**ARUNIMA SINGH THAKUR**

**180905218 CSE C-31**

**DS LAB 6&7**

**CLOCK SYNCHRONIZATION & MUTUAL EXCLUSION (ELECTION ALGORITHM)**

1. **Berkeley’s Algorithm**

**4 clients - KMC, MIT, TAPMI, SOLS institute clocks**

**1 master clock server**

**server.py**

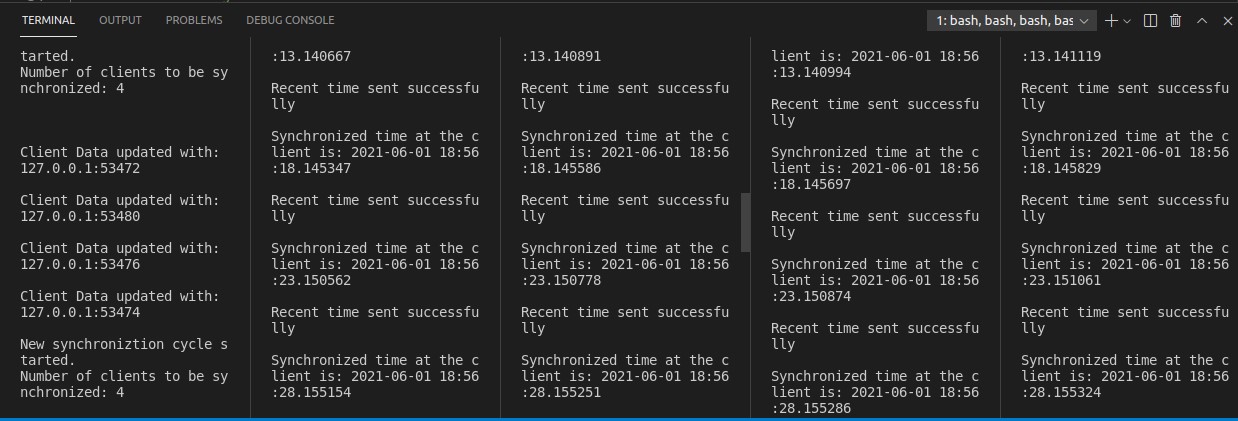
|  |
| --- |
| from dateutil import parser |
| import threading |
| import datetime |
| import socket |
| import time |
| # data structure used to store client address and clock data |
| client\_data = {} |
|  |
| ''' nested thread function used to receive |
| clock time from a connected client ''' |
|  |
| def startRecieveingClockTime(connector, address): |
| while True: |
| clock\_time\_string = connector.recv(1024).decode() |
| clock\_time = parser.parse(clock\_time\_string) |
| clock\_time\_diff = datetime.datetime.now() - clock\_time |
| client\_data[address] = { |
| "clock\_time": clock\_time, |
| "time\_difference": clock\_time\_diff, |
| "connector": connector |
| } |
| print("Client Data updated with: " + str(address), end="\n\n") |
| time.sleep(5) |
|  |
| ''' master thread function used to open portal for |
| accepting clients over given port ''' |
|  |
| def startConnecting(master\_server): |
| # fetch clock time at slaves / clients |
| while True: |

