**Department of Computing**

**Course Code: CS332**

**Class: BSCS-9AB**

**Lab 05: Understanding the Physical clock algorithms**

**CLO1: Design Distributed Protocols**

**Date: 28th February 2022**

**Time: 09:00-1200**

**Instructors: Dr. Farzana Jabeen**

|  |
| --- |
| **Submitted By:**  **Fatima Seemab**  **291310**  **BSCS 9B** |

**Christian Algorithm**

**Client:**

|  |
| --- |
| import socket import pandas from dateutil import parser  def synchronizeTime():  s = socket.socket()  port = 8000   s.connect(('127.0.0.1', port))  request\_time = pandas.to\_datetime("today")  server\_time = parser.parse(s.recv(1024).decode())  response\_time =pandas.to\_datetime("today")  unsychnronized\_time =pandas.to\_datetime("today")  client\_time = server\_time + ((response\_time-request\_time)/2)  print("The travel time between client and server is: ",((response\_time-request\_time)/2))  print("The client time before sychronization is:", unsychnronized\_time)  print("The client time after sychronization is:", client\_time)   # s.close()   # Driver function if \_\_name\_\_ == '\_\_main\_\_':  # synchronize time using clock server  synchronizeTime() |

**Server:**

|  |
| --- |
| import socket import pandas server = socket.socket() print("------------------------------------------") print("socket is created") port = 8000 print("socket is binded to port 8000") server.bind(('', port)) server.listen() print("server is listening") print("------------------------------------------") while True:  client, address = server.accept()  print("client is connected")  print('Server connected to', address)  client.send(str(pandas.to\_datetime("today")).encode())  print("sending date time to client")  client.close()  print("connection for the client is closed")  print("Waiting for other clients..........") server.close() |

**Output:**

**Server:**

|  |
| --- |
| **Text  Description automatically generated** |

|  |
| --- |
| Text  Description automatically generated |

**client**

**Barkeley Algorithm**

**Client**

|  |
| --- |
| import socket import datetime from dateutil import parser from datetime import datetime def synchronizeTime():  client = socket.socket()  port = 5000   client.connect(('127.0.0.1', port))   message = (datetime.now())  print("The time client sending is:",message)  client.send(str(message).encode())  while True:  server\_time = (client.recv(1024).decode())  client\_time = (datetime.now())  # response\_time =pandas.to\_datetime("today")  # unsychnronized\_time =pandas.to\_datetime("today")  # client\_time = server\_time + ((response\_time-request\_time)/2)  print("The time according to this client should be: ", client\_time)  print("The new time sent by the server node is : ", (server\_time))  # print("The client time before sychronization is:", unsychnronized\_time)  # print("The client time after sychronization is:", client\_time)   # client.close()   # Driver function if \_\_name\_\_ == '\_\_main\_\_':  # synchronize time using clock server  synchronizeTime() |

**Server**

|  |
| --- |
| import socket from dateutil import parser  import datetime  clients={}  def create\_master\_node():  server=socket.socket()  ip\_address="127.0.0.1"  port=5000  server.bind((ip\_address,port))  server.listen(5)  print("master nose is waiting for the clients.......")  connect\_slaves(server) def connect\_slaves(server):  while True:  client, address = server.accept()  print("client is connected")  message = parser.parse(client.recv(512).decode())  server\_time = datetime.datetime.now()   clock\_difference = server\_time.timestamp() - message.timestamp()   print("time from the client", address, "is", message)  print("The clock difference between server and client is: ", clock\_difference)  clients[address] = {  "client": client,  "time": message,  "clock\_difference": clock\_difference  }  synchronize\_time()  def average():  difference=[]  for value in clients.values():  print()  difference.append(value["clock\_difference"])  if len(clients)>1:  Sum=sum(difference)  average=Sum/len(clients)  print("average time difference is:",average)  return average  else:   return difference[0]  def synchronize\_time():  print("Number of clients to be synchronized: " + str(len(clients)))    if len(clients) > 0:  average\_clock\_difference = average()  for client\_addr, client in clients.items():  time = datetime.datetime.now()  synchronized\_time = time.timestamp() + average\_clock\_difference  synchronized\_time = datetime.datetime.fromtimestamp(synchronized\_time)  print("synchronized time is:", synchronized\_time)  client['client'].send(str(synchronized\_time).encode())  print("data sent...")  else:  print("No client data.")  print("-------------------------------------------------------")  create\_master\_node() |

|  |
| --- |
| Graphical user interface, text  Description automatically generated |

**Output:**