CS 214 Fall 2025
Project IV: Nim

David Menendez

Due: December 8, at 11:59 PM (ET)

For this project, you and your partner will design and implement a server for an on-line multiplayer nim game service. You will write a program, `nimd`, which receives incoming connection requests from game clients, arranges matches, and tracks winners and losers.

1 Nim

Nim is a very simple game for two players. The game board begins with a set of stones, arranged into several piles. Players alternate turns. On a turn, players may remove as many stones as they want from any single pile. The game ends when no stones remain. The player who removed the last stone is declared the winner.

2 Software

Your program, `nimd`, will wait for incoming TCP connection requests on a specified port number. Normally, the port number would be a "well-known" number for the protocol, but to avoid collisions with other students, `nimd` will use its first argument as its port number.

`nimd` will listen for incoming connection requests and receive player name information from clients. Once two unmatched clients have connected, `nimd` will place them into a nim game by telling each player their opponent's name. During the game, `nimd` will inform each player of the board status and the player whose turn is next, and then wait for that player to send a move.

3 Protocol

The Nim game protocol (NGP) uses strings of ASCII text[1] as its message format and relies on TCP for transport.

3.1 Message format

Messages in NGP consist of several "fields", each terminated by a single vertical bar (`|`). The number of fields required is determined by the message type.

Every message, regardless of type, begins with three fields:

[1]"String" as in "sequence of characters". These are not C strings and do not include terminators.

```
0|22|OPEN|John Nim-Example|
0|05|WAIT|
0|25|NAME|1|Alice Opponentson|
0|17|PLAY|1|1 3 5 7 9|
0|09|MOVE|3|5|
0|18|OVER|2|0 0 0 0 0||
0|24|FAIL|22 Already Playing|
```

Figure 1: Examples of NGP messages

- The *protocol version*, a single decimal digit. For this project, the version number will always be 0 (zero).
- The *message length*, two decimal digits. This indicates the number of bytes in the message, excluding the protocol version and message length and their terminators. Note that this means messages cannot exceed 104 bytes in total.
- The *message type*, four ASCII characters.

Depending on the message type, there may be one or more additional fields.

The end of the message is indicated both by the message length and by the terminator following the last field. Every message must end with a vertical bar, and must contain the expected number of vertical bars (at least three), depending on the message type.

`nimd` should detect and report improperly formatted messages.

3.2 Message types

Messages in NGP may take one of these forms:

OPEN Sent by the client after opening a connection. Contains one field, the name of the player.

WAIT Sent by the server after receiving PLAY. No fields.

NAME Sent by the server once two players have been matched into a game. Has two fields: The player's number in the turn order (1 or 2) and the opponent's name.

PLAY Sent by the server when it is time for a player to move. Has two fields: the number of the player who will play next and the current state of the board.

MOVE Sent by the client after receiving PLAY (if it is that client's turn). Has two fields: the pile from which to remove stones, and the number of stones to remove.

OVER Sent by the server when the game has concluded. Has three fields: the number of the winning player, the final board state, and either an empty string or the string "Forfeit".

FAIL Sent by the client or server in response to an inappropriate message. Has one field, a string containing an error message listed in table 1.

See fig. 1 for some examples of NGP messages.

Player names The player's name may be any sequence of characters, excluding the vertical bar. You may also reject names containing control characters and non-ASCII characters, at your discretion.

For simplicity, we will impose a maximum name length of 72 characters.

Board state The board state included in the PLAY and OVER messages consists of several decimal integers separated by spaces. While the protocol allows for any number of piles, with no maximum pile size, `nimd` will always have 5 piles with at most 9 stones per pile.

3.3 Message flows

Setup handshake Upon initiating a connection, the client will send OPEN and the player's name to the server. The server will respond with WAIT.

Once this exchange has happened, `nimd` will consider that client ready to be placed in a match.

Matching Once two clients have been placed into a match, `nimd` will indicate to both clients that they should move by sending NAME along with the player's number and their opponent's name. This will immediately be followed by the first turn.

Turn The server will send PLAY to both clients, indicating the which player shall play next and the current state of the board.

The player who can play next will send MOVE.

This flow repeats until the board is empty.

End of game After the last move, the server will send OVER, indicating the winner, the final state of the board, and whether the game was won normally or by forfeit. The server and client will end the connection at that time.

3.4 Error conditions

Errors can occur when a message cannot be properly parsed, is incorrectly framed, has been sent at an inappropriate time, or describes a move not allowed in Nim.

Most errors will result in the server sending FAIL along with an error code. Table 1 lists the possible error codes and their meanings, and whether the server should close the connection after sending the message.

3.4.1 Session errors

If a client disconnects before being placed in a game (that is, after sending OPEN, but before receiving NAME), it should be dropped by the server and not considered for matching.

If a client disconnects during a game (that is, after receiving NAME, but before receiving OVER), it is considered a forfeit. The server will send OVER to the remaining player indicating that they won by forfeit.

