CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not have in their hard drive, may download it from another active peer who does. All communication among peers or between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows; additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open connection, and in response the server provides the peer with a list of other peers who have the RFC; if no such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending the (text) file containing the RFC to A over the same connection; once the file transmission is completed,

the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

- the RFC number (of type integer),
- the title of the RFC (of type string), and
- the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:
• 200 OK
• 400 Bad Request
• 404 Not Found
• 505 P2P-CI Version Not Supported
Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S
The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:
• `ADD`, to add a locally available RFC to the server's index,

• LOOKUP, to find peers that have the specified RFC, and
• LIST, to request the whole index of RFCs from the server.
Also, three header fields are defined:
• Host: the hostname of the host sending the request,
• Port: the port to which the upload server of the host is attached, and
• Title: the title of the RFC
For instance, peer thishost.csc.ncsu.edu who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a ADD request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an ADD simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a LOOKUP may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a LIST lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.
Note: the peer should print the responses it receives from the server to the standard output in the same format.
Submission and Deliverables
Submit your source code *(no object files!)* using submit by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a

`makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no

such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.

• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and

its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).

A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per

line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.
Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and

provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a

new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),

`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```



```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called "Requests for Comments" (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or

between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.
• There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
• When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.
Each item of the linked list of peers contains two elements:
1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:
• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).
Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.
Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a new process that handles all communication with this peer. In particular, this process receives the information from the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes "carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:
• 200 OK
• 400 Bad Request
• 404 Not Found
• 505 P2P-CI Version Not Supported
Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S
The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:
• `ADD`, to add a locally available RFC to the server's index,
• `LOOKUP`, to find peers that have the specified RFC, and
• `LIST`, to request the whole index of RFCs from the server.
Also, three header fields are defined:
• `Host:` the hostname of the host sending the request,
• `Port:` the port to which the upload server of the host is attached, and
• `Title:` the title of the RFC
For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```



```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and

• implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

• There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.

• When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).

• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.

• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.

• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.
When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.
3/4
The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:
• 200 OK
• 400 Bad Request
• 404 Not Found
• 505 P2P-CI Version Not Supported
Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S
The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:
• `ADD`, to add a locally available RFC to the server's index,
• `LOOKUP`, to find peers that have the specified RFC, and
• `LIST`, to request the whole index of RFCs from the server.
Also, three header fields are defined:
• `Host:` the hostname of the host sending the request,
• `Port:` the port to which the upload server of the host is attached, and
• `Title:` the title of the RFC
For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this

P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not

scalable to very large number of peers and/or RFCs, it will do for this project.
Each item of the linked list of peers contains two elements:
1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.
2/4
Each item of the linked list representing the index of RFCs contains these elements:
• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).
Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.
Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.
When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.1

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes “space,” `<cr>` denotes
“carriage return,” and `<lf>` denotes “line feed.”

```
method <sp> RFC number <sp> version <cr> <lf>
```

```
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available

and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not have in their hard drive, may download it from another active peer who does. All communication among peers or between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows; additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open connection, and in response the server provides the peer with a list of other peers who have the RFC; if no such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data structures: a list with information about the currently active peers and the index of RFCs available at each peer. For simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

- the RFC number (of type integer),
- the title of the RFC (of type string), and
- the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a new process that handles all communication with this peer. In particular, this process receives the information from the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:
• 200 OK
• 400 Bad Request
• 404 Not Found
• 505 P2P-CI Version Not Supported
Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S
The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:
• `ADD`, to add a locally available RFC to the server's index,

• `LOOKUP`, to find peers that have the specified RFC, and
• `LIST`, to request the whole index of RFCs from the server.
Also, three header fields are defined:
• `Host:` the hostname of the host sending the request,
• `Port:` the port to which the upload server of the host is attached, and
• `Title:` the title of the RFC
For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.
Note: the peer should print the responses it receives from the server to the standard output in the same format.
Submission and Deliverables
Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a

`makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open

connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.

• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local

port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).

A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
```

```
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a

different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.
Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called "Requests for Comments" (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which

keeps information about the active peers and maintains an index of the RFCs available at each active peer.

• When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).

• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.

• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.

• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple

simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request

• 404 Not Found
• 505 P2P-CI Version Not Supported
Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S
The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:
• `ADD`, to add a locally available RFC to the server's index,
• `LOOKUP`, to find peers that have the specified RFC, and
• `LIST`, to request the whole index of RFCs from the server.
Also, three header fields are defined:
• `Host:` the hostname of the host sending the request,
• `Port:` the port to which the upload server of the host is attached, and
• `Title:` the title of the RFC
For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
```

```
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

- the RFC number (of type integer),
- the title of the RFC (of type string), and
- the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are

updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.
When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```



