

Proyecto 2

Sockets

Estudiantes: Samuel Giordano M.
Luciano Mora S.

Profesor: César Azurdia M.

Auxiliares: Alejandro Cuevas
Nicolás Ortega
Pablo Ortega
Sandy Bolufe

Ayudantes: Claudio Urbina
Jean Cherubini

Fecha de realización: 6 de abril de 2018

Fecha de entrega: 27 de octubre de 2018

Santiago, Chile

Índice de Contenidos

1. Introducción	1
2. Marco Teórico	2
3. Descripción del servidor	5
3.1. Servidor	5
3.1.1. Iniciación del puerto de conexión	5
3.1.2. Integración de los clientes nuevos al chat	5
3.1.3. Recepción de mensajes y comandos	5
3.2. Cliente	6
3.2.1. Conexión al puerto del Chat	6
3.2.2. Envío y recepción de mensajes	6
4. Discusión y conclusiones	7
Anexo A. Programa del servidor	11
Anexo B. Programa del Cliente	14
Referencias	16

Lista de Figuras

1	Movimiento de la información desde la capa de aplicación remitente hasta el sistema principal destinatario.	2
2	Funcionamiento del Socket desde el punto de vista del servidor.	3
3	Funcionamiento del Socket desde el punto de vista del cliente.	4
4	Inicio del servidor.	7
5	Inicio del cliente.	7
6	Conexión del servidor.	7
7	Conexión del cliente.	7
8	Mensajes del servidor.	8
9	Mensajes del cliente.	8
10	Comandos del servidor	8
11	Comandos del cliente	9
12	Mensaje privado cliente-emisor.	9
13	Mensaje privado Cliente-receptor.	9

Lista de Códigos

A.1.	Código fuente del servidor.	11
B.1.	Código fuente del servidor.	14

1. Introducción

Hoy en día se podrían mencionar varios conceptos que han revolucionado la ciencia, la ingeniería y la vida de las personas en las sociedades. Uno de ellos son las llamadas Tecnologías de Información y Comunicación (TICs por sus siglas en Inglés).

Es tanto el impacto que han tenido las TICs que algunos autores afirman incluso que se trata de una revolución social, evolucionando la sociedad a una nueva “sociedad de la información” que eventualmente generará nuevos puestos de trabajo, readaptando la estructura laboral actual, etc [1]. Independiente de las valoraciones que se puedan hacer de estas afirmaciones y de las críticas que los autores de este reporte pueden tener, lo cierto es que esto es un objeto de debate y es menester detenerse a estudiar uno de los fenómenos más cotidianos y de vanguardia tecnológica que existe actualmente.

En términos de las definiciones, en [2] se definen las TICs como “ las que giran en torno a tres medios básicos: la informática, la microelectrónica y las telecomunicaciones; pero giran, no sólo de forma aislada, sino lo que es más significativo de manera interactiva e interconexionadas, lo que permite conseguir nuevas realidades comunicativas”. Tal y como se menciona la interconexión de diferentes áreas permite la concepción de nuevas realidades de comunicación. Los servicios de mensajería instantánea, el internet y recientemente el internet de las cosas forman parte de este paradigma.

En el siguiente trabajo se hace un análisis de sockets y además se implementa un chat, una de las aplicaciones más conocidas de los sockets.

Los resultados y la implementación pueden encontrarse en los archivos fuente que pueden ser encontrados en los anexos.

2. Marco Teórico

Una forma básica de transmitir información desde un ordenador a otro es la conexión física por cable. Sin embargo, cuando se tiene un conjunto de ordenadores conectados entre sí se debe definir una “forma de comunicarse” entre sí. En [3] se hace la analogía con un grupo de personas que se pone de acuerdo para hacer un debate en francés, inglés o español. Dicha forma de comunicación entre máquinas se denomina **protocolo de comunicación**. En [4] se define un protocolo como “conjuntos de normas para formatos de mensaje y procedimientos que permiten a las máquinas y los programas de aplicación intercambiar información”.

El protocolo utilizado por el internet es TCP/IP y este define de forma minuciosa como se mueve el mensaje desde el remitente hasta el destinatario. Para hacerlo efectivo se define un modelo de capas que van modificando la estructura de los datos para el envío de la información. En la figura siguiente se muestra un esquema que muestra el modelo de capas del protocolo:

