

CSIS0234B Computer and Communication Networks (Class B)
Assignment 1
Netjack

Tutor: vjwmkwan@csis.hku.hk, Deadline Feb 20, 2004, 5:00pm.

In the course web page, you can find a simple program that plays the card game “Blackjack”. The program acts as the dealer, allowing the user to play Blackjack with it. Your task is to turn this single-player program played on a single computer into a multi-player program played on multiple computers over the Internet, using the protocol specified below.

This is a group assignment allowing 1–2 students in each group.

1. Rule of the game

The program plays a simplified variant of the Blackjack game¹, with a dealer which uses 6 decks and the Stand-On-17 (S17) rule. Here’s the precise description of the game. Blackjack is a casino game played with playing cards, where players decide whether they get one more card (“hit”) or stop getting cards (“stand”). The objective of the game is to compete against the “dealer”² on how close they can get to a “hand value” of 21, without exceeding it. The hand value of a hand of cards is defined as follows:

1. Each playing card is assigned a “card value”, which is the number on the card if there is one, or 10 if it is a face card (Jack, Queen and King), or 11 if it is an Ace.
2. If the sum of the card values is above 21, and there is an Ace that has the card value 11, then the card value of one such Ace is changed to 1, and this step is repeated.
3. The hand value is the sum of card values of the hand after step 2.

Initially, the dealer shuffles a set of cards containing 6 standard decks (so there are 312 cards). Then he works mechanically to play games repeatedly. Each game is described as follows.

1. If the number of cards remaining is less than 79, the dealer shuffles all the cards again.
2. The dealer invites for bets. Those who bet become the players.
3. The game starts, by the dealer drawing cards and passing them to himself and then each player in two rounds. In each round every player gets a card, so at the end every player gets two cards. All cards are faced up (i.e., every player can read the card) except that the first of the dealer’s card is faced down (only the dealer can read it).
4. The dealer and each player check whether they Blackjack: get an initial hand value of 21. If the dealer Blackjacks, he wins the bets of all other players who do not Blackjack (and return the bet of all players who also Blackjack), and this game ends. If the dealer does not Blackjack, all players who Blackjack wins 1.5 times of his bet.
5. For each player who didn’t Blackjack, the dealer repeatedly lets him decide whether to hit or stand, until he decides to stand or his hand value is at least 21. Each time the player decides to hit, the dealer draws one more card for him. If he gets a hand value larger than 21 (“burst”), he loses his bet, and his cards are removed from table immediately.

¹The simplifications include the removal of the capability for the player to choose “surrender”, “split pairs”, “double” and “take insurance”. It is also a “shoe-game”: most cards are face-up.

²In the description, the dealer is not counted as a player.

6. If there is any player who didn't burst or Blackjack, the dealer turns up his card which was faced down. Then he hits until his hand value is 17 or more. If the dealer bursts, each player who didn't burst and didn't Blackjack wins his bet. Otherwise, the hand value of each such player is compared against the hand value of the dealer: if the former is less, the player loses his bet, if the former is more, the player wins his bet, and if they are equal his bet is returned.

2. Protocol

You should write two programs, one to act as the server which is the dealer, the other to act as a client which is a player. The server runs on a TCP port that is specified by the `-p` option of the server program. Once it starts, it should not end until explicitly killed (by a signal, e.g., the user pressed Control-C).

To allow easy debugging of a program, a line-oriented protocol is used. The protocol is strongly inspired by the SMTP protocol. In general, each message sent by the server is a line, terminated by the LF character (ASCII 10). It consists of a 3-digit message type, a space, then the space-separated arguments of the message (if any), optionally followed by a space and a comment. The 3-digit message type is coded in a way that tells the client what it is expected to do, and what state is the server currently in. The first digit is 1 if the message is "information", where the client needs doing nothing; it is 2 if it is a "request", where the client is allowed to respond by sending a command; and it is 5 if it is an "error", where the client has sent something wrong. The second digit is 0 if no game is in progress, 1 if the dealer is doing the initial passing of cards and checking for Blackjack, 2 if the dealer is getting decisions from the players, 3 if the game is about to end, and 9 for general messages that can occur in any state. The client is not expected to understand the comments: they are only intended for debugging purpose (and for human who enjoy playing the game using just `telnet`).

