ECE 4220

Zachary Rump

2017-04-10

Lab 5 Report

Objectives

The primary objective for Lab 5 was to learn about inter-process communication and synchronization in a distributed environment.

Lab Description

The specific objectives of the Lab:

1. Learn to use sockets for communication.
2. Implement a Master-Slave setup.

Implementation

The implementation for this lab consisted of a single server program. The program initializes itself as a 'slave', then opens a network (UDP) socket (port specified on command line) and receives messages, e.g. WHOIS, VOTE. If the program receives a WHOIS request and it is the master it responds on the broadcast address. If it receives a VOTE request it will generate a random number between 1-10, use it as the last octet, and send it's vote on the broadcast address. Additionally, it will receive the votes from all the other servers and tally them to determine the new master.

Flowcharts

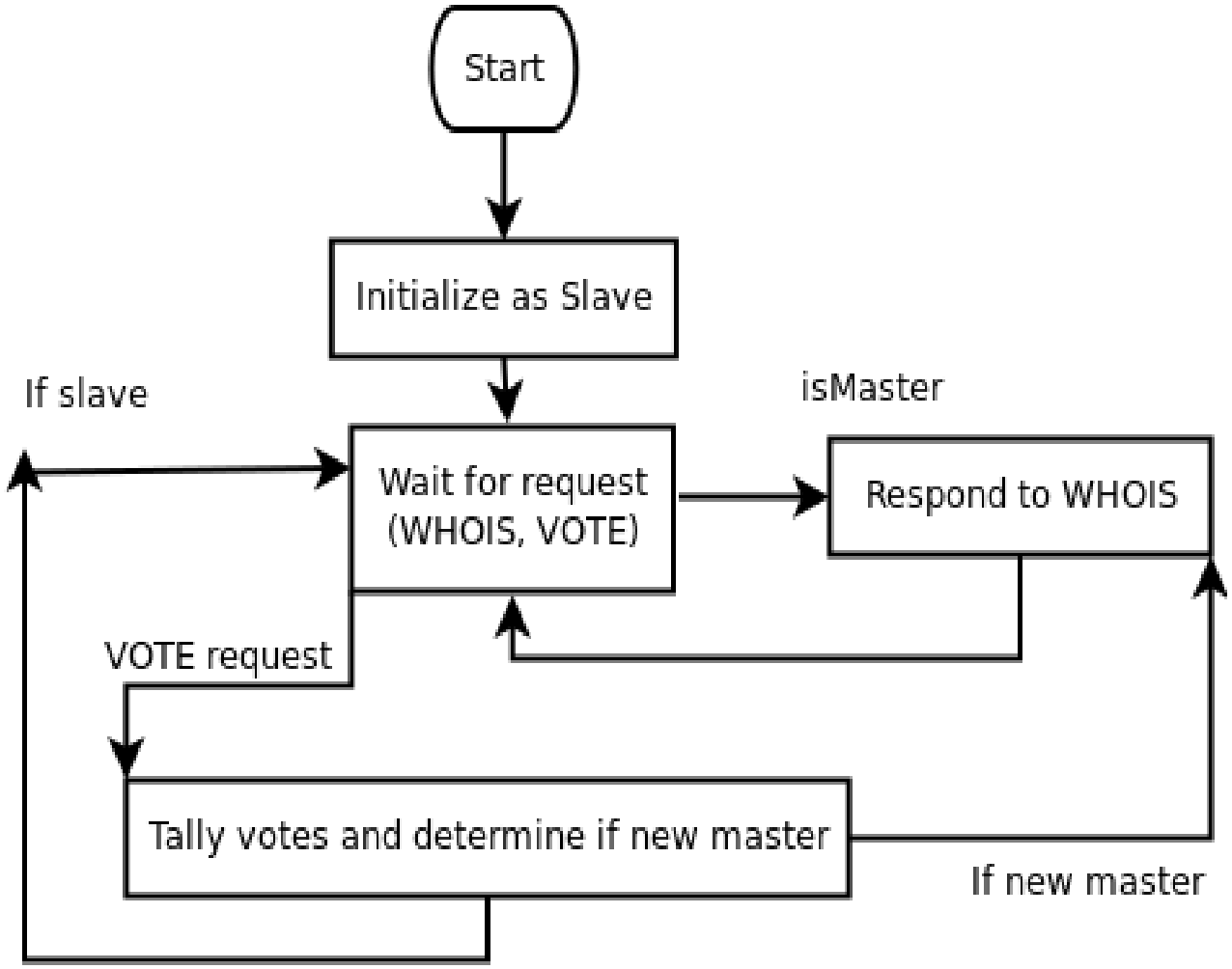

Figure 1. Flowchart for Lab 4.

