

Homework 1: BasicIM

COSC435 - Homework 1 (BasicIM)

A Simple, Unencrypted Instant Messenger [100 points]

Assigned August 29th, 2019; last modified September 9th, 2019.

Due ~~September 12th, 2019~~ September 13th at 11:59pm.

Overview

To introduce you to network programming, you will build a simple unencrypted instant messenger: BasicIM. BasicIM consists of two parts: a single instance of a *server* and a variable number of BasicIM *clients*. Each client connects to the server, the latter of which is responsible for receiving instant messages from a client and echoing those messages to all other connected clients. In other words, BasicIM provides group chat, similar to [IRC](https://en.wikipedia.org/wiki/Internet_Relay_Chat) [_](https://en.wikipedia.org/wiki/Internet_Relay_Chat), and uses [TCP sockets](https://docs.python.org/3/howto/sockets.html) [_](https://docs.python.org/3/howto/sockets.html) to communicate.

A very helpful reference for this project is [this Python sockets tutorial](https://docs.python.org/3/howto/sockets.html) [_](https://docs.python.org/3/howto/sockets.html).

The BasicIM Client

The BasicIM client program performs two "simultaneous" (more on that later) functions:

1. it reads from [standard input](https://en.wikipedia.org/wiki/Standard_streams) [_](https://en.wikipedia.org/wiki/Standard_streams) and sends whatever was typed by the user to the BasicIM server
2. it reads from a network socket (connected to the BasicIM server) and receives messages (via the server), which it then displays to standard output (i.e., it prints out the received messages).

Your BasicIM client should use TCP to send messages between the server and client instances. Additionally, you should use [Google Protocol Buffers](https://developers.google.com/protocol-buffers/docs/pythontutorial) [_](https://developers.google.com/protocol-buffers/docs/pythontutorial) to [serialize structured data](https://en.wikipedia.org/wiki/Marshalling_(computer_science)) [_](https://en.wikipedia.org/wiki/Marshalling_(computer_science)).

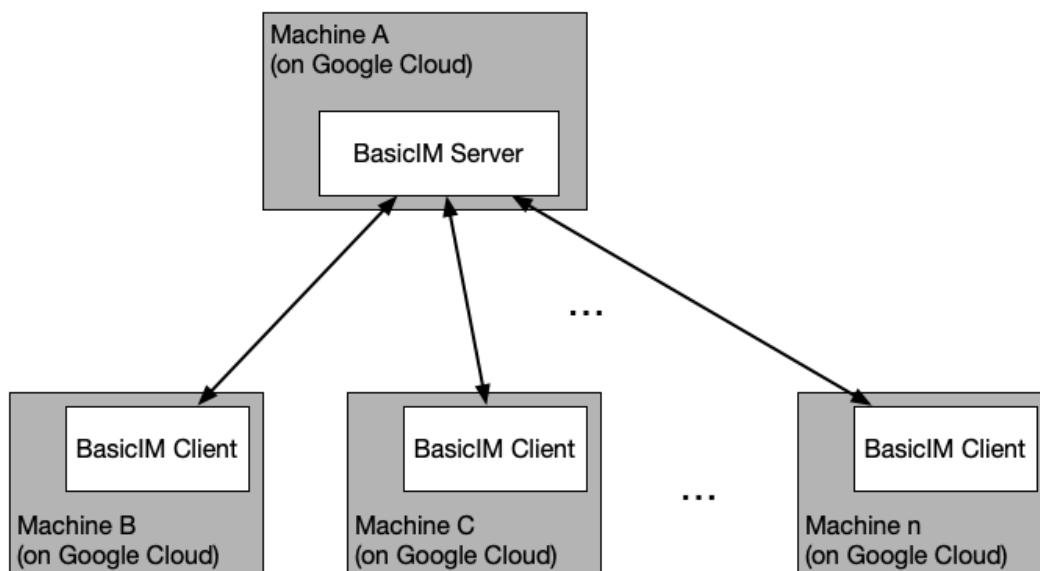
The BasicIM Server

The BasicIM server performs two functions:

1. it waits/listens for new connections from clients

- upon receiving a message from a BasicIM client (encoded using Google Protocol Buffers), it sends a copy of that message to every other client.

