

Assignment Report

Content:

1. [Language requirement & environment setup](#)
2. [Program design](#)
3. [Data structure design](#)
4. [Application layer message format](#)
5. [How system work](#)

Part 01: Language requirement and environment setup

Language	Python
Version	3.8.10
Libraries Requirement	Socket, threading, pickle, datetime, time, sys, os

Part 02: Program design

There are 2 main sections in this assignment design --- Server and Client.

- 1) Server
The server starts a new ClientThread when there is a user login. In the ClientThread, there defines many self-functions in format "process_*****()", which can process different client command and cases. In the run() of ClientThread, there is a while loop which can continuously receive and send message with the client.
- 2) Client
The Client has 4 different classes – (PrivateSendThread, PrivateReceiveThread, SendThread, ReceiveThread). The SendThread and ReceiveThread are the main part which process all the user input and server reply. For the p2p section, Client will start up with its own "PrivateSendThread" and "PrivateReceiveThread", and extra socket connected to the target client. This can allow the client to send-receive to all the private tunnels and also to the server.

Part03: Data Structure design

- Client:
 1. Global variables

Variable Name	Type	Description
message_send	Message (defined in Message.py)	save message that need to send via Client socket
message_received	Message (defined in Message.py)	save message that received by Client socket
confirm_wait	Boolean	True: when there is requester asking for private messaging. False: when there is no private requester
requester_list	List of string	list of private messaging requester
username	String	Client username
password	String	Client password
private_send_thread_list	List of PrivateSendThread (defined in TCPClient3.py)	Send_Threads for all private messaging tunnel
private_receive_thread_list	List of PrivateReceiveThread (defined in TCPClient3.py)	Receive_Threads for all private messaging tunnel

- Server
 1. Global variables

Variable Name	Type	Description
UserData	Dictionary format: {user: passwd,}	Load data from <i>credential.txt</i> , update when new user login.
OnLine_list	List of tuples format: [(user, login_time),]	Data of online_users, update when login or logout
Log_history	List of tuples format: [(user, login_time),]	Data of user log history, only update when login.
login_blocked_list	List of tuples format: [(user, TimeThread),]	Information of blocked accounts and relevant Timers.
clientThread_list	List of ClientThread	Store all the ClientThreads
blocker_list	Dictionary format: {user: [blocked_users],}	The unidirectional relationship of blocking
stored_message_list	Dictionary format: {user: [messages],}	All stored messages and their target users

(Note: for the local variables, please check the relevant code, the variable names can explain its role)

Part 04: Application layer message format

```
class MessageType(Enum):
    ... NOCOMMAND = -1
    ... ARGUMENT_ERROR = -2
    ...
    ... LOGIN_USERNAME = 0.1
    ... LOGIN_PASSWD = 0.2
    ... LOGIN_NEWPASSWD = 0.3
    ...
    ... MESSAGE = 1
    ... BROADCAST = 2
    ... WHOELSE = 3
    ... WHOELSEINCE = 4
    ... BLOCK = 5
    ... UNBLOCK = 6
    ... LOGOUT = 7
    ...
    ... STARTPRIVATE = 8
    ... TELL_TARGET_USER_SETUP_PRIVATE_SENDERTHREAD = 8.1
    ... PRIVATE = 9
    ... STOPPRIVATE = 10
    ... STOPPRIVATE_PRIVATE_STOPPED = 10.1
    ...
    ... YES = 11
    ... NO = 12
```

```
class ServerMessageType(Enum):
    ... ERROR = -100
    ...
    ... REQUEST_NEWUSER = 101
    ... REQUEST_NEEDPASSWORD = 102
    ... ANNOUNCEMENT = 103
    ... TIMEOUT = 104
    ... ACCOUNT_BLOCK = 105
    ...
    ... ASK_FOR_PRIVATE_CONNECTION = 106
    ... SEND_REQUESTER_SOCKET_ADDRESS = 107
    ... SETUP_PRIVATE_SENDERTHREAD = 108
    ...
    ... STOP_AND_DELETE_PRIVATE = 109
    ... STOPPED_PRIVATE_CONFIRM = 110
    ...
    ... LOGOUT_ANNOUNCEMENT = 111
```

```
class Message(object):
    ... def __init__(self, content, message_type):
    ...     self.content = content
    ...     self.type = message_type
    ...
    ... def getContent(self):
    ...     return self.content
    ...
    ... def getType(self):
    ...     return self.type
```

```
class ServerMessage(object):
    ... def __init__(self, content, reply_type):
    ...     self.content = content
    ...     self.type = reply_type
    ...
    ... def getContent(self):
    ...     return self.content
    ...
    ... def getType(self):
    ...     return self.type
```

For the Message Types of Clients and Server, They are all defined in the Message.py
As the diagrams shown, there are 19 MessageType and 12 ServerMessageType. The Message class are the packet that send from Client to Server, and ServerMessage class is the packet that send from Server to Client. Both of them have two self-functions to get the message content and message type.

