

| Experiment no. | Titel                             |
|----------------|-----------------------------------|
| 1              | Client Server                     |
| 2              | Client/Server using RPC/RMI       |
| 3              | Election Algorithm                |
| 4              | Clock Synchronization algorithms  |
| 5              | Lamport's Logical Clock           |
| 6              | Synchronization-Producer-Consumer |
| 7              | Stateful Server                   |
| 8              | Group Communication               |
| 9              | Load Balancing Algorithm          |
| 10             | Mutual Exclusion Algorithm        |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <p style="text-align: center;">JNIESTRT'S</p> <p style="text-align: center;"><b>SMT. INDIRA GANDHI COLLEGE OF ENGINEERING</b></p> <p style="text-align: center;">GHANSOLI, NAVI MUMBAI – 400701</p> <p style="text-align: center;">(Approved by AICTE New Delhi &amp; Govt. of Maharashtra, Affiliated to University of Mumbai)</p> <p style="text-align: center;"><b>COMPUTER ENGINEERING DEPARTMENT</b></p> <p style="text-align: center;"><b>ACADEMIC YEAR: 2020-21</b></p> |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|

## **Experiment No.: 01**

**Aim: Client/Server.**

| Date of Performance | Date of Submission | Marks (10)  |   |   |   |   | Sign / Remark |
|---------------------|--------------------|-------------|---|---|---|---|---------------|
|                     |                    | A           | B | C | D | E |               |
|                     |                    | 2           | 3 | 2 | 2 | 1 |               |
|                     |                    |             |   |   |   |   |               |
|                     |                    | Total Marks |   |   |   |   |               |
|                     |                    |             |   |   |   |   |               |

### **EXPERIMENT NO.:01**

**Aim:**Client/Server

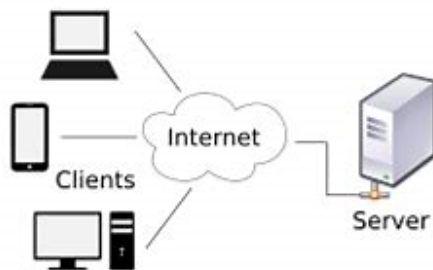
**Theory:**

A network application known as client-server architecture, sometimes known as a

client-server model, divides workloads and tasks between clients and servers housed on the same system or connected via a computer network. Client-server architecture often consists of workstations, PCs, or other devices belonging to many users that are linked to a central server over the Internet or another network. When the server receives a request for data from the client, it complies and transmits the requested data packets back to the user. A client sends a request using a network-capable device, the server accepts and processes the request, and then it provides the client with the response. This is an overview of how a client-server network operates.

### **Components of Client Server Architecture**

Client-server architecture requires three components to function. Workstations, servers, and networking devices comprise the three components. Let us go over them in depth for a better understanding.



**WORKSTATIONS** – Client computers are another term for workstations. Workstations act as servers' subordinates, sending requests to access shared files and databases. As a primary repository of files, programs, databases, and management policies, a server requests information from workstations and performs several functions. Workstations are governed by guidelines set by the server.

**SERVERS** – Servers are defined as high-performance computing devices that serve as centralized storage locations for network files, programs, databases, and policies. Servers have ample storage space and memory to handle multiple requests coming in from various workstations at the same time. In a client-server architecture, servers can simultaneously serve as mail servers, database servers, file servers, and domain controllers.

**NETWORKING DEVICES** – In a client-server design, networking devices function as a method for connecting workstations and servers. Numerous network operations square measure applied to employ a type of networking device. A hub,

as an example, is employed to link a server to various workstations. Information between 2 devices is often effectively transferred via repeaters. To segregate network segmentation, bridges square measure is used.

#### Working of the client-server network

Users must input the website URL or file when dealing with servers as clients. The DNS server then returns the IP address of the web server in response to the browser's request for the DNS server to seek up the web server's address. The server then returns the website's files once the browser makes an HTTP request to the WEB Server's IP. The webpage is shown when the browser renders the files.

#### Types of Client-Server Architecture

Multiple levels of functionality are present in a client-server architecture. More information will improve our understanding.

##### 1-TIER Architecture

In this subclass of client-server design, all choices, configuration options, and selling logic are contained on a single device. While 1-tier design provides a wide range of services, managing such an Associate in Nursing design might take time and effort. The volatility of information is typical of fault. It frequently results in the duplication of effort. The display, business, and information layers are just a few components that make up the 1-tier design. These layers incorporate the usage of a unique set of computer codes. In native systems or on shared media, the knowledge of this layer is usually unbroken.