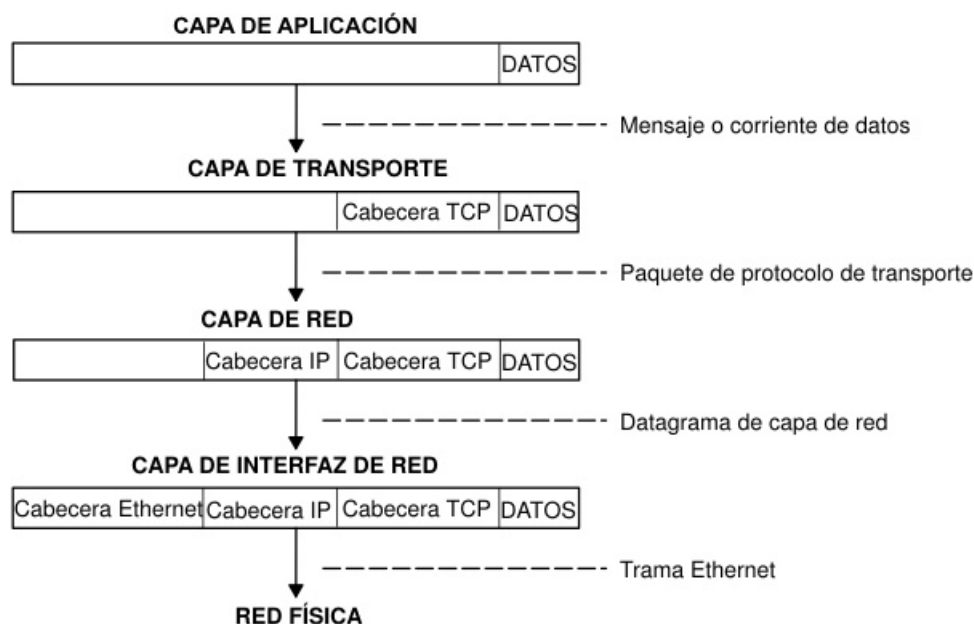


Figura 1: Movimiento de la información desde la capa de aplicación remitente hasta el sistema principal destinatario.

Una de las tantas formas de transmitir datos entre dos programas es la programación de sockets. Como se define en [3] un socket “es un canal de comunicación entre dos programas que corren en ordenadores diferentes o en el mismo ordenador”. Además, desde el punto de vista de la programación corresponde a un fichero (dato almacenado en algún recurso de memoria) que es abierto de una forma especial. Una vez dentro del fichero, se puede agregar caracteres a través de funciones estándar de algún lenguaje de programación. De esta forma, se tiene que el socket es un extremo dentro de una conexión punto a punto entre dos programas.

Los sockets funcionan de la siguiente forma:

- En primer lugar, se crea un socket con una dirección de puerto conocida en un servidor de una máquina (un Host local, por ejemplo).

- Luego, se espera que un cliente haga una petición al servidor donde se ha creado el socket, donde el cliente conoce el nombre de la máquina con la cual se quiere conectar y la dirección del socket.
- Si se logra entablar conexión con el servidor, se crea un nuevo socket con el fin de atender las peticiones de otros clientes que quieran hacer peticiones, mientras en paralelo se recogen las peticiones del cliente conectado. Al mismo tiempo, se crea un socket en la máquina del cliente con el fin de que pueda agregar los mensajes correspondientes que se quieren hacer llegar al servidor.
- Finalmente, se realiza la transmisión de los mensajes utilizando el protocolo TCP/IP, que fue descrito de forma breve anteriormente.

En las imágenes de a continuación se resume el funcionamiento de los Sockets.

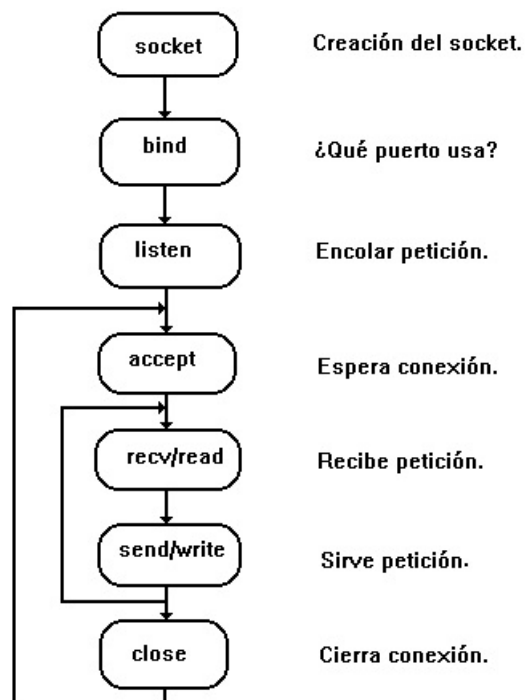


Figura 2: Funcionamiento del Socket desde el punto de vista del servidor.