```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
```

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```
while a list request would be:
```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```
The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.
```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```
In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:
```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```
The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,

• creating client processes that contact a well-known server and exchange data over the Internet,
• defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
• creating and managing a centralized index at the server based on information provided by the peers, and
• implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

• There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
• When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a new process that handles all communication with this peer. In particular, this process receives the information from the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes “space,” `<cr>` denotes “carriage return,” and `<lf>` denotes “line feed.”

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which

the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
```

```
ADD RFC 2345 P2P-CI/1.0
```
4/4
```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```
Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:
```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```
while a list request would be:
```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```
The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.
```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```
In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:
```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```
The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

1/4

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For

simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.
Each item of the linked list of peers contains two elements:
1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.
2/4
Each item of the linked list representing the index of RFCs contains these elements:
• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).
Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.
Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.
When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes “space,” `<cr>` denotes

“carriage return,” and <lf> denotes “line feed.”

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, GET, and only two header fields, Host (the hostname of the peer from which
the RFC is requested) and OS (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: Date (the date the response was sent), OS (operating system of responding host),
Last-Modified (the date/time the file was last modified), Content-Length (the length of the file in bytes),
and Content-Type (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- ADD, to add a locally available RFC to the server’s index,
- LOOKUP, to find peers that have the specified RFC, and
- LIST, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- Host: the hostname of the host sending the request,
- Port: the port to which the upload server of the host is attached, and
- Title: the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not have in their hard drive, may download it from another active peer who does. All communication among peers or between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows; additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open connection, and in response the server provides the peer with a list of other peers who have the RFC; if no such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is

not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

- the RFC number (of type integer),
- the title of the RFC (of type string), and
- the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at

the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).

A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
```

```
header field name <sp> value <cr> <lf>
<cr> <lf>
```
There are three methods:
• `ADD`, to add a locally available RFC to the server's index,
• `LOOKUP`, to find peers that have the specified RFC, and
• `LIST`, to request the whole index of RFCs from the server.
Also, three header fields are defined:
• `Host:` the hostname of the host sending the request,
• `Port:` the port to which the upload server of the host is attached, and
• `Title:` the title of the RFC
For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:
```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```
Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:
```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```
while a list request would be:
```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```
The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.
```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```
In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:
```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```
The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.
Note: the peer should print the responses it receives from the server to the standard output in the same format.
Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.

• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).

A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
```

```
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a

different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.
Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called "Requests for Comments" (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which

keeps information about the active peers and maintains an index of the RFCs available at each active peer.

• When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).

• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to handle the communication to each new peer.

• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open connection, and in response the server provides the peer with a list of other peers who have the RFC; if no such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.

• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data structures: a list with information about the currently active peers and the index of RFCs available at each peer. For simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple

simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request

• 404 Not Found
• 505 P2P-CI Version Not Supported
Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S
The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:
• `ADD`, to add a locally available RFC to the server's index,
• `LOOKUP`, to find peers that have the specified RFC, and
• `LIST`, to request the whole index of RFCs from the server.
Also, three header fields are defined:
• `Host:` the hostname of the host sending the request,
• `Port:` the port to which the upload server of the host is attached, and
• `Title:` the title of the RFC
For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
```

```
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

- the RFC number (of type integer),
- the title of the RFC (of type string), and
- the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are

updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.
When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```



```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
```