##### 2- Tier Architecture

The best setting is formed by this design. This style helps maintain information and business logic on the client's or server's aspect. It happens with the shopper's computer program and the server's information. Due to the lack of intermediaries between client and server in the 2-tier architecture, it is faster than the 1-tier system. It's regularly employed to ensure client clarity. One of the famous examples of a 2-tier design is the online ticket reservation process.

##### 3- Tier Architecture

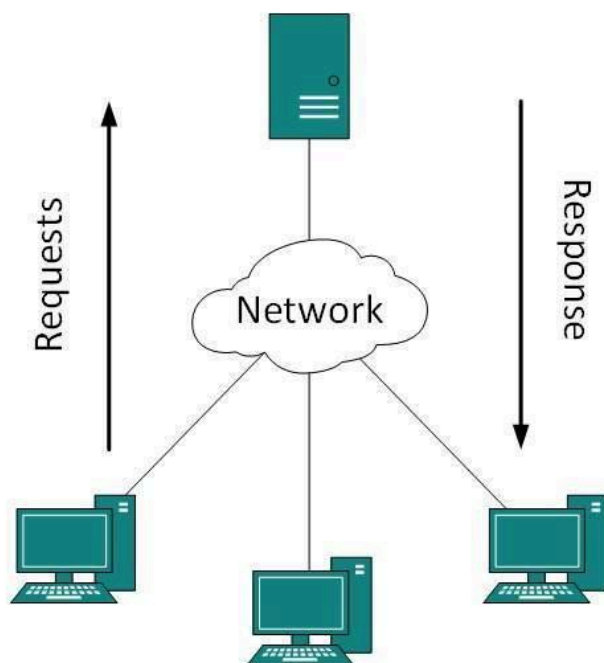
Contrary to 2-tier architecture, which has no middleware, 3-tier client-server design places middleware between the client and the server. The middleware will first acknowledge a request from the client to retrieve specific data from the server. Then it will be delivered to the server for additional processing. The identical

procedure will be used when the server replies to the client. The three main layers that make up the 3-tier architecture framework are the display layer, application layer, and database tier. All three layers are governed at various ends. The middleware and server are in charge of the application layer and database tier, respectively, while the client's device is in order of the presentation layer. Since it includes a third layer that offers data control, the three-tier architecture is more secure, has an invisible database structure, and ensures data integrity.

### N-Architecture

Multi-tier design and N-tier design are similar terms. This variety of design is the scaled-down version of the opposite 3. Every performance, show, application process, and information management functionality is often placed in this style as a separate layer.

In the client-server model, any process can act as Server or Client. It is not the type of machine, size of the machine, or its computing power which makes it a server; it is the ability of serving requests that makes a machine a server.



A system can act as Server and Client simultaneously. That is, one process is acting as Server and another is acting as a client. This may also happen that both client and server processes reside on the same machine.

### Sockets

In this paradigm, the process acting as Server opens a socket using a well-known (or known by client) port and waits until some client request comes. The second process acting as a Client also opens a socket but instead of waiting for an incoming request, the client processes 'requests first'.

When the request is reached to the server, it is served. It can either be an information sharing or resource request.

**Input:**

**Client:**

```
import socket                # Import socket module

host = socket.gethostname()  # Get local machine name
port = 12346                 # Reserve a port for your service.
conn = socket.socket()       # Create a socket object

conn.connect((host, port))

conn.sendall(b'Connected. Wait for data...')

while 1:

    intosend = input("message to send:")
    conn.sendall(intosend.encode('utf-8'))
    #data received back from sever
    data = conn.recv(1024)
    print("Data: ", data.decode('utf-8'))
    conn.close()             # Close the socket when done

print(data.decode("utf-8"))
```

**Server:**

```
import socket
import random
s = socket.socket()          # Create a socket object
host = socket.gethostname()  # Get local machine name
```

```

port = 12345                                # Reserve a port for your service.
s = socket.socket()
s.bind((host, port))                        # Bind to the port

s.listen(5)                                # Now wait for client connection.

conn, addr = s.accept()
print('Got connection from ', addr[0], '(', addr[1], ')')

while True:
    data = conn.recv(1024)
    print(data.decode("utf-8"))
    if not data:
        break
    conn.sendall(data)

conn.close()

print('Thank you for connecting')

```

## Output:

The screenshot shows a code editor with a project named 'DCLab'. The file 'Exp\_1\_client.py' is open, displaying the following Python code:

```

1  import socket                                # Import socket module
2
3