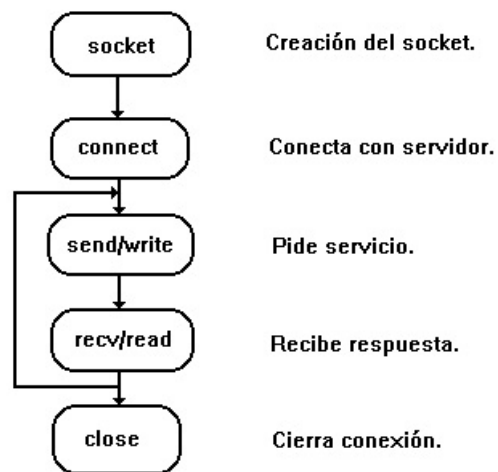


Figura 3: Funcionamiento del Socket desde el punto de vista del cliente.

3. Descripción del servidor

El chat fue implementado usando la librería `sockets` de `python`, donde se programó un código para el servidor y otro para el o los clientes. A continuación, se explica el funcionamiento de ambos códigos.

3.1. Servidor

3.1.1. Iniciación del puerto de conexión

En primer lugar, se debe definir el puerto o socket por el cual se establecerán los intercambios de información entre el servidor y los clientes. Por ende, en el código se definió el nombre del host y el puerto por el cual los clientes podrán conectarse al chat. El programa muestra en pantalla el nombre del host y el puerto donde espera conexiones.

3.1.2. Integración de los clientes nuevos al chat

Luego de activar el socket de intercambio, se definió una iteración que ejecuta las acciones necesarias para que el chat funcione. Con ese objetivo, antes de la iteración se crea un grupo de listas (`SOCKET_LIST`, `Identif` y `Nicknames`) la cuales tienen como objetivo tener un registro de los participantes del chat y sus respectivos nombres e identificadores. Se desea que el servidor sea capaz de detectar paralelamente cuando haya un nuevo cliente que quiera hacerse parte del chat y también cuando alguno de los clientes conectados haya enviado un mensaje, todo de forma ininterrumpida. Para ello se implementa la función `select.select`, que permite delegar el mantenimiento del estado del socket al sistema operativo, para que el servidor pueda realizar ambas acciones a medida que sea necesario. Es decir, iterativamente el programa define si es que alguno de los participantes del chat, contenidos en la lista `SOCKET_LIST` ha realizado alguna acción. Por lo tanto, cuando el servidor recibe la solicitud de algún cliente para integrarse al chat, el programa incluye al socket del servidor en la lista `readers` y como consecuencia procede a aceptar al nuevo cliente, darle la bienvenida y a guardar sus datos en las listas previamente mencionadas. El servidor también envía un aviso en caso de que el nombre de usuario escogido por el cliente ya esté en uso y no lo incluye hasta que el cliente envíe un nombre en desuso.

3.1.3. Recepción de mensajes y comandos

Paralelamente, cuando se detecta un mensaje de parte de alguno de los clientes hacia el servidor, el programa incluye al socket de este cliente en la lista `readers` y en consecuencia el programa llama a recibir y analizar los mensajes recibidos. El programa se encarga de identificar los datos del cliente de origen del mensaje y imprime el mensaje.

Como una excepción en la recepción de mensajes, si es que el mensaje recibido por parte de alguno de los clientes corresponde a un comando, el programa detecta el comando y realiza la acción asociado a este. Se define la función `transmitir`, la cual se ejecuta cuando alguno de los clientes utiliza el comando “:p” asociado al envío de mensajes privados, encaminando el mensaje al destinatario sin imprimirlo en la consola del servidor.

3.2. Cliente

3.2.1. Conexión al puerto del Chat

En primer lugar, el programa del cliente solicita acceso al puerto del chat, lo que activa la recepción del saludo de parte del servidor. Luego de esto, el programa solicita la elección de un nombre de usuario, el cual es aprobado por el servidor siempre y cuando no esté en uso y luego el programa pasa a la iteración de la conversación.

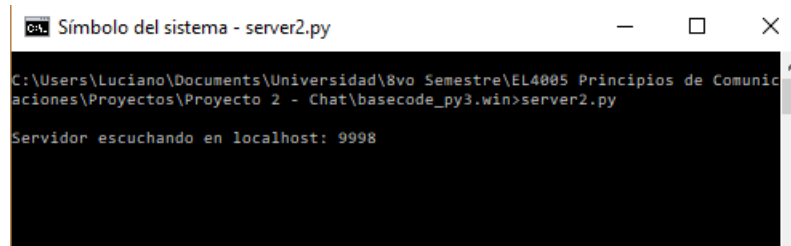
3.2.2. Envío y recepción de mensajes

Ya en la iteración, lo ideal es que el programa permita de forma simultánea, el envío y la recepción de mensajes que llegan desde el servidor. Para ello, se vuelve a implementar la herramienta `select.select` que, como ya se mencionó, delega el mantenimiento del estado del socket al sistema operativo y permite que el cliente imprima los mensajes recibidos y permita escribir un mensaje en la consola independientemente.

