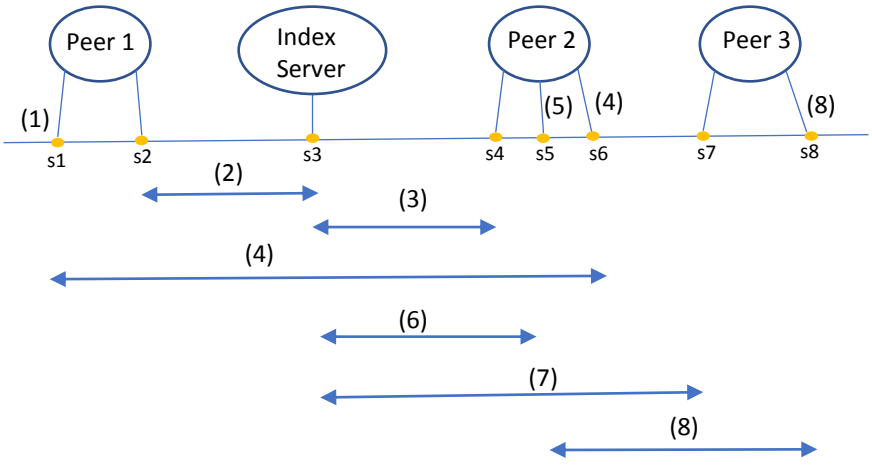
COE768 Project: P2P Application

I. General Description

In this project, a P2P (Peer-to-Peer) application will be developed. The application consists of an index server and a number of peers. Under this application, peers can exchange content among themselves through the support of the index server. A peer that has a piece of content (a movie, a song or a text file) available for download by other peers is called the content server of that content. Similarly, a peer that wants to download a piece of content is called the content client of that content. A peer can be both a content server of a set of content and a content client of another set of content. The content server registers its content to the index server. In turn, the content client finds the address of a content server from the index server. The communication between the index server and a peer is based on UDP while the content download is based on TCP. The following figure illustrates the P2P mechanism.



In the figure, sockets s2, s3, s4 and s7 are UDP sockets. Peer 1, Peer 2 and Peer 3 use their respective UDP sockets to communicate with the Index server. The numbers in the figure represents a chronological sequence of events. To start with, Peer 1 wants to make a piece of its content available, so it creates a TCP passive socket s1 (event (1)) as all TCP servers do (Lab 3). Peer 1 then registers its content to the index server (event (2)). The registration should include the address of the server consisting IP address of Peer 1 and port number associated with s1. After the content registration, Peer2 can download the content by first contacting the index server to find out the address of the content server (event (3)). After finding the address, Peer2 establishes a TCP connection with Peer1 and proceeds with the download (event (4)). Once the download is completed, Peer 2 will also register itself as a content server of the download content (events (5) and (6)). When Peer 3 inquires the same content from the index server, the index server can respond with the server address of either Peer 1 or Peer 2. In the figure, the index server responds with the address of Peer 2 (event (7)). Finally, Peer 3 download the content from Peer 2 (event (8)).

II. Protocol Data Unit Format

The protocol data unit (PDU) has the following simple format:

However, depending on the PDU type, addition data structure may be imposed in the Data field. The Type field has the size of one byte. It specifies the PDU type. There are eight PDU types. The table below summarizes the functions of each type.

PDU type	Function	Direction
R	Content Registration	Peer to Index Server
D	Content Download Request	Content Client to Content Server
S	Search for content and the associated content server	Between Peer and Index Server
T	Content De-Registration	Peer to Index Server
C	Content Data	Content Server to Content Client
O	List of On-Line Registered Content	Between Peer and Index Server
A	Acknowledgement	Index Server to Peer
E	Error	Between Peers or between Peer and Index Server

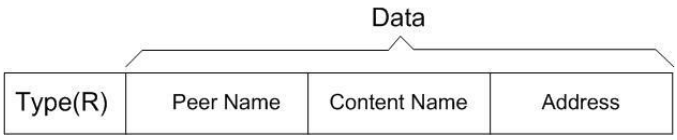
The Data field of all the PDU types, except the C-type PDU, has a maximum size of 100 bytes. The C-type PDU is used to carry content data. The size of its Data field is the size of the content. Since the size of the content could be larger than the maximum size of the TCP packet (For Ethernet, it is less than 1460 bytes), the Data field may have to be broken up into a number of packets for transmission. The file data transmission mechanism used in Lab 3 could be adopted here.

