Operating Systems. Homework #5.

Submission - January 13, 23:59.

In this exercise, you will implement a toy client/server architecture: a *printable characters counting server*.

You need to implement two programs:

1. `pcc_server`: The server accepts TCP connections from clients. A client that connects sends the server a stream of *N* bytes (*N* depends on the client, and is not a global constant). The server counts the number of printable characters in the stream (a *printable character* is a byte whose value is in the range [32,126]). Once the stream ends, the server sends the count back to the client over the same connection.
   In addition, the server maintains a data structure in which it counts the number of times each printable character was observed in all the connections. When the server receives a SIGINT, it prints these counts and exits.
2. `pcc_client`: The client creates a TCP connection to the server and sends it *N* bytes read from `/dev/urandom`, where *N* is a user-supplied argument. The client then reads back the count of printable characters from the server, prints it, and exits.

Cheating Policy

Part of the course requirements is to program assignments by yourself. You can discuss concepts and ideas with others, but looking at other people's code, looking at code from previous semesters, sharing your code with others, coding together, etc. are all not allowed. Students caught violating the requirement to program individually will receive a "250" grade in the course and will have to repeat it. *Even if you're under pressure, ask for help or even don't submit, rather than cheat and risk having to repeat the course.*

If you put your work in your TAU home directory, set file and directory permissions to be accessible only by you.

Late Submission Policy

You have 5 grace days throughout the semester, you can use all of them, some of them or none of them for this exercise. A 10 point deduction will be applied for every late day past the grace days.

Submission Guidelines

Submit two files named **pcc_server.c and pcc_client.c**. **Document it** properly – with a main comment at the beginning of the file, and an explanation for ***every non-trivial*** part of your code. Help the grader understand your solution and the flow of your code.

We check

1. C program compiles without warning with "`gcc -O3 -Wall -std=gnu99`"

Client Specification

*Command line argument*s:

1. *Server ip*: Assume it is a valid IP address.
2. *Server port*: Assume it is a 16-bit unsigned integer.
3. *Length*: The number of bytes to read from `/dev/urandom` and send to the server. Assume it is an unsigned integer.

High-level flow:

1. Create a TCP connection to the specified *server port* on the specified *server ip*.
2. Open `/dev/urandom` for reading.
3. Transfer the specified *length* bytes from `/dev/urandom` to the server over the TCP connection.
4. Obtain the count of printable characters computed by the server, *C* (an unsigned int), and print it to the user in the following manner:

```
printf("# of printable characters: %u\n", C);
```

5. Exit with code 0.

Remarks:

- You define the protocol used over the TCP connection between the client and the server. All that matters is that in the end, the client receives the number of printable characters computed on the server, and stores it in an `unsigned int` that is printed as described above.
- On error, print an error message containing the `errno` string (i.e., with `perror()` or `strerror()`) and exit with a non-zero return code.
- There's no need to clean up file descriptors or free memory when exiting.
- Do not assume a bound on the size specified by the *length* argument, except that it's specified by an `unsigned int` (i.e., at most 2^{32}-1).

Server Specification

Command line arguments:

1. *Server port*: Assume this is a 16-bit unsigned integer.

High-level flow:

1. Initialize a data structure `pcc_total` that will count how many times each printable character was observed in all client connections.
2. Listen to incoming TCP connections with queue of size 10
3. Enter a loop, in which you:
    A. Accept a TCP connection on the specified *server port*.
    B. When a connection is accepted, read a stream of bytes from the client, compute its printable character count and write the result to the client over the TCP connection. Also updates the `pcc_total` global data structure.
4. If the user hits Ctrl-C, perform the following actions:
    A. If the server is in the middle of processing a client, finish computing their printable character counts and update the `pcc_total` global data structure.
    B. Print out the number of times each printable character was observed by clients. The format of the printout is the following line, for each printable character:
    ```
    char '%c' : %u times\n
    ```
    C. Exit with exit code 0.

Remarks:

- You can `bind()` to the address INADDR_ANY to accept connections on all network interface. See the `ip(7)` manual page for more details.
- You define the data structures required to implement the above specification.
- If the connection to the client terminates (e.g. the client closed the connection unexpectedly) don't exit the server (print an error message to stderr) and keep accepting new client connections.
- On other errors, print an error message containing the `errno` string (i.e., with `perror()` or `strerror()`) and exit with a non-zero return code.
- There's no need to clean up file descriptors or free memory when exiting.

Example Execution

client console	server console

`$ pcc_client server 2233 4`
`# of printable characters: 1`
	`$ pcc_server 2233`
	`^C`
	`char ' ' : 0 times`
	`char '!' : 1 times`
	`…`
	`char '~' : 0 times`

Background Info and Tips

- `/dev/urandom` is a pseudo file that returns random bytes. Therefore, it has no size; you can `read()` from it repeatedly and keep getting random bytes forever.
- Read the manual pages of `ip(7)`, `connect(2)`, `listen(2)`, `accept(2)`, and `bind(2)`.
- The IP address 127.0.0.1 specifies the local host (it is called a *loopback*) address. If you don't know the IP address of your machine, or are working on a VM without an Internet connection, you can still connect to 127.0.0.1.
- For testing/debugging, consider using a regular file with predefined content instead of `/dev/urandom`.
- You can use the NetCat utility (**`nc`**) to simulate a server or client.
- You can use the `ngrep` utility to monitor the traffic on specific port. Pay attention that root permissions are required to run it, so use sudo.
  Example:

Console 1

```
$ nc -l 2233   # Server listens on
               # port 2233
Hello, world # Server gets the
               #    string    and prints
               # it.
```

Console 2

```
$ nc 127.0.0.1 2233 # Client connects
Hello, world        # to port 2233, and
                    #   sends   the string
```

Console 3

```
$ sudo ngrep -d any port 2233                # Start intercepting IP traffic
[sudo] password for eug:                   # on any device, port 2233
interface: any
filter: (ip or ip6) and ( port 2233 )
####                                      # Server accepted connection
T 127.0.0.1:41104 -> 127.0.0.1:2233 [AP]    # Transfer started
  Hello, world.                             # Dumping the data
####                                        # Connection closed
```

Submission

Submit a ZIP archive named ex5_012345678.zip, where 012345678 is your ID. The archive should contain at least 2 files, `pcc_client.c`, and `pcc_server.c`. *Mac users! Don't submit hidden folders!*

Your code must compile without warning with the following compilation commands:

```
gcc -std=gnu99 -O3 -Wall -o pcc_server pcc_server.c
gcc -std=gnu99 -O3 -Wall -o pcc_client pcc_client.c
```