Provider Scheduling

Please note this project is built in Python using the python 2.7 interpreter. This is important because the SocketServer class was integrated into socket class in versions 3 and up and will throw a class error in python 3.

**Client:**

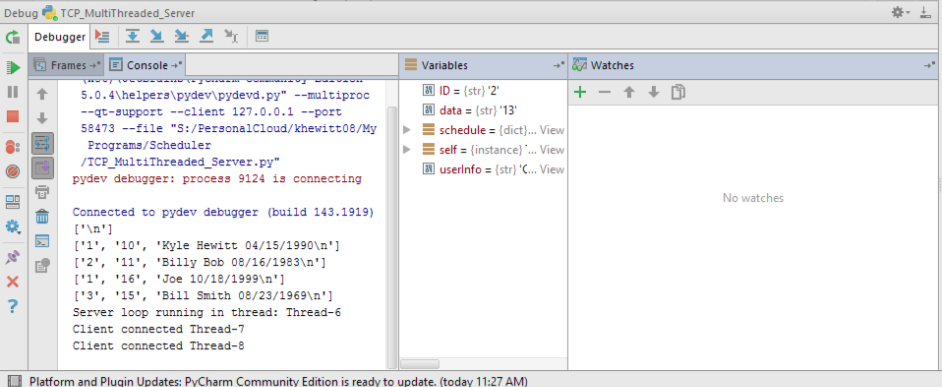
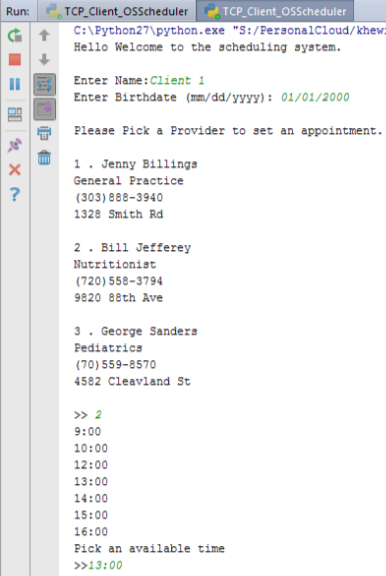
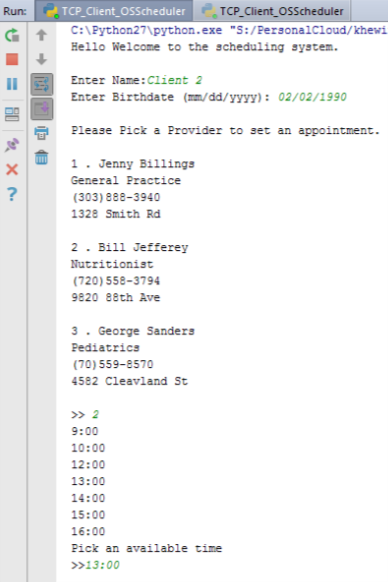
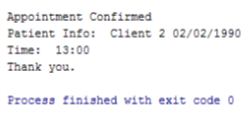
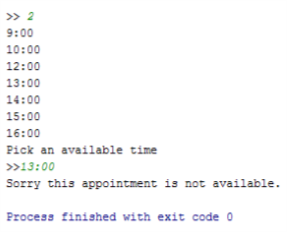
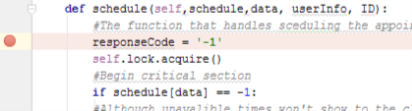
The scheduling client begins by declaring the HOST and PORT attributes. In this case HOST is “localhost” and PORT is 4445. Then a sock object is created from the socket class passing in the attributes socket.AF\_INET to signify an internet type application and socket.SOCK\_STREAM to indicate a stream (TCP) connection. Then the client attempts to use the server, throwing a socket.error if it cannot establish the connection. The client program is almost entirely functional so no classes or functions are used. The client program starts by printing out a message to the user and then asking for their personally identifiable information. This is implemented into the program because having something as simple as name and birthdate could make recalling appointments made from the server very easily. This is not currently implemented in this program. After the user enters their name and birthdate, the client sends the user’s info to the server with the string code ‘1’. The client server protocol used is a simple accumulator protocol, with ‘1’ signifying that it’s the first communication from that client. This prompts the server to send over the list of available providers as a single string separated by a colon separator. This is used to split the string into a list of providers to be printed out for user selection. The provider information contains the PA name, practice category, phone number and address. The user selects the number to the left of the provider information to pick which provider to request availability for. This section was written strategically so that if a provider is added to the list, no change in code would be needed to handle any number of providers sent by the server as long as they are separated with a colon. Once the user selects the number of their provider, it is passed to the server with a protocol number 2, telling the server this was a provider selection and to send back available times for that provider. The server keeps track of the appointments in dictionaries so the order that the available times arrive to the client is unpredictable, so in order to print out available times an accumulator is used starting at 9 and checks if the accumulator value is in the available times and if so its printed on the screen for user selection. The user will only be shown times that don’t have any scheduled appointments at the time of the request. If a user selects a time that is not shown or that time is booked between the time of the server response, a denial message will be displayed and the client program will terminate. The client can enter the hour of the appointment they would like to book (i.e. 13) or as displayed (i.e. 13:00). If the latter is used the ‘:00’ is removed and only the hour of the request is sent to the server with no protocol number because the server is still inside protocol routine 2 waiting for client response. At this point the server can send back ‘-1’, indicating that the appointment was denied, or ‘200’, indicating that the appointment was successfully scheduled. If ‘200’ is received the user receives a message with a confirmation of their info and the time of the scheduled appointment. The socket is then closed and the client program terminated. If at any point a network error occurs, the socket is closed and the client program is terminated.