Table 1: Error codes used with FAIL

`10 Invalid`	Message cannot be read	Close
`21 Long Name`	Player name too long	Close
`22 Already Playing`	Player already playing a game	Close
`23 Already Open`	OPEN sent more than once	Close
`24 Not Playing`	MOVE sent before NAME	Close
`31 Impatient`	MOVE sent during opponent's turn	
`32 Pile Index`	Specified pile out of range	
`33 Quantity`	Number to remove too large or too small	

3.4.2 Presentation errors

Message framing is determined both by the message length parameter that occurs second in the message format, and by the number of fields for that message type. In particular, this implies that the last byte of the message should be a vertical bar, and that the number of vertical bars following the message code will be one more than the number of fields for that message type. When attempting to determine whether a complete message has been received, services should check both conditions, and not make a blocking call to `read()` unless neither condition has been met.

For simplicity, endpoints are not required to attempt recovery from framing errors. Instead, they should send FAIL with the error message `10 Invalid` and close the connection.

If the message length is not two decimal digits followed by a vertical bar, then the message length cannot be determined.

If the message length is less than 5, then it cannot contain a complete message type code.

If the message type code is not four characters or does not match one of the types described in section 3.2, the number of required parameters cannot be determined.

If the number of bytes received is less than the stated message length, check whether all the necessary fields are present and terminated. If they are, the message is too short.

If the number of bytes received is greater than or equal to the stated message length, check whether the necessary fields are all present and terminated within the stated length. If they are not, the message is too long.

To avoid unnecessary blocking, endpoints should try to detect byte sequences that cannot be valid as early as possible. For example, after receiving `0|1|`, it is already possible to determine that the message is invalid.

Note that the message length refers to the bytes in the message *following* the message length and its terminator. The entire message, including the version and length indicator, will be 5 bytes longer. Hence the maximum message length of 104 bytes.

3.4.3 Protocol errors

Messages may only be sent at certain defined times and must comply with the rules of Nim.

If the client sends OPEN with a name longer than 72 characters, the server should respond with FAIL and the message code `21 Long Name`, and then close the connection.

If the client sends OPEN with a name that is already used by a player in an on-going game, the server should respond with FAIL and the message code `22 Already Playing`, and then close the connection.

If the client sends OPEN a second time, the server should respond with FAIL and the message code `23 Already Open`, and then close the connection.

If the client sends MOVE before the game begins (that is, before receiving WAIT and NAME), the server should respond with FAIL and the message code `24 Not Playing`, and then close the connection.

If the client sends MOVE when it is not their turn, the server should respond with FAIL and the message code `31 Impatient`.[2]

If the client makes a move that is not allowed, the server should respond with FAIL and either `32 Pile Index` or `33 Quantity`, depending on whether the pile index or the amount to remove is out of range.

4 Implementation

Your implementation of `nimd` will accept as an argument the port number it will use to listen for incoming connections. As player clients connect, it will pair them into games, assign roles to each player, and maintain the state of the game grid until the game has ended. Doing so will involve at least three sockets: one for listening, and one for each active connection.

It is recommended that `nimd` write a log to STDOUT, noting incoming connections and moves as they are made.

For simplicity, `nimd` will use the same starting configuration for all games: five piles containing 1, 3, 5, 7, and 9 stones, respectively.

4.1 Single game (70 points)

For partial credit, you may design `nimd` to handle at most one active game at a time. Once two players have connected, you can start the game and simply ignore incoming connection requests until the game ends. The protocol is designed such that `nimd` will only receive input from one client or the other, depending on the turn, so it is always possible to use `read()` to receive the next expected message.

4.2 Concurrent games (100 points)

For full credit, `nimd` should use `fork()` or multithreading (or some other method) to manage multiple concurrent games.

Additionally, `nimd` must maintain a list of all players currently in an active game. If a player attempts to start a game while already connected to the server, the server should respond with FAIL, error message "already present", and disconnect.

4.3 Extra credit (+20 points)

For extra credit, the concurrent game server must be able to receive and handle messages as soon as they arrive. For example, a client should not send MOVE unless they have been sent PLAY indiciating it is their turn. In a scenario where it is player 1's turn, player 1 has not yet sent a move, but player 2 sends MOVE, the server should immediately respond with FAIL 31 Impatient without waiting to receive a message from player 1.

[2]This is only detectable if you are attempting the extra credit.

Similarly, if player 2 disconnects while player 1 is considering their move, the server should declare player 1 the winner by forfeit without waiting for player 1's move.

5 Submission

Determine in advance which partner will be responsible for submitting the completed assignment. Communication and coordination is key to a successful project!

Submit a Tar archive containing a directory P4, containing:

- Your README, including the names of the authors, your test plans, and whether your server handles a single game, concurrent games, or the extra credit
- Your AUTHOR file, containing only one or two NetIDs
- Your source code file(s), including testing code
- Your make file
- Any test inputs used by your testing process