Remy Oukaour, 107122849
CSE 310, Group 2

Overview

All three parts of the project have been implemented:

1. The client program `nim.py` and server program `nimserver.py` communicate via a protocol similar to HTTP. The client program accepts the `help`, `login`, `remove`, and `bye` commands.
2. The server program handles multiple simultaneous client connections. The client program accepts the `games`, `who`, and `play` commands.
3. The client program accepts the `observe` and `unobserve` commands.

Concurrency- or network-related bugs may exist, but have not been encountered.

User documentation

Nim server (nimserver.py)

To run the Nim server, enter:

```
python nimserver.py
```

By default it will serve on the current machine's hostname, port 7849. To use another host or port, enter:

```
python nimserver.py HOST PORT
```

For help with usage, enter:

```
python nimserver.py -h
```

When the server starts, it outputs:

```
Listening on HOST:PORT... (^C to shut down)
```

No interaction is needed except to exit it. When it receives an interrupt signal, it outputs:

```
Shutting down...
```

However, the server will not actually exit until all clients have disconnected.

When a client connects, it outputs:

```
Connection from HOST:PORT on socket N, thread T
```

When a client disconnects, it outputs:

```
Disconnection by HOST:PORT on socket N, thread T
```

For every client request, it outputs:

```
HOST:PORT [DATETIME] "REQUEST" STATUS REASON RESPONSE
```

Every client uses a persistent connection in its own thread.
The number of simultaneous clients that can be handled is limited only by the number of allowed sockets and threads on the host machine.

If the server cannot serve due to connection problems, it outputs:

```
Unable to serve!
```

For instance, if a bad hostname is provided, it outputs:

```
Unable to serve!
getaddrinfo failed
```

Nim client (nim.py)

To run the Nim client, connecting to a specific server, enter:

```
python nim.py HOST
```

By default it will connect to port 7849. To use another port, enter:

```
python nim.py HOST PORT
```

For help with usage, enter:

```
python nim.py -h
```

When the client successfully connects to a server, it will output:

```
Welcome to Nim on HOST:PORT!
Type 'help' for help, 'bye' to exit.
```

The user is then presented with a > prompting them to enter a command.

The commands are:

- `help` — Lists the commands along with short descriptions.
- `login NAME` — Logs in to the server with the desired name. Usernames must be 1 to 32 characters, using only letters, numbers, underscore, dash, plus sign, or period.
- `games` — Lists all the current ongoing games, by ID and player names.
- `who` — Lists all the logged-in users available to play a game.
- `play NAME` — Begins playing a game with the named user, if they are available.
- `remove N S` — During a game, removes *N* objects from set *S*.
- `observe ID` — Starts observing an ongoing game.
- `unobserve ID` — Stops observing an ongoing game, if it is being observed.
- `bye` — Logs off the server and exits the program.

The client has built-in checks to avoid some errors before sending commands to the server. It ensures that commands and their parameters are valid, and corrects the user if they are not. Valid commands are sent to the server and the response, successful or otherwise, is displayed. For instance, the command `remove 0 1` would be rejected (since one cannot remove 0 objects), but `remove 1 1` is valid and would be sent (although the server might respond with an error if the move is out of turn or against the rules).

Connection problems may result in other error messages. For instance, trying to connect to a machine not running a Nim server outputs:

```
Could not connect to HOST:PORT!
No connection could be made because the target machine actively refused it
Exiting...
```

A typical use case for the Nim client would be to log in, see who is available to play, and start a game with someone (or wait for someone else to start a game). Then take turns removing objects from the sets, with the winner being the one to remove the last object. One can also see the ongoing games and observe one of them without participating. When finished, log off and exit.

System documentation

Nim protocol specification

The Nim protocol is used to communicate between clients and servers for the game of Nim. Its packet structure is modeled after HTTP; however, unlike HTTP, it is stateful and uses a persistent TCP connection for each client. It uses port 7849 for communication by default. This project implements Nim version 3.0.

Nim packets consist of a start line, zero or more headers, and an optional message body:

```
LETTER      = "A"-"Z" | "a"-"z"
DIGIT       = "0"-"9"
CRLF        = "\r" "\n"
SPACE       = ( " " | "\t" )+
token       = ( LETTER | DIGIT | "_" | "-" | "+" | "-" )+
nim-version = "NIM" "/" ( DIGIT | "." )+
nim-packet  = start-line ( header CRLF )* CRLF [ body ]
start-line  = request-line | status-line
header      = field-name ":" [ field-value ]
field-name  = token
field-value = <text without leading and trailing whitespace>
body        = <text>
```

The `Content-Length` header should indicate the number of bytes in the message body. No other headers are needed.