|  |
| --- |
| # accepting a client / slave clock client |
| master\_slave\_connector, addr = master\_server.accept() |
| slave\_address = str(addr[0]) + ":" + str(addr[1]) |
| print(slave\_address + " got connected successfully") |
| current\_thread = threading.Thread( |
| target=startRecieveingClockTime, |
| args=(master\_slave\_connector, |
| slave\_address, )) |
| current\_thread.start() |
|  |
| def getAverageClockDiff(): |
| current\_client\_data = client\_data.copy() |
| time\_difference\_list = list(client['time\_difference'] |
| for client\_addr, client in |
| client\_data.items()) |
| sum\_of\_clock\_difference = sum( |
| time\_difference\_list, datetime.timedelta(0, 0)) |
| average\_clock\_difference = sum\_of\_clock\_difference / |
| len(client\_data) |
| return average\_clock\_difference |
|  |
| def synchronizeAllClocks(): |
| while True: |
| print("New synchroniztion cycle started.") |
| print("Number of clients to be synchronized: " + |
| str(len(client\_data))) |
| if len(client\_data) > 0: |
| average\_clock\_difference = getAverageClockDiff() |
| for client\_addr, client in client\_data.items(): |
| try: |
| synchronized\_time = \ |
| datetime.datetime.now() + \ |
| average\_clock\_difference |
| client['connector'].send(str( |
| synchronized\_time).encode()) |
| except Exception as e: |
| print("Something went wrong while " + |
| "sending synchronized time " + |
| "through " + str(client\_addr)) |
| else: |
| print("No client data." + |
| " Synchronization not applicable.") |
| print("\n\n") |
| time.sleep(5) |
|  |
| # function used to initiate the Clock Server / Master Node |
| def initiateClockServer(port=8080): |
| master\_server = socket.socket() |
| master\_server.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1) |
| print("Socket at master node created successfully\n") |
| master\_server.bind(('', port)) |
| # Start listening to requests |
| master\_server.listen(10) |
| print("Clock server started...\n") |
| # start making connections |
| print("Starting to make connections...\n") |
| master\_thread = threading.Thread( |
| target=startConnecting, args=(master\_server, )) |
| master\_thread.start() |
| # start synchroniztion |
| print("Starting synchronization parallely...\n") |
| sync\_thread = threading.Thread(target=synchronizeAllClocks, args=()) |
| sync\_thread.start() |
|  |
| if \_\_name\_\_ == '\_\_main\_\_': |
| initiateClockServer(port=8080) |

**client.py**

|  |
| --- |
| from timeit import default\_timer as timer |
| from dateutil import parser |
| import threading |
| import datetime |
| import socket |
| import time |
| import sys |
| # client thread function used to send time at client side |
|  |
| def startSendingTime(slave\_client): |
| while True: |
| # provide server with clock time at the client |
| slave\_client.send(str( |
| datetime.datetime.now()).encode()) |
| print("Recent time sent successfully", |
| end="\n\n") |
| time.sleep(5) |
|  |
| def startReceivingTime(slave\_client): |
| while True: |
| # receive data from the server |
| Synchronized\_time = |
| parser.parse(slave\_client.recv(1024).decode()) |
| print("Synchronized time at the client is: " + |
| str(Synchronized\_time), end="\n\n") |
|  |
| def initiateSlaveClient(port=8080): |
| slave\_client = socket.socket() |
| # connect to the clock server on local computer |
| slave\_client.connect(('127.0.0.1', port)) |
| print(sys.argv[1]+" digital clock") |
| # start sending time to server |
| print("Starting to receive time from server\n") |
| send\_time\_thread = threading.Thread( |
| target=startSendingTime, |
| args=(slave\_client, )) |
| send\_time\_thread.start() |
| # start recieving synchronized from server |
| print("Starting to recieving " + |
| "synchronized time from server\n") |
| receive\_time\_thread = threading.Thread( |
| target=startReceivingTime, |
| args=(slave\_client, )) |
| receive\_time\_thread.start() |
|  |
| if \_\_name\_\_ == '\_\_main\_\_': |
| # initialize the Slave / Client |
| initiateSlaveClient(port=8080) |
|  |

**Screenshot:**



1. **Cristian’s Algorithm**

**Taking Laptop as server (UTC reciever)**

**Mobile as a client for synchronizing**

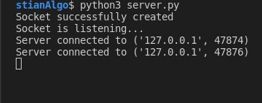
**server.py**

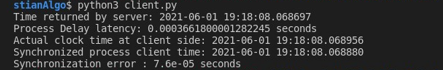
|  |
| --- |
| import datetime |
| import time |
| import socket |
|  |
| def initiateClockServer(): |
| s = socket.socket() |
| print("Socket successfully created") |
| port = 8011 |
| s.bind(('127.0.0.1', port)) |
| s.listen(5) |
| print("Socket is listening...") |
| while True: |
| connection, address = s.accept() |
| print('Server connected to', address) |
| serverDateTime = str(datetime.datetime.now()) |
| connection.send(serverDateTime.encode()) |
| connection.close() |
|  |
| if \_\_name\_\_ == '\_\_main\_\_': |
| initiateClockServer() |