Part 05: How the system work

Client:

● Non-P2P Command:

In the client, the user input will be processed by the main SendThread, and the server reply will be processed by the main ReceiveThread. All the command and replies will be classified based on its MessageType or ServerMessageType, and then process in different if-statement cases.

```
class ReceiveThread(Thread):
    ... def __init__(self, clientSocket:socket): ...
    ...
    ... def run(self):
    ...     global message_received
    ...     global username
    ...     global password
    ...     global confirm_wait
    ...
    ...     while self.Alive:
    ...         data = self.clientSocket.recv(1024)
    ...         message_received = pickle.loads(data)
    ...
    ...         if message_received.getType() == ServerMessageType.ANNOUNCEMENT: ...
    ...         elif message_received.getType() == ServerMessageType.LOGOUT_ANNOUNCEMENT: ...
    ...         elif message_received.getType() == ServerMessageType.TIMEOUT: ...
    ...         elif message_received.getType() == ServerMessageType.ERROR: ...
    ...         elif message_received.getType() == ServerMessageType.ASK_FOR_PRIVATE_CONNECTION: ...
    ...         elif message_received.getType() == ServerMessageType.SEND_REQUESTER_SOCKET_ADDRESS: ...
    ...         elif message_received.getType() == ServerMessageType.SETUP_PRIVATE_SENDERTHREAD: ...
    ...         elif message_received.getType() == ServerMessageType.STOP_AND_DELETE_PRIVATE: ...
    ...         elif message_received.getType() == ServerMessageType.STOPPED_PRIVATE_CONFIRM: ...
```

```
class SendThread(Thread):
    ... def __init__(self, clientSocket:socket): ...
    ...
    ... def run(self):
    ...     global message_send
    ...     global message_received
    ...     global username
    ...     global password
    ...     global confirm_wait
    ...     global requester_list
    ...     global confirm_wait
    ...     global requester_list
    ...     global confirm_wait
    ...     global requester_list
    ...
    ...     while self.Alive:
    ...         message_send = input()
    ...         (self.message_content, self.message_type) = MessageContentByType(message_send)
    ...
    ...         if confirm_wait == True and requester_list != []: ...
    ...         else:
    ...             if self.message_type == MessageType.NOCOMMAND or self.message_type == MessageType.YES or self.message_type == MessageType.NO: ...
    ...             elif self.message_type == MessageType.MESSAGE: ...
    ...             elif self.message_type == MessageType.BROADCAST: ...
    ...             elif self.message_type == MessageType.WHOELSE: ...
    ...             elif self.message_type == MessageType.WHOELSEINCE: ...
    ...             elif self.message_type == MessageType.BLOCK: ...
    ...             elif self.message_type == MessageType.UNBLOCK: ...
    ...             elif self.message_type == MessageType.LOGOUT: ...
    ...             elif self.message_type == MessageType.STARTPRIVATE: ...
    ...             elif self.message_type == MessageType.PRIVATE: ...
    ...             elif self.message_type == MessageType.STOPPRIVATE: ...
    ...             elif self.message_type == MessageType.STOPPED_PRIVATE_CONFIRM: ...
    ...             elif self.message_type == MessageType.ARGUMENT_ERROR: ...
```

● P2P Command:

For the p2p commands, once the SendThread get user input as a p2p command, it will look through the private_thread_lists and process this command with the relevant PrivateThreads or send it to the server to ask target client making private messaging tunnel. The PrivateSendThread take the role of private message sending, and the PrivateReceiveThread take the role of private message receiving.

```
class PrivateReceiveThread(Thread):
    ... def __init__(self, private_socket:socket): ...
    ...
    ... def run(self):
    ...     self.private_socket.listen()
    ...     self.target_socket, self.target_address = self.private_socket.accept()
    ...
    ...     self.message = ''
    ...     while True:
    ...         try:
    ...             data = self.target_socket.recv(1024)
    ...             self.message = pickle.loads(data)
    ...         except Exception:
    ...             break
    ...         print(f"[private] {username} : {self.message.getContent()}")
    ...
    ...     def getSocket(self):
    ...         return self.private_socket
    ...
    ...     def endSocket(self):
    ...         self.private_socket.close()
```

```
class PrivateSendThread(Thread):
    ... def __init__(self, private_socket:socket): ...
    ...
    ... def run(self): ...
    ...
    ... def sendmessage(self, message:str):
    ...     self.private_socket.sendall(pickle.dumps(message))
    ...
    ... def getSocket(self):
    ...     return self.private_socket
    ...
    ... def endSocket(self):
    ...     self.private_socket.close()
```

For the server, as the diagrams shown, similar to what happened in client, the server will receive the client message and then process via different MessageTypes. And there is no process function for the “private” command cause the private message go through the private socket in client, while for “startprivate” and “stopprivate” commands, the server will help the message forwarding and communication between two clients.

```
class for client
used to define the instance for each connection from each client

203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

