

Tanto en el caso de Select, como en el de Concurrencia, se hace uso de la librería “datetime”, la cual nos permite tener acceso a fechas, incluyendo la fecha, hora, minuto y segundo actual.

Código del cliente con un modelo Cliente-Servidor, con la implementación del formato Select.

```
4 import socket
5 import sys
6 import datetime
7
8 mensajes = [
9     'Este mensaje ',
10    'es enviado ',
11    'en partes.',
12    'chau'
13 ]
14 dir_servidor = ('localhost', 10000)
15
16 # Creo socket
17 socks = [
18     socket.socket(socket.AF_INET, socket.SOCK_STREAM),
19     socket.socket(socket.AF_INET, socket.SOCK_STREAM),
20 ]
21
22 # conectar el socket al puerto en el cual el servidor está escuchando
23 print('conectando a {} puerto {}'.format(*dir_servidor),
24       file=sys.stderr)
25 for s in socks:
26     s.connect(dir_servidor)
27     hora_conexion = datetime.datetime.now() #registra la hora de la conexion
28     print('conexion a las {}'.format(hora_conexion))
29 for mensaje in mensajes:
30     datos_salientes = mensaje.encode()
31     bandera = False
32     # envio mensajes en ambos sockets
33     for s in socks:
34         if (datos_salientes != b'chau'):
35             print('{}: enviando {}'.format(*args: s.getsockname(),
36                                             datos_salientes),
37                   file=sys.stderr)
38             s.send(datos_salientes)
39         else:
40             hora_cierre = datetime.datetime.now()
41             duracion = hora_cierre - hora_conexion
42             print('cerrando socket {} a las {}'.format(*args: s.getsockname(), hora_cierre),
43                   file=sys.stderr)
44             print("Duracion: {}".format(duracion))
45             bandera = True
46     s.close()
47
48 if bandera:
49     break
50
51 # leo respuestas en ambos sockets
52 for s in socks:
53     data = s.recv(1024)
54     print('{}: recibido {}'.format(*args: s.getsockname(),
55                                    data),
56           file=sys.stderr)
57     if not data:
58         print("VERIF")
59         hora_cierre = datetime.datetime.now()
60         duracion = hora_cierre - hora_conexion
61         print('cerrando socket {} a las {}'.format(*args: s.getsockname(), hora_cierre),
62               file=sys.stderr)
63         print("Duracion: {}".format(duracion))
64     s.close()
```

El Cliente funciona de la misma manera que el Cliente en base en C, crea las conexiones, a partir de las cuales se envían y reciben mensajes con el servidor. El único cambio visible se puede visualizar en la línea 26, donde cambia la forma de escribir el CONNECT().