Experiments and Results

Didn't actually get the demo working in class, but I did manage to test parts of the functionality with other servers. Mostly tested it by using the loopback address (127.0.0.1) as the broadcast and running one of the example programs to send messages. Using this method I was at least able to get the server to respond to a WHOIS, and VOTE.

Discussion

The most painful part of this lab was dealing with the conversion between the sockaddr integer value and the string ip address. I.e. string parsing. Also the logic to keep track of the votes. Main thing I learned was just about the socket functions and data structs in general. Not very fun to use initially, but now that I have a cursory familiarity it should be easier to learn more about them.

Post Lab Questions

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <arpa/inet.h>
#include <netdb.h> // gethostbyname
#include <string.h>
#include <strings.h> // bcopy
#include <errno.h>
#include <time.h>

#define MSG_SIZE 40
#define IP_MAX 10
#define IP_MIN 1
#define VOTE_TIMEOUT_SEC 5
//#define BROADCAST_ADDR 10.3.52.255
#define BROADCAST_ADDR 127.0.0.1

int VOTES[IP_MAX + 1] = { 0 };
int last_octet; // To store the identifier for this instance
// Tally votes. Return 1 if this program is the new master, 0 otherwise;
int newMaster(void) {
  int i;
  int max_votes = -1;
  int new_master = -1;
  for(i=1; i<IP_MAX+1; i++)
    {
      printf("[%d]: %d\n", i, VOTES[i]);
      if(VOTES[i] > max_votes)
{
  new_master = i;
  max_votes = VOTES[i];
}
      else if(max_votes == VOTES[i])
{
  if(i > new_master)
    {
      new_master = i;
    }
}
    }
  if(new_master == last_octet)
    {
      return 1;
    }
  return 0;
}


int main(int argc, char **argv) {
  int sock, n;
  struct sockaddr_in server;
  struct sockaddr_in from;
  struct hostent *hp = NULL;
  unsigned long broadcast_addr = inet_addr("BROADCAST_ADDR");
  unsigned long server_addr;
  int length = sizeof(server);
```

```
socklen_t fromlen = sizeof(struct sockaddr_in);
unsigned int opt_val_bool = 1; // To pass to setsockopt()
unsigned int isMaster = 0;
char buf[MSG_SIZE]; // for receive messages
char str_buf[MSG_SIZE]; // for sprintf
char tok_buf[MSG_SIZE]; // for strtok
char ip_str[INET_ADDRSTRLEN];

struct timeval read_timeout = { .tv_sec = VOTE_TIMEOUT_SEC, .tv_usec = 0};


// Check input args for port #
if(argc < 3) {
  fprintf(stderr, "Usage is %s <hostname> <port>\n", argv[0]);
  return EXIT_FAILURE;
}

// Random numbers seed
srand(time(NULL));
// Set server properties
server.sin_family = AF_INET;
// Get port # from input
server.sin_port = htons(atoi(argv[2]));
// Get ip address by hostname
hp = gethostbyname(argv[1]);
if(0 == hp) {
  // Error
  fprintf(stderr, "Error: unknown host\n");
  return EXIT_FAILURE;
}
// IP Address
bcopy((char *)hp->h_addr, (char *)&server.sin_addr, hp->h_length);
length = sizeof(server);

// Get ip address as string
struct sockaddr_in *ipv4Addr = (struct sockaddr_in*)&server;
struct in_addr ipAddr = ipv4Addr->sin_addr;
inet_ntop(AF_INET, &ipAddr, ip_str, INET_ADDRSTRLEN);
strncpy(&tok_buf, &ip_str, strlen(ip_str));
char *my_id = strtok(&ip_str, ".");
int i;
for(i=0; i<3; i++)
  {
    my_id = strtok(NULL, ".");
  }
last_octet = atoi(my_id);
printf("My identifier: %d\n", last_octet);

// Connectionless ipv4 socket (domain, type, protocol=0)
sock = socket(AF_INET, SOCK_DGRAM, 0);
if(sock < 0)
  {
    // Error creating socket
```

