ARIZONA STATE UNIVERSITY

CSE 434, SLN 11285 — Computer Networks — Spring 2019

Instructor: Dr. Violet R. Syrotiuk

Socket Programming Project

Available Sunday 01/27/2019; milestone due 02/10/2019, full project due Sunday 02/24/2019

1 Introduction

The objective of this project is to develop a Bingo game application that utilizes a client-server for game management, and a multi-threaded peer-to-peer (P2P) approach for each ongoing game.

The logical architecture of the application is illustrated in Figure 1. The *manager* is a centralized server used for managing players and games. Before it does anything else, each peer must register with the manager.

When a peer wants to start a Bingo game, it requests the information of $k \geq 2$ other peers from the manager to be players in the game. The manager selects k players, returns them to the peer, and stores all participant information for the game. The initiating peer contacts each peer player to join the game, exchanging new port information. Once the game is confirmed, a new *caller* thread is spawned at the peer initiating the game to communicate with new peer *player* threads via a new socket. Once each *player* thread initializes a Bingo card, the caller runs in rounds, making a distinct call until at least one player wins; see §1.1. Once a winner is declared, all player threads terminate. Then, the caller thread terminates, and this event is then used to inform the manager that the game is over resulting in an update of ongoing games.

Each Bingo peer is a client of the manager server, and also combines P2P functionality. A peer acts as a client in a game when it is a player in a game of Bingo. A separate *player* thread is spawned for each game it plays. A peer acts as a server in a game when it is a caller in a game of Bingo. A separate *caller* thread is spawned for each game it calls.

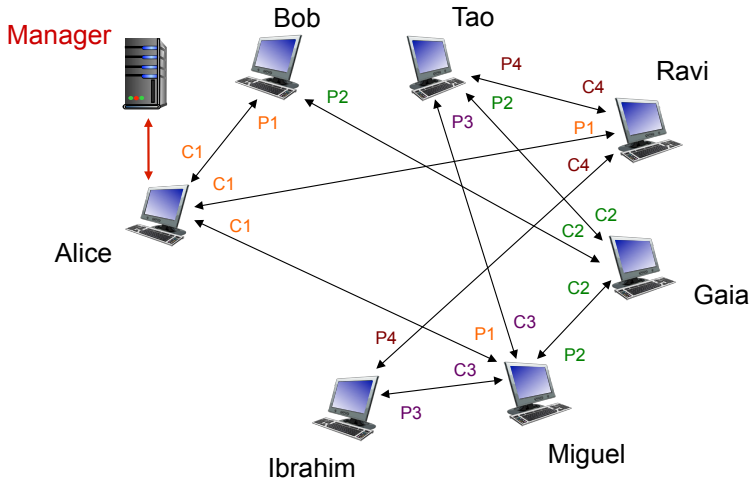

Figure 1: Logical architecture of the Bingo game application. For simplification, threads are not illustrated.

Figure 1 shows a scenario in which seven peers have registered with the manager and the last communication was with Alice; there are four Bingo games ongoing. One has Alice as the caller with Bob, Ravi, and Miguel as players. The second has Gaia as the caller also with three players: Tao, Bob, and Miguel. The third game has Miguel as caller with Tao and Ibrahim as players. Finally, in the fourth game, Ravi is

the caller also with Tao and Ibrahim as players. Alphanumeric labels on each edge indicate communication between a caller (C) and players (P) in the same (numbered) game.

The separate threads are not illustrated to simplify the presentation in the figure; the participants of each game are to run as separate threads and communicate with each other on a separate socket. In the scenario, Alice, Gaia, Miguel, and Ravi are each running one caller thread, while the others have no caller threads. In terms of player threads, Alice and Gaia have none, Ravi runs one, Bob, Miguel, and Ibrahim each run two, and Tao runs three.

1.1 The Game of Bingo

The most common Bingo cards contain 25 squares arranged in a 5×5 matrix. The five columns of the card are labelled 'B', 'I', 'N', 'G', and 'O' from left to right. A typical Bingo game utilizes the numbers 1 through 75. However, the range of numbers that can appear on the card is restricted by column, with the 'B' column only containing numbers between 1 and 15 inclusive, the 'I' column containing only 16 through 30, 'N' containing 31 through 45, 'G' containing 46 through 60, and 'O' containing 61 through 75; *duplicate numbers in a column are not allowed*. The center space is usually marked "Free," and is considered automatically matched. Figure 2 shows one of the approximately 5.52×10^{26} Bingo cards possible.