Por último, es preciso mencionar que el tipo de Socket utilizado es orientado a la conexión debido a que es una aplicación de chat en la cual varios clientes deben conectarse a un servidor con el fin de hacer peticiones.

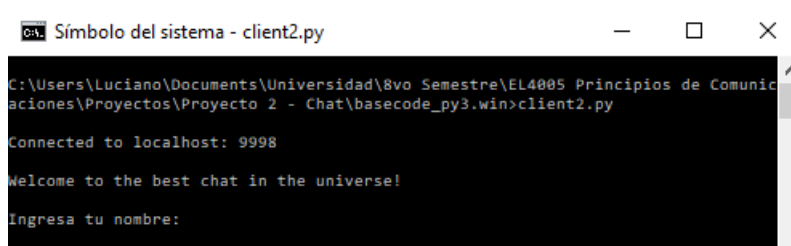
4. Discusión y conclusiones

A continuación, se muestran las pruebas hechas a la implementación del chat.



```
Símbolo del sistema - server2.py
C:\Users\Luciano\Documents\Universidad\8vo Semestre\EL4005 Principios de Comunicaciones\Proyectos\Proyecto 2 - Chat\basecode_py3.win>server2.py
Servidor escuchando en localhost: 9998
```

Figura 4: Inicio del servidor.



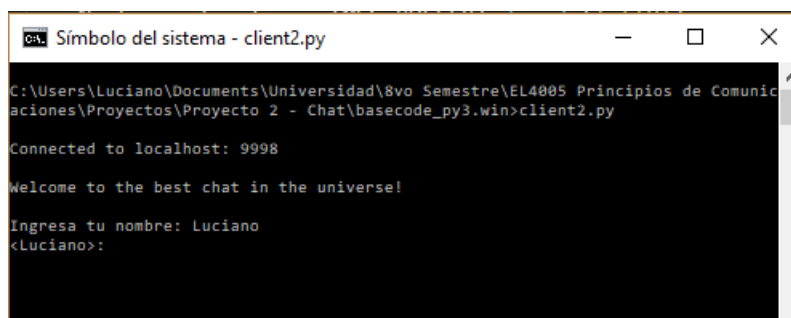
```
Símbolo del sistema - client2.py
C:\Users\Luciano\Documents\Universidad\8vo Semestre\EL4005 Principios de Comunicaciones\Proyectos\Proyecto 2 - Chat\basecode_py3.win>client2.py
Connected to localhost: 9998
Welcome to the best chat in the universe!
Ingresa tu nombre:
```

Figura 5: Inicio del cliente.



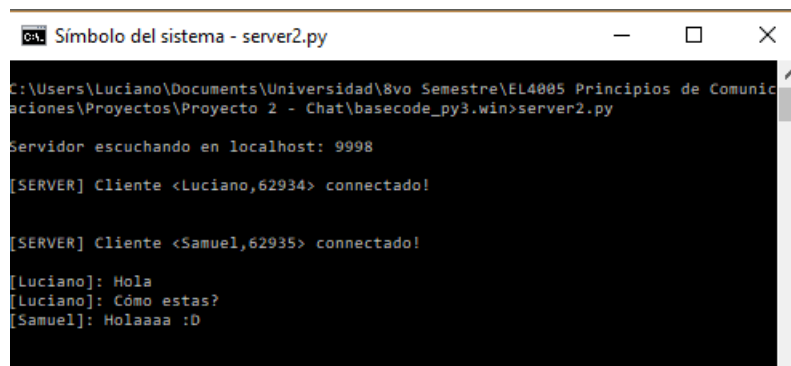
```
Símbolo del sistema - server2.py
C:\Users\Luciano\Documents\Universidad\8vo Semestre\EL4005 Principios de Comunicaciones\Proyectos\Proyecto 2 - Chat\basecode_py3.win>server2.py
Servidor escuchando en localhost: 9998
[SERVER] Cliente <Luciano,62934> conectado!
[SERVER] Cliente <Samuel,62935> conectado!
```

Figura 6: Conexión del servidor.



```
Símbolo del sistema - client2.py
C:\Users\Luciano\Documents\Universidad\8vo Semestre\EL4005 Principios de Comunicaciones\Proyectos\Proyecto 2 - Chat\basecode_py3.win>client2.py
Connected to localhost: 9998
Welcome to the best chat in the universe!
Ingresa tu nombre: Luciano
<Luciano>
```