```
    fprintf(stderr, "Error in socket(): %s\n", strerror(errno));
    return EXIT_FAILURE;
  }

 // bind socket to address of host + port # (command line)
 if(bind(sock, (struct sockaddr *)&server, length) < 0 )
  {
    // Error binding
    fprintf(stderr, "Error in bind(): %s\n", strerror(errno));
    return EXIT_FAILURE;
  }

 // set socket perms to allow broadcast
 if(setsockopt(sock, SOL_SOCKET, SO_BROADCAST, (void *)&opt_val_bool, sizeof(opt_val_bool)) < 0 )
  {
    // Error setting options
    fprintf(stderr, "Error in setsockopt(): %s\n", strerror(errno));
    return EXIT_FAILURE;
  }

 while(1)
  {
    // Wait for message from client.
     n = recvfrom(sock, buf, MSG_SIZE, 0, (struct sockaddr *)&from, &fromlen);
     if(n < 0) {
  perror("recvfrom");
}
     // DEBUG
     printf("Received datagram. It says: %s, %d\n", buf, n);
     if(0 == strcmp(buf, "WHOIS\n"))
{
  bzero(buf, MSG_SIZE);
  printf("WHOIS ack.\n");
  if(isMaster)
    {
      server_addr = server.sin_addr.s_addr;
      server.sin_addr.s_addr = broadcast_addr;
      sprintf(&str_buf, "Zach on board %s is the master", ip_str);
      // Write message to broadcast address.
      // TODO change to broadcast
      n = sendto(sock, &str_buf, strlen(str_buf), 0, (struct sockaddr *)&server, fromlen);
      if(n < 0) {
perror("sendto");
      }
      server.sin_addr.s_addr = server_addr;
    }
  else // Wait for another master to reply or a VOTE
    {
      n = recvfrom(sock, buf, MSG_SIZE, 0, (struct sockaddr *)&from, &fromlen);
      if(n < 0) {
perror("recvfrom");
      }
      printf("Received datagram. It says: %s %d", buf, n);
```

```
      if(0 == strcmp(buf, "VOTE\n"))
{
  // Clear votes
  memset(VOTES, 0, (IP_MAX + 1)*sizeof(int));
  bzero(tok_buf, MSG_SIZE);
  printf("Voting initiated.\n");
  // No master reply =>
  // Send and receive votes
  // Get random number between 1-10 for the last octet
  // Zeroes represent the min number
  int r = rand() % (IP_MAX + 1 - IP_MIN) + IP_MIN;
  //VOTES[r]++;
  sprintf(str_buf, "# 10.3.52.%d", r);
  // Send vote to broadcast
  server.sin_addr.s_addr = broadcast_addr;
  n = sendto(sock, &str_buf, MSG_SIZE, 0, (struct sockaddr *)&server, fromlen);
  if(n < 0) {
    perror("sendto");
  }

  // Set timeout and read votes
  read_timeout.tv_sec = VOTE_TIMEOUT_SEC;
  if(setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &read_timeout, sizeof(read_timeout)) < 0) {
    perror("Error setting socket timeout");
  }
  do {
    n = recvfrom(sock, buf, MSG_SIZE, 0, (struct sockaddr *)&from, &fromlen);
    // Check buf for leading '#' and parse ip
    if('#' == buf[0]) {
      printf("Vote received\n%s", buf);
      strncpy(&tok_buf, &buf, strlen(buf));
      char *vote_num = strtok(&tok_buf, ".");
      int i;
      for(i=0; i<3; i++)
{
  vote_num = strtok(NULL, ".");
}
      int vote_index = atoi(vote_num);
      VOTES[vote_index]++;
      printf("Vote: %d\n", vote_index);
    }
    bzero(buf, MSG_SIZE);
    bzero(tok_buf, MSG_SIZE);

    // Add vote
  } while(n > 0);

  // Disable timeout
  read_timeout.tv_sec = 0;
   if(setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &read_timeout, sizeof(struct timeval)) < 0) {
    perror("Error un-setting socket timeout");
  }
   // Change address back from broadcast
```

```
   server.sin_addr.s_addr = server_addr;
   // Tally votes
   if(newMaster())
     {
       printf("I am the new master\n");
       isMaster = 1;
     }
 } // Voting not initiated
else
  {
    // Master replied
  }
     }
 } // endifwhois
    } // end while

return EXIT_SUCCESS;

}
```