B	I	N	G	O
14	20	32	52	71
10	27	42	55	64
7	23	FREE	58	69
11	28	34	56	72
15	25	33	53	66

Figure 2: An example Bingo card.

A game of Bingo has one caller, and (to make it interesting) two or more players. The game runs in rounds until at least one player declares itself a winner. In each round, the caller makes a call consisting of a column label, *i.e.*, one of 'B', 'I', 'N', 'G', or 'O', and a number restricted by the column label; *the call in each round must be unique*. Players look for a match with the call on their card. For example, if in a round the caller calls 'G54' then the call does not match any row in column 'G' on the card in Figure 2, whereas a call of 'B7' matches row three of column 'B.' A player wins if, after a sequence of calls, all squares in a row, column, or diagonal of the player's Bingo card are matched.

2 Requirements of Socket Programming Project

You are to write two programs for this socket programming project:

1. One program, `manager`, implements the manager server. The `manager` must be able to process all commands listed in §2.1 issued from a client via a text-based user interface.
2. The second program, `bingo`:
   (a) interacts with the `manager` as a client, and
   (b) implements the P2P Bingo game; see §2.2 for details.

You should determine the command line parameters that each of the `manager` and `bingo` program requires to support the needed communication. For information about port numbers see §3.

2.1 Client-Server: The Manager

The program `manager` is an always-on server located at a fixed IP address and port. It maintains a "database" of players, and the participants involved in all ongoing Bingo games.

The commands the manager must support follow; *i.e.*, these are the commands that can be issued by a client. The manager always sends a response to the client issuing the command. All communication between the manager (server) and its clients must be through a socket.

1. `register name IP-address port`, to register a triple associated with the client at the manager. This command returns `SUCCESS` if the `name` is unique among all peers registered and returns `FAILURE` otherwise. On success, the manager stores the `name`, `IP-address`, and `port` number of the client. This "default" IP address and port will be used in game set-up only; another socket on a different set of ports will be used for the game itself. Once registered, the client may become a caller or a player in a Bingo game; see §2.2.
2. `query players`, to query the players currently registered with the manager. This command returns a return code equal to the number of registered players, and a list of triples associated with each player. If there are no players registered, the return code is set to zero and the list is empty.
3. `start game k`, to start a Bingo game with `k` players with the issuing client/peer as caller of the game. If there are at least $k+1$ players registered with the manager then a new Bingo game may be initiated. In this case, the command returns a return code consisting of an identifier for the game. The manager selects k players from those registered at random, excluding the caller, and also returns a list of triples associated with each player selected to the client/peer. The game information, *i.e.*, the game identifier, caller, and players, is also stored at the manager. If there are insufficient peers to start the game then a return code of `FAILURE` is returned and the list is empty.
4. `query games`, to query the Bingo games currently ongoing. This command returns a return code equal to the number of ongoing games, and a list that includes the game information for each game, *i.e.*, the game identifier, caller, and players in each game. If there are no games ongoing, the return code is set to zero and the list is empty.
5. `end game-identifier`, to indicate that the game with `game-identifier` and caller corresponding to the client/peer that initiated the game has completed. This command returns `SUCCESS` if the the game identifier and caller match that stored by the manager and returns `FAILURE` otherwise. On success, the manager deletes the game information from the list of ongoing games.
6. `degregister name`, to deregister as a peer and exit the application. This command returns `SUCCESS` if and only if the peer with the given `name` is not involved in any ongoing Bingo game, as a caller or as a player. In that case, the triple associated with the peer is deleted from the manager's records and the peer can safely exit the Bingo application. If the peer is involved in an ongoing game, this command returns `FAILURE`.

2.2 Peer-to-Peer (P2P): The Bingo Game

The `bingo` process acts as a client when interacting with the `manager`. You must provide a text-based user interface for this communication. Through this interface, commands of the format listed in §2.1 are sent to the manager via a socket; responses returned from the manager must be displayed. The `start` and `end` commands involve additional processing, as explained next.

