Universidad de Costa Rica

Facultad de Ingeniería

Escuela de Ciencias de la Computación e Informática

CI-0123

Grupo 01

Proyecto
Fase 1
Comunicación punto a punto

Profesores:

Gabriela Barrantes

Francisco Arroyo

Elaborado por:

Luis Carvajal Rojas B31494

Marcial Carrillo Vega A81379

Diego Contreras Estrada B62074

Andrés Navarrete Boza B55017

Índice

1. Problema

Siguiendo con el contador de palabras de la primera parte, se desea ahora lograr que el "servidor" (el proceso que cuenta), pueda responderle a procesos "cliente" (lectores) que se encuentran en diferentes máquinas. Para no complicar esta entrega, puede suponer que solo hay un proceso cliente vivo a la vez. Los procesos cliente y el servidor deben ser implementados en TCP. De nuevo, el proceso contador debe almacenar la oración y el número de palabras, pero ahora también debe guardar la dirección IP y el puerto del proceso solicitante. El proceso servidor debe contar con un acceso de consola. Deben definir una manera de que, por solicitud de consola, se puedan imprimir los datos almacenados para un IP y puerto dados. OJO: Todas las condiciones y restricciones de la fase I siguen aplicando. Tengan presente que cualquier solución va a requerir al menos dos hilos por cliente y dos hilos por servidor, debido a los posibles bloqueos y encolamientos. Tengan cuidado con esa concurrencia.

2. Decisiones y Especificaciones

- Se utiliza Python
- Para terminar los procesos se usa la palabra 'exit' y para imprimir 'print' en el server
- Se usa una cola como estructura de datos
- Se usan archivos de configuración para cambiar la IP y el puerto
- Se utiliza el puerto 5000
- Se controla la falta de conexión y la desconexión

3. Pseudocódigo

3.1. Server

```
Se inicia servidor
  Se abren 3 hilos
  UserInputManager{ #Es un hilo
    si no es 'exit'
      trabaje
```

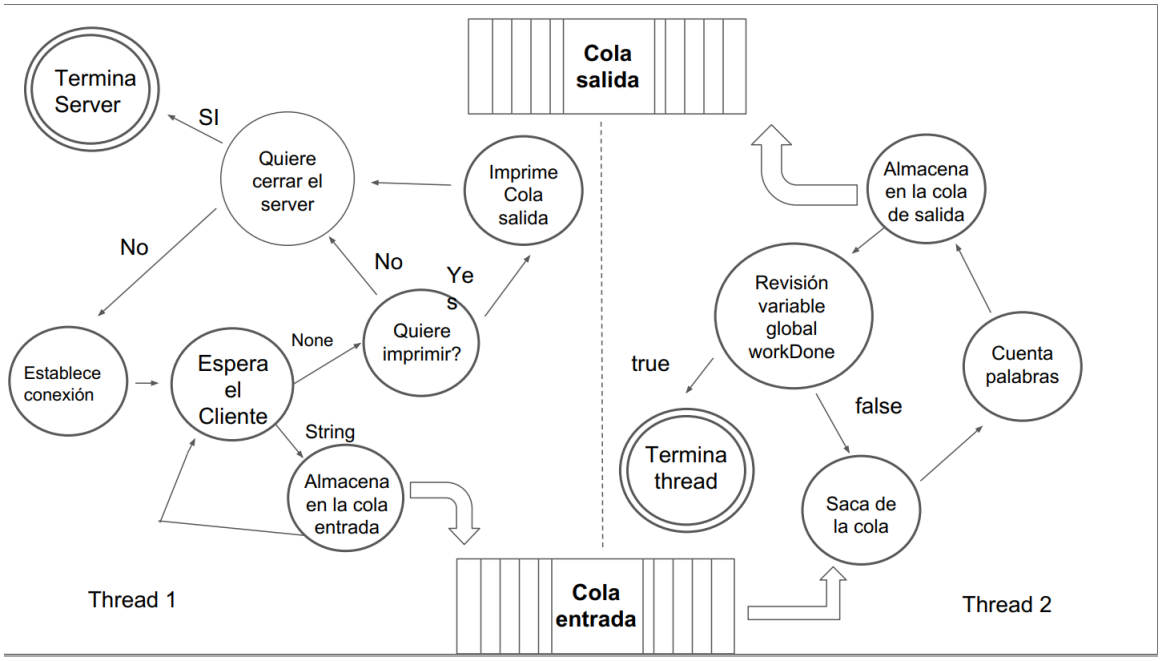
```
    si es 'print'
      imprime datos
  }

  BufferManager{ #otro hilo
    Guarda lo que le va llegando
    Pregunta si aun hay conexin
  }

  CountManager{
    contar las palabras e imprimir
  }
```