Figura 7: Conexión del cliente.



```
Símbolo del sistema - server2.py
C:\Users\Luciano\Documents\Universidad\8vo Semestre\EL4005 Principios de Comunicaciones\Proyectos\Proyecto 2 - Chat\basecode_py3.win>server2.py

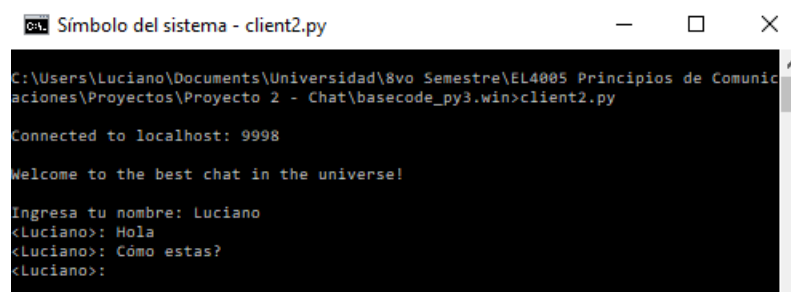
Servidor escuchando en localhost: 9998

[SERVER] Cliente <Luciano,62934> conectado!

[SERVER] Cliente <Samuel,62935> conectado!

[Luciano]: Hola
[Luciano]: Cómo estas?
[Samuel]: Holaaaa :D
```

Figura 8: Mensajes del servidor.



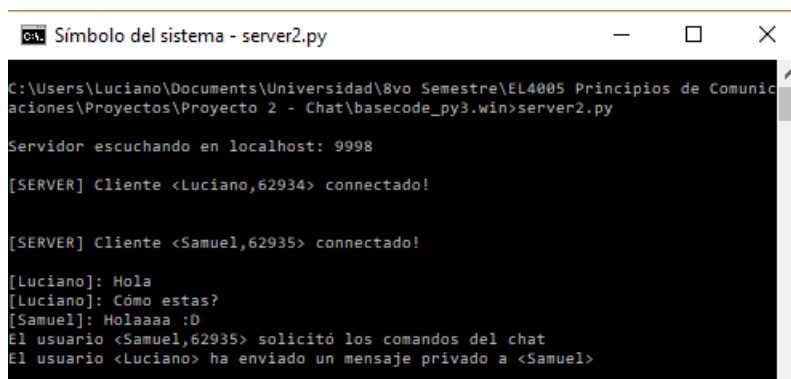
```
Símbolo del sistema - client2.py
C:\Users\Luciano\Documents\Universidad\8vo Semestre\EL4005 Principios de Comunicaciones\Proyectos\Proyecto 2 - Chat\basecode_py3.win>client2.py

Connected to localhost: 9998

Welcome to the best chat in the universe!

Ingresa tu nombre: Luciano
<Luciano>: Hola
<Luciano>: Cómo estas?
<Luciano>:
```

Figura 9: Mensajes del cliente.



```
Símbolo del sistema - server2.py
C:\Users\Luciano\Documents\Universidad\8vo Semestre\EL4005 Principios de Comunicaciones\Proyectos\Proyecto 2 - Chat\basecode_py3.win>server2.py

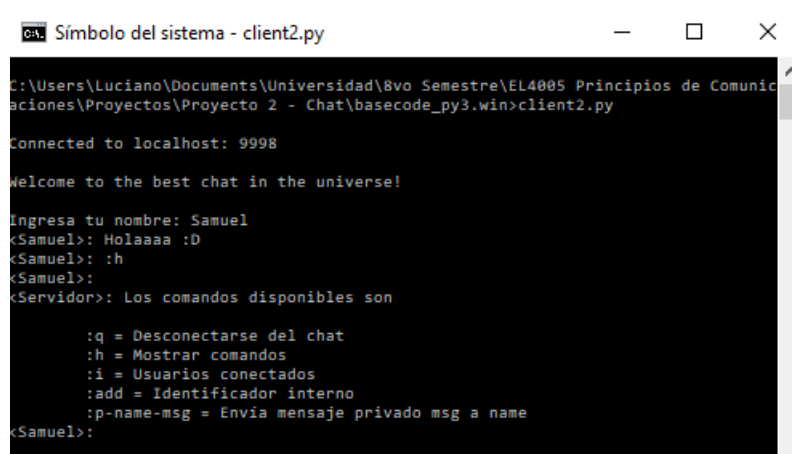
Servidor escuchando en localhost: 9998

[SERVER] Cliente <Luciano,62934> conectado!

[SERVER] Cliente <Samuel,62935> conectado!

[Luciano]: Hola
[Luciano]: Cómo estas?
[Samuel]: Holaaaa :D
El usuario <Samuel,62935> solicitó los comandos del chat
El usuario <Luciano> ha enviado un mensaje privado a <Samuel>
```