A server message can have one or more arguments. Each argument is a number, which is encoded as an ASCII string in decimal. Cards are also represented by numbers: 1 means Spade Ace, 2 means Spade 2, 3 means Spade 3, ..., 11 means Spade J, 12 means Spade Q, 13 means Spade King, 14 means Heart Ace, ..., 26 means Heart King, 27 means Diamond Ace, ..., 40 means Club Ace, ..., 52 means Club King. 0 means a card which is faced down.

The following server messages are defined:

- 201: The server is waiting for a bet to start off the game. This is sent once to each client connected during the initial waiting period.
- 102 *n*: A game is about to start in *n* seconds. This message is sent once to each client connected during the period between the first bet and the game starts.
- 103 *p*: You are the *p*-th player. The first player, naturally, gets the value 1. The player receives the message after he makes his bet.
- 104 *p*: Player *p* joins the game. This message is sent to all clients other than the *p*-th player.
- 111: A game is started. The message is sent to every client.
- 112 *p c*: A card *c* is given to player *p*, where *p* = 0 if it's the dealer. The message is sent to every client.
- 113 *c*: Dealer Blackjacks, with faced down card *c*. The message is sent to every client.
- 114 *p*: Player *p* Blackjacks. The message is sent to every client.

- 121: Decision period starts. The message is sent to every client.
- 222: Your decision. This message is sent to the player who is expected to make a decision to hit or stand.
- 123: Timeout, stand assumed. The player gets this message if he does not respond correctly to a 222 message within 10 seconds.
- 124 $p\ c$: Player p gets card c . It is similar to message 112. If a new connection is made at a time when a game is in progress, he gets a sequence of such messages at the beginning to tell him all the cards on table.
- 125 p : Player p bursts and loses his bet. This is sent to every client.
- 126 $p\ n$: Player p has hand value n . This is sent to every client after player p decided to stand.
- 131 c : Dealer uncovers card c . This is sent to every client.
- 132 c : Dealer gets card c . This is sent to every client.
- 133: Dealer bursts. This is sent to every client.
- 134 n : Dealer has hand value n . This is sent to every client.
- 135: Game ended. This is sent to every client.
- 136 $p\ n$: Player p wins $\$n$. Here $n > 0$ indicates winning, $n = 0$ indicates a push (draw), and $n = -bet$ (where bet is the bet initially put by the player) indicate losing the game. This message is sent to every client.
- 137: A shuffle is made after the game ends. The message is sent to every client.
- 190 n : There are n clients connected. This is sent to all clients whenever a client comes or a client leaves.
- 596: Maximum number of clients reached, so connection is refused. The server only allows 10 clients at the same time.
- 597: Your command is unexpected and is ignored.
- 598: Your command is not understood and is ignored.
- 599: Your command has wrong argument and is ignored.

The client gives one of three commands. Again it is line-oriented. Each line starts with a case-insensitive command word, followed by a space and its argument (if any), optionally followed by a space and a comment which is ignored by the server. The following commands are recognized:

1. `hit`: A decision to hit. No argument.
2. `stand`: A decision to stand. No argument.
3. `bet`: Make a bet of n dollars. For ease of Blackjack handling, n must be even, and must be within the range from 2 to 200. (The server generates a 599 message otherwise.)

If a client terminates the connection before a game ends, the game will treat it as if it leaves only after the game ends, and all its remaining decisions considered “stand”.

3. Example dialog

Here we show a possible sequence of message sent by each party, with comments to make them clear. The symbol at the beginning indicates the direction of the message and the recipients.

```
->A      201      Waiting for a bet to start.
->All    190 1     There is 1 client.
```

```
->B      201      Waiting for a bet to start.
->All    190 2     There are 2 clients.
<-A     bet 5
->A     599      Bet must be even.
<-A     bet 10
->A     103 1     You are Player 1.
->All    102 10    A game to start in 10 seconds.
->B     104 1     Player 1 joins.
<-B     bet 8
->B     103 2     You are Player 2.
->A     104 2     Player 2 joins.
->All    111      Game started.
->All    112 0 0   I get face-down card.
->All    112 1 3   Player 1 gets Spade 3.
->All    112 2 26  Player 2 gets Heart K.
->All    112 0 27  I get Diamond Ace.
->All    112 1 26  Player 1 gets Heart K.
->All    112 2 40  Player 2 gets Club A.
->All    114 2     Player 2 Blackjack.
->All    121      Decision period starts.
->A     222      You have S3 and HK, I have DA.  Your decision?
->C     111      Game started.
->All    190 3     There are 3 clients.
->C     124 0 0   I get face-down card.
->C     124 1 3   Player 1 gets Spade 3.
->C     124 2 26  Player 2 gets Heart K.
->C     124 0 27  I get Diamond Ace.
->C     124 1 26  Player 1 gets Heart K.
->C     124 2 40  Player 2 gets Club A.
<-C     bet 8
->C     597      Game in progress.
<-A     hit
->All    124 1 29  Player 1 gets Diamond 3.
->A     222      You have S3, HK and D3, I have DA.  Your decision?
<-A     stand
->All    126 1 16  Player 1 has hand value 16.
->All    131 2     Hidden card was Spade 2.
->All    132 37    I get Diamond Jack.
->All    132 43    I get Club 4.
->All    134 17    My hand value is 17.
->All    135      Game ended.
->All    136 1 -10 Player 1 loses $10.
->All    136 2 12  Player 2 wins $12.
->All    201      Waiting for a bet to start.
...     ...
```