2.2.1 The `start` Command

A successful `start game k` starts a new Bingo game between the peer issuing the command – the *caller* – and `k` players returned by the `manager`. The caller first obtains a *new* port number to use for communication with each player during the Bingo game. This exchange occurs on peers' default IP address and port. After obtaining a new port from each player, the peer spawns a new caller thread. Each player in the game spawns a new thread after providing a new port number for the game, initializes a Bingo card for the game, and waits listening for calls on the new port number. Subsequently, communication for the game is through new

sockets using the default IP address, but new port numbers. (The threads do not change hosts, only port numbers.) The caller runs the game in rounds, sending the call to each player and listening for a winner. The caller should output the call for the `game-identifier`, while each player should output its Bingo card indicating matches, and its response to the caller. If the game has a winner, the caller announces the winner to all players, and each player terminates. Then the caller itself terminates.

More specifically, recall that the response to a successful `start game k` command is a return code that is a game identifier, and a list of `k` players including the player's name, default IP address, and default port number. That is, included as part of the response from the manager is a list of the form:

$$\begin{array}{lll} \text{name}_1 & \text{IP-addr}_1 & \text{p}_1 \\ \text{name}_2 & \text{IP-addr}_2 & \text{p}_2 \\ \vdots & \vdots & \vdots \\ \text{name}_k & \text{IP-addr}_k & \text{p}_k \end{array}$$

To make the exchange prior to the game starting clear, suppose that the `bingo` process issuing the `start game k` command is the peer with name_0 running on IP-addr_0 and port number p_0 (its default IP address and port). First, this peer name_0 obtains a new port number from each player, *i.e.*, p_{new_i} for each $i = \{1, \ldots, k\}$, by communicating with each player name_i on its default IP address and port (*i.e.*, IP-addr_i and p_i). After the exchange, a new caller process is spawned at the peer name_0 and it communicates with newly spawned player threads at each player peer. All subsequent communication for the duration of the game is between the caller at IP-addr_0 and p_{new_0}, with each player name_i at IP-addr_i and p_{new_i} for each $i = \{1, \ldots, k\}$.

2.2.2 The `end` Command

Once a caller has been informed by a player that it is a winner, it informs all players in the game that the game has a winner, *i.e.*, instead of a call, it send a winner indication. This causes each player thread to terminate. Then the caller thread terminates. Assuming that the caller thread spawned by the `bingo` process it not detached, it recognizes the termination of the caller thread and it sends the `end game-identifier` command to the manager.

2.3 Bingo Protocol Definition

Recall that a *protocol* defines the format, order of messages sent and received among network entities, and actions taken on message transmission and receipt.

While the order of messages sent and received between your `manager` and `bingo` peer, and actions to take, have been defined in §2.1, you must define your message format. This is achieved by defining a structure with all the fields required. For example, you could define the command field as an integer, *e.g.*, if the `command` field is equal to one, then this corresponds to the `register` command. Alternatively, you could implement the command field as a string.

In general, you must define an upper bound on strings, *e.g.*, the length of a `name`, and other bounds that your application assumes. It is also useful to define return codes to help you differentiate the types of failures that can occur.

You must also define the protocol between your `bingo` caller and `bingo` player based on the guidelines in §2.2. Your message format and time-space diagrams showing the message order must be provided as part of your solution to this project; see the requirements in §4.

3 Port Numbers

Both TCP and UDP use 16-bit integer port numbers to differentiate between processes. Both TCP and UDP define a group of well known ports to identify well-known services. For example, every TCP/IP implementation that supports FTP assigns the well-known port of 21 (decimal) to the FTP server.

Clients on the other hand, use ephemeral, or short-lived, ports. These port numbers are normally assigned to the client. Clients normally do not care about the value of the ephemeral port; the client just needs to be certain that the ephemeral port is unique on the client host.