Figura 10: Comandos del servidor



```
Símbolo del sistema - client2.py
C:\Users\Luciano\Documents\Universidad\8vo Semestre\EL4005 Principios de Comunicaciones\Proyectos\Proyecto 2 - Chat\basecode_py3.win>client2.py

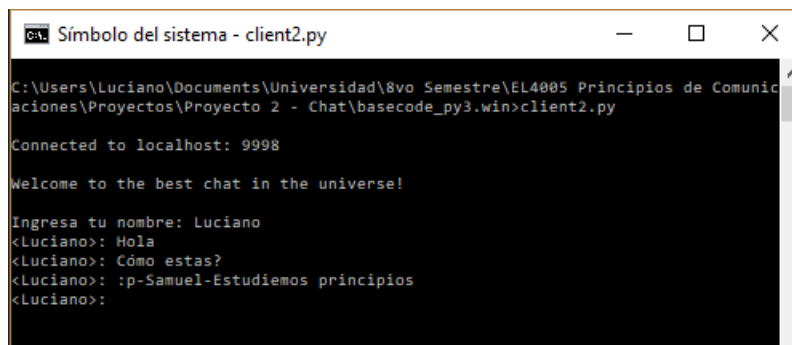
Connected to localhost: 9998

Welcome to the best chat in the universe!

Ingresa tu nombre: Samuel
<Samuel>: Holaaaa :D
<Samuel>: :h
<Samuel>:
<Servidor>: Los comandos disponibles son

      :q = Desconectarse del chat
      :h = Mostrar comandos
      :i = Usuarios conectados
      :add = Identificador interno
      :p-name-msg = Envía mensaje privado msg a name
<Samuel>:
```

Figura 11: Comandos del cliente



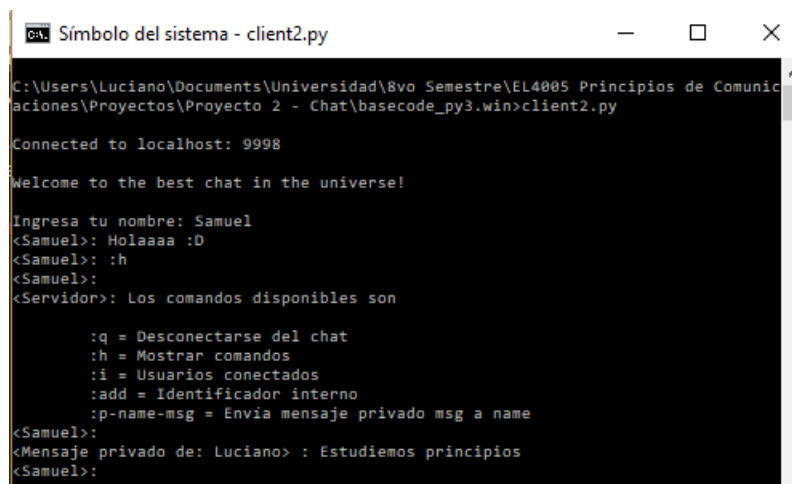
```
Símbolo del sistema - client2.py
C:\Users\Luciano\Documents\Universidad\8vo Semestre\EL4005 Principios de Comunicaciones\Proyectos\Proyecto 2 - Chat\basecode_py3.win>client2.py

Connected to localhost: 9998

Welcome to the best chat in the universe!

Ingresa tu nombre: Luciano
<Luciano>: Hola
<Luciano>: Cómo estas?
<Luciano>: :p-Samuel-Estudemos principios
<Luciano>:
```

Figura 12: Mensaje privado cliente-emisor.



```
Símbolo del sistema - client2.py
C:\Users\Luciano\Documents\Universidad\8vo Semestre\EL4005 Principios de Comunicaciones\Proyectos\Proyecto 2 - Chat\basecode_py3.win>client2.py

Connected to localhost: 9998

Welcome to the best chat in the universe!

Ingresa tu nombre: Samuel
<Samuel>: Holaaaa :D
<Samuel>: :h
<Samuel>:
<Servidor>: Los comandos disponibles son

      :q = Desconectarse del chat
      :h = Mostrar comandos
      :i = Usuarios conectados
      :add = Identificador interno
      :p-name-msg = Envía mensaje privado msg a name
<Samuel>:
<Mensaje privado de: Luciano> : Estudemos principios
<Samuel>:
```

Figura 13: Mensaje privado Cliente-receptor.

Como se observa en las imágenes, la implementación del código funciona correctamente respecto a la mayor parte de los requisitos expuestos en el enunciado del proyecto. Sin embargo, lo que no se logra es implementar correctamente la función "select".^{en} el programa desarrollado.