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```
while a list request would be:
```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```
The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.
```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```
In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:
```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```
The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,

• creating client processes that contact a well-known server and exchange data over the Internet,
• defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
• creating and managing a centralized index at the server based on information provided by the peers, and
• implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

• There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
• When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a new process that handles all communication with this peer. In particular, this process receives the information from the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes “space,” `<cr>` denotes “carriage return,” and `<lf>` denotes “line feed.”

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which

the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
```

```
ADD RFC 2345 P2P-CI/1.0
```
4/4
```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```
Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:
```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```
while a list request would be:
```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```
The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.
```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```
In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:
```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```
The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

1/4

CSC 573 – Internet Protocols
Project #1

Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not have in their hard drive, may download it from another active peer who does. All communication among peers or between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows; additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open connection, and in response the server provides the peer with a list of other peers who have the RFC; if no such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.
Each item of the linked list of peers contains two elements:
1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.
2/4
Each item of the linked list representing the index of RFCs contains these elements:
• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).
Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.
Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.
When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.1

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host somehost.csc.ncsu.edu.
Then, A sends to B a request message formatted as follows, where <sp> denotes "space," <cr> denotes
"carriage return," and <lf> denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, GET, and only two header fields, Host (the hostname of the peer from which
the RFC is requested) and OS (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.
3/4
The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:
• 200 OK
• 400 Bad Request
• 404 Not Found
• 505 P2P-CI Version Not Supported
Five header fields are defined: Date (the date the response was sent), OS (operating system of responding host),
Last-Modified (the date/time the file was last modified), Content-Length (the length of the file in bytes),
and Content-Type (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S
The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:
• ADD, to add a locally available RFC to the server's index,
• LOOKUP, to find peers that have the specified RFC, and
• LIST, to request the whole index of RFCs from the server.

Also, three header fields are defined:
• `Host:` the hostname of the host sending the request,
• `Port:` the port to which the upload server of the host is attached, and
• `Title:` the title of the RFC
For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.
Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,

if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not have in their hard drive, may download it from another active peer who does. All communication among peers or between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows; additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open connection, and in response the server provides the peer with a list of other peers who have the RFC; if no such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each

peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.

• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data structures: a list with information about the currently active peers and the index of RFCs available at each peer. For simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a new process that handles all communication with this peer. In particular, this process receives the information from the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may

send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).

A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```



```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and

phrase is returned, and the data part of the response is empty.
Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as

long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from

the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),

and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```



```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host

containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called "Requests for Comments" (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which

keeps information about the active peers and maintains an index of the RFCs available at each active peer.

• When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).

• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to handle the communication to each new peer.

• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open connection, and in response the server provides the peer with a list of other peers who have the RFC; if no such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.

• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data structures: a list with information about the currently active peers and the index of RFCs available at each peer. For simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple

simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request

• 404 Not Found
• 505 P2P-CI Version Not Supported
Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S
The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:
• `ADD`, to add a locally available RFC to the server's index,
• `LOOKUP`, to find peers that have the specified RFC, and
• `LIST`, to request the whole index of RFCs from the server.
Also, three header fields are defined:
• `Host:` the hostname of the host sending the request,
• `Port:` the port to which the upload server of the host is attached, and
• `Title:` the title of the RFC
For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
```

```
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called "Requests for Comments" (RFCs). RFCs are available

for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

- the RFC number (of type integer),
- the title of the RFC (of type string), and
- the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to

the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.
When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes “space,” `<cr>` denotes
“carriage return,” and `<lf>` denotes “line feed.”

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host), `Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes), and `Content-Type` (always text/plain for this project).

A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```



```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

```
Title: Requirements for IPsec Remote Access Scenarios
```
while a list request would be:
```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```
The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.
```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```
In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:
```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```
The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,

• creating client processes that contact a well-known server and exchange data over the Internet,
• defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
• creating and managing a centralized index at the server based on information provided by the peers, and
• implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

• There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
• When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a new process that handles all communication with this peer. In particular, this process receives the information from the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes “space,” `<cr>` denotes “carriage return,” and `<lf>` denotes “line feed.”

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which

the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
```

```
ADD RFC 2345 P2P-CI/1.0
```

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```
Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:
```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```
while a list request would be:
```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```
The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.
```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```
In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:
```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```
The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For

simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.
Each item of the linked list of peers contains two elements:
1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.
2/4
Each item of the linked list representing the index of RFCs contains these elements:
• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).
Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.
Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.
When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.1

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes “space,” `<cr>` denotes

“carriage return,” and <lf> denotes “line feed.”