The following figure presents the overall architecture of BasicIM. Clients and the server run on separate "compute instances" (you can think of these as machines) on Google Cloud. Clients communicate only with the BasicIM server, via TCP connections.



Program Description

Your solution must be written in Python 3.7 and must run on the Google Cloud Platform. More on the latter below. See also [Creating Your Servers](https://georgetown.instructure.com/courses/82306/pages/creating-your-servers)

(<https://georgetown.instructure.com/courses/82306/pages/creating-your-servers>) for setting up your virtual machines on Google Cloud. (Seriously, read [Creating Your Servers](https://georgetown.instructure.com/courses/82306/pages/creating-your-servers) (<https://georgetown.instructure.com/courses/82306/pages/creating-your-servers>)... it contains instructions for claiming your free Google Cloud credits.)

The BasicIM Client

Your BasicIM client takes as command-line arguments the hostname or IP address of the BasicIM server and a nickname or alias of the user. Specifically, your program should be invoked as:

```
python3 basicIMclient.py -s servername -n nickname
```

where *servername* is the IP or hostname of the machine that is already running the BasicIM server and *nickname* is an alias chosen by the user. The *-s* and *-n* arguments are both mandatory, and your program should accept these arguments in any order (i.e., "python3 basicIMclient.py -n CoolMicah -s foo.com" is valid, as is "python3 basicIMclient.py -s foo.com -n CoolMicah"). Your life will be much

happier if you use [argparse](https://docs.python.org/3/howto/argparse.html) [_ \(https://docs.python.org/3/howto/argparse.html\)](https://docs.python.org/3/howto/argparse.html) to parse the command-line arguments -- it does magical things for you. Fortunately, your very kind instructor [has provided some sample argparse code](https://georgetown.instructure.com/courses/82306/files/2840925/download?wrap=1) [_ \(https://georgetown.instructure.com/courses/82306/files/2840925/download?wrap=1\)](https://georgetown.instructure.com/courses/82306/files/2840925/download?wrap=1) [_ \(https://georgetown.instructure.com/courses/82306/files/2840925/download?wrap=1\)](https://georgetown.instructure.com/courses/82306/files/2840925/download?wrap=1) which you can use as a template. (You don't need to cite this code.)

The client should first connect to the BasicIM server, which should already be running (more on the server below) and listening for incoming connections on TCP port 9999. Once connected, BasicIM should read from standard input (essentially, the keyboard) and send all inputted text to the BasicIM server (which you'll also write). It should also receive messages, which are serialized with [Google Protocol Buffers](https://developers.google.com/protocol-buffers/docs/pythontutorial) [_ \(https://developers.google.com/protocol-buffers/docs/pythontutorial\)](https://developers.google.com/protocol-buffers/docs/pythontutorial) -- thus, your program should deserialize these messages. BasicIM messages, which are serialized with Google Protocol Buffers, consist of:

- the nickname of the sender
- the contents of the instant message

Note that in BasicIM, all messages are "broadcast", meaning that if one client sends a message, all other online clients should receive that message. There are no direct messages in BasicIM.

When your BasicIM client receives a message from the server, it should deserialize it to discern the nickname of the sender and the contents of the instant message. It should then print out (i.e., send to standard output) the following:

```
nickname: message
```


For example, if the user with the nickname "CoolMicah" typed a message, "Wow, COSC435 is REALLY great!", then all other BasicIM clients should output:

```
CoolMicah: Wow, COSC435 is REALLY great!
```

Important: Your program should not print any prompts or messages (other than the nickname, a colon followed by a space, and the content of the instant messages). As discussed in more detail below, your solution will be graded by an autograder, an automated software program, which is super sensitive to non-compliance. In other words, if you get cute and add special prompts or fancy output, this will confuse the autograder and you'll be penalized.

