CS/ECE 438: Computer Networks **Spring 2026**

Machine Problem 0

Handed Out: January 22, 2026 *Due: 11:59pm, February 1, 2026*

TA: Martin Chong

1 Introduction

This MP introduces socket programming in C and the mechanism for submitting assignments. GitHub will host your submissions and the course staff will manage virtual machines to simulate multiple network entities. This assignment will help you prepare your environment so that all following assignments will be simpler to code and submit. The course staff will expect you to know how to use this system as the course continues.

The introductory exercise in this MP will have you obtain, compile, run, and extend simple C network programs. Extensions to the given code will introduce you to one method of framing data into individual messages when using a byte stream abstraction such as TCP communication.

2 What Is Expected in this MP?

Inside the `.release` folder (after setup), you will find a folder named mp0, which contains the programs `client.c`, `server.c`, `talker.c`, and `listener.c` — all from Beej's Guide to Network Programming:

`https://beej.us/guide/bgnet/`

Beej's guide is an excellent introduction to socket programming, and very approachable. Compile the files using `gcc` to create the executable files `client`, `server`, `talker`, and `listener`. The provided Makefile will compile all 4 (simply run `make` inside the directory). The real assignments will require you to submit a Makefile, so if you aren't already experienced with make, please familiarize yourself with the provided Makefile, and ensure that you can adapt it to a new project.

Login to two different virtual machines, and execute `client` on one and `server` on the other. This makes a TCP connection. Next, execute `talker` on one machine and `listener` on the other. This sends a UDP packet.

Note that the connection oriented pair, `server` and `client`, use a different port than the datagram oriented pair, `listener` and `talker`. Try using the same port for each pair, and run the pairs simultaneously. Do the pairs of programs interfere with each other?

Next, change `server.c` to accept a file name as a command line argument and to deliver the length and contents of the file to each client. Assume that the file contains no more than 100 bytes of data. Send the length of the file (an integer between 0 and 100) as an 8-bit integer. Change `client.c` to read first the length, then that number of bytes from the TCP socket, and then print what was received.

The client output should look like this:

```
client:  connecting to <hostname>
client:  received <filelen> bytes
This is a sample file that is sent over a TCP connection.
```

where <hostname> is the address of the server, <filelen> is the number of bytes received, and the rest of the output is the file contents. That's it. Sounds simple, doesn't it? Indeed, for experienced Unix/C programmers this MP is trivial. Others should find it a nice way to get started on network programming.

You will need to have (or quickly acquire) a good knowledge of the ANSI C programming language, including the use of pointers, structures, typedef, and header files. If you have taken CS 241, you should already have the necessary background. Don't simply download the source code and compile the programs, but make sure that you read and understand how the sockets are created and the connection established. Beej's guide is a very useful tool in this sense.

3 Setting Up Your VirtualBox VM Environment

The autograder runs your code in VMs — 64-bit Ubuntu 24.04 LTS VMs, running on VirtualBox. Therefore, to test your code, you will need two 64-bit Ubuntu 24.04 LTS VMs of your own. (Even if you're already running Linux on your personal machine, later assignments will use multiple VMs, so you might as well start using the VM now.)

WARNING: COMPILATION CAN BE A LOT LESS PORTABLE THAN YOU THINK, ESPECIALLY WHEN MAC OS OR EWS IS INVOLVED. Please don't assume that it will be ok after testing it only on your personal machine or EWS. (Just don't use EWS at all; it is not well suited to classes that involve networked programming assignments.)

The Ubuntu image is available at `https://ubuntu.com/download`

A tutorial for installing Ubuntu on VirtualBox on Windows and Linux can be found at
`https://ubuntu.com/tutorials/how-to-run-ubuntu-desktop-on-a-virtual-machine-using-virtualbox`

For Macs (or ARM-based computers), we suggest using Docker since VirtualBox does not support ARM architecture. Please refer to Section 5 for details.

If your version of VirtualBox displays the Unattended Installation option, it can be skipped.

After the Ubuntu install process (within the VM), you should install the ssh server. You can run these commands once the OS is installed:

```
sudo apt update && sudo apt upgrade -y
sudo apt-get install openssh-server gcc g++ make gdb wget iperf tcpdump
```

Use `sudo apt-get install <package>` to install any programs you'll need, like `gdb` or `valgrind`. We also suggest getting `iperf` and `tcpdump`, which will be useful later.