client.py

|  |
| --- |
| import socket |
| import datetime |
| from dateutil import parser |
| from timeit import default\_timer as timer |
|  |
| def synchronizeTime(): |
| s = socket.socket() |
| port = 8011 |
| s.connect(('127.0.0.1', port)) |
| request\_time = timer() |
| server\_time = parser.parse(s.recv(1024).decode()) |
| response\_time = timer() |
| actual\_time = datetime.datetime.now() |
| print("Time returned by server: " + str(server\_time)) |
| process\_delay\_latency = response\_time - request\_time |
| print("Process Delay latency: " + str(process\_delay\_latency) + " |
| seconds") |
| print("Actual clock time at client side: " + str(actual\_time)) |
|  |
| client\_time = server\_time + \ |
| datetime.timedelta(seconds=(process\_delay\_latency) / 2) |
| print("Synchronized process client time: " + str(client\_time)) |
| error = actual\_time - client\_time |
| print("Synchronization error : " + str(error.total\_seconds()) + " |
| seconds") |
| s.close() |
|  |
| if \_\_name\_\_ == '\_\_main\_\_': |
| synchronizeTime() |

**Screenshot:**



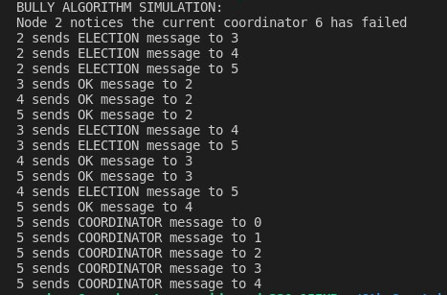


1. **Bully Algorithm CODE:**

|  |
| --- |
| import sys |
|  |
| noOfNodes = int(sys.argv[1]) |
| initiatorNode = int(sys.argv[2]) |
|  |
| def bully\_algorithm(): |
| print("BULLY ALGORITHM SIMULATION:") |
| print('Node %s notices the current coordinator %s has failed' |
| % |
| (initiatorNode, noOfNodes)) |
| biggerNodes = [] |
| for i in range(initiatorNode+1, noOfNodes): |
| print("%s sends ELECTION message to %s" % (initiatorNode, |
| i)) |
| biggerNodes.append(i) |
| for i in biggerNodes: |
| print("%s sends OK message to %s" % (i, initiatorNode)) |
|  |
| while len(biggerNodes) != 1: |
| i = biggerNodes[0] |
| for j in range(i+1, noOfNodes): |
| print("%s sends ELECTION message to %s" % (i, j)) |
| for k in range(i+1, noOfNodes): |
| print("%s sends OK message to %s" % (k, i)) |
| biggerNodes.remove(i) |
| newCoordinatorNode = biggerNodes[0] |
| for i in range(0, newCoordinatorNode): |
| print("%s sends COORDINATOR message to %s" % |
| (newCoordinatorNode, i)) |
|  |
| if \_\_name\_\_ == '\_\_main\_\_': |
| bully\_algorithm() |

**Screenshot:**





1. **Ring Algorithm CODE:**

|  |
| --- |
| import sys |
|  |
| noOfNodes = int(sys.argv[1]) |
| initiatorNode = int(sys.argv[2]) |
|  |
|  |
| def ring\_algorithm(): |
| print("RING ALGORITHM") |
| print('Node %s notices the current coordinator %s has failed' |
| % |
| (initiatorNode, noOfNodes)) |
| ELECTION = [] |
|  |
| i = initiatorNode |
| while True: |
| print("%s sends ELECTION message to %s" % (i, (i+1) % |
| noOfNodes)) |
| ELECTION.append(i) |
| print("Election Message elements: ", ELECTION) |
| i = (i+1) % noOfNodes |
| if i == initiatorNode: |
| break |
| max\_ele = max(ELECTION) |
| for i in range(0, noOfNodes): |
| print("%s sends %s COORDINATOR message to %s" % |
| (initiatorNode, max\_ele, i)) |
|  |
| if \_\_name\_\_ == '\_\_main\_\_': |
| ring\_algorithm() |

**Screenshot:**



