TP1 – C

Integrantes: Lucas Debarbieri, Sebastian Lernoud, Andres Luza, Gonzalo Crucita

Programa Cliente:

```
# El programa cliente de ejemplo utiliza dos sockets para demostrar cómo el servidor con select() administra
múltiples conexiones al mismo tiempo.
# El cliente comienza conectando cada socket TCP/IP al servidor.

import socket
import sys
from datetime import datetime, date, time, timezone

mensajes = [
    "4",
    "5",
    "6"
]
dir_servidor = ('localhost', 10000)



orden = input("Desea conectar el cliente con el servidor? ")
while orden=="si":
    # Creo socket
    socks = [
        socket.socket(socket.AF_INET, socket.SOCK_STREAM),
        socket.socket(socket.AF_INET, socket.SOCK_STREAM),
    ]
    # onectar el socket al puerto en el cual el servidor está escuchando
    print('conectando a {} puerto {}'.format(*dir_servidor),
        file=sys.stderr)
    for s in socks:
        s.connect(dir_servidor)
```

```
    for mensaje in mensajes:
        datos_salientes = mensaje.encode()

        # envío mensajes en ambos sockets
        for s in socks:
            print('{}: enviando {!r}'.format(s.getsockname(),
                                             datos_salientes),
                file=sys.stderr)
            s.send(datos_salientes)

        # leo respuestas en ambos sockets
        for s in socks:
            data = s.recv(1024)
            print('{}: recibido {!r}'.format(s.getsockname(),
                                             data),
                file=sys.stderr)
            if not data:
                print('cerrando socket', s.getsockname(),
                    file=sys.stderr)
                s.close()
    for s in socks:
        print("Cerrando socket",s.getsockname(),file=sys.stderr)
        s.close()
    orden = input("Desea que el el socket cliente siga activo?")
```

Programa Servidor:

```
# Servidor con Select()
# El módulo select proporciona acceso a funciones específicas de E/S.
# La función POSIX select(), está disponible en Unix y Windows.
# El módulo tam ién incluye poll(), pero solo para Unix, y varias opciones que solo funcionan con variantes específicas de Unix.
# El ejemplo del servidor de eco lo mejoramosampliar para tener más de una conexión simultánea.
# La nueva versión comienza creando un socket TCP/IP que no se bloquea y configurado para escuchar en una dirección.
```

UCA

```
# Los argumentos para select() son tres listas que contienen canales de comunicación a monitorear.
# La primera es una lista de los objetos para verificar los datos entrantes que se leerán,
# el segundo contiene objetos que recibirán datos salientes cuando haya espacio en el buffer,
# y el tercero aquellos que pueden tener un error (generalmente un combinación de los objetos del canal de entrada y salida).
# El siguiente paso en el servidor es configurar las listas que contienen fuentes de entrada y los destinos de salida que se pasarán a select().
# El bucle principal del servidor agrega y elimina las conexiones de estas listas.
# Dado que esta versión del servidor va a esperar poder escribir a un socket antes de enviar cualquier dato (en lugar de enviar inmediatamente la respuesta),
# Cada conexión de salida necesita una cola para actuar como un buffer para los datos que se enviarán a través de él.
# La parte del programa principal del servidor hace un bucle, llamando a select() para bloquear y esperar la actividad de la red.
# select() devuelve tres nuevas listas, que contienen subconjuntos del contenido de las listas pasadas.
# Todos los sockets en la lista readable tiene datos entrantes almacenados en búfer y disponibles para ser leídos.
# Todos los sockets en la lista writable tienen espacio libre en su búfer y se puede escribir en ellos.
# Los sockets devueltos en excepcional han tenido un error (la definición real de «condición excepcional» depende de la plataforma).
# Los sockets «legibles» representan tres casos posibles.
# Si el socket es el socket principal del «servidor», el que se usa para escuchar conexiones, entonces la condición «legible» significa que está listo para aceptar otra conexión entrante.
# Además de añadir la nueva conexión a la lista de entradas para monitorear, esta sección establece que el socket del cliente no se bloquee.
# El siguiente caso es una conexión establecida con un cliente que ha enviado datos. Los datos se leen con recv(),
# luego se colocan en la cola para que puedan ser enviados a través del socket y de vuelta al cliente.
# Hay menos casos para las conexiones escribibles. Si hay datos en la cola para una conexión, se envía el siguiente mensaje.
# De otra manera, la conexión se elimina de la lista de conexiones de salida para que la próxima vez a través del bucle select() no indique que el socket está listo para enviar datos.
# select() también toma un cuarto parámetro opcional, que es el número de segundos a esperar antes de interrumpir el monitoreo si no se han activado canales.
# El uso de un valor de tiempo de espera permite a un programa principal ejecutar select() como parte de un ciclo de procesamiento más grande,
# tomando otras acciones entre la comprobación de entrada de red.
```

```
# Cuando el tiempo de espera expira, select() devuelve tres listas vacías.
# Actualizar el ejemplo del servidor para usar un tiempo de espera requiere agregar el argumento extra a la llamada
select() y manejo de las listas vacías que select() devuelve.

# Referencias : PyMOTW-3 - Doug Hellmann

def factorial(n):
    res=1
    for i in range(1,n+1):
        res *= i
    return res



import select
import socket
import sys
import queue
from datetime import datetime, date, time, timezone

# 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]
```

UCA

```
# Sockets que espero enviar
salidas = []

# Cola de mensajes salientes
cola_mensajes = {}
terminarPrograma = False

#Lista con horarios de conexion
listaConexion = []
listaDesconexion = []
while entradas and not terminarPrograma:

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

    print('esperando 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 "leíble" está listo para aceptar conexiones
            con, dir_cliente = s.accept()
            print('  conexión desde: ', dir_cliente,
                  file=sys.stderr)
            con.setblocking(0)
            entradas.append(con)
            hora_conexion = datetime.now()
```

```
            listaConexion.append((dir_cliente,hora_conexion))

            # 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 {} desde {}'.format(
                    data, s.getpeername()), file=sys.stderr,
                )
                data_operar = int(data)
                data_devolver = factorial(data_operar)
                data = str(data_devolver).encode()
                cola_mensajes[s].put(data)

                #print("Entro: {}\nSalio: {}".format(data_operar,data_devolver))
                # 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
                print('  cerrando...', dir_cliente,
                      file=sys.stderr) #Imprime bien solo la direccion del primero
                # dejo de escuchar en la conexión
                if s in salidas:
                    salidas.remove(s)
                if s in entradas:
                    entradas.remove(s)
                s.close()
                #print(s)

                # while len(entradas)!=0:
                #     entradas.pop()
```

```
            # Rremueve mensaje de la cola
            del cola_mensajes[s]
            #Solo queda el servidor en la lista entonces lo saco asi termina
            # la ejecucion
            # if len(entradas)==1:
            #     entradas.pop()
    # 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)
        except:
            print("Error")
        else:
            print(' enviando {} a {}'.format(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]
```

```
    if len(entradas)==1 and input("Desea terminar la conexion? ") == "si":
        terminarPrograma=True

hora_desconexion = datetime.now()

for tuplaConexion in listaConexion:
    print("Conectado {} a las: {}".format(tuplaConexion[0],tuplaConexion[1]))
    print("{} Desconectado a las : {}".format(tuplaConexion[0],hora_desconexion))
    tiempoDeConexion = hora_desconexion-tuplaConexion[1]
    print("Estuvo {} microsegundos".format(tiempoDeConexion))
```