**Server:**

The server utilizes both volatile and non-volatile memory to increase performance and reduce I/O operations. The schedules are kept in key-value pairs in dictionary data structures. By default, all the libraries are initialized with the appointment times ranging from 9 to 16 as the values and each of the values are set to -1 to indicate that the value is not valid appointment info. Once the server program is running and the libraries are initialized to default values, it then reads from a file to fill in appointments that were created prior to this instance of the server. In the file an empty line (‘ ’) indicates the end of the document. Each appointment entry is on a separate line, starting with a number indicating which provider, the time and the personally identifiable information for the appointment holder. Each element in the file is separated by a colon indicating where to tokenize the lines. Once the libraries are updated with the already booked appointments, the server program enters its main function to initialize the server process. Like the client program, the server initialization begins with HOST and PORT declaration. Then a server object is created by passing the server attributes and handler into the ThreadedTCPServer class object. Then the server object is then passed into the thread class and told to serve forever. This is the command that gives the server its multithreaded properties. The server thread is then started and a confirmation message is printed to the console. From there the handler listens for incoming requests from clients. When a client attempts to connect to the server, a new thread is spawned and the thread is assigned a new port for TCP communications between the server and the client and the thread is passed into the ThreadedTCPRequestHandler class. The handler always runs the function setup(self) first which prints the client information and the thread name that the client is running on. Setup also initializes the synchronization lock for the critical section of the scheduling. Once the setup function has finished the main body of the handler, handle, is executed. In the function handle, the global libraries are declared and then enters the while true loop that the server will run in until the client is finished. The beginning of the while loop always starts with a receive instruction because the protocol number that is passed in by the client determines where the server code goes. If protocol number ‘1’ is received it means that it’s the first communication from the client and the rest of the data received will be the user’s personally identifiable information. The server then sends back the list of available providers and begins listening to a response from the client again. Once protocol number ‘2’ is received, the server knows that the data contained from the client will be the selection of which provider they would like to schedule an appointment with. Each of the available providers is assigned a number matching the number that’s printed onto the screen when the provider information is displayed to the client’s user. Once the correct block is found the sendaval() function is called, passing the name of the dictionary containing the provider’s appointment availability. This function looks through all the values in the dictionary and whenever a -1 is found, the corresponding key is concatenated to the string of available times, separated by whitespace. The string of available times is then returned to the handle function and sent to the waiting client. The server handler then waits for a response from the client with the selected time. Once a response is received from the client, the schedule function is called, passing the dictionary of the provider, the data just received from the client, the user info collected at the beginning of the thread’s execution, and the provider’s associated ID number for I/O. The schedule function contains the critical section of the code. The response code is defaulted to ‘-1’, meaning the appointment was not scheduled. Then the lock that was initiated in the handler’s self-function is called to acquire the lock if it is free. Once the lock is acquired no other threads can enter the following section until the thread with the lock has completed. The first thing the critical section checks is that the client sent a valid available time. Even though the client only sees available times, the user can still enter a time that’s not listed and must be checked. Also, if two threads reach the critical section at the same time requesting the same time from the same provider, both clients saw this time as a valid time and we must avoid double booking and overwriting other appointments. If the value with the associated key is not ‘-1’ then the unchanged schedule is returned along with the default response code of -1. If the time is free (value of -1) then the dictionary is updated with the selected time as the key and the user’s info as the value. Then the file is open and the new appointment is written on a new line of the file. The file is then closed and a confirmation is printed to the console along with the updated dictionary. The response code is then changed from ‘-1’ to ‘200’ , the lock is released and the dictionary and response code are returned to the handler. The critical section is placed in a try finally block to ensure that the lock gets released regardless of the outcome of the critical section and prevent deadlock situations. Then in the handler the request code, user info and time selected are sent back to the client regardless of the outcome of the critical section and the client handles the situation depending on the request code value. Once this last communication is done, the thread breaks out of the while true loop and the socket is closed and the thread resources are released.

**Testing:**

Testing the functionality of the critical section was done with the debugger in PyCharm. I set the stop marker on the line just prior to the command to acquire the lock in the debugger and then ran two separate clients under normal execution requesting the same provider at the same time. The server stops both threads at the line before the lock acquire and when I let the server continue to run, one client got a confirmation and the other got a denial, showing that the lock worked correctly.



Stops the thread execution right before the call to aquire the lock so that the threads are coming in at almost the same time.

3. Client 2 requesting provider 2 at 13:00 waiting on server response

2. Client 1 requesting provider 2 at 13:00 waiting on server response

5. Since client 2 aquired lock first and scheduled this appointment, when client 1’s thread aquired the lock the time was no longer availible and returned -1

4. Client 2 Successfully scheduled showing that it aquire the lock first

1. Debugger: Server with 2 clients connected waiting to enter critical section