4. The Netjack server (70%)

All intelligence of the game rule should be in the server, not in the client. This allows the server to change the game rule easily without modifying the clients. The client only needs to remember all

the cards that the server indicates and display them. Here are some hints for writing the server.

- The protocol is written so that it is possible to play a game using the `telnet` program by typing the protocol directly. So write the server first, before the client.
- A new client can pop up any time, and the server has to get the connection and respond to them (either by rejecting the connection, or by letting it watch the game in progress). Also, you have to implement timeouts at some points of the interactions. To add to the complications, you also have to deal with the possibility that a client gets stuck before sending the end-of-line character. So you must not use standard `read()` and `write()` system calls until you already know that the connection can be read or written without blocking. So you must use `select()` to check whether any of the channels is ready for input or output, and perform the operation only when it can be done without stopping the rest of the program.
- The above might make you think that the sample program must be substantially changed. This is not the case. What you need is a generic function that deals with all “bookkeeping” interactions. In this function, wait for all the possible inputs using `select`, collect the input into an input buffer until a LF character is received; collect all the output in an output buffer, and send it to the client once the socket is ready for output. Once you have such a function, to output you simply need to append to the output buffer. Then implement input in another function, which calls this function to wait until a line is read, and handle the line. Handling is different for bet and for hit/stand decision, but these are the only locations in the server which requires input.
- Connections to clients that do not read the input, and hence cause large output buffers, should be closed to avoid the server using too much memory. Similarly, disconnect those clients that send lines which are too long, e.g., longer than 1024 bytes.
- Remember that the data arguments of `read()` and `write()` is not NULL-terminated strings. That is the reason why these system calls has a length argument. Also, you should not include a NULL character in the network messages.
- When generating messages, follow the specification strictly. On the other hand, when accepting input, you may be a bit less strict, possibly allowing messages that deviate from the specification slightly. E.g., it might be easier for your program to ignore the distinction between spaces and tabs if you use an *istringstream* to parse the input.

5. The Netjack client (30%)

The client program connects to the server program and allows the user to play with the server in a more user-friendly fashion than typing on the `telnet` program. It should provide an interface similar to using `telnet`, with the following requirements:

- The only comments that the client may use are the ones about errors. All other comments are not used. Instead, the client derives the messages to send to the user from the message code in the received messages. This allows the program to play against a server which provide no comment at all.
- The client must remember what cards are held by each player, and show the cards grouped by the players when the decision period is started and when the game is about to end (i.e., after the 134 message is received). To aid the player in making his decision, the cards held

by the dealer and the player must also be presented when a decision is asked (i.e., when 222 is received).

The simplest way to write the client is to have two separate processes, one for dealing with the input from the socket and output to the display, while the other for dealing with the input from the keyboard and output to the socket. The latter simply requires directly feeding everything the user types to the remote end of the connection, and is hence more or less trivial. On the other hand, the former would require interpretation of the server messages. You can start by just printing out the server message, and once you know the communication is done correctly, you can start adding code to recognize the server messages.

6. References

The tutorial 1 and tutorial 2 should give you most of the system calls you need. You will also need to understand exactly how `select(2)` works. Read the man page. Note also that the socket library of Solaris requires extra libraries to be linked with the program. See the man page `socket(3socket)`.

7. Hand-ins

Submit the source code for your two programs to the handin system of the department. Do not send any executable file. Be sure that your program will compile and work on Linux.

Apart from the source code, submit a file `readme.txt` to tell the TA your group members, as well as any known problem that may prevent the TA to use her client to connect to your server or vice versa.