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  # onectar 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      # envío mensajes en ambos sockets
33      for s in socks:
34          if (datos_salientes != b'chau'):
35              print('{}: enviando {!r}'.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: {}\n".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 {!r}'.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: {}\n".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:

```python
import select
import socket
import sys
import queue
import datetime

# Creando un socket TCP/IP
servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
servidor.setblocking(0)

# Hago Bind del socket al puerto
dir_servidor = ('localhost', 10000)
print('iniciando en {} port {}'.format(*dir_servidor),
      file=sys.stderr)
servidor.bind(dir_servidor)

# Escucho conexiones entrantes
servidor.listen(5)

# Sockect que espero leer
entradas = [servidor]

# Sockets que espero enviar
salidas = []

# Cola de mensajes salientes
cola_mensajes = {}

while entradas:

    # Espero a que al menos uno de los sockets este listo para ser procesado

    print('\nesperando el próximo evento', file=sys.stderr)
    readable, writable, exceptional = select.select(entradas,
                                                    salidas,
                                                    entradas)

    if not (readable or writable or exceptional):
        print('  tiempo excedido....',
              file=sys.stderr)
        continue
```

```
    # Manejo entradas
    for s in readable:

        if s is servidor:
            # Un socket "leible" está listo para aceptar conexiones
            con, dir_cliente = s.accept()
            hora_conexion = datetime.datetime.now() #registra la hora de la conexion
            print('  conexión desde: ', dir_cliente,
                  file=sys.stderr)
            print("a las {}\n".format(hora_conexion))
            con.setblocking(0)
            entradas.append(con)

            # Le asigno a la conexión una cola en la cuál quiero enviar
            cola_mensajes[con] = queue.Queue()

        else:
            data = s.recv(1024)
            if data:
                # Un socket leíble tiene datos
                print('  recibido {!r} desde {}'.format(
                    *args: data, s.getpeername()), file=sys.stderr,
                )
                cola_mensajes[s].put(data)
                # Agrego un canal de salida para la respuesta
                if s not in salidas:
                    salidas.append(s)
            else:
                # Si está vacío lo interpreto como una conexión a cerrar
                hora_cierre = datetime.datetime.now()
                duracion = hora_cierre - hora_conexion
                print('  cerrando...{} a las {}'.format( *args: dir_cliente, hora_cierre),
                      file=sys.stderr)
                print("Duracion: {}\n".format(duracion))
                # dejo de escuchar en la conexión
                if s in salidas:
                    salidas.remove(s)
                entradas.remove(s)
                s.close()

                # Rremueve mensaje de la cola
                del cola_mensajes[s]
    # Administro salidas
    for s in writable:
        try:
            next_msg = cola_mensajes[s].get_nowait()
        except queue.Empty:
            # No hay mensaje en espera. Dejo de controlar para posibles escrituras

            print('  ', s.getpeername(), 'cola vacía',
                  file=sys.stderr)
            salidas.remove(s)
        else:
            print(' enviando {!r} a {}'.format( *args: next_msg, s.getpeername()), file=sys.stderr)
            s.send(next_msg)

    # Administro condiciones excepcionales"
    for s in exceptional:
        print('excepción en', s.getpeername(),
              file=sys.stderr)
        # Dejo de escuchar en las conexiones
        entradas.remove(s)
        if s in salidas:
            salidas.remove(s)
        s.close()

        # Remuevo cola de mensajes
        del cola_mensajes[s]
```