III. Protocol Description

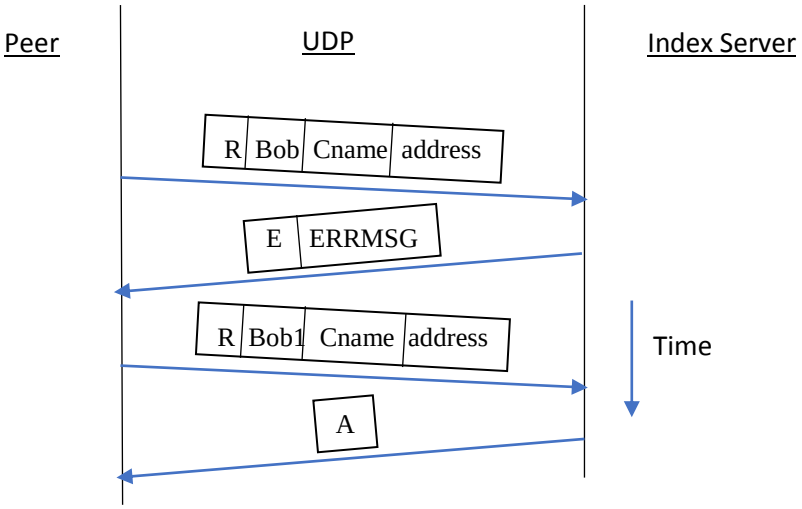
The communication between a peer and the index server is based on UDP. The index server provides services of content registration, deregistration and search. In addition, it will respond with the list of registered content upon the request from a peer.

1. Content Registration

A peer can register its content to the index server by sending an R-type PDU. The data portion of the PDU contains the peer name, content name, and the address of the content server for that content. The following figure shows the format.



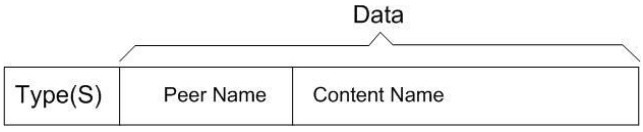
To simplify the implementation, we fix the sizes of the peer name and content name fields to 10 bytes each. When the index server receives an R-type PDU, it will first check if another peer with the same name has registered the same content. If this happens, the index server will send an E-type PDU to prompt the peer to choose another peer name. If there is no conflict of peer name, the content server will register the content and store the associated address. Subsequently, it sends back an A-type PDU to acknowledge the success of the content registration. Note that before sending a registration request, a peer must first create a TCP socket for content download. The port number associated with the socket is part of the address registered to the index server. The following figure illustrates the registration procedure performed between a peer and the index server in the situation where the peer needs to choose another peer name due to the peer name conflict.



If a peer registers more than one piece of content, it should open a separate TCP socket for each piece of content.