The start line of request packets consists of the method name, zero or more parameters, and the protocol version:

```
request-line = method SPACE parameters nim-version CRLF
method       = ( "A"-"Z" )+
parameters   = ( parameter SPACE )*
parameter    = token
```

The methods and corresponding parameters defined by Nim 1.0 are:

- `LOGIN` *`NAME`* — Logs in with the provided username. The server should respond with status 405 if the user is already logged in; 401 if the username is taken; and 201 if the user was successfully logged in.
- `REMOVE` *`N S`* — During a game, removes *N* objects from set *S*. The server should respond with status 405 if the user is not playing a game or if it is not their turn; 404 if there is no set *S*; 402 if *N* is less than 1 or greater than the amount in set *S*; 200 if the move was successful; and 204 if the move ended the game. The response's body should describe the modified game state. If the move was successful, the response should be queued for the user's opponent.
- `BYE` — Logs off the server and disconnects. The server should respond with status 202. If the user was playing a game, a message indicating their disconnection should be queued for their opponent.

- `PING` — Asks the server for any queued messages. The server should respond with status 200. The response's body should contain the queued messages, if any.

The ones introduced in Nim 2.0 are:

- `GAMES` — Lists all the current ongoing games. The server should respond with status 200. The response's body should list the current ongoing games.
- `WHO` — Lists all the logged-in users available to play a game. The server should respond with status 200. The response's body should list the logged-in users available to play a game.
- `PLAY` *`NAME`* — Begins playing a game with the named user. The server should respond with status 405 if the user is not logged in or is already playing a game; 404 if the opponent does not exist; 403 if the opponent is the user making the request; 401 if the opponent is already playing a game; and 203 if a game was successfully started. If a game was started, the response's body should describe the initial game state and should be queued for the user's opponent.

The ones introduced in Nim 3.0 are:

- `OBSERVE` *`ID`* — Starts observing an ongoing game. The server should respond with status 404 if the game does not exist; 401 if the user is already observing the game; 403 if the user is playing the game; and 200 if the user was successfully made an observer. Any further responses to `REMOVE` requests for the game should be queued for the user.
- `UNOBSERVE` *`ID`* — Stops observing an ongoing game. The server should respond with status 404 if the game does not exist; 401 if the user is not observing the game; and 200 if the user was successfully removed as an observer.

The response to any request may instead have status code 300, with the body containing any queued messages for the user; followed by the actual response in another packet.

The start line of response packets consists of the protocol version, the numeric status code, and the reason phrase:

```
response-line = nim-version SPACE status-code SPACE reason-phrase CRLF
status-code   = DIGIT+
reason-phrase = ( token | SPACE )+
```

The status codes and reason phrases defined by Nim 1.0 (and retained through 3.0) are:

- 200 OK — The request was successful.
- 201 Hello — The `LOGIN` request was successful.
- 202 Bye — The `BYE` request was successful.
- 203 Begin Game — The `PLAY` request was successful.
- 204 End Game — The `REMOVE` request ended the game.
- 300 Continued — The client should get another response without making a request.
- 400 Error — The request was unsuccessful due to client error.
- 401 Impossible — The request cannot possibly be fulfilled.
- 402 Illegal Move — The `REMOVE` request took too many or too few objects.
- 403 Forbidden — The request is not allowed or redundant.
- 404 Not Found — The request involved a resource that does not exist.

- 405 Method Not Allowed — The request method is not allowed in the client's current state.
- 418 I'm A Teapot — The client is a teapot.
- 500 Internal Error — The request was unsuccessful due to server error.
- 501 Not Implemented — The request method is not implemented by the server.
- 503 Service Unavailable — The server is currently unavailable.
- 505 Nim Version Not Supported — The request version is not supported by the server.

The 2xx status codes indicate a successful request; the 3xx ones indicate that further action is required by the client; the 4xx ones indicate client error; and the 5xx ones indicate server error.

`nim` module structure

The main functionality of this program has been placed in a Python module named `nim`. This allows it to be reused for different client or server programs, or any program that wants to use the Nim protocol. This project implements a text-based client and server, but one could easily build GUI-based ones with the same module.