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, GET, and only two header fields, Host (the hostname of the peer from which
the RFC is requested) and OS (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: Date (the date the response was sent), OS (operating system of responding host),
Last-Modified (the date/time the file was last modified), Content-Length (the length of the file in bytes),
and Content-Type (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- ADD, to add a locally available RFC to the server’s index,
- LOOKUP, to find peers that have the specified RFC, and
- LIST, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- Host: the hostname of the host sending the request,
- Port: the port to which the upload server of the host is attached, and
- Title: the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not have in their hard drive, may download it from another active peer who does. All communication among peers or between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows; additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open connection, and in response the server provides the peer with a list of other peers who have the RFC; if no such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is

not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

- the RFC number (of type integer),
- the title of the RFC (of type string), and
- the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at

the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).

A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
```

```
header field name <sp> value <cr> <lf>
<cr> <lf>
```
There are three methods:
• `ADD`, to add a locally available RFC to the server's index,
• `LOOKUP`, to find peers that have the specified RFC, and
• `LIST`, to request the whole index of RFCs from the server.
Also, three header fields are defined:
• `Host:` the hostname of the host sending the request,
• `Port:` the port to which the upload server of the host is attached, and
• `Title:` the title of the RFC
For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:
```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```
Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:
```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```
while a list request would be:
```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```
The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.
```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```
In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:
```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```
The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.
Note: the peer should print the responses it receives from the server to the standard output in the same format.
Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to

handle the communication to each new peer.
• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).

A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
```

```
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a

different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.
Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called "Requests for Comments" (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and

provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a

new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),

`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```



```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called "Requests for Comments" (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not

have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.
• There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
• When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.
Each item of the linked list of peers contains two elements:
1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:
• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).
Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.
Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate

record and inserts it at the front of the linked list representing the index.
When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
```

```
<cr> <lf>
data
```
Four status codes and corresponding phrases are defined:
- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:
```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```
Application Layer Protocol: P2S
The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:
```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```
There are three methods:
- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:
- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:
```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```
Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:
```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```
while a list request would be:
```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
```

`Port: 5678`

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,

• creating and managing a centralized index at the server based on information provided by the peers, and
• implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

• There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
• When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:
1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:
• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a new process that handles all communication with this peer. In particular, this process receives the information from the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes "carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message would look like this:

```
GET RFC 1234 P2P-CI/1.0
```

```
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```
1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

3/4

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).

A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this

P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not

scalable to very large number of peers and/or RFCs, it will do for this project.
Each item of the linked list of peers contains two elements:
1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.
2/4
Each item of the linked list representing the index of RFCs contains these elements:
• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).
Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.
Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.
When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.1

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes “space,” `<cr>` denotes
“carriage return,” and `<lf>` denotes “line feed.”

```
method <sp> RFC number <sp> version <cr> <lf>
```

```
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available

and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not have in their hard drive, may download it from another active peer who does. All communication among peers or between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows; additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open connection, and in response the server provides the peer with a list of other peers who have the RFC; if no such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data structures: a list with information about the currently active peers and the index of RFCs available at each peer. For simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

- the RFC number (of type integer),
- the title of the RFC (of type string), and
- the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a new process that handles all communication with this peer. In particular, this process receives the information from the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:
• 200 OK
• 400 Bad Request
• 404 Not Found
• 505 P2P-CI Version Not Supported
Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S
The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:
• `ADD`, to add a locally available RFC to the server's index,

• LOOKUP, to find peers that have the specified RFC, and
• LIST, to request the whole index of RFCs from the server.
Also, three header fields are defined:
• Host: the hostname of the host sending the request,
• Port: the port to which the upload server of the host is attached, and
• Title: the title of the RFC
For instance, peer thishost.csc.ncsu.edu who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a ADD request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an ADD simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a LOOKUP may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a LIST lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.
Note: the peer should print the responses it receives from the server to the standard output in the same format.
Submission and Deliverables
Submit your source code *(no object files!)* using submit by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a

`makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no

such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.

• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and

its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).