2. Content Download

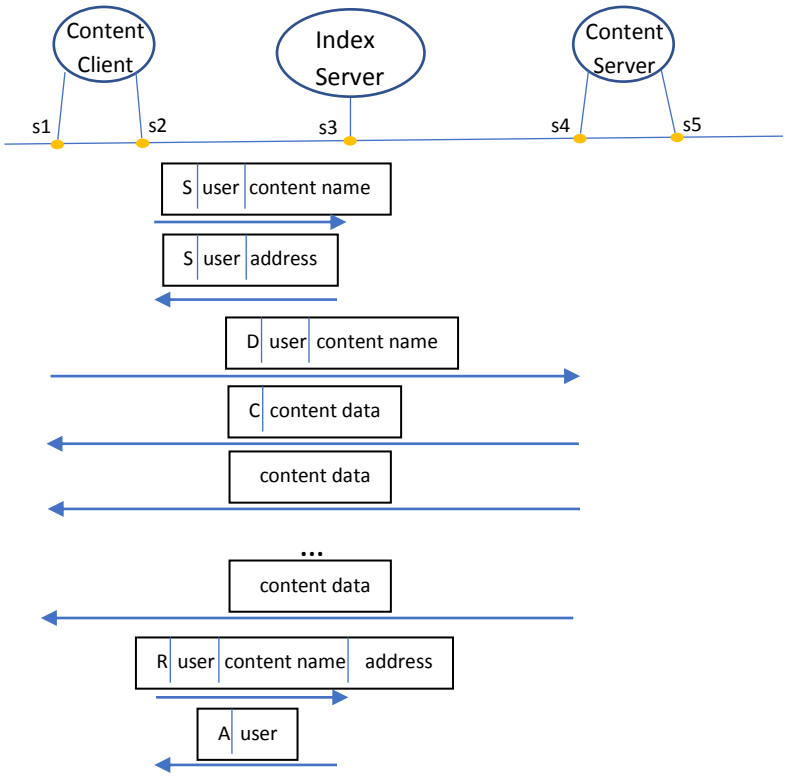
To download a piece of content, a peer first contacts the index server to look for the address of the corresponding content server. It does that by sending an S-type PDU with the following format:



The Peer Name is the name of the peer. The index server responds by sending either an S-type PDU which contains the address of a content server or an E-type PDU which implies that no such content is available.

If the peer receives an S-type PDU back from the index server, it will extract the address from the PDU and setup a TCP connection with the content server. If the TCP connection is successfully established, the peer sends a D-type PDU to the content server to trigger a download. The D-type PDU holds the name of the content in the data field. Upon the reception of the D-type PDU and if the content is available, the content server will deliver the content by sending a C-type PDU which contains the content. As mentioned above, a C-type PDU could be too large to transmit in one TCP packet, in such a case, the PDU must be broken up into number of packets. Once all the data are sent, the content server will terminate the TCP connection, thus, inform the client that the download is completed. This approach is similar to that of Lab 3.

After downloading the content, the peer registers the content to the index server, thus, becomes the server of the content itself. Note that the index server may have more than one server for a given content. In order to distribute the load evenly, upon the reception of the search request of a content, the index server chooses a content server which has been used least. For example, suppose Peer 1 is the first to register as the content server of a given content; subsequently, Peer 2 downloads the content from Peer 1 and also became the server of the content. If another peer wants to download the content, the index server should send the address of the content server associated with Peer 2. It is because the content server of Peer 2 has not been used for download while the content server of Peer 1 has been used at least once. The following figure illustrates the PDU transactions for the content search, downloading and the subsequent content registration.



In the figure, s2, s3 and s5 are UDP sockets while s1 and s4 are TCP sockets. The content client uses s2 to search and register content and s1 to download the content.

3. Content Listing

The user can find out the list of registered contents by sending an O-type PDU to the index server. The index server will respond with an O-type PDU that contains the list of registered contents.

4. Content De-Registration

The application should provide an option to allow a peer to de-register the content. To do this, the peer sends a T-type PDU to the index server to de-register the content. Upon a successful deregistration, the index server will send back an ack (A-type PDU).

5. Quit

In order to keep the content registration information at the index server up-to-date, whenever a peer wants to quit, it must first de-register all its registered content. This can be accomplished by sending a series of T-type PDUs to the index server before quitting. Each T-type PDU will de-register one content.

