Washington University
Department of Computer Science

CS422: Operating Systems

Programming Project
Due Tuesday, November 18^{th}, 1997

1 Overview

In this assignment, you will implement an IP multicast-based Naming Service locator. Client applications can use this locator to determine where a Naming Service is located in a LAN, *without* having to know much about the LAN host topology. In this assignment, the client will use IP multicasting[1] to send queries to multiple Naming Service locators simultaneously. The system will behave as follows:

- The client *multicasts* a message to the multicast IP address and port number using `sendto`. The content of the message doesn't matter since the message will just serve to identify the client to the name servers.
- The Name Servers read the message with `recvfrom`. Now that they have address of the client, they each reply with a *unicast* message. The content of this message must include the port number where the Name Server is listening for TCP connections.
- The client reads the first message it receives with `recvfrom`. Any other messages from other Name Servers are just ignored. The client now has the address of a Name Server and the port number it is listening on for TCP connections.
- The client makes a TCP connection with that Name Server. It then sends the hostname across the connection. The nameserver resolves the name into an address `gethostbyname` and then sends it back across the connection.

2 Design and Implementation Issues

2.1 Client Functionality

For the purposes of the assignment, the client driver program can be straightforward. You should implement a function called:

```
int resolve_name_service (u_short port,
                          sockaddr_in &naming_service_addr,
                          Time_Value &tv);
```

This function will multicast to the designated `port` in an attempt to locate a Naming Service. If it gets a response in `tv` amount of time, then 0 will be returned and `naming_service_addr` will contain the IP address and port number where the Naming Service resides. Otherwise, -1 will be returned and `errno` will report the problem.

If the Naming Service is located successfully, your client should connect to it using the TCP and perform `read` operations until the Naming Server closes down the connection.[2]

Naturally, the tricky part will be implementing the `resolve_name_service` function. I suggest that you use C++ wrappers, as described below.

[1] Actually, we'll be using the UDP socket interface to IP multicasting.
[2] If you want extra credit, you can implement a more complete "Naming Service" than this simple protocol.

2.2 Encapsulating Multicast with C++ Wrappers

The Name Service locator will be designed using IP multicasting via UDP sockets. Note that you must perform marshaling and demarshaling of any binary data you pass between hosts in your solution. In particular, your solution should work correctly when run on host machines with different byte orders.

The underlying C interface to IP multicasting is quite low-level and difficult to understand and use correctly. Therefore, you'll need to write a C++ wrapper that encapsulates this interface. The following C++ class may help you get started. Note that this uses some of the features in ACE. Feel free to use these libraries in your solution.

```
typedef int HANDLE;
// Make UNIX look like Win32...

class SOCK_Mcast
{
  HANDLE open (int protocol_family = PF_INET, int protocol = 0);
  // Allocate the underlying socket descriptor.

  int subscribe (const INET_Addr &mcast_addr,
                 int protocol_family = PF_INET,
                 int protocol = 0,
                 char* interface = "le0");
  // Subscribe to a multicast group.  The interface is hardware
  // specific.  Use netstat -i to find whether your interface is
  // le0 or other.

  int unsubscribe (void);
  // Unsubscribe from a multicast group.

  ssize_t send (const void *buf, size_t n, int flags = 0) const;
  // Send BUF to default multicast group (subscribe must have been called).

  ssize_t send (const void *buf, size_t n,
                const INET_Addr &addr, int flags = 0) const;
  // Send BUF to multicast group ADDR (subscribe need not have been called).

  ssize_t recv (void *buf, size_t n, INET_Addr &addr, int flags = 0) const;
  // Receive BUF from a multicast group (subscribe must have been called).

  int set_option (int option, char optval);
  // Set an ip option that takes a char as input.
  // e.g. IP_MULTICAST_LOOP.

private:
  struct ip_mreq multicast_address_;
  // Underlying IP multicast address.

  INET_Addr send_address_;
  // Default address (if subscribed to a multicast group).

  HANDLE socket_;
  // The underlying socket HANDLE.
};
```

A client will bind to the server via the `subscribe` method on `SOCK_Mcast` class. You'll need to explicitly pass in the multicast group address. Once the client application has subscribed it can multicast Naming Service queries by calling the `send` method on the `SOCK_Mcast` object.

2.3 Server Functionality

Your Naming Service server will need to handle locator requests that arrive via UDP multicast. It will also need to handle TCP connections, as well. Therefore, I recommend that you use the `Reactor` from your previous assignment in order to wait for both the UDP multicast requests and the TCP connections in a single thread of control.

Once the client and server components are written, compiled, and linked together you'll need to start up the server program manually on all the hosts you select for your replicated Naming Service.

3 Concluding Remarks

In order to understand how to implement these methods you'll need to learn more about IP multicasting. You can find out more about this at `www.cs.wustl.edu/~schmidt/cs422/mcast.ps.gz`.