Como se puede apreciar en las Figuras 11 y 13, el cliente requiere que se termine de enviar un mensaje para reiniciar la iteración y recién poder recibir una respuesta o mensaje proveniente del servidor.

Para solucionar este inconveniente, se trata también de incluir un tiempo límite ("timeout") en la función "select" para reiniciarlo, sin embargo, tampoco funciona como es esperado y el problema persiste.

Dejando este detalle de lado, el programa cumple con las funcionalidades básicas de un chat.

Anexo A. Programa del servidor

Código A.1: Código fuente del servidor.

```
1  #!/usr/bin/python3
2  import socket
3  import sys
4  import select
5
6  # Message Buffer size
7  MSG_BUFFER = 1024
8
9  #Lista de sockets
10 SOCKET_LIST = []
11
12 # Obtaining the arguments using command line
13 try:
14     HOST = sys.argv[1]
15 except:
16     HOST = 'localhost'
17 try:
18     PORT = int(sys.argv[2])
19 except:
20     PORT = 9998
21
22 # Creating the client socket. AF_INET IP Family (v4)
23 # and STREAM SOCKET Type.
24 serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
25 serverSocket.bind((HOST, PORT))
26 serverSocket.listen(10)
27
28 # Incluir objeto socket del servidor a la lista de conexiones
29 SOCKET_LIST.append(serverSocket)
30
31 # lista con nombres de usuario y identificadores
32 Identif = []
33 Nicknames = []
34
35 # Función para enviar mensajes a todos los clientes
36 def transmitir (server_socket, sock, nombre, msj):
37     for socket in SOCKET_LIST:
38         # descarte de destinatarios: servidor y self
39         if socket != server_socket and socket != sock :
40             try :
41                 comp = '<'+nombre+'>: '+msj
42                 socket.send(comp.encode('utf8'))
43             except :
44                 # cierra el socket si esta inactivo
45                 socket.close()
46                 # y se remueve de la lista de sockets
47                 if socket in SOCKET_LIST:
```

```
48         SOCKET_LIST.remove(socket)
49
50 print('\nServidor escuchando en %s: %s' % (HOST, PORT))
51
52 while True:
53
54     # usamos select para delegar del mantenimiento el estado del socket al sistema
    # operativo
55     readers, _ , _ = select.select(SOCKET_LIST,[],[])
56
57     for actual in readers:
58         if actual is serverSocket:
59             # cuando la iteración pasa por el servidor revisamos las nuevas conexiones
60             sclient, addr = serverSocket.accept() # Accepting client
61
62             #sclient.setblocking(0)
63             SOCKET_LIST.append(sclient)
64
65             # mensaje de bienvenida para el nuevo cliente
66             welc = 'Welcome to the best chat in the universe!'
67             sclient.send(welc.encode('utf8'))
68
69             Nick = sclient.recv(MSG_BUFFER).decode('utf8')
70
71             while Nick in Nicknames:
72                 in_use = 'yes'
73                 sclient.send(in_use.encode('utf8'))
74                 Nick = sclient.recv(MSG_BUFFER).decode('utf8')
75
76             in_use = 'no'
77             sclient.send(in_use.encode('utf8'))
78
79             # agregamos el identificador y el nombre del nuevo cliente a las listas
    # respectivas
80             Identif.append(addr[1])
81             Nicknames.append(Nick)
82
83             print("\n[SERVER] Cliente <%s,%s> connectado!\n" % (Nick, addr[1]))
84
85         else:
86             addr = actual.getpeername() # obtenemos a la tupla que contiene al identificador
    # del cliente
87             lugar = Identif.index(addr[1]) # obtenemos el indice del identificador
88             nombre = Nicknames[lugar] # como es el mismo, usamos el indice del
    # indetificador para obtener el nombre
89
90             msg = actual.recv(MSG_BUFFER).decode('utf8')
91
92             # Si el mensaje es alguno de los comandos:
93
94             # solicitud de desconexión
95             if msg == ':q':
```

```

96         print('El usuario <%s,%s> solicitó desconectarse del chat' % (nombre, addr
[1]))
97         desp = '<Servidor>: Te has desconectado. Vuelve pronto!' # me despido
98         actual.send(desp.encode('utf8'))
99         actual.close() #cierro el socket del cliente y saco sus datos de los
registros
100         SOCKET_LIST.remove(actual)
101         Nicknames.remove(nombre)
102         Identif.remove(addr[1])
103         continue
104
105     # solicitud de comandos
106     if msg == ':h':
107         print('El usuario <%s,%s> solicitó los comandos del chat' % (nombre, addr
[1]))
108         comnd = '<Servidor>: Los comandos disponibles son\n\n          :q =
Desconectarse del chat\n          :h = Mostrar comandos\n          :i = Usuarios
conectados\n          :add = Identificador interno\n          :p-name-msg = Envía mensaje
privado msg a name'
109         actual.send(comnd.encode('utf8'))
110         continue
111
112     # solicitud lista de usuarios
113     if msg == ':i':
114         print('El usuario <%s,%s> solicitó la lista de usuarios' % (nombre, addr[1])
)
115         lista_users='User          Indentificador'
116         u=0
117         while u < len(Nicknames): #creo lista de datos para enviar
118             lista_users+=' \n' + Nicknames[u] + '          '+ str(Identif[u])
119             u+=1
120         users = '<Servidor>: \nLos usuarios disponibles son\n' + lista_users
121         actual.send(users.encode('utf8'))
122         continue
123
124     # solicitud de identificador
125     if msg == ':add':
126         print('El usuario <%s,%s> solicitó su identificador' % (nombre, addr[1]))
127         ident = 'Tu identificador es '+str(addr[1])
128         actual.send(ident.encode('utf8'))
129         continue
130
131     # mensaje privado
132     if msg[:3] == ':p-':
133         c_nom = msg[3:] # nombre-msg
134         end = c_nom.find('-') # posicion final del nombre
135         destino = c_nom[:end] # despejo nombre
136         print('El usuario <%s> ha enviado un mensaje privado a <%s>' % (nombre,
destino))
137
138         s_nom = '<Mensaje privado de: '+nombre+ '>: ' + c_nom[end+1:] # despejo y
arreglo msg

