ECE 358, S'16 — Assignment 1

Total # Points = 20, Due: Tue, June 14, 11:59:59pm

Design and implement a peer-to-peer content management system. It comprises the following programs you must write.

- `addpeer` – this causes a new peer to be created and added to your system. The optional command-line arguments are the ip-address and port associated with one other peer in your system. If these command-line arguments are not provided, you assume that this is the first peer in the system. You should output the non-loopback ip address and port at which the peer is created. The peer is always created on this host. If the host has more than one non-loopback ip address, you pick any of those addresses to print. Example usage and output:

```
$ addpeer
1.2.3.4 50400
$ addpeer 1.2.3.4 50400
3.4.5.6 12013
$ addpeer 1.2.3.4 50400
3.4.5.6 34232
```

- `removepeer` – this takes as argument the ip-address and port associated with a peer. It causes the peer to be removed from the system. All the content managed by the peer must be redistributed to the other peers before this peer disappears. See also the "load-balancing requirement" below. If this is the only peer in the system, then all the content vanishes with it. Example usage and output:

```
$ removepeer 1.2.3.4 50400
$ removepeer 3.4.5.6 12013
$ removepeer 3.4.5.6 34232
$ removepeer 3.4.5.6 34232
Error: no such peer
```

- `addcontent` – this takes as argument the ip-address and port of some peer in your system, and a piece of content. It adds a new piece of content into the system. You choose a peer to host the content – this is not necessarily the peer that we provide as command-line argument. The content, which is provided as command-line argument, is just a string. This command outputs, to stdout, a unique positive integer key by which this content may be retrieved from your peer-to-peer system. Example usage and output:

```
$ addcontent 1.2.3.4 50400 "Alias Grace"
12
$ addcontent 3.4.5.6 34232 "Come, Thou Tortoise"
328
$ addcontent 1.2.3.4 50400 "No Great Mischief"
33
$ addcontent 3.4.5.6 50400 "Fifth Business"
Error: no such peer
```

- `removecontent` – this removes some content that is hosted by a peer in your system. The ip-address and port of some peer, and the key associated with the content are provided as command-line arguments. As with `addcontent`, the peer provided at the command-line is not necessarily the peer that hosts the content. Example usage and output:

```
$ removecontent 1.2.3.4 50400 44
Error: no such content
$ removecontent 3.4.5.6 12013 12
```

- `lookupcontent` – given an ip-address and port of a peer, and a key as command-line arguments, this retrieves the content from your peer-to-peer system and prints it to stdout. Example usage and output:

```
$ lookupcontent 1.2.3.4 50400 12
Error: no such content
$ lookupcontent 1.2.3.4 50400 328
Come, Thou Tortoise
```

Load-balancing requirement At any moment, you should ensure that each peer maintains an equally fair share of the content items. That is, if at a particular moment we have p peers and c pieces of content, then each peer must host between $\lfloor c/p \rfloor$ and $\lceil c/p \rceil$ pieces of content. For us to query and check this, you have the following additional requirement.

We will write a tcp client that will *connect()* to the ip-address and port of a peer. Once the peer accepts the connection request, we will *send()* the string "allkeys." We will then *recv()*. The peer must send back the keys of all the pieces of content it hosts as a string, separated by comma (","), and terminated by 0. E.g., the string the peer responds with may be "23,4,12,0." Your peer may *shutdown()* the connection as soon as it sends this. (We will *shutdown()* from our end anyway.)

No filesystem use you should not use the filesystem for anything. That means no reading from/writing to files, no temporary files, no named pipes (FIFOs), no devices accessed via the filesystem, etc. Everything you store must be in memory. We will eyeball your source code for violations.

ecelinux Your solution should work on the ecelinux machines. The machines ecelinux2, 3, 5 and 6 are the ones we are allowed to use. If you're accessing from off-campus, you may need to come in through ecelinux4. You can do your development on your own setup, but you'll finally need to ensure that everything works on ecelinux. Because that is where we will do the marking.

It possible that only the ports 10,000–11,000 are availble for your use on the ecelinux machines. We make available a simple subroutine called *mybind()*, which you can use in place of *bind()*.

What you turn-in You should turn in a file named `a1-358s16.zip`, to the appropriate dropbox on Learn. You should turn in only one submission per group. When we unzip your `a1-358s16.zip`, we should get exactly the following. Our automated marking scripts expect exactly this structure when we unzip your file. A sample, conformant `a1-358s16.zip` is uploaded to this module on learn.

```
README
Makefile
all source code, perhaps in sub-folders
```

When our script issues "make," the above commands must be built and deposited in the folder in which we unzip and issue "make."

How we will mark We will not try crazy pathological inputs. But, apart from "normal" test cases, we will also try border cases. E.g., `addcontent` when there is no peer, `removecontent` when there is none to be removed, `removepeer` till all peers are removed and then `lookupcontent` for some content we added earlier, etc.

Group members You should mention the names and student IDs of the members of your group in the `README` file. That is the only content we will look for in the `README` file. (More instructions on group-membership to follow.)