RFC 1700 contains the list of port number assignments from the Internet Assigned Numbers Authority (IANA). The port numbers are divided into three ranges:

- *Well-known ports:* 0 through 1023. These port numbers are controlled and assigned by IANA. When possible the same port is assigned to a given server for both TCP and UDP. For example, port 80 is assigned for a Web server for both protocols, though all implementations currently use only TCP.
- *Registered ports:* 1024 through 49151. The upper limit of 49151 for these ports is new; RFC 1700 lists the upper range as 65535. These port numbers are not controlled by the IANA. As with well-known ports, the same port is assigned to a given service for both TCP and UDP.
- *Dynamic* or *private ports:* 49152 through 65535. The IANA dictates nothing about these ports. These are the ephemeral ports.

3.1 Port Number Assignment

For this project, each group G is assigned a set of 1000 unique registered port numbers to use: $[1000 + (G \times 1000), 1000 + (G \times 1000) + 999]$. Do not use port numbers outside the range assigned for your group as you may then interfere with other projects!

4 Project Requirements

This project may be completed individually, or by a group of at most two people. It is expected that each group maintains a *private* code repository as it is developed for this project, using *e.g.*, `github`. You may be required to show the commits you have made to this repository through its development.

While this project has a milestone due date of 02/10/2019, there is no associated milestone submission. The milestone is intended for you to reach a certain target functionality by that date; see §4.3 for a suggested timeline.

Warning: Do not delay starting this project! There will be no extensions to the due date of Sunday 02/24/2019.

4.1 Deliverables

This socket programming project consists of two deliverables:

1. A design and implementation of your Bingo application. This includes:
   (a) The design and implementation of a communication protocol for Bingo game management between the manager server and a peer client.
   (b) The design and implementation of a communication protocol for Bingo play between a P2P server acting as the caller, and two or more P2P clients acting as players, interacting in a Bingo game.
2. A demonstration of your Bingo application on the racks in BYENG 217 between at least three peers running on different hosts.

Your Bingo application must be implemented in C or C++, and you must use sockets to implement the client/server (peer/manager) communication and the P2P (caller/player) communication.

4.2 Submission Details

Submit electronically before 11:59pm on Sunday, 02/24/2019, a zip file[1] named `Groupx.zip` where `x` is your group number.

This zip file should unzip into two directories, one named `Documents` and other named `Code`.:

1. **Design document. 30%** In the `Documents` directory you must provide a report (.pdf format preferred) that includes:
   (a) The design of your communication protocol for Bingo game management between the manager server and a peer client. This includes diagrams of the message format, and time-space diagrams showing the message order. (Some of this is described for you in §2.1 but must be converted into time-space diagrams.)
   (b) The design of your communication protocol for Bingo play between a P2P server acting as the caller, and two or more P2P clients acting as players, interacting in a Bingo game.
   (c) Describe your implementation design decisions, *i.e.*, major data structures, algorithms, choice of TCP or UDP sockets, threading, command line arguments, *etc.* .
   (d) Describe the status of your project, *i.e.*, what is working, what is not working.
   (e) Provide a link to your code repository.
2. **Code and user documentation. 70%** In the `Code` directory you must provide a report (.pdf format preferred) that includes:
   (a) All source code files making up your implementation of your Bingo application program.
   (b) A `makefile` that produces your `manager` and `bingo` executables.
   (c) You must provide user documentation that describes how to set-up and run your application.

A demonstration of your project will be scheduled in the two weeks after the due date. What you submit on the due date is what will be compiled and run for the demo, *i.e.*, in fairness to everyone, you may not continue working on the project after the due date. During the demo, you will be required to demonstrate the functionality of your Bingo application program (a rubric will be posted in advance of demos).

4.3 Suggested Timeline

Here is a timeline you should consider following to complete this project by the due date of Sunday, 02/24/2019:

Date	Milestone
01/27/2019	Project description comes out and is discussed in class on Monday, 01/28/2019.
02/03/2019	Complete protocol design, *i.e.*, first draft of message format(s), time-space diagrams.
02/10/2019	Complete implementation of commands/responses between client and `manager`. Have implementation of Bingo game well under way.
02/17/2019	Complete implementation the Bingo game application.
02/24/2019	Complete testing and debugging on racks in BYENG 217 for submission.

5 For Honours Project Credit

If you are interested in extending this project for honours credit please come talk to me about your idea.

[1] **Do not** use any other archiving program except `zip`.