how cylinders and sectors work.



This diagram shows that there are multiple tracks or also known as cylinders and within those cylinders are multiple sectors. In this program the server takes a file and divides the file in cylinders and within those cylinders a certain amount of sectors. The sectors are saved as an array of struct where the struct holds an array of 128 bytes of characters. The client sends a request for information on the disk geometry on a file, contents of a file at a cylinder and sector, and sends a request to write to a cylinder and sector of a file. If asked which cylinder and sector that is to be read the server will read from the respective struct. For example if they wanted to read from cylinder 2 sector 3 the server will read from the 6th struct in the array. The 128 bytes of characters from that struct will then be sent to the client and printed out. Below is an example of how each command works. The top terminal is the server and the bottom terminal is the client.



```
edwin@edwin-VirtualBox:/media/sf_CS4440/Project
Received: I
Received: R 2 3
Received: W 2 3 My name is Edwin Tran
Received: R 2 3
```

edwin@edwin-VirtualBox: /media/sf_CS44

```
edwin@edwin-VirtualBox:/media/sf_CS4440/Project

[I]nfo  -Gets Disk Geometry
[R]ead  -Read from disk 'R [Cylinders] [Sectors]
[W]rite -Writes to disk 'W [Cylinders] [Sectors]
[E]xit   -Exits the program

> I
SERVER> 3 3
> R 2 3
SERVER> 1llo there what is your name?
Hello there what is your name?
Hello there what is your name?
Hello there what is your name?
Hello
> W 2 3 My name is Edwin Tran
SERVER> 1
> R 2 3
SERVER> 1 My name is Edwin Tran name?
Hello there what is your name?
Hello there what is your name?
Hello there what is your name?
Hello
> 
```