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.

```
    for s in readable:

        if s is servidor:
            # Un socket "leíble" está listo para aceptar conexiones
            con, dir_cliente = s.accept()
            hora_conexion = datetime.datetime.now() #registra la hora de la conexion
            print('  conexión desde: ', dir_cliente,
                  file=sys.stderr)
            print("a las {}\n".format(hora_conexion))
            con.setblocking(0)
            entradas.append(con)

            # Le asigno a la conexión una cola en la cuál quiero enviar
            cola_mensajes[con] = queue.Queue()

        else:
            data = s.recv(1024)
            if data:
                # Un socket leíble tiene datos
                print('  recibido {!r} desde {}'.format(
                    *args: data, s.getpeername()), file=sys.stderr,
                )
                cola_mensajes[s].put(data)
                # Agrego un canal de salida para la respuesta
                if s not in salidas:
                    salidas.append(s)
            else:
                # Si está vacio lo interpreto como una conexión a cerrar
                hora_cierre = datetime.datetime.now()
                duracion = hora_cierre - hora_conexion
                print('  cerrando...{} a las {}'.format( *args: dir_cliente, hora_cierre),
                      file=sys.stderr)
                print("Duracion: {}\n".format(duracion))
                # dejo de escuchar en la conexión
                if s in salidas:
                    salidas.remove(s)
                entradas.remove(s)
                s.close()

                # Rremueve mensaje de la cola
                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.

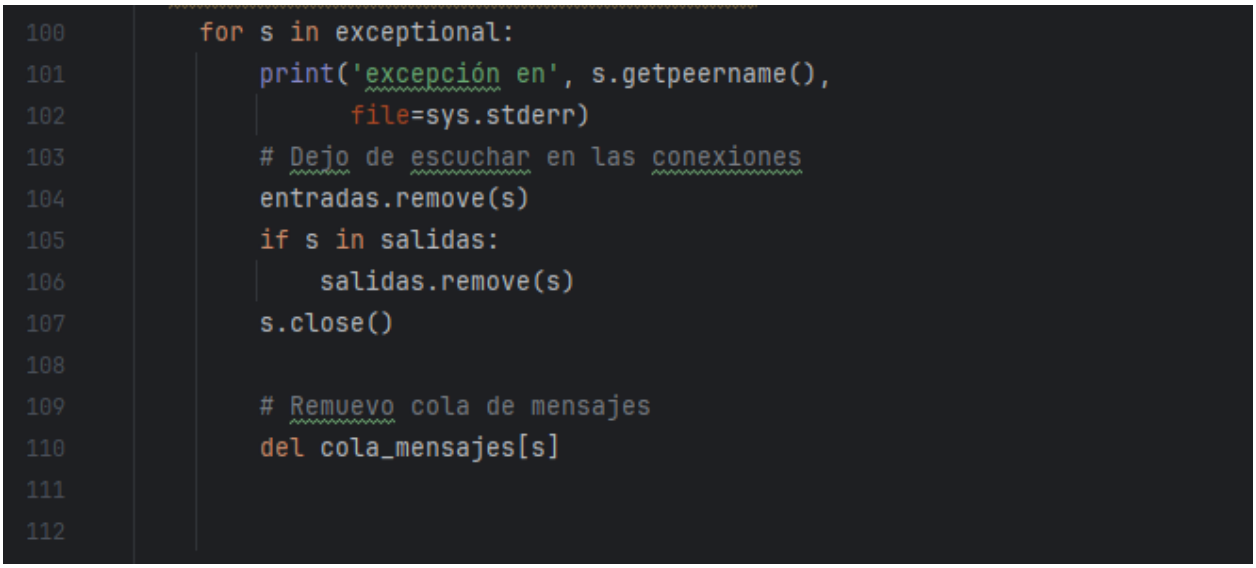
Por otro lado, en el else, cuando hay una conexión con datos los guarda y los printea, de no haber datos, asume que el cliente quiere cerrar la conexión.

```
    for s in writable:
        try:
            next_msg = cola_mensajes[s].get_nowait()
        except queue.Empty:
            # No hay mensaje en espera. Dejo de controlar para posibles escrituras

            print('  ', s.getpeername(), 'cola vacía',
                  file=sys.stderr)
            salidas.remove(s)
        else:
            print(' enviando {!r} a {}'.format( *args: next_msg, s.getpeername()), file=sys.stderr)
            s.send(next_msg)
```

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 envio 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.

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

```python
import socket
import sys

# Creando un socket TCP/IP
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Conecta el socket en el puerto cuando el servidor esté escuchando
server_dir = ('localhost', 6667)
print ('Conectando a:', server_dir[0], ', puerto:', server_dir[1])
sock.connect(server_dir)

avisoConex = sock.recv(1024)
print(avisoConex)

contador = 0
while contador <= 6:
    try:
        # Enviando datos
        print("\nPaso:", contador)
        mensaje = b'hola'
        print('Enviando mensaje desde el cliente:', mensaje)
        sock.sendall(mensaje)

        # Buscando respuesta
        data = sock.recv(1024)
        print('Recibiendo del servidor:', data)


    finally:
        contador += 1

msj = b'chau'
sock.sendall(msj)
duracionConex = sock.recv(1024)
print('Cerrando socket.\nDuracion: {}'.format(duracionConex))
sock.close()

```

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:

```python
import socket
import threading
import datetime

host = "127.0.0.1"
port = 6667

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("Socket Creado")
sock.bind((host, port))
print("socket bind Completado")
sock.listen(1)
print("socket en modo escucha - pasivo\n")

def proceso_hijo(*args):
    conn = args[0]
    addr = args[1]
    hora_conexion = datetime.datetime.now() #registra la hora de la conexion del cliente
    print('Conexion con {} a las {}.'.format(addr, hora_conexion))

    try:

        conn.send("Servidor: Conectado con cliente".encode('UTF-8'))
        bandera= b'chau'
        data = conn.recv(1024)
        while data != bandera:
            print('Recibido del Cliente: ', data)
            print('Conexion con {}.'.format(addr))
            if data:
                print('Enviando mensaje de vuelta al cliente\n')
                conn.sendall(data)
            #else:
            #    print('No hay mas datos\n', addr)
            #    break
            data = conn.recv(1024)

        print('No hay mas datos\n', addr)
        hora_desconexion = datetime.datetime.now()
        duracion_conexion = hora_desconexion - hora_conexion
        print('Conexion con {} cerrada a las {}. Duracion: {}'.format(addr, hora_desconexion, duracion_conexion))
        conn.send(str(duracion_conexion).encode('utf-8'))
        #le envio la duracion de tiempo que estuvo conectado al cliente en string encodeada


    finally:
        conn.close()

while True:
    conn, addr = sock.accept()
    threading.Thread(target=proceso_hijo, args=(conn, addr)).start()

```

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.