Note: WSL (Windows Subsystem for Linux) may work as one of your machines, but keep in mind that some compiler actions might be different. If you're using WSL, be careful about the undefined coding behaviors (e.g., uninitialized variables, memory leaks, etc.).

4 Set Up Networking Inside VMs (VirtualBox only)

VirtualBox's default network setup is a NAT (which we'll learn about later!) interface to the outside world, provided by the host computer. This allows the VM to access the Internet, but the host computer and other VMs will not be able to talk to it. We're going to replace the NAT interface with one that allows those communications.

Before making this change, use that Internet access to install the packages mentioned above.

Now it's time to replace the network interface.

- Make sure the VM is fully shut down, then go to its Settings → Network section.
- Switch the Adapter Type from "NAT" to "Host-only Adapter", and click ok.
- On the Adapter 2 tab, enable the adapter and set it to "NAT". This shares the Internet connection from your Host machine for updating and fetching from your GitHub repo.
- Restart, and now the VM can now talk to other VMs on the same host computer.

With SSH enabled on the VM, you should be able to access it from Visual Studio Code. Check the IP address inside the VM with this command: `ip a`

5 Docker Instructions

This section covers the basics of how to set up Docker for the MPs. This will be useful for those with a Mac with Apple silicon. This is not a detailed instruction guide, so you may have to adapt these instructions based on your device.

5.1 Docker Setup

- Docker can be installed from this link: `https://docs.docker.com/engine/install/`
  For now, we will stick to running Docker from the command line.
- Once Docker is installed, check the current images on your machine with the command:
  `docker images`
- To check running containers:
  `docker ps`
- To download the latest Ubuntu 24.04 image, use the command
  `docker pull ubuntu:24.04`
  This should show up as the `ubuntu` image with tag `24.04`

Docker uses containers instead of launching full-fledged virtual machines. These images are temporary instances, and their storage is destroyed when you stop the container. If we want persistent storage, we need to mount a hard drive corresponding to a folder in your machine.

- First, create two working directories on your machine, one for each container instance. The MPs in this course require two separate virtual machines/containers.
- Create a network using the following command:
  `docker network create <network name>`
- Then launch your Docker container using the following command:
  ```
  docker run -dit --cap-add=NET_ADMIN --net <network name> -v
  '<persistant/directory>:/main' --user $(id -u):$(id -g) --ip <ip address>
  --name <container name> <image name>
  ```
- The above command means the following:
  `docker run -dit` - starts the container with image <image name>
  `<container name>` - specifies the name of the container
  `-v` - specifies the host directory that will be used as persistent storage for the container. This can be found inside the Docker container in folder `/main`
- Once the container is launched, we can connect to it using the command:
  `docker attach <container name>`
  This should open a Bash shell
- To exit the container shell without shutting down the container, use command
  `control+P` followed by `control+Q`
- To shutdown the container, you can type `exit` from inside its Bash shell. To restart it, use the command
  `docker restart <container name>`
- To check all containers, even those that are shutdown, use the command
  `docker container ls -all`

5.2 First steps

Once you have connected to the container, you can install all the dependencies using `apt-get`. The container is extremely lightweight, so you should check if any commands you normally find installed in Ubuntu by default are still available. Text editors may not be available, so install those too (e.g., `nano, micro, neovim`). The other dependencies specified earlier should also be installed.

We also suggest setting up the `git` ssh keys during the first-time setup. Once all your dependencies are installed, we would like to create a new image on top of the Ubuntu image that we can use for doing the assignments. Remember, only the persistent directory `/main/` is retained after the container is deleted. All other files, including the install files and ssh keys will be destroyed when the container is deleted. To create a new image:

- From the host command line, run:
  `docker commit <container name> <new container image name>`

To create a new Ubuntu image with all your installs. You will be using this image from now on for your MPs. This will save time since you don't have to create a new container with all the dependency installs.

- You can now use the previously created image to create a duplicate container to run the MPs.

5.3 Networking across containers

To allow Docker instances to communicate with each other, they need to be on the same network.

- To create a docked network:
  `docker network create <network name>`
- To connect your container to this network:
  `docker network connect <network name> <container name>`
- To check network properties, such as the IP address of the container:
  `docker network inspect <network name>`

If you set up ports correctly, you should be able to connect one container to another. For example, to ping one container to the other, first attach port 80 of both containers to ports on your host machine using `-p 8001:80` for one container and `-p 8002:80` for the other. You should now be able to ping one machine from another with `ping <other container's IP>`

6 What Tips and Tricks Will be Useful?

Copy-pasting directly from PDFs is a bad idea. Dashes, quotes, and kerned characters may get completely misrepresented when pasted in a terminal.