Also, note that it's OK (and expected) for the messages typed on a client to appear on that user's terminal/screen. (Otherwise, it's difficult for the user to see what s/he is typing.) However, the sender of a message should not have that message printed AGAIN once it is sent to the BasicIM server. In other words, the BasicIM server should not repeat a message back to its original sender.

Here's the tricky bit: Received messages should be *immediately* written to standard output. That is, if Alice types a message "Hello!" and hits [ENTER] (i.e., the enter key), then "Hello!" should immediately appear on some other client Bob's screen (along with Alice's nickname). You therefore cannot program your solution to (1) wait for the user to type something, (2) send it, (3) wait for something to arrive in the network, (4) print it, (5) goto step 1. (This fails because Bob could send something to Alice (via the BasicIM server) during step 1, which wouldn't be printed until step 4.)

To allow for asynchronous message delivery, you will need to use the `select()` (<http://man7.org/linux/man-pages/man2/select.2.html>) `system call` (https://en.wikipedia.org/wiki/System_call) to block and wait for input either on standard input or the network socket (that is, the socket that is connected to the BasicIM server). Descriptions of Python's `select` call [are available online](https://docs.python.org/2/library/select.html) (<https://docs.python.org/2/library/select.html>). You should also definitely check out [this sample code](https://georgetown.instructure.com/courses/82306/files/2867436/download?wrap=1) (<https://georgetown.instructure.com/courses/82306/files/2867436/download?wrap=1>)  that I've provided.

Note that the code sample doesn't solve exactly the problem you'll need to solve for this assignment. It is meant only to illustrate Python's `select()` call.

If the user types "exit [ENTER]" (i.e, exit followed by enter/return), then the BasicIM client should close its connection to the BasicIM server and exit (i.e., terminate). The program should not consider the case (i.e., capitalization) when interpreting whether the user typed "exit"; in other words, "EXIT [ENTER]", "exit [ENTER]", and "eXiT [ENTER]" should all cause BasicIM to close.

The BasicIM Server

The BasicIM Server must be started before any of the clients. (Otherwise, the clients have nothing to connect to.) The server program runs continuously once started. (You can terminate it by pressing CTRL-C if you need to.)

The BasicIM Server program takes no command-line arguments. It should be invoked via:

```
python3 basicIMserver.py
```

The server should bind to all addresses and listen for incoming connections from BasicIM clients on port 9999 (TCP). Hence, your code, somewhere, should have a line that looks like:

```
s.bind(('', 9999))
```

where `s` is your socket object. Note that the empty string in the first portion of the tuple sent to `s.bind()` means "listen for connections from any network interface". This is important to include, since compute engine instances on Google Cloud have multiple network interfaces.

Importantly, BasicIM Server should support dynamic lists of clients. That is, it should not crash when a client leaves.

Program naming conventions and other fun requirements

Your program will consist of multiple source files. **You must include with your submission the .proto Google Protocol Buffers definition.** However, you should also include the "compiled" Python classes [that were produced from the Google Protocol Buffers compiler](https://developers.google.com/protocol-buffers/docs/pythontutorial#compiling-your-protocol-buffers) (<https://developers.google.com/protocol-buffers/docs/pythontutorial#compiling-your-protocol-buffers>).

Because your program will be graded by an autograder, you need to be extra careful that you name your programs correctly. (Your supporting Python files can be named whatever you like.) Specifically,

- the server should be named `basicIMserver.py`
- the client should be named `basicIMclient.py`

These are case sensitive!

Computing resources

You may develop your code wherever you see fit (e.g., on [Google Cloud](https://georgetown.instructure.com/courses/82306/pages/creating-your-servers) (<https://georgetown.instructure.com/courses/82306/pages/creating-your-servers>), your personal laptop, a computer in a lab, etc.). However, your code must operate correctly in the autograder, which itself is based on an Debian-based Linux distribution (as is, non-coincidentally, your Google Cloud instances). It is strongly recommended that you develop and test (especially the latter) your code on a virtual machine (VM) that you can create on Google Cloud.