```

```
139
140     #Reviso si el usuario de destino está conectado
141     if destino in Nicknames:
142         s_destino = SOCKET_LIST[Nicknames.index(destino)+1]
143         s_destino.send(s_nom.encode('utf8'))
144
145
146     # de lo contrario notifico al emisor
147     else:
148         no_esta = 'Usuario no conectado'
149         actual.send(no_esta.encode('utf8'))
150
151
152
153     #otros mensajes
154     else:
155         print('[ ' + nombre + ']: ' + msg)
156         continue
157
158 serverSocket.close()
```

Anexo B. Programa del Cliente

Código B.1: Código fuente del servidor.

```
1 #!/usr/bin/python3
2 import sys
3 import socket
4 import select
5
6 cSocket = socket.socket()
7 try:
8     host = sys.argv[1]
9 except:
10     host = 'localhost'
11 try:
12     port = sys.argv[2]
13 except:
14     port = 9998
15
16 # Connecting
17 cSocket.connect((host, port))
18
19 socket = [cSocket]
20
21 print('\nConnected to %s: %s \n' % (host, port))
22
23 # contacto inicial
24 saludo = cSocket.recv(1024).decode('utf8')
25 print(saludo)
26 Nick = input('\nIngresa tu nombre: ')
```



```
27 cSocket.send(Nick.encode('utf8'))
28
29 # verificamos que el nick no esté en uso
30 in_use = cSocket.recv(1024).decode('utf8')
31 while in_use == 'yes':
32     Nick = input('Nombre en uso. Escoge otro nombre: ')
33     cSocket.send(Nick.encode('utf8'))
34     in_use = cSocket.recv(1024).decode('utf8')
35
36
37 while True:
38
39     # usamos select para delegar el mantenimiento del estado del socket al sistema
    operativo
40     timeout_expired = False
41     readable, writable, _ = select.select(socket, socket, [], timeout_expired)
42
43     for actual in readable:
44
45         if not readable:
46             timeout_expired = True
47         else:
48             msj_rcb = actual.recv(1024).decode('utf8')
49             print (msj_rcb)
50
51
52     for actual in writable:
53         if not writable:
54             timeout_expired = True
55
56         else:
57             message = input('<'+Nick+'>: ')
58             if message == '':
59                 continue
60             cSocket.send(message.encode('utf8'))
61
62
63 cSocket.close()
```

Referencias

- [1] C. Belloch. *Las tecnologías de la información en el aprendizaje*. Universidad de Valencia, 2015.
- [2] CABERO, J. y LOSCERTALES, F. “¿Cómo nos ven los demás? La imagen del profesor y la enseñanza en los medios de comunicación”, Sevilla, Secretariado de Publicaciones de la Universidad de Sevilla, 1998.
- [3] Javier Abellán. (2007,4 Feb). Programación de sockets en C de Unix/Linux [Online]. Available: http://www.chuidiang.org/clinux/sockets/sockets_simp.php.
- [4] IBM Knowledge Center. Protocolos TCP/IP [Online]. Aviable: https://www.ibm.com/support/knowledgecenter/es/ssw_aix_72/com.ibm.aix.networkcomm/tcpip_protocols.htm.