26	s.connect(dir_servidor)
----	-------------------------

Código del Servidor:

```
2 import select
3 import socket
4 import sys
5 import queue
6 import datetime
7
8 # Creando un socket TCP/IP
9 servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10 servidor.setblocking(0)
11
12 # Hago Bind del socket al puerto
13 dir_servidor = ('localhost', 10000)
14 print('iniciando en {} port {}'.format(*dir_servidor),
15       file=sys.stderr)
16 servidor.bind(dir_servidor)
17
18 # Escucho conexiones entrantes
19 servidor.listen(5)
20
21 # Socket que espero leer
22 entradas = [servidor]
23
24 # Sockets que espero enviar
25 salidas = []
26
27 # Cola de mensajes salientes
28 cola_mensajes = {}
29
30 while entradas:
31
32     # Espero a que al menos uno de los sockets este listo para ser procesado
33
34     print('\nesperando el próximo evento', file=sys.stderr)
35     readable, writable, exceptional = select.select(entradas,
36                                                     salidas,
37                                                     entradas)
38
39     if not (readable or writable or exceptional):
40         print(' tiempo excedido...',
41               file=sys.stderr)
42         continue
```

```

43 # Manejo entradas
44 for s in readable:
45
46     if s is servidor:
47         # Un socket "leible" está listo para aceptar conexiones
48         con, dir_cliente = s.accept()
49         hora_conexion = datetime.datetime.now() #registra la hora de la conexion
50         print(' conexión desde: ', dir_cliente,
51               file=sys.stderr)
52         print("a las {}".format(hora_conexion))
53         con.setblocking(0)
54         entradas.append(con)
55
56         # Le asigno a la conexión una cola en la cuál quiero enviar
57         cola_mensajes[con] = queue.Queue()
58
59     else:
60         data = s.recv(1024)
61         if data:
62             # Un socket leible tiene datos
63             print(' recibido {!r} desde {}'.format(
64                   *args: data, s.getpeername()), file=sys.stderr,
65                   )
66             cola_mensajes[s].put(data)
67             # Agrego un canal de salida para la respuesta
68             if s not in salidas:
69                 salidas.append(s)
70         else:
71             # Si está vacío lo interpreto como una conexión a cerrar
72             hora_cierre = datetime.datetime.now()
73             duracion = hora_cierre - hora_conexion
74             print(' cerrando...{} a las {}'.format(*args: dir_cliente, hora_cierre),
75                   file=sys.stderr)
76             print("duracion: {}".format(duracion))
77             # Dejo de escuchar en la conexión
78             if s in salidas:
79                 salidas.remove(s)
80             entradas.remove(s)
81             s.close()
82
83             # Remueve mensaje de la cola
84             del cola_mensajes[s]
85
86 # Administro salidas
87 for s in writable:
88     try:
89         next_msg = cola_mensajes[s].get_nowait()
90     except queue.Empty:
91         # No hay mensaje en espera. Dejo de controlar para posibles escrituras
92
93         print(' ', s.getpeername(), 'cola vacia',
94               file=sys.stderr)
95         salidas.remove(s)
96     else:
97         print(' enviando {!r} a {}'.format(*args: next_msg, s.getpeername()), file=sys.stderr)
98         s.send(next_msg)
99
100 # Administro condiciones excepcionales"
101 for s in exceptional:
102     print('excepción en', s.getpeername(),
103           file=sys.stderr)
104     # Dejo de escuchar en las conexiones
105     entradas.remove(s)
106     if s in salidas:
107         salidas.remove(s)
108     s.close()
109
110 # Remuevo cola de mensajes
111 del cola_mensajes[s]
112

```

Por el lado del código del Servidor, cambia la forma de escribir el LISTEN ():

```

19 servidor.listen(5)

```

Incluyendo el BIND ():

```
16 servidor.bind(dir_servidor)
```

Donde la variable dir_servidor es:

```
13 dir_servidor = ('localhost', 10000)
```

Para realizar un servidor aparente utilizamos el proceso Select de la siguiente manera:

```
32 # Espero a que al menos uno de los sockets este listo para ser procesado
33
34 print('\nesperando el próximo evento', file=sys.stderr)
35 readable, writable, exceptional = select.select(entradas,
36                                                  salidas,
37                                                  entradas)
```

Donde:

- Readable: lo que tiene que leer el servidor
- Writable: lo que se enviará
- Exceptional: se utiliza para excepciones en caso de errores.

```

44  for s in readable:
45
46      if s is servidor:
47          # Un socket "leible" está listo para aceptar conexiones
48          con, dir_cliente = s.accept()
49          hora_conexion = datetime.datetime.now() #registra la hora de la conexion
50          print(' conexión desde: ', dir_cliente,
51                file=sys.stderr)
52          print("a las {}".format(hora_conexion))
53          con.setblocking(0)
54          entradas.append(con)
55
56          # Le asigno a la conexión una cola en la cuál quiero enviar
57          cola_mensajes[con] = queue.Queue()
58
59      else:
60          data = s.recv(1024)
61          if data:
62              # Un socket leible tiene datos
63              print(' recibido {!r} desde {}'.format(
64                    *args: data, s.getpeername()), file=sys.stderr,
65                    )
66              cola_mensajes[s].put(data)
67              # Agrego un canal de salida para la respuesta
68              if s not in salidas:
69                  salidas.append(s)
70          else:
71              # Si está vacío lo interpreto como una conexión a cerrar
72              hora_cierre = datetime.datetime.now()
73              duracion = hora_cierre - hora_conexion
74              print(' cerrando...{} a las {}'.format(*args: dir_cliente, hora_cierre),
75                    file=sys.stderr)
76              print("Duracion: {}".format(duracion))
77              # dejo de escuchar en la conexión
78              if s in salidas:
79                  salidas.remove(s)
80              entradas.remove(s)
81              s.close()
82
83              # Remueve mensaje de la cola
84              del cola_mensajes[s]

```

En el ciclo for se analiza cada conexión si va a ser la primera vez, o ya posee información.

En el primer if, cuando llega una conexión el servidor, guarda la hora en la que se conecto, printea la dirección y encola la conexión.

