Arizona State University W. P. Carey School of Business

CIS 345 – Business Information Systems Development II

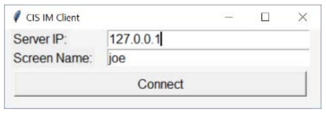
Assignment 8: Instant Messenger

Learning Outcomes

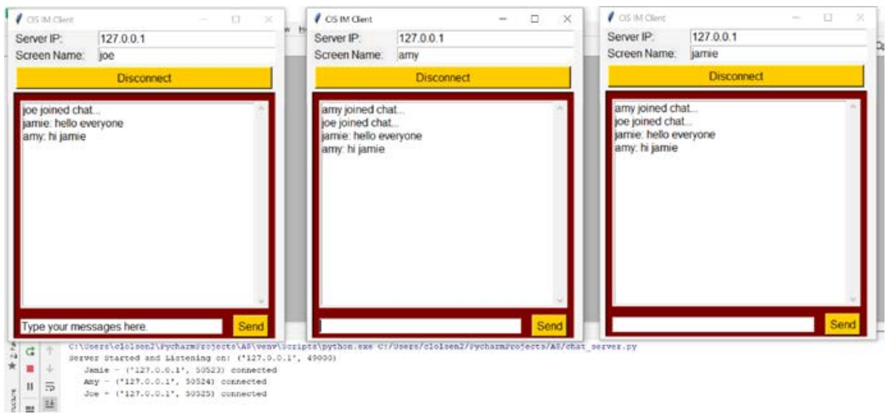
1.1. Implement tkinter GUI Python Application
1.2. Implement and utilize networking sockets
1.3. Develop callback functions/event handler functions tied to a GUI
1.4. Integrate networking logic with your GUI to send messages to the server

Program Overview

A server chat_server has been developed and deployed with your organization. Management has assigned you the task of developing a chat client GUI application. No changes can be made to the server because it has already been tested and deployed. Your client must work with the server and provide a nice user interface (UI). The UI design team has come up with some ideas of the look and feel for the chat client application (see screenshots). The application must be able to take a username and connect to the server. Once connected, the client will start a receive_message microservice. This service will run in its own Thread and will constantly receive messages sent from the server and display them within the GUI client application. While this service is running, the chat client will allow the user to enter messages within the GUI and send them to the server. The server will then take the messages and broadcast them to all users in chat (connected to server). The server allows private messaging a single user by typing first the @screen_name, a space, and then the private message. Doing this will send the message to only that one user. The user of the chat client must have the ability to Connect and Disconnect to/from chat servers and be able to enter the IP address of the server they wish to connect to.



Server IP: 127.0.0.1
Screen Name: joe



Server IP: 127.0.0.1
Screen Name: joe

```
joe joined chat...
jamie: hello everyone
amy: hi jamie
```

Type your messages here.

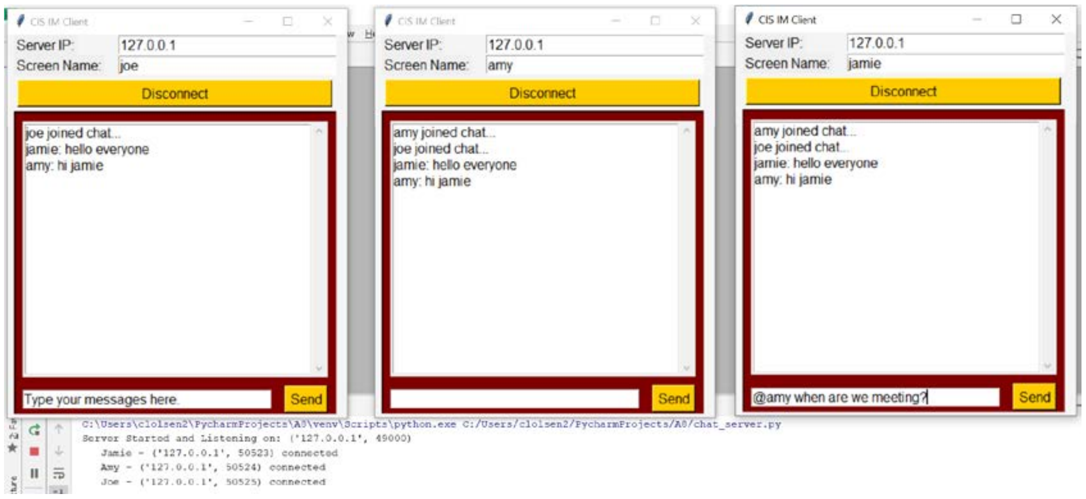
Server IP: 127.0.0.1
Screen Name: amy

```
amy joined chat...
joe joined chat...
jamie: hello everyone
amy: hi jamie
```