IV. Programming Information

The following system calls are recommended for your project.

getsocketname system call

When a peer registers as a content server, it opens a socket to be used by other peers for downloading the content. The address to which that socket is bounded to must be sent to the index server during the registration. The address consists of IP address and port number. Since the IP address is known (it ties to the machine where the peer running on), the peer just needs the port number information. In all the previous Labs, the port number of the server is assigned by the user. In this project, it is more appropriate to let the TCP module in the OS to select the port number dynamically. The following is the code to do just that.

```
struct sockaddr_in reg_addr
s = socket(AF_INET, SOCK_STREAM, 0)
reg_addr.sin_family = AF_INET;
reg_addr.sin_port = htons(0);
reg_addr.sin_addr.s_addr = htonl(INADDR_ANY);
bind(s, (struct sockaddr *)&reg_addr, sizeof(reg_addr))
```

By assign the port number to 0 in line 4, the peer requests the TCP module to choose a unique port number. To obtain the address, the peer can call *getsockname*.

```
alen = sizeof (struct sockaddr_in);
getsockname(p_sock, (struct sockaddr *) &reg_addr, &alen);
```

The port number is stored in *reg_addr.sin_port*; the IP address in reg_addr.sin_addr.s_addr. (You probably find the IP address stored in *reg_addr* is just 0.0.0.0, which means "this machine", so this information is not very useful.)

Select System Call

From the discussion so far, it is obvious that the peer is attached to multiple sockets. As a content server, the peer that provides download for multiple contents needs to listen to multiple TCP sockets. In addition, it needs to listen to the stdin in order to interact with the user. The *read* and *accept* system calls can only listen to one socket a time, thus, themselves alone are not enough to support the mechanism required by the peer.

To listen to multiple sockets, instead, the peer uses *select*. The system call *select* blocks the return, similar to *read* and *accept*, until one or more events (usually associated with pending TCP connection or data arrival) happen at the sockets it listens to. After the return, the peer can find out which socket(s) should be serviced and call *read* and/or *accept* to handle the event(s). The code below gives an example on how to setup the arguments before calling *select*.

```
fd_set rfds, afds;
sock = socket(AF_INET, SOCK_STREAM, 0);
FD_ZERO(&afds);
FD_SET(sock, &afds);          /* Listening on a TCP socket  */
FD_SET(0, &afds);             /* Listening on stdin   */
memcpy(&rfds, &afds, sizeof(rfds));
select(FD_SETSIZE, &rfds, NULL, NULL, NULL)
```

The type fd_set is a bit array. On line 4, if the argument sock, which is the descriptor of a TCP socket, has the value of 3, then FD_SET sets the 3rd bit of afds to 1; similarly, FD_SET sets the 0th bit of afds to 1 on line 5. When *select* is called on line 7, the return will be blocked until a TCP connection is pending at the TCP socket and/or data arrives at stdin. Upon the return of *select*, rfds indicates which socket (sockets) is (are) required for service. The following code shows a typical way to handle the return of *select*.

```
if (FD_ISSET(0, &rfds)) {
        n = read(0, buf, BUFSIZE);
            .
            .
            .
}
if(FD_ISSET(sock, &rfds)){
        new_sd = accept(sock, (struct sockaddr *)&client, &alen);
            .
            .
            .
} …
```

If the first bit (0^{th} position) of rfds is set, it implies that data has arrived at stdin; similarly, if the 4^{th} bit (3^{rd} position) of rfds is set, it implies there is a pending TCP connection. To handle the first case, a *read* is called to receive the data. For the second case, *accept* is called to complete the TCP connection.

Maximum Group size

This is a group project. Group size is 2.

Demonstration Requirements

1. The application you will develop should demonstrate the following functions
    a. A peer (Peer 1) registers a content to the index server.
    b. Another peer (Peer 2) requests a search of a content from the index server and receives the address of a content server.
    c. Subsequently, Peer 2 uses the address information to download the requested content from the content server.
    d. After the download, Peer 2 should automatically register as a content server of the downloaded content. The third peer requests the same content should be able to download the content from Peer 2.
    e. A peer can request from the index server the list of registered content.
    f. When a user of a peer chooses to quit, all the content registered by the peer should be deregistered.

2. Go over your programs, locate and explain the parts of the code that deal with the following implementations:
    a. Content registration.
    b. Content download.
    c. Content deregistration

Report

The deadline for report submission is Friday of the 13^{th} week of 2024 Fall term. Submit the report to D2L course site. The formal report format can be found in the D2L site.