Por otro lado, en el else, cuando hay una conexión con datos los guarda y los printea, de no haber datos, asume que el cliente quiere cerrar la conexión.

```

86  for s in writable:
87      try:
88          next_msg = cola_mensajes[s].get_nowait()
89      except queue.Empty:
90          # No hay mensaje en espera. Dejo de controlar para posibles escrituras
91
92          print(' ', s.getpeername(), 'cola vacía',
93                file=sys.stderr)
94          salidas.remove(s)
95      else:
96          print(' enviando {!r} a {}'.format(*args: next_msg, s.getpeername()), file=sys.stderr)
97          s.send(next_msg)
98

```

Este ciclo for recorre las conexiones y le envia información de ser necesario.

Utilizamos un Try-Catch en caso de no querer enviar información, lo dejo de controlar para escritura. En caso contrario, se realiza el envío de información.

```
100         for s in exceptional:
101             print('excepción en', s.getpeername(),
102                   file=sys.stderr)
103             # Dejo de escuchar en las conexiones
104             entradas.remove(s)
105             if s in salidas:
106                 salidas.remove(s)
107             s.close()
108
109             # Remuevo cola de mensajes
110             del cola_mensajes[s]
111
112
```

Este ciclo permite controlar si hay errores en las conexiones, y en caso de haberlo, que sean cerradas.

Salida por terminal:

```
o pato@DESKTOP-S8KAMG2:~/vscode-server/.vscode/archivos/Tp1$
/bin/python3 /home/pato/.vscode-server/.vscode/archivos/Tp1/
Socket_Servidor_Select.py
iniciando en localhost port 10000

esperando el próximo evento
conexión desde: ('127.0.0.1', 40284)
a las 2024-05-13 00:26:45.484333

esperando el próximo evento
conexión desde: ('127.0.0.1', 40296)
a las 2024-05-13 00:26:45.484488

recibido b'Este mensaje ' desde ('127.0.0.1', 40284)

esperando el próximo evento
recibido b'Este mensaje ' desde ('127.0.0.1', 40296)
enviando b'Este mensaje ' a ('127.0.0.1', 40284)

esperando el próximo evento
('127.0.0.1', 40284) cola vacía
enviando b'Este mensaje ' a ('127.0.0.1', 40296)

esperando el próximo evento
('127.0.0.1', 40296) cola vacía

esperando el próximo evento
recibido b'es enviado ' desde ('127.0.0.1', 40284)

esperando el próximo evento
recibido b'es enviado ' desde ('127.0.0.1', 40296)
enviando b'es enviado ' a ('127.0.0.1', 40284)

esperando el próximo evento
('127.0.0.1', 40284) cola vacía
enviando b'es enviado ' a ('127.0.0.1', 40296)

pato@DESKTOP-S8KAMG2:~/vscode-server/.vscode/archivos/Tp1$
python3 Socket_Cliente_Select.py
conectando a localhost puerto 10000
conexion a las 2024-05-13 00:26:45.484231
conexion a las 2024-05-13 00:26:45.484306
('127.0.0.1', 40284): enviando b'Este mensaje '
('127.0.0.1', 40296): enviando b'Este mensaje '
('127.0.0.1', 40284): recibido b'Este mensaje '
('127.0.0.1', 40296): recibido b'Este mensaje '
('127.0.0.1', 40284): enviando b'es enviado '
('127.0.0.1', 40296): enviando b'es enviado '
('127.0.0.1', 40284): recibido b'es enviado '
('127.0.0.1', 40296): recibido b'es enviado '
('127.0.0.1', 40284): enviando b'en partes.'
('127.0.0.1', 40296): enviando b'en partes.'
('127.0.0.1', 40284): recibido b'en partes.'
('127.0.0.1', 40296): recibido b'en partes.'
cerrando socket ('127.0.0.1', 40284) a las 2024-05-13 00:26
:45.485356
Duracion: 0:00:00.001050

cerrando socket ('127.0.0.1', 40296) a las 2024-05-13 00:26
:45.485475
Duracion: 0:00:00.001169

pato@DESKTOP-S8KAMG2:~/vscode-server/.vscode/archivos/Tp1
$
```