3.2. Cliente

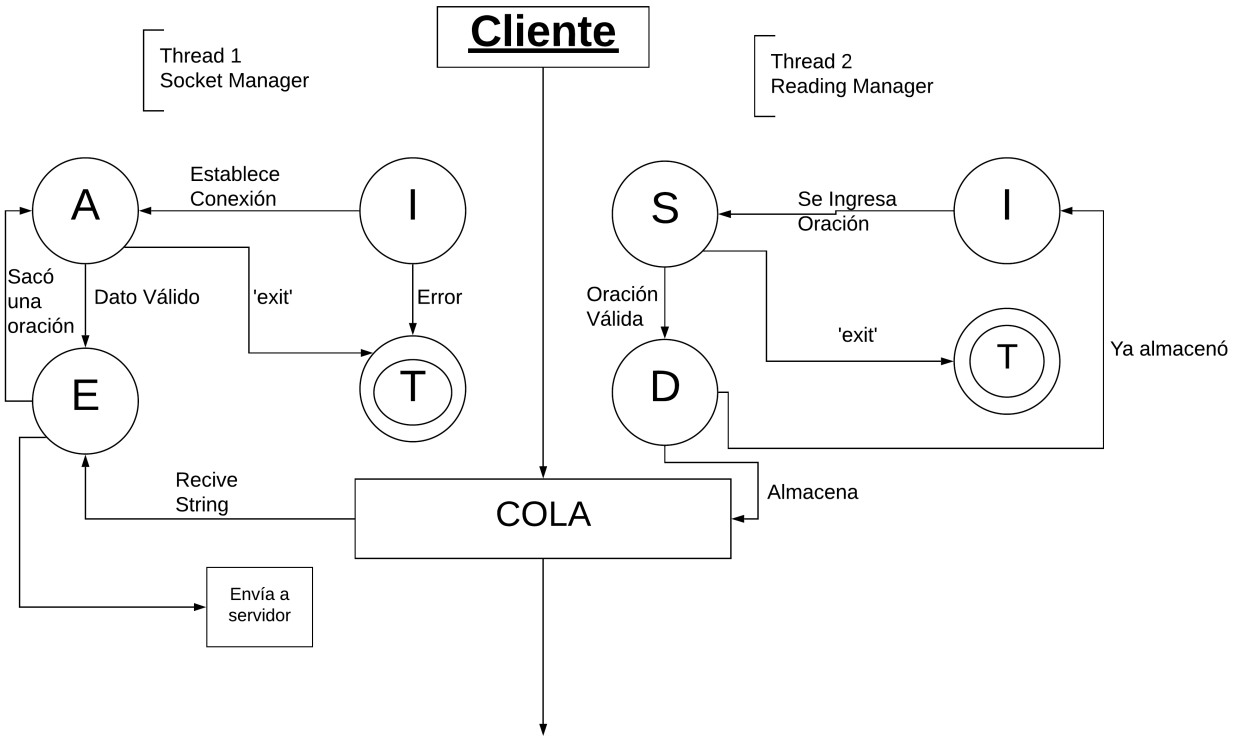
```
Se inician el cliente
  Se abren 2 hilos
    socketManager{
     Iniciar clente y esperar respuesta
    }
    readingManager{
      Leer las entrasdas de usuario
      si es 'exit'
        sale
    }
```

4. Máquinas de Estado

4.1. Server



5. Cliente



6. Manual y Requerimientos

1. Correr el server (python3 server.py)
2. Correr el cliente (python3 client.py)
3. Escribir oraciones en el cliente
4. Escribir 'print' en el server para ver datos
5. Escribir 'exit' en el server y luego en el cliente

Requerimientos:

- En los archivos client.ini y server.ini, escribir la ip y el puerto deseado
- Python 3
- Servidor y cliente deben estar en la misma red