Server IP: 127.0.0.1
Screen Name: jamie

```
amy joined chat...
joe joined chat...
jamie: hello everyone
amy: hi jamie
```

```
C:\Users\clolsen2\PycharmProjects\A8\venv\Scripts\python.exe C:/Users/clolsen2/PycharmProjects/A8/chat_server.py
Server Started and Listening on: ('127.0.0.1', 49000)
   Jamie - ('127.0.0.1', 50523) connected
   Amy - ('127.0.0.1', 50524) connected
   Joe - ('127.0.0.1', 50525) connected
```



Server IP: 127.0.0.1
Screen Name: joe

```
joe joined chat...
jamie: hello everyone
amy: hi jamie
```

Type your messages here.

Server IP: 127.0.0.1
Screen Name: amy

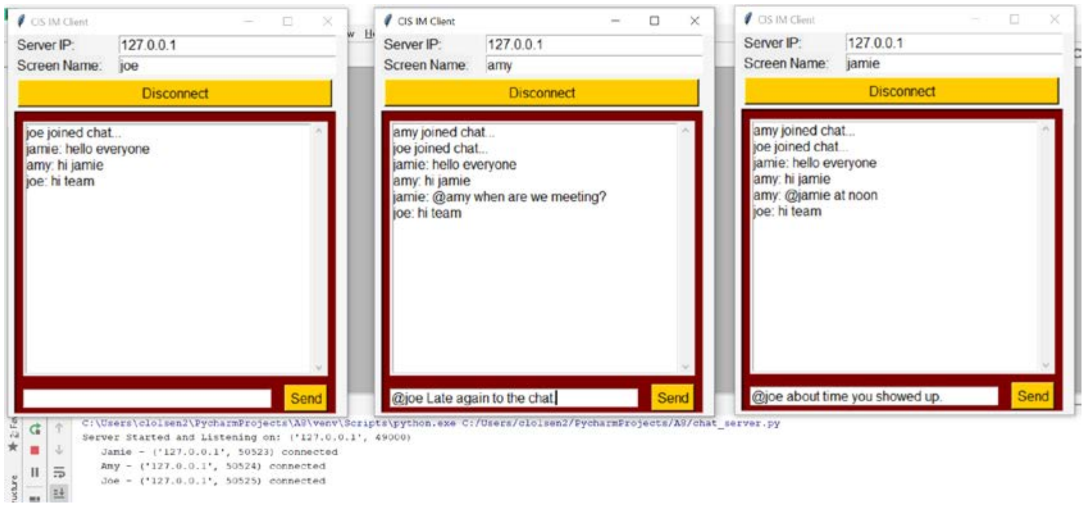
```
amy joined chat...
joe joined chat...
jamie: hello everyone
amy: hi jamie
```

Server IP: 127.0.0.1
Screen Name: jamie

```
amy joined chat...
joe joined chat...
jamie: hello everyone
amy: hi jamie
```

@amy when are we meeting?

```
C:\Users\clolsen2\PycharmProjects\A8\venv\Scripts\python.exe C:/Users/clolsen2/PycharmProjects/A8/chat_server.py
Server Started and Listening on: ('127.0.0.1', 49000)
   Jamie - ('127.0.0.1', 50523) connected
   Amy - ('127.0.0.1', 50524) connected
   Joe - ('127.0.0.1', 50525) connected
```



Server IP: 127.0.0.1
Screen Name: joe

```
joe joined chat...
jamie: hello everyone
amy: hi jamie
joe: hi team
```

Server IP: 127.0.0.1
Screen Name: amy

```
amy joined chat...
joe joined chat...
jamie: hello everyone
amy: hi jamie
jamie: @amy when are we meeting?
joe: hi team
```