A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per

line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.
Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and

provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a

new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),

`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```



```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called "Requests for Comments" (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or

between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.
• There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
• When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.
Each item of the linked list of peers contains two elements:
1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:
• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).
Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.
Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a new process that handles all communication with this peer. In particular, this process receives the information from the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes "carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:
• 200 OK
• 400 Bad Request
• 404 Not Found
• 505 P2P-CI Version Not Supported
Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S
The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:
• `ADD`, to add a locally available RFC to the server's index,
• `LOOKUP`, to find peers that have the specified RFC, and
• `LIST`, to request the whole index of RFCs from the server.
Also, three header fields are defined:
• `Host:` the hostname of the host sending the request,
• `Port:` the port to which the upload server of the host is attached, and
• `Title:` the title of the RFC
For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```



```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and

• implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

• There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.

• When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).

• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.

• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.

• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.
When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.
3/4
The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:
• 200 OK
• 400 Bad Request
• 404 Not Found
• 505 P2P-CI Version Not Supported
Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S
The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:
• `ADD`, to add a locally available RFC to the server's index,
• `LOOKUP`, to find peers that have the specified RFC, and
• `LIST`, to request the whole index of RFCs from the server.
Also, three header fields are defined:
• `Host:` the hostname of the host sending the request,
• `Port:` the port to which the upload server of the host is attached, and
• `Title:` the title of the RFC
For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this

P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not

scalable to very large number of peers and/or RFCs, it will do for this project.
Each item of the linked list of peers contains two elements:
1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.
2/4
Each item of the linked list representing the index of RFCs contains these elements:
• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).
Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.
Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.
When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.1

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes “space,” `<cr>` denotes
“carriage return,” and `<lf>` denotes “line feed.”

```
method <sp> RFC number <sp> version <cr> <lf>
```

```
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available

and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not have in their hard drive, may download it from another active peer who does. All communication among peers or between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows; additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open connection, and in response the server provides the peer with a list of other peers who have the RFC; if no such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data structures: a list with information about the currently active peers and the index of RFCs available at each peer. For simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

- the RFC number (of type integer),
- the title of the RFC (of type string), and
- the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a new process that handles all communication with this peer. In particular, this process receives the information from the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:
• 200 OK
• 400 Bad Request
• 404 Not Found
• 505 P2P-CI Version Not Supported
Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S
The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:
• `ADD`, to add a locally available RFC to the server's index,

• `LOOKUP`, to find peers that have the specified RFC, and
• `LIST`, to request the whole index of RFCs from the server.
Also, three header fields are defined:
• `Host:` the hostname of the host sending the request,
• `Port:` the port to which the upload server of the host is attached, and
• `Title:` the title of the RFC
For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.
Note: the peer should print the responses it receives from the server to the standard output in the same format.
Submission and Deliverables
Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a

`makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open

connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.

• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local

port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.

Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.

Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

1 Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).

A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
```

```
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a

different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.
Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called "Requests for Comments" (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which

keeps information about the active peers and maintains an index of the RFCs available at each active peer.

• When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).

• Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.

• When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.

• Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

• the RFC number (of type integer),
• the title of the RFC (of type string), and
• the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are
updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.

The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.

When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple

simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request

• 404 Not Found
• 505 P2P-CI Version Not Supported
Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S
The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:
• `ADD`, to add a locally available RFC to the server's index,
• `LOOKUP`, to find peers that have the specified RFC, and
• `LIST`, to request the whole index of RFCs from the server.
Also, three header fields are defined:
• `Host:` the hostname of the host sending the request,
• `Port:` the port to which the upload server of the host is attached, and
• `Title:` the title of the RFC
For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```

4/4

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```

while a list request would be:

```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```

The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.

```
version <sp> status code <sp> phrase <cr> <lf>
```

```
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```

In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:

```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```

The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.

CSC 573 – Internet Protocols
Project #1
Spring 2014

Project Objectives

In this project, you will implement a simple peer-to-peer (P2P) system with a centralized index (CI). Although this
P2P-CI system is rather elementary, in the process I expect that you will develop a good understanding of P2P and
client-server systems and build a number of fundamental skills related to writing Internet applications, including:

- becoming familiar with network programming and the socket interface,
- creating server processes that wait for connections,
- creating client processes that contact a well-known server and exchange data over the Internet,
- defining a simple application protocol and making sure that peers and server follow precisely the specifications for their side of the protocol in order to accomplish particular tasks,
- creating and managing a centralized index at the server based on information provided by the peers, and
- implementing a concurrent server that is capable of carrying out communication with multiple clients simultaneously.