Never add compiled executables and object files to git. You might lose points in subsequent assignments if you do. Use `git add` to add only the source files (`src` folder), `Makefile`, and `config.ini` files. Also, learn how a `.gitignore` file works!

MP0 is ungraded but still very important to get you started. Performing this assignment successfully will make submitting futuacre assignments much easier.

7 Git Instructions

7.1 Course Setup (only once for the entire semester)

The first time you're accessing the CS/ECE 438 repository this semester, you will need to have a CS/ECE 438 repository set up for you. This process is simple:

1. Visit `https://edu.cs.illinois.edu/create-gh-repo/sp26_cs438` and follow the instructions to join the *illinois-cs-coursework* organization on GitHub
2. The web service will generate your CS/ECE 438 repository and provide you with your repository name: `https://github.com/illinois-cs-coursework/sp26_cs438_NETID`

7.2 Workspace Setup (necessary only once per computer/directory)

Username and password entry:

Your username is the one you use for GitHub. For the password, you need to setup a Personal Access Token and enable SSO (single sign-on) for the *illinois-cs-coursework* organization. **Once you generate the token, make sure to save it since you won't see it again.** Follow this guide to generate it: `https://www.shanebart.com/clone-repo-using-token/`

To enable SSO, click the button next to the token you just generated and authorize it for the organization.

You may also opt to use an SSH key linked to your GitHub account. To set this up, visit this link: `https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account`

After adding the key, authorize it for the *illinois-cs-coursework* organization.

Cloning your repository:

To clone your repository, run `git clone` inside the directory you want to house your code in:

```
git clone https://<TOKEN>@github.com/illinois-cs-coursework/sp26_cs438_NETID.git <FOLDER>
```

You can replace <FOLDER> with whatever folder you want created (for example `cs438`) and <TOKEN> with the one you generated. You can submit this MP and all subsequent ones through this repository.

Finally, move into the directory you just cloned: `cd <FOLDER>`

Released code:

The code we release as a starting point for any MP will be present in the `.release` repository. You need to add this repository as a remote:

```
git remote add release https://github.com/illinois-cs-coursework/sp26_cs438_.release.git
```

You're now all set to begin to work on your assignment!

7.3 Assignment Setup (necessary only once per assignment)

To retrieve the latest assignments for CS/ECE 438, you need to fetch and merge the release repository into your repository. This can be done with the following commands:

```
git pull
git fetch release
git merge release/main -m "Merging release repository" --allow-unrelated-histories
```

7.4 Assignment Submission (do this often!)

The first step in assignment submission is to increment the version number in your `config.ini` file. Your code will not be picked up by the autograder if you fail to increment the version number.

Every time you want to submit your work, you will need to `add`, `commit`, and `push` your work to your git repository. This can always be done using the following commands on a command line while within your CS/ECE 438 directory:

```
git add -u
git commit -m "your commit message"
git push origin main
```

You can also check the working tree status of your repository by running `git status`. Be sure not to skip `origin main` when trying to `push` or `pull`.

IMPORTANT: The autograder will use Version 0 of your repo to save your repo locally, so commit a Version 1 so it can create the _grades branch

7.5 Verifying Submission

You can always verify your submission by visiting `https://github.com/illinois-cs-coursework/` and viewing the files in your repository. Only the files that appear in your GitHub repository will be graded.

7.6 How to See Your Grade

The autograder runs periodically on your new submissions (again, don't forget to update the version number!) and the results are updated in a different branch (`_grades`) inside your directory.

For your convenience, we have created a script to see the results: `./see_results.sh`

The script swaps the branch to _grades, shows the results, and swaps the branch back. If you run the see_results.sh file on mp0 before the autograder has run, your directory will move to the _grades branch. You will have to manually execute `git checkout main` to get back to your working branch. **DO NOT work on the _grades branch!**

Caution: During the last few hours leading of the submission deadline, queues could be multiple hours long. We advise you to get your work done early.

7.7 Final Grade for an MP

To finalize the grade for an MP, either keep the `final_grade_version` in your `config.ini` file as **NULL** to choose your final submitted version for grading, or replace NULL with the version number you would like to use instead. Remember that points will be deducted based on how many days after the deadline you submitted that particular version.