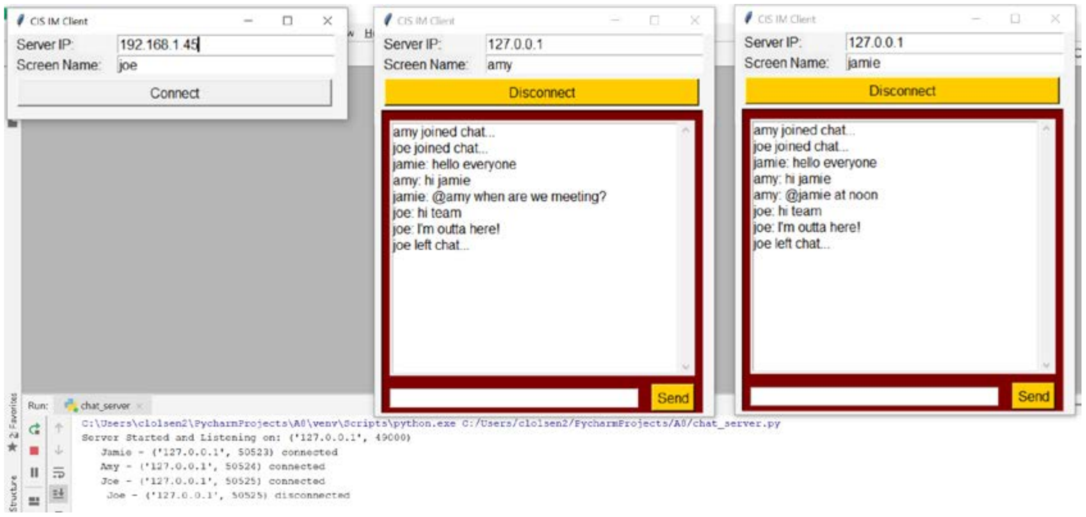
@joe Late again to the chat

Server IP: 127.0.0.1
Screen Name: jamie

```
amy joined chat...
joe joined chat...
jamie: hello everyone
amy: hi jamie
amy: @jamie at noon
joe: hi team
```

@joe about time you showed up.

```
C:\Users\clolsen2\PycharmProjects\A8\venv\Scripts\python.exe C:/Users/clolsen2/PycharmProjects/A8/chat_server.py
Server Started and Listening on: ('127.0.0.1', 49000)
   Jamie - ('127.0.0.1', 50523) connected
   Amy - ('127.0.0.1', 50524) connected
   Joe - ('127.0.0.1', 50525) connected
```



Server IP: 192.168.1.45
Screen Name: joe

Server IP: 127.0.0.1
Screen Name: amy

```
amy joined chat...
joe joined chat...
jamie: hello everyone
amy: hi jamie
jamie: @amy when are we meeting?
joe: hi team
joe: I'm outta here!
joe left chat...
```

Server IP: 127.0.0.1
Screen Name: jamie

```
amy joined chat...
joe joined chat...
jamie: hello everyone
amy: hi jamie
amy: @jamie at noon
joe: hi team
joe: I'm outta here!
joe left chat...
```

```
C:\Users\clolsen2\PycharmProjects\A8\venv\Scripts\python.exe C:/Users/clolsen2/PycharmProjects/A8/chat_server.py
Server Started and Listening on: ('127.0.0.1', 49000)
   Jamie - ('127.0.0.1', 50523) connected
   Amy - ('127.0.0.1', 50524) connected
   Joe - ('127.0.0.1', 50525) connected
    Joe - ('127.0.0.1', 50525) disconnected
```

Instructions & Requirements

- Create a new project and download the chat_server.py module and add it to your project.
- Add a new module called gui_im_client.py.
- Add required comments with name and course info on line 1 of all .py files.

Design Requirements

You are free to customize colors. Placement can be close (doesn't have to be exact). Your goal is to make a professional looking chat client application.

GUI Design

Window Design will include the following widgets:

1. Tk – create a window to hold all widgets
2. Two Label – Server IP and Screen name. Place at top left are of window
3. Two Entry – get entered IP and screen name. Link to StringVar objects
4. Button – Connect/Disconnect button. Open or close your socket.
5. Chat Frame – One to hold the chatting items (maroon background in pics)
6. Frame – contains the messages listbox within the chat frame
7. Scrollbar – for Listbox widget within frame. Scrolls vertically.
8. Listbox – insert messages received from server for the user to read
9. Entry – get messages to send to the server. Link to StringVar object
10. Button – call send_message function to send the entered message to server

I recommend using grid to place item in window and the chat frame, but pack for the frame containing the list box. Listbox can then be packed and utilize pack option fill=BOTH to have the list completely fill frame.

To create a scroll bar for your Listbox code the following:

- Create scrollbar – scrollbar = Scrollbar(frame_name)
  frame_name is the name of your frame you want the scrollbar in
- Then write code to create listbox and add to form.
  - Add this keyword when constructing to list of other options
    `Listbox( …, yscrollcommand=scrollbar.set, … )`

  - The above command will move scrollbar when you use arrows to move in listbox
- Lastly, add command option to scrollbar after you coded the list box:
  `scrollbar.config(command=listbox_name.yview`
  Where listbox_name is the name of the variable/object you assigned the widger
- Use pack_forget() or grid_forget() to hide any frame and then resize window using window.geometry().