Peer-to-Peer with Centralized Index (P2P-CI) System for Downloading RFCs

Internet protocol standards are defined in documents called “Requests for Comments” (RFCs). RFCs are available
for download from the IETF web site (http://www.ietf.org/). Rather than using this centralized server for downloading RFCs, you will build a P2P-CI system in which peers who wish to download an RFC that they do not
have in their hard drive, may download it from another active peer who does. All communication among peers or
between a peer and the server will take place over TCP. Specifically, the P2P-CI system will operate as follows;
additional details on each component of the system will be provided shortly.

- There is a centralized server, running on a well-known host and listening on a well-known port, which keeps information about the active peers and maintains an index of the RFCs available at each active peer.
- When a peer decides to join the P2P-CI system, it opens a connection to the server to register itself and provide information about the RFCs that it makes available to other peers. This connection remains open as
long as the peer remains active; the peer closes the connection when it leaves the system (becomes inactive).
- Since the server may have connections open to multiple peers simultaneously, it spawns a new process to
handle the communication to each new peer.
- When a peer wishes to download a specific RFC, it provides the RFC number to the server over the open
connection, and in response the server provides the peer with a list of other peers who have the RFC; if no
such active peer exists, an appropriate message is transmitted to the requesting peer. Additionally, each peer may at any point query the server to obtain the whole index of RFCs available at all other active peers.
- Each peer runs a *upload server* process that listens on a port *specific to the peer* ; in other words, this port is
not known in advance to any of the peers. When a peer A needs to download an RFC from a peer B, it opens a connection to the upload port of peer B, provides the RFC number to B, and B responds by sending
the (text) file containing the RFC to A over the same connection; once the file transmission is completed, the connection is closed.

The Server

The server waits for connections from the peers on the well-known port 7734. The server maintains two data
structures: a list with information about the currently active peers and the index of RFCs available at each peer. For
simplicity, you will implement both these structures as linked lists; while such an implementation is obviously not
scalable to very large number of peers and/or RFCs, it will do for this project.

Each item of the linked list of peers contains two elements:

1. the hostname of the peer (of type string), and
2. the port number (of type integer) to which the upload server of this peer is listening.

Each item of the linked list representing the index of RFCs contains these elements:

- the RFC number (of type integer),
- the title of the RFC (of type string), and
- the hostname of the peer containing the RFC (of type string).

Note that the index may contain multiple records of a given RFC, one record for each peer that contains the RFC.

Initially (i.e., when the server starts up), both linked lists are empty (do not contain any records). The linked lists are

updated as peers join and leave the system. When a peer joins the system, it provides its hostname and upload port to
the server (as explained below) and the server creates a new peer record and inserts it at the front of the linked list.
The peer also provides the server with a list of RFCs that it has. For each RFC, the server creates an appropriate
record and inserts it at the front of the linked list representing the index.
When a peer leaves the system (i.e., closes its connection to the server), the server searches both linked lists and
removes all records associated with this peer. As we mentioned earlier, the server must be able to handle multiple
simultaneous connections from peers. To this end, it has a main process that initializes the two linked-lists to empty
and then listens to the well-known port 7734. When a connection from a peer is received, the main process spawns a
new process that handles all communication with this peer. In particular, this process receives the information from
the peer and updates the peer list and index, and it also returns any information requested by the peer. When the peer
closes the connection, this process removes all records associated with the peer and then terminates.[1]

The Peers

When a peer wishes to join the system, it first instantiates an upload server process listening to any available local
port. It then creates a connection to the server at the well-known port 7734 and passes information about itself and
its RFCs to the server, as we describe shortly. It keeps this connection open until it leaves the system. The peer may
send requests to the server over this open connection and receive responses (e.g., the hostname and upload port of a
server containing a particular RFC). When it wishes to download an RFC, it opens a connection to a remote peer at
the specified upload port, requests the RFC, receives the file and stores it locally, and the closes this download
connection to the peer.

The Application Layer Protocol: P2P

The protocol used to download files among peers is a simplified version of the HTTP protocol we discussed in class.
Suppose that peer A wishes to download RFC 1234 from peer B running at host `somehost.csc.ncsu.edu`.
Then, A sends to B a request message formatted as follows, where `<sp>` denotes "space," `<cr>` denotes
"carriage return," and `<lf>` denotes "line feed."