Additional requirements and hints

Please make sure that your program conforms to the following:

- **You should use Python version 3.7** (which is installed in Google Cloud). Don't use Python 2.7x -- it will be deprecated [very soon](https://pythonclock.org/) (<https://pythonclock.org/>).
- You may only use standard libraries as well as those already installed on Google Cloud VM. *Hint:* For this assignment, you should only need to import the `argparse`, `select`, `socket`, and `sys` modules. If you want to get fancy, you may also import `signal` (i.e., to trap CTRL-C keypresses).
- You may not collaborate on this homework. **This project should be done individually.** You may use the [Python documentation pages](https://docs.python.org/) (<https://docs.python.org/>) (or equivalent documentation pages

for your programming language) as a programming language reference, and may look at any programming tutorials. However, you may not base your code off of online tutorials that essentially solve this programming assignment. Please consult the teaching staff via a private Piazza note if you are unsure whether a particular online resource is allowed.

- The automatic grader is VERY picky. Do not provide a prompt to the user, and only write received messages to standard out. We will be using automated testing tools to evaluate your solutions, and printing additional messages or characters makes such automation far more difficult.
- Your program should not take in any additional command-line options other than the ones described above.
- It is OK if messages are only sent after the user presses [ENTER] after entering a line of text. However, incoming messages should be displayed immediately after they are received by the kernel.

In homework 2, (assigned shortly after the due date!), we will add a layer of encryption to our IM applications.

Grading

This homework is worth 100 points. A **non-comprehensive** list of deductions is provided in the rubric below.

We will award partial credit when possible and appropriate. To maximize opportunities for partial credit, please rigorously comment your code. If we cannot understand what you intended, we cannot award partial credit.

Description	Deduction
Compilation / interpreter errors	50
Non-use of Google Protocol Buffers (or, .proto file(s) not included in submission)	33
Compiles, but IMs are neither successfully transmitted nor received	30
Compiles, but IMs are either only transmitted or only received	20
Doesn't support dynamic client joins and leaves	25
Doesn't support nicknames	15
Received messages only appear after user presses [ENTER] (indicates that <i>select</i> is used improperly)	15
General instability	10

Run-time error (e.g., crash) on large input	10
Any other run-time error (e.g., crash)	5
Non-conforming command-line options (hinders automated testing)	5
Includes unnecessary prompts (hinders automated testing)	5

Note that this grading rubric is not intended to be comprehensive. Also, although the deductions above may total more than 100 points, it's not possible to get a negative score on this assignment.

Submission Instructions

You will use the [Autolab](https://autolab.georgetown.edu) (<https://autolab.georgetown.edu>) system for submission and automated grading.

IMPORTANT NOTE: There are two separate "assignments" listed in Autolab for this homework. The first, *hw1-compilecheck* is, as the name implies, a simple check that your program can at least be run by the autograder. There are an unlimited number of submissions to *hw1-compilecheck*. Note that assignments submitted to *hw1-compilecheck* will not be graded. This is purely for your benefit.

The more critical assignment is *hw1-graded*. Uploads to *hw1-graded* on Autolab will be graded. **You may upload at most three times** to *hw1-graded*. The final upload (whether it's upload number one, two, or three) will be the one that is graded. So, think before you upload your submission. Importantly, make sure that it compiles using *hw1-compilecheck*. Don't waste one of your three uploads to *hw1-graded* on something that doesn't compile.

For both *hw1-compilecheck* and *hw1-graded*, submit your solution as a single zip file to [Autolab](https://autolab.georgetown.edu) (<https://autolab.georgetown.edu>). **Super important: make sure that `basicIMclient.py` and `basicIMserver.py` are in the top-level directory of the zip file.** If you put it in "hw1/" or what-have-you, the autograder can't magically find it.

Upload your assignment to *hw1-graded* before 11:59pm on September 12th, 2019. Late assignments will be penalized according to the [Grading Policy](https://georgetown.instructure.com/courses/82306/pages/policy-grading) (<https://georgetown.instructure.com/courses/82306/pages/policy-grading>).

Please post questions (especially requests for clarification) about this homework to [Piazza](https://piazza.com/georgetown/fall2018/cosc435/home) (<https://piazza.com/georgetown/fall2018/cosc435/home>).