Functionality Requirements

Events Handler Functions / Callback Functions:

1. IP address Entry widget – Key press events must be handled to only allow valid keys to be used – bind on <Key> that call your function. Valid keys:

   ```
   valid_keys = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '.', '\b', '']
   ```

   '\b' is the backspace key and '' is the delete key
   If user presses a non-valid key return "break" otherwise do nothing

2. Connect function – You will need access to global variables such as your socket, IP address, screen name, connect/disconnect button, window, and chat frame. Perform the following:
   a. If IP is greater than 6 characters and a screen name has been entered
      i. Create ADDR an address using entered IP and port 49000
      ii. Construct your socket using IPv4 and TCP sock stream protocol
      iii. Connect socket to the ADDR
      iv. Encode and Send screen name to server using socket
      v. Place the above in Exception handling and if an Exception occurs
         1. Call close() within socket
         2. Assign None to socket object
      vi. If no exception, basically the else case of try:
         1. Create Thread for receiving messages and store it
            `X = Thread(target=recveive_func, daemon=True)`
            receive_func is item number # below
         2. Start receive message thread
      vii. Set window to full size so chat frame can be seen
      viii. Set connect button config options:
         1. bg – a new color to show connected
         2. text – 'Disconnect'
         3. command – to the disconnect function item # 3 below
      ix. Place chat frame using grid on window:
         1. Sticky = N+S+W+E
   b. Else the IP or screen name is not entered:
      i. Messagebox.showinfo(…) – Error, tell them how to enter the data

3. Disconnect function – You will need access to global variables such as your socket, IP address, screen name, connect/disconnect button, window, and chat frame. Perform the following:
   a. Try
      i. Send EXIT message of '[Q]' using socket encoded
   b. Except
      i. Pass if an exception no error messages needed
   c. Finally
      i. Close the socket using .close() and assign None to socket object

   d. Set connect/disconnect button widget:
      i. bg – ‘SystemButtonFace’ is the default color it starts with
      ii. text – ‘Connect
      iii. command – back to the connect function item #2 above
   e. Remove chat frame using .grid_forget()
   f. Change window geometry to smaller size per screenshots
   g. Clear the entered message Entry widget

4. Receive message function – You will need access to global variables such as your socket and screen name. Perform the following in an infinite **while loop**:
   a. Try
      i. Receive 1024 bytes using socket and save as received message
   b. Except OSError
      i. Assign None to the received message object
      ii. Break out of while loop
   c. If not received message (meaning it got nothing – due to server closed)
      i. Call disconnect function
      ii. Break out of loop
   d. Decode the received message and insert into your Listbox widget at the END

5. Send message function – You will need access to global variables such as your socket and entered message. Perform the following:
   a. Get the message from the StringVar object and store in a variable
   b. If the message == EXIT sequence of ‘[Q]’
      i. Call disconnect function
   c. Else if message is not blank
      i. Try
         1. Send encoded message using socket
      ii. Except OSError
         1. Call disconnect function
   d. Set message StringVar object to a blank string

6. Window closing function – You will need access to global variable socket. This function will be called if the user clicks on the X in upper right corner to close the GUI application. You’ll need to add this code before the .mainloop() call:

```
window.protocol("WM_DELETE_WINDOW", window_closing)
```

   Within the window closing function - Perform the following:
   a. If socket (this checks if you have a socket – True if not None)
      i. Call disconnect function
   b. Use your Tk window object to call .quit()

Testing: Run the provided chat_server.py using a command prompt or terminal. Run one of your clients using Pycharm so that you will be able to debug your application. When you believe you have it working. Open additional terminals/command prompt windows and execute your client application using python filename.py.

Tests

- Connect and Disconnect from server
- Try connecting without entering screen name or IP or both
- Connect and send a message – Is it broadcast to all clients?
- Send a private message using @name message format – does only intended user get it?
- Try sending a blank message
- Close the application using X in upper right both when connected and then when disconnected.
- Do all of the above work successfully?

Assignment-specific Submission Instructions (if any)

All CIS 345 submissions must adhere to standards detailed in the following documents available on Blackboard, for full credit.

- CIS 340 345 Assignment Submission Instructions
- CIS 340 345 Programming Conventions
- CIS 340 345 Commenting Guide

General Grading Criteria

1. Assignments will be scored out of 30 points.
2. Assignments will be on source code AND output.
3. Do not worry about the program crashing due to large numbers.
4. **You can make a working game without the reuse of event handlers. You can score up to 26/30 points without reusing event handlers**

Grading Criteria	Points
GUI - All widgets are present and configured correctly	5
Application will Connect and Disconnect from server without producing errors on the server	8
Receive service created on its own thread and receives messages as soon as they are sent by server	4
Client application sends messages to all users and can private message individual users	4
Other function: IP address entry only allows valid keys to be entered based on requirements. On closing window the client disconnects.	6
Style and Standards CIS 340 345 Programming Conventions are followed CIS 340 345 Commenting Guide is followed File names and project names are accurate Class file has name, class, assignment number, and class time written on Line 1.	3