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There is only one method defined, `GET`, and only two header fields, `Host` (the hostname of the peer from which
the RFC is requested) and `OS` (the operating system of the requesting host). For instance, a typical request message
would look like this:

```
GET RFC 1234 P2P-CI/1.0
Host: somehost.csc.ncsu.edu
OS: Mac OS 10.4.1
```

[1] Since multiple processes may manipulate the linked lists of the server simultaneously, these lists must be locked so that at most one process
may access them to read or modify at any one time. For simplicity, we will not require that you implement locking.

The response message is formatted as follows:

```
version <sp> status code <sp> phrase <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
...
<cr> <lf>
data
```

Four status codes and corresponding phrases are defined:

- 200 OK
- 400 Bad Request
- 404 Not Found
- 505 P2P-CI Version Not Supported

Five header fields are defined: `Date` (the date the response was sent), `OS` (operating system of responding host),
`Last-Modified` (the date/time the file was last modified), `Content-Length` (the length of the file in bytes),
and `Content-Type` (always text/plain for this project).
A typical response message looks like this:

```
P2P-CI/1.0 200 OK
Date: Wed, 12 Feb 2009 15:12:05 GMT
OS: Mac OS 10.2.1
Last-Modified: Thu, 21 Jan 2001 9:23:46 GMT
Content-Length: 12345
Content-Type: text/text
(data data data ...)
```

Application Layer Protocol: P2S

The protocol used between a peer and the server is also a request-response protocol, where requests are initiated by
peers. The format of a request message is as follows:

```
method <sp> RFC number <sp> version <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>
```

There are three methods:

- `ADD`, to add a locally available RFC to the server's index,
- `LOOKUP`, to find peers that have the specified RFC, and
- `LIST`, to request the whole index of RFCs from the server.

Also, three header fields are defined:

- `Host:` the hostname of the host sending the request,
- `Port:` the port to which the upload server of the host is attached, and
- `Title:` the title of the RFC

For instance, peer `thishost.csc.ncsu.edu` who has two RFCs, RFC 123 and RFC 2345 locally available
and whose upload port is 5678 would first transmit the following two requests to the server:

```
ADD RFC 123 P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
Title: A Proferred Official ICP
ADD RFC 2345 P2P-CI/1.0
```



```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Domain Names and Company Name Retrieval
```

Once a peer downloads a new RFC from another peer, it may transmit a `ADD` request to add a new record into the
server's index. A lookup request from this host would like this:

```
LOOKUP RFC 3457 P2P-CI/1.0
```

```
Host: thishost.csc.ncsu.edu
Port: 5678
Title: Requirements for IPsec Remote Access Scenarios
```
while a list request would be:
```
LIST ALL P2P-CI/1.0
Host: thishost.csc.ncsu.edu
Port: 5678
```
The response message from the server is formatted as follows, where the status code and corresponding phrase are
the same as defined above.
```
version <sp> status code <sp> phrase <cr> <lf>
<cr> <lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
RFC number <sp> RFC title <sp> hostname <sp> upload port number<cr><lf>
...
<cr><lf>
```
In other words, the data part of the response lists one RFC per line, along with the information about the host
containing the RFC. The response to an `ADD` simply echoes back the information provided by the host:
```
P2P-CI/1.0 200 OK
RFC 123 A Proferred Official ICP thishost.csc.ncsu.edu 5678
```
The response to a `LOOKUP` may contain multiple lines for a given RFC, each line containing information about a
different peer having the RFC. Finally, the response to a `LIST` lists all the records in the server's database, one per
line. If the request contains an error or the requested RFC is not found in the index, an appropriate status code and
phrase is returned, and the data part of the response is empty.

Note: the peer should print the responses it receives from the server to the standard output in the same format.

Submission and Deliverables

Submit your source code *(no object files!)* using `submit` by midnight on the day due. There are several weeks
until the due date for you to work on this project, therefore no late submissions will be accepted. Include a `makefile` with your submission, or a file with instructions on how to compile and run your code. We would like
to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore,
if you fail to include such a file, we will *subtract 5 points* from your project grade.