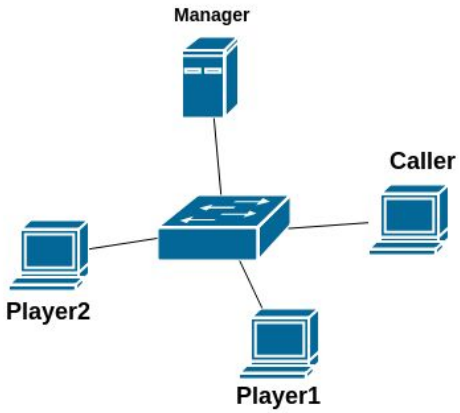
Bingo Application

CSE434 Spring 2019



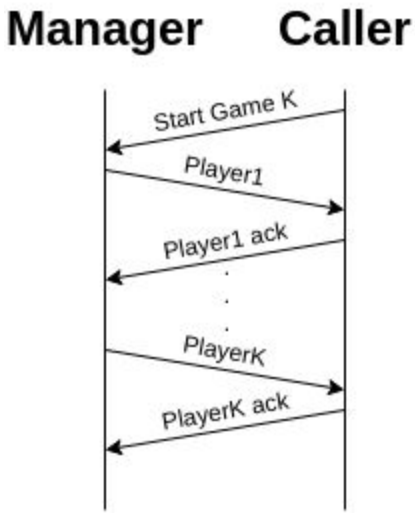
Authors:
Devon Bautista
Cesar D. Tamayo

1. Current Status:

The current version of the code is able to register hosts and start games with multiple players. The manager is able to track games and registered players. The bingo hosts have a thread that automatically listens for games after registration, and starts a new thread when asked to join a game. The caller is able to connect to players and call numbers until there is a winner.

2. Protocol Design Decisions:

a. Caller-Manager:



The first program that needs to run is the manager since it's the one registering players when they execute the bingo program on the other hosts. When the user runs the manager executable in the hosts, it starts listening by default on the port entered as a parameter. After this, users can then run bingo on the other hosts specifying the manager ip and port.

When users run bingo they are asked to enter the name, IP address and default port that will be used to register in the manager. This way, the manager can send their info to callers requesting players. After a successful registration, the user can then choose between options in a Menu. One of the options is the "start game K" command which requests K players and a GameID from the manager to start a game. The manager first checks if it has enough registered players to provide the caller and if it doesn't then it sends a message back to the caller. If it does, then it sends each player data in a sequence of K messages (shown in Figure1). Each message contains the GameID, the player IP address and the default port number. At that point, the caller is ready to start the game with the received players.

Message Format:

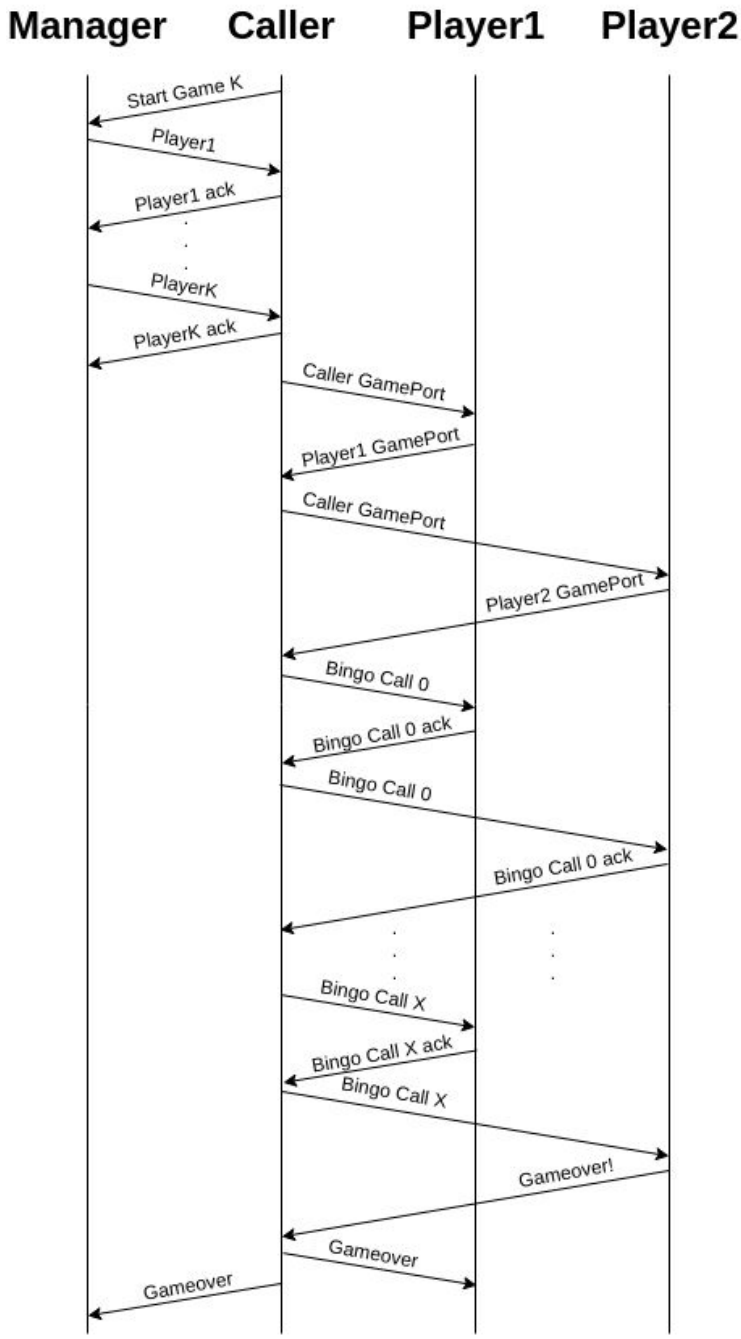
For every message we send in our code, we used the following struct, which contains a command code (*command*) and then a union that allows to extract command-specific data depending on the command code. For the case of the caller-manager communication, we used *clr_cmd_startgame_t* and *mgr_rsp_startgame_t* which correspond to the caller command and the manager response. The caller message contains the value of K, its IP address and Port number. This information is used by the manager to send the requested players and also to link the caller details to the game data stored while the game is ongoing. The manager response, on the other hand contains the gameID, the player name, IP, default port and the number of players left.

```
typedef struct {
   int command;
   union {
      /* Register command/response */
      mgr_cmd_register_t mgr_cmd_register;    /**< Register player command data */
      mgr_rsp_register_t mgr_rsp_register;    /**< Register player response data */
```

```
        /* Deregister command/response */
        mgr_cmd_deregister_t mgr_cmd_deregister;    /**< Deregister player command data */
        mgr_rsp_deregister_t mgr_rsp_deregister;    /**< Deregister player response data */

        /* Calling command/response (caller-player) */
        clr_cmd_bingocall_t clr_cmd_bingocals;  /**< Bingo Call command data */
        ply_rsp_bingocall_t ply_rsp_bingocall;  /**< Player response to Bingo Call data */

        /* Start game command/response */
        clr_cmd_startgame_t clr_cmd_startgame;  /**< Caller Start Command data */
        mgr_rsp_startgame_t mgr_rsp_startgame;  /**< Manager Response to startGame command */
        port_handshake_t port_handshake;        /**< Port negotiation information */

        /* Query players response */
        mgr_rsp_queryplayers_t mgr_rsp_queryplayers;    /**< Manager response to Query Players
command */
    };
} msg_t;

typedef struct clr_cmd_startgame_t {
    int k;
    char callerIP[BUFMAX];
    int callerPort;
} clr_cmd_startgame_t;

typedef struct {
    int gameID;                 /**< New game's ID (if started) */
    int playersLeft;            /**< Sentinel for caller to know how many responses left */
    char playerIP[BUFMAX];      /**< Player's default IP address */
    char playerName[BUFMAX];    /**< Player's name */
    unsigned int playerPort;    /**< Player's default port */
} mgr_rsp_startgame_t;
```

b. Caller-Players:

After the caller had all the data for the players that will participate, it starts negotiating port numbers for the new threads that will handle the game. For each of the players, the caller sends its new port for that particular game ("Caller GamePort" in Figure1) and the player responds with its new port for that particular game ("PlayerX GamePort" in Figure1). After the caller has received the new port numbers from all the players, then it will start calling numbers to all of the players and waiting for acknowledgement from them. Every time a player receives a number, it checks its board and it checks if it won. If it did, it sends a GAMEOVER signal to the caller. Then the caller notifies the other players in the game and the game is over. The caller also notifies the manager that the game is over.



Message Format:

Using the same generic struct, we expanded some structs specifically for the gameplay and the port negotiations. The struct used for port negotiation *port_handshake_t* contains only one field called gamePort that allows for called and players to exchange selected ports. For the gameplay, we have the clr_cmd_bingocall_t that contains the number called by the player and the response from the player just contains a return code that tells the caller if the value was received.

```
typedef struct clr_cmd_bingocall_t {
   int bingoNumber;    /**< The number being called (e.g. 'B7') */
} clr_cmd_bingocall_t;

typedef struct ply_rsp_bingocall_t {
   int ret_code;       /**< Return code from caller for bingo call */
```

```
} ply_rsp_bingocall_t;

typedef struct port_handshake_t {
   unsigned int gamePort;      /**< New pprt to negotiate */
} port_handshake_t;
```

c. Manager Pseudocode:

```
Check for errors in the arguments.
Try to convert port.
Instantiate Manager object.
Instantiate ServerSocket and start it.
while (true):
        If something is received in the server socket:
                Check command field in the message.
                        If command is REGISTER:
                                Save player data in list of registered players.
                                Build response message and send it to caller.
                        If command is DEREGISTER: deregister player.
                                Try to deregister the player.
                                Add SUCCESS command code if succeeded, FAILURE otherwise.
                                Send message back to caller.
                        If command is START_GAME:
                                Check if there are enough players registered.
                                If there are:
                                        Save caller data.
                                        Send K players.
                                        Save game data in memory.
                                If not, send FAILURE to caller.
                        If command is QUERY_PLAYERS:
                                Send all registered players.
        If not, restart the socket.
```

d. Bingo Pseudocode:

```
Check for errors in the arguments.
Try to convert port.
Instantiate Bingo object.
Get registration data from user: Name, IP and Port.
Register to Manager.
Create a thread that listens for callers starting games.
Print Menu with options.
```

If user selects EXIT, close program.
If user selects START GAME:
    Check if player is registered and if it's not: throw error.
    Get value of K from user.
    Build message body and send it.
    Receive players from manager.
    For each player:
        Negotiate game port number.
    For each player:
        Call numbers until gameover signal is received.
If user selects DEREGISTER:
    Check if player is registered and if it's not: throw error.
    Create clientSocket to connect to manager.
    Populate message and send DEREGISTER command to manager.
If user selects REGISTER:
    Check if player is registered and if it's not: throw error.
    Create clientSocket to connect to manager.
    Populate message and send REGISTER command to manager.
If user selects QUERY_PLAYERS:
    Create clientSocket to connect to manager.
    Populate message body and send QUERY_PLAYERS command to manager.

3. How to Use:

1. Download repository.
2. Navigate to repository in the terminal.
3. Run make.
4. If it's a manager host, run:
   ./manager <portNumber>
   portNumber represents the port number the manager will be listening to.
5. If it's a bingo host, run:
   ./bingo/bingo <managerIP> <managerPort>
6. Follow instructions on the Menu for Bingo hosts.

4. Repository: https://github.com/synackd/bingo.git