7. Código fuente

7.1. Server

```
import socket
import threading
import queue
import os
import configparser

class Data:
    def __init__(self, oracion, numPalabaras, addr):
        self.oracion = oracion
        self.numPalabaras = numPalabaras
        self.ip = addr[0]
        self.port = addr[1]

class UserInputManager(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)

    def printResults(self):
        for data in resultData:
            print("%s \t %d \t %s %s" % (data.oracion, data.numPalabaras, data.ip,
     data.port))

    def run(self):
        while True:
            userInput = input()
            if userInput == "print":
                self.printResults()
            elif userInput == "exit":
                # sys.exit(1)
                os._exit(1)
                #aqui falta tirar a false una variable global y terminar el hilo que
     trabaja esto


class BufferManager(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)

    def run(self):
        config = configparser.ConfigParser()
        config.sections()
        config.read('server.ini')
        host = config['DEFAULT']['IP']
```

```
42          port = int(config['DEFAULT']['Port'])
43          print("waiting connection")
44          mySocket = socket.socket()
45          mySocket.bind((host, port))
46
47          mySocket.listen()
48          conn, addr = mySocket.accept()
49          print("Connection from: " , addr)
50
51          while True: #ESte ciclo debe de depender de una variable global
52              string = conn.recv(1024).decode()
53              if not string:
54                  conn.close()
55                  print("connection lost, waiting for connection...")
56                  mySocket = socket.socket()
57                  mySocket.bind((host, port))
58                  mySocket.listen()
59                  conn, addr = mySocket.accept()
60                  print("Connection from -: " + str(addr))
61              else:
62                  cola.put(Data(string, 0, addr))
63                  response = "message received"
64                  conn.send(response.encode())
65          conn.close()
66
67
68  class CountManager(threading.Thread):
69      def __init__(self):
70          threading.Thread.__init__(self)
71      def run(self):
72          while workDone[0] == False:
73              data = cola.get(True,None)
74              if data != -1:
75                  result = len(data.oracion.split())
76                  data.numPalabaras = result
77                  resultData.append(data)
78          print("exitCountManager")
79
80
81  workDone = []
82
83  printCommand = []
84
85  cola = queue.Queue()
86
87  resultData = []
88
89  workDone.append(False)
90
```

```
# Create new threads
thread1 = BufferManager()
thread2 = CountManager()
thread3 = UserInputManager()

# Start new Threads
thread1.start()
thread2.start()
thread3.start()

thread1.join()
thread2.join()
thread3.join()
```

7.2. Cliente

```
import socket
import threading
import queue
import sys
import configparser

class socketManager(threading.Thread):
  def __init__(self):
    threading.Thread.__init__(self)
    self.continueWorking = True

  def run(self): #Cliente
    global startAsking
    global normal_exit
    global stopAll

    config = configparser.ConfigParser()
    config.sections()
    config.read('client.ini')
    host = config['DEFAULT']['IP']
    port = int(config['DEFAULT']['Port'])

    # Trying to make a connection with the server
    try:
      mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      mySocket.connect((host,port))
      print('Connection Sucessful')
    except:
      print("Server is down")
      startAsking = False
      self.continueWorking = False
```

```
32          normal_exit = True
33
34      while self.continueWorking and not stopAll:
35          startAsking = True
36
37          if not inputQueue.empty():
38              outboundMessage = inputQueue.get()
39              if outboundMessage != 'exit':
40                  mySocket.send(outboundMessage.encode())
41                  response = mySocket.recv(1024).decode()
42
43                  if response == "":
44                      print("Server down press enter ...")
45                      mySocket.close()
46                      self.continueWorking = False
47                      normal_exit = True
48                  else:
49                      print ('Server: ', response)
50              else:
51                  mySocket.close()
52                  self.continueWorking = False
53
54
55  class readingManager(threading.Thread):
56      def __init__(self):
57          threading.Thread.__init__(self)
58
59      def run(self):
60          global normal_exit
61          global stopAll
62          while not normal_exit:
63              if startAsking:
64                  message = ".."
65                  while message != 'exit' and not normal_exit:
66                      message = input()
67                      if message == "exit":
68                          normal_exit = True
69                      else:
70                          inputQueue.put(message)
71          stopAll = True
72
73
74  inputQueue = queue.Queue()
75
76  normal_exit = False
77  startAsking = False
78  stopAll = False
79
80  thread1 = readingManager()
```

```
81 thread2 = socketManager()
82
83 thread1.start()
84 thread2.start()
85
86 thread1.join()
87 thread2.join()
```