```

def __init__(self, client):
    """
    This class would be used to define the instance for each connection from each client
    For example, client-1 makes a connection request to the server, the server will call
    class (ClientThread) to define a thread for client-1, and when client-2 make a connection
    request to the server, the server will call class (ClientThread) again and create a thread
    for client-2, each client will be running in a separate thread, which is the multi-threading
    """
    # You 7 hours ago 2 authors (You and others)
    (class ClientThread(Thread)):

    def __init__(self, clientAddress, clientSocket):

    def run(self):
        global UserData, Online list, log_History, login_blocked_list, clientThread_list, blocker_list
        # 1. Get the user's login info, 2 days ago * 86400 * 24
        self.message = clientSocket.recv(1024)
        while self.clientAlive:
            # use recv() to receive message from the client
            try:
                except timeout:
            except IOError or ConnectionResetError:
                self.message_content = self.message.getcontent()
                self.message_type = self.message.getType()

                # before process everything, remove all unlocked user
                for (user, timer) in login_blocked_list[:]:
                    if timer.is_alive() == False:
                        login_blocked_list.remove((user, timer))

                if self.message_type == MessageType.LOGIN_USERNAME:
                    self.message_type = MessageType.LOGIN_PASSWORD
                elif self.message_type == MessageType.LOGIN_PASSWORD:
                    self.message_type = MessageType.MESSAGE
                elif self.message_type == MessageType.MESSAGE:
                    self.message_type = MessageType.BROADCAST
                elif self.message_type == MessageType.ANNOUNCE:
                    self.message_type = MessageType.ANNOUNCE
                elif self.message_type == MessageType.ANNOUNCE:
                    self.message_type = MessageType.BLOCK
                elif self.message_type == MessageType.ANNOUNCE:
                    self.message_type = MessageType.ANNOUNCE
                elif self.message_type == MessageType.ANNOUNCE:
                    self.message_type = MessageType.STARTPRIVATE
                elif self.message_type == MessageType.STARTPRIVATE:
                    self.message_type = MessageType.STOPPRIVATE
                elif self.message_type == MessageType.STOPPRIVATE_PRIVATE_STOPPED:
                    self.message_type = MessageType.RESET
                elif self.message_type == MessageType.RESET:
                    self.message_type = MessageType.RESET
                elif self.message_type == MessageType.RESET_TARGET_USER_SETUP_PRIVATE_SENDPRIVATE:

```

```

801
802     You can create more customized APIs here, e.g., logic for processing user authentication
803     Each api can be used to handle one specific function, for example:
804     def process_login(self):
805         message = "user credentials request"
806         self.clientSocket.send(message.encode())
807
808     """
809     You: 2 days ago • 11:10 rebuild
810
811 def process_login(self):--
812
813
814 def process_login_outsider(self):--
815
816
817 def process_login_newuser(self):--
818
819
820 def process_login_success(self):--
821
822 ===== Welcome to the greatest messaging application ever! =====
823
824
825
826
827 def process_login_broadcast(self):--
828
829
830 def process_login_couldlocked(self, checked_user):--
831
832
833 def process_message(self):--
834
835
836 def process_broadcast(self):--
837
838
839 def process_whoolse(self):--
840
841
842 def process_whoolseince(self):--
843
844
845 def process_block(self):--
846
847
848 def process_unblock(self):--
849
850
851 def process_logout(self):--
852
853
854 def process_logout_broadcast(self):--
855
856
857 def process_timeout(self):--
858
859
860 def process_startprivate(self):--
861
862
863 def process_stopprivate(self):--
864
865
866 def process_stopped_private.feedback(self):--
867

```

```

$ python TCPServer3.py 4000 60 60
----- Server is running -----
----- Waiting for connection request from clients.-----
----- New connection created for: ('127.0.0.1', 50240)
[recv] User already exists!! Check password!!
[recv] password = zxcvb
[check] Password Correct.
[check] Successfully login time - 2021-11-18 22:12:09
----- New connection created for: ('127.0.0.1', 50242)
[recv] User already exists!! Check password!!
[recv] password = zxcvb
[check] Password Correct.
[check] Successfully login time - 2021-11-18 22:12:21
[Message]: send '3311 is good' from 'yst001' to 'yst'
[recv] received startprivate message from yst
[request] yst Request yst001 for private connection.
[Request] Request has been refused!
[recv] received startprivate message from yst
[request] yst Request yst001 for private connection.
[Request] Request has been confirmed!
[broadcast] yst is broadcasting 3311 is also good
[broadcast] 'hans' is currently offline
[broadcast] 'yoda' is currently offline
[broadcast] 'wader' is currently offline
[broadcast] 'r2d2' is currently offline
[broadcast] 'c3p0' is currently offline
[broadcast] 'leia' is currently offline
[broadcast] 'obiban' is currently offline
[broadcast] 'luke' is currently offline
[broadcast] 'chevy' is currently offline
[broadcast] 'palpatine' is currently offline
[broadcast] yst001 is currently online
[broadcast] 'yst002' is currently offline
[broadcast] 'yst003' is currently offline
[broadcast] 'yst01' is currently offline
[store] Storing message
----- yst disconnected - ('127.0.0.1', 50240)
----- Logout time - 2021-11-18 22:13:23
----- yst001 disconnected - ('127.0.0.1', 50242)
----- Logout time - 2021-11-18 22:13:25

```