

# CSE333 Exercise 10

**Out:** Friday, February 25

**Due:** Wednesday, March 2 by 11 am PST

**Rating:** 3 (note)

## Goals

- Adapt client-side networking code (TCP) in C/C++.
- Use the utility netcat (`nc`) to interact with clients over the network.

## Problem Description

Write a C++ program that connects (via TCP) to a server specified by a user-supplied hostname and port number, sends the bytes of a specified local file to the server, and then closes the connection and exits. Your program should accept the command-line arguments *in the following order*:

1. The **hostname** of the server
2. The **port number** of the server
3. The name of a **local file**

Because you must use the POSIX `write()` to write bytes to the server, we also want you to use the POSIX `read()` to read in the local file.

An example execution of the completed application is:

```
bash$ ./ex10 localhost 5555 test.txt
```

## Provided Files

We have provided you with the following **four** source files, which can be downloaded from this web directory (`./ex10_files`) or with the commands:

```
bash$ wget https://courses.cs.washington.edu/courses/cse333/22wi/exercises/ex10_files/<filename>
```

- **SocketUtil.h** — Provides the public interface for various client-side networking utility functions. **DO NOT MODIFY.**

- **SocketUtil.cc** — Contains empty implementations of the utility functions declared in the header file.
- **ex10.cc** — Contains an empty `main` function for the client-side networking program.
- **Makefile** — Provided for your convenience in compiling the executable `ex10`.

**Note:** You will only submit `SocketUtil.cc` and `ex10.cc`.

## Implementation Notes

### Code Adaptation

Feel free to adapt sample code from lecture and section as part of your solution if it helps, but be sure you understand what your code does when you're done.

### User Input

Since you will be reading in user input via command-line arguments, you should make sure your code handles various inputs from the user, which may be in an unexpected format. Note that each of the three inputs in this exercise has a *different* expected format.

### Error Handling & Robustness

When you are using POSIX functions, you should handle errors by retrying in the case of recoverable errors ( `EAGAIN` and `EINTR` ) and returning an error status in the case of a non-recoverable error. Make sure that you clean up system resources in *all* possible cases.

## Testing Notes

### Server Setup

To test your program, you will need to setup a server to receive the data that your `ex10` executable will send. The recommended way to do this is using the `nc` utility:

```
bash$ nc -l <port> > output.bytes
```

This command will run a netcat listener (`-l`) on port `<port>` (e.g., `5555`), which needs to match the port you provide to the `ex10` executable, and redirect any received bytes to the file `output.bytes`. Note that this will create the file if it didn't exist or overwrite if it does exist.

Note that `nc` will exit once it has processed a single connection, so you'll need to rerun `nc` each time you test your client.

### Local Testing

If you are running the server on the same computer/host that you are testing your code on, you can use the special loop-back IP address `127.0.0.1` or `localhost` to refer to the same host. Please note that each `attu` (e.g., `attu1`, `attu2`) counts as a separate host, so if you are testing on `attu`, both client and server must be running on the same one for this to work. To log into a specific `attu` machine, run:

```
bash$ ssh <netid>@attu<#>.cs.washington.edu
```

where <#> should be replaced by a number 1 – 8.

## Style Focus

Don't forget that good practices from previous exercises still apply!

### General

For the sake of our autograder, you may not modify `SocketUtil.h`, which also means that you should not modify the function signatures in `SocketUtil.cc`.

### C/C++ Idioms


To work with the POSIX networking API, we, unfortunately, have to mix C and C++ idioms in our code. But you should still try to use C++ idioms whenever possible (e.g., use `cout` instead of `printf`, use C++ casting).

## Submission

You will submit: `SocketUtil.cc` and `ex10.cc`.

Your code must:

- Compile without errors or warnings on CSE Linux machines (lab workstations, `attu`, or CSE home VM).
- Have no runtime errors, memory leaks, or memory errors ( `g++` and `valgrind` ).
- Be contained in the files listed above and compile with the provided Makefile.
- Have a comment at the top of your `.cc` files with your name(s) and CSE or UW email address(es).
- Be pretty: the formatting, modularization, variable and function names, commenting, and so on should be consistent with class style guidelines. Additionally, the linter shouldn't have any complaints about your code ( `cpplint.py` ).
- Be robust: your code should deal with hard-to-handle/edge cases and bogus user input (if there are any) gracefully.

Submit your code on  Gradescope (<https://www.gradescope.com>). Don't forget to add your partner if you have one.

