```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>
#define MSG_CNFM 0
#define TRUE 1
#define FALSE 0
#define ML 1024
#define MPROC 32

/*
Function to create a new connection to port 'connect_to'
1. Creates the socket.
2. Binds to port.
3. Returns socket id
*/
int connect_to_port(int connect_to)
{
		int sock_id;
		int opt = 1;
		struct sockaddr_in server;
		if ((sock_id = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
		{
				perror("unable to create a socket");
				exit(EXIT_FAILURE);
		}
		setsockopt(sock_id, SOL_SOCKET, SO_REUSEADDR, (const void*)&opt, sizeof(int));
		memset(&server, 0, sizeof(server));
		server.sin_family = AF_INET;
		server.sin_addr.s_addr = INADDR_ANY;
		server.sin_port = htons(connect_to);
		if (bind(sock_id, (const struct sockaddr *)&server,
		sizeof(server)) < 0)
		{
				perror("unable to bind to port");
				exit(EXIT_FAILURE);
		}
		return sock_id;
}
/*
sends a message to port id to
*/
void send_to_id(int to, int id, char message[ML])
{
		struct sockaddr_in cl;
		memset(&cl, 0, sizeof(cl));
		cl.sin_family = AF_INET;
		cl.sin_addr.s_addr = INADDR_ANY;
		cl.sin_port = htons(to);
		sendto(id, (const char *)message, strlen(message), MSG_CNFM, (const struct sockaddr *)&cl, sizeof(cl));
}
/*
starts the election, returns 1 if it wins the round
*/
int election(int id, int *procs, int num_procs, int self)
{
		int itr;
		char message[ML];
		strcpy(message, "ELECTION");
		int is_new_coord = 1; // assume you are the winner until you close
		for (itr = 0; itr < num_procs; itr += 1)
		{
				if (procs[itr] > self)
				{
						printf("sending election to: %d\n",
						procs[itr]);
						send_to_id(procs[itr], id, message);
						is_new_coord = 0; // a proc with id > self exists thus cannot be coord
```

```
                        }
                }
                return is_new_coord;
}
/*
announces completion by sending coord messages
*/
void announce_completion(int id, int *procs, int num_procs, int self)
{
                int itr;
                char message[ML];
                strcpy(message, "COORDINATOR");
                for (itr = 0; itr < num_procs; itr += 1)
                if (procs[itr] != self)
                send_to_id(procs[itr], id, message);
}
int main(int argc, char* argv[])
{
                // 0. Initialize variables

                int self = atoi(argv[1]);
                int n_proc = atoi(argv[2]);
                int procs[MPROC];
                int sock_id, bully_id;
                int itr, len, n, start_at;
                char buff[ML], message[ML];
                struct sockaddr_in from;
                for (itr = 0; itr < n_proc; itr += 1)
                        procs[itr] = atoi(argv[3 + itr]);
                start_at = atoi(argv[3 + n_proc]) == 1? TRUE : FALSE;
                // 1. Create socket
                printf("creating a node at %d %d \n", self, start_at);
                sock_id = connect_to_port(self);
                // getchar();
                // 2. check is process is initiator
                if (start_at == TRUE)
                {
                        election(sock_id, procs, n_proc, self);
                }
                // 3. if not the initiator wait for someone else
                while(TRUE)
                {
                        memset(&from, 0, sizeof(from));
                        n = recvfrom(sock_id, (char *)buff, ML, MSG_WAITALL,(struct sockaddr *)&from, &len);
                        buff[n] = '\0';
                        printf("Recieved messed: %s\n", buff);
                        if (!strcmp(buff, "ELECTION"))
                        {
                                strcpy(message, "E-ACK"); // send election ack
                                sendto(sock_id,(const char *)message,strlen(message),MSG_CNFM,(const struct sockaddr
*)&from,sizeof(from));
                                if (election(sock_id, procs, n_proc, self))
                                {
                                        announce_completion(sock_id, procs,
                                        n_proc, self);
                                        printf("ANNOUNCING SELF AS NEW COORD\n");
                                }
                        }
                        else if (!strcmp(buff, "E-ACK"))
                                continue; // nothing do, your job is done
                        else if (!strcmp(buff, "COORDINATOR"))
                                bully_id = from.sin_port;


                }
}
```