Activar Windows
Ve a Configuración para activar

```
esperando el próximo evento
('127.0.0.1', 40296) cola vacía

esperando el próximo evento
recibido b'en partes.' desde ('127.0.0.1', 40284)
recibido b'en partes.' desde ('127.0.0.1', 40296)

esperando el próximo evento
enviando b'en partes.' a ('127.0.0.1', 40284)
enviando b'en partes.' a ('127.0.0.1', 40296)

esperando el próximo evento
('127.0.0.1', 40284) cola vacía
('127.0.0.1', 40296) cola vacía

esperando el próximo evento
cerrando...('127.0.0.1', 40296) a las 2024-05-13 00:26:45.
485489
Duracion: 0:00:00.001001

esperando el próximo evento
cerrando...('127.0.0.1', 40296) a las 2024-05-13 00:26:45.
485602
Duracion: 0:00:00.001114

esperando el próximo evento

```

```
('127.0.0.1', 40284): recibido b'Este mensaje '
('127.0.0.1', 40296): recibido b'Este mensaje '
('127.0.0.1', 40284): enviando b'es enviado '
('127.0.0.1', 40296): enviando b'es enviado '
('127.0.0.1', 40284): recibido b'es enviado '
('127.0.0.1', 40296): recibido b'es enviado '
('127.0.0.1', 40284): enviando b'en partes.'
('127.0.0.1', 40296): enviando b'en partes.'
('127.0.0.1', 40284): recibido b'en partes.'
('127.0.0.1', 40296): recibido b'en partes.'
cerrando socket ('127.0.0.1', 40284) a las 2024-05-13 00:26
:45.485356
Duracion: 0:00:00.001050

cerrando socket ('127.0.0.1', 40296) a las 2024-05-13 00:26
:45.485475
Duracion: 0:00:00.001169

pato@DESKTOP-S8KAWG2:~/vscode-server/.vscode/archivos/Tp1
$
```

Activar Windows
Ve a Configuración para activar

Código del cliente con un modelo Cliente-Servidor, con la implementación del formato Concurrente.

```
1  import socket
2  import sys
3  ----
4  # Creando un socket TCP/IP
5  sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6
7  # Conecta el socket en el puerto cuando el servidor esté escuchando
8  server_dir = ('localhost', 6667)
9  print ('Conectando a:', server_dir[0], ', puerto:', server_dir[1])
10 sock.connect(server_dir)
11
12 avisoConex = sock.recv(1024)
13 print(avisoConex)
14
15 contador = 0
16 while contador <= 6:
17     try:
18         # Enviando datos
19         print("\nPaso:", contador)
20         mensaje = b'hola'
21         print('Enviando mensaje desde el cliente:', mensaje)
22         sock.sendall(mensaje)
23
24         # Buscando respuesta
25         data = sock.recv(1024)
26         print('Recibiendo del servidor:', data)
27
28     finally:
29         contador += 1
30
31 msj = b'chau'
32 sock.sendall(msj)
33 duracionConex = sock.recv(1024)
34 print('Cerrando socket.\nDuracion: {}'.format(duracionConex))
35 sock.close()
36
37
```

Al igual que en el ejemplo de C-S con formato Select, el Cliente funciona de la misma manera que el Cliente en base en C, crea las conexiones, a partir de las cuales se envían y reciben mensajes con el servidor. La forma de hacer el CONNECT() es la misma que mencionada anteriormente.

Hacemos uso de los métodos recv y send, o en este caso sendall, para recibir la cantidad de bytes enviadas por argumento, y enviar lo que se pasa por argumento (en el caso de sendall continua enviando hasta que todos los bytes hayan sido enviados)

Por ultimo enviamos un mensaje que solicita una desconexion, y como respuesta recibe el cierre del socket y la duración de la comunicación.

Código del Servidor:

```
1  import socket
2  import threading
3  import datetime
4
5  host = "127.0.0.1"
6  port = 6667
7
8  sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9  print("Socket Creado")
10 sock.bind((host, port))
11 print("socket bind Completado")
12 sock.listen(1)
13 print("socket en modo escucha - pasivo\n")
14
15 def proceso_hijo(*args):
16     conn = args[0]
17     addr = args[1]
18     hora_conexion = datetime.datetime.now() #registra la hora de la conexion del cliente
19     print('Conexion con {} a las {}'.format(addr, hora_conexion))
20
21     try:
22
23         conn.send("Servidor: Conectado con cliente".encode('UTF-8'))
24         bandera= b'chau'
25         data = conn.recv(1024)
26         while data != bandera:
27             print('Recibido del Cliente: ', data)
28             print('Conexion con {}'.format(addr))
29             if data:
30                 print('Enviando mensaje de vuelta al cliente\n')
31                 conn.sendall(data)
32             #else:
33             # print('No hay mas datos\n', addr)
34             # break
35             data = conn.recv(1024)
36
37         print('No hay mas datos\n', addr)
38         hora_desconexion = datetime.datetime.now()
39         duracion_conexion = hora_desconexion - hora_conexion
40         print('Conexion con {} cerrada a las {}. Duracion: {}'.format(addr, hora_desconexion, duracion_conexion))
41         conn.send(str(duracion_conexion).encode('utf-8'))
42         #le envio la duracion de tiempo que estuvo conectado al cliente en string encodeada
43
44
45     finally:
46         conn.close()
47
48 while True:
49     conn, addr = sock.accept()
50     threading.Thread(target=proceso_hijo, args=(conn, addr)).start()
51
```