The submodule `nimlib` is modeled after Python's built-in `httplib` module. It defines the values and classes needed to use the Nim protocol:

- Constants for the version of the Nim protocol supported by the module (3.0), the default port for Nim communication (7849), and the minimum and maximum number of sets and objects (3 to 5 sets, 1 to 7 objects per set).
- `methods` — A mapping from request methods to their expected parameters.
- Constants for the response status codes.
- `responses` — A mapping from status codes to human-readable reason strings.
- `NimException` — The class of exceptions raised for Nim-related errors.
- `NimPacket` — An abstract class; represents a parsed Nim packet. Both request and response packets have headers, a body, and raw data.
- `NimRequest` — Represents a parsed Nim request packet. Request packets have a protocol version, request method, and parameters.
- `NimResponse` — Represents a parsed Nim response packet. Response packets have a protocol version, status code, and reason string.

The submodule `server` defines the classes needed to implement a Nim server:

- `NimUser` — Represents a user connected to the server. Users maintain state: their client socket, username, current game, and queued messages.
- `NimGame` — Represents an ongoing game. Games maintain state: their ID, sets, players, and observers.
- `NimServer` — Represents a Nim server. Modeled after Python 3's `http.server.HTTPServer` class. Servers maintain state: their users, logged-in users' names, and ongoing games.
- `ForkingNimServer` — Extends `NimServer` to start a new process for each connection.

- `ThreadingNimServer` — Extends `NimServer` to start a new thread for each connection.
- `BaseNimRequestHandler` — Represents a connection to a server. Upon instantiation, it calls the `setup()`, `handle()`, and `finish()` methods; the `handle()` method repeatedly receives requests via the `parse_request()` method and delegates handling them to other methods (which are responsible for calling the `send_response()` method to respond). This class is extended by actual Nim server implementations.

The submodule `client` defines the functions and classes needed to implement a Nim client:

- `is_natural` — Checks whether or not a number is natural (a positive integer). Object amounts, set IDs, and game IDs in Nim are all natural numbers.
- `is_nim_username` — Checks whether a string is a valid Nim username.
- `NimConnection` — Represents a single, persistent transaction with a server. Modeled after Python's built-in `httplib.HTTPConnection` class. A connection is created upon instantiation, the `request()` and `getresponse()` methods allow for communication, and the `close()` method closes the connection.
- `NimClient` — Represents a Nim client. This class maintains a `NimConnection` to a server and has specific methods for communicating with it.

Server structure

The server program `nimserver.py` defines two classes:

- `NimTextRequestHandler` — Extends `BaseNimRequestHandler` to output logging information for each client's connection, request, and disconnection.
- `NimTextServer` — Parses command-line arguments to instantiate and run a `NimServer` that uses a `NimTextRequestHandler` for each client.

Client structure

The client program `nim.py` defines the `NimTextClient` class, which parses command-line arguments to instantiate a `NimClient`. The main loop repeatedly prompts the user for commands and parses them into requests for the `NimClient`. A background thread sends a `PING` request to the server every second when the main thread is waiting for user input, in order to receive queued messages.

Testing documentation

Some test scenarios that demonstrate edge cases and other unexpected conditions:

- Start a server and connect at least 10 clients to it. **Result:** The server is capable of serving at least that many clients, as expected.
- Start at least 5 games between 10 clients. **Result:** The server is capable of handling at least that many games, as expected.

- Start a game between two users, with a third observing, and have one enter the `bye` command. **Result:** Their opponent and the observer will be notified that the user has quit. If the observer quits without unobserving the game, it will not be interrupted.
- Start two games between two users each, and have one user observe the other game. **Result:** The user will successfully play one game while observing another.
- Modify line 179 of `client.py` to send an invalid `GAMEZ` request instead of `GAMES`, and comment out lines 72 and 73 to allow the request to be sent. Have a modified client enter the `games` command. **Result:** The server responds with `501 Not Implemented`, and the client outputs:
  ```
  Unsupported method (GAMEZ)
  ```
- Replace `NIM_VERSION` on line 81 of `client.py` with `4.0`. Start a modified client. **Result:** The server responds to every packet with `505 Nim Version Not Supported`, and the client outputs:
  ```
  Unsupported Nim version (4.0)
  ```
  (This happens every second without user input due to the client `PING`ing the server in a background thread.)
- Shut down a server while clients are still connected. **Result:** The server will output:
  ```
  Shutting down...
  ```
  but will continue to run until the clients disconnect, as expected.
- Kill a server's process while clients are still connected. **Result:** The clients will output:
  ```
  An existing connection was forcibly closed by the remote host
  ```
  when they next send a command, and will quit. (Their background `PING` threads will output the same thing without waiting for user input, but this will not quit the clients.)