En principio se crea el socket correspondiente al servidor, y se lo pone en estado de Escucha para conexiones posibles. Luego se tiene un while siempre activo (o se puede poner una condición para que se cierre el socket si así se lo desea) que acepta conexiones entrantes y enlaza a esta con la función proceso_hijo()

Esta función tiene como parámetro una serie de argumentos (2 en general, el socket al cual se acepta la conexión y su dirección). Registra la hora de inicio de la conexión, e intenta enviar y recibir datos mientras no se reciba una solicitud de cierre de conexión (el mensaje encode "chau"). Cuando esta solicitud llega se registra la hora de desconexion, y antes de cerrarla se envía la duración al Cliente. Luego de eso cierra la conexión.

Salida por terminal:

```
pato@DESKTOP-S8KAMG2:~/vscode-server/.vscode/archivos/Tp1$
/bin/python3 /home/pato/.vscode-server/.vscode/archivos/Tp1/
Socket_Servidor_Concurrente_01.py
Socket Creado
socket bind Completado
socket en modo escucha - pasivo

Conexion con ('127.0.0.1', 54210) a las 2024-05-13 00:33:28.
762237.
Recibido del Cliente: b'hola'
Conexion con ('127.0.0.1', 54210).
Enviando mensaje de vuelta al cliente

Recibido del Cliente: b'hola'
Conexion con ('127.0.0.1', 54210).
Enviando mensaje de vuelta al cliente

Recibido del Cliente: b'hola'
Conexion con ('127.0.0.1', 54210).
Enviando mensaje de vuelta al cliente

Recibido del Cliente: b'hola'
Conexion con ('127.0.0.1', 54210).
Enviando mensaje de vuelta al cliente

Recibido del Cliente: b'hola'
Conexion con ('127.0.0.1', 54210).
Enviando mensaje de vuelta al cliente

Recibido del Cliente: b'hola'
Conexion con ('127.0.0.1', 54210).
Enviando mensaje de vuelta al cliente

No hay mas datos
('127.0.0.1', 54210)
Conexion con ('127.0.0.1', 54210) cerrada a las 2024-05-13 0
0:33:28.764607. Duracion: 0:00:00.002370

pato@DESKTOP-S8KAMG2:~/vscode-server/.vscode/archivos/Tp1$
$ python3 Socket_Cliente_02.py
Conectando a: localhost , puerto: 6667
b'Servidor: Conectado con cliente'

Paso: 0
Enviando mensaje desde el cliente: b'hola'
Recibiendo del servidor: b'hola'

Paso: 1
Enviando mensaje desde el cliente: b'hola'
Recibiendo del servidor: b'hola'

Paso: 2
Enviando mensaje desde el cliente: b'hola'
Recibiendo del servidor: b'hola'

Paso: 3
Enviando mensaje desde el cliente: b'hola'
Recibiendo del servidor: b'hola'

Paso: 4
Enviando mensaje desde el cliente: b'hola'
Recibiendo del servidor: b'hola'

Paso: 5
Enviando mensaje desde el cliente: b'hola'
Recibiendo del servidor: b'hola'

Paso: 6
Enviando mensaje desde el cliente: b'hola'
Recibiendo del servidor: b'hola'
Cerrando socket.
Duracion: b'0:00:00.002370'
pato@DESKTOP-S8KAMG2:~/vscode-server/.vscode/archivos/Tp1$
$
Recibiendo del servidor: b'hola'
Cerrando socket.
Duracion: b'0:00:00.002370'
pato@DESKTOP-S8KAMG2:~/vscode-server/.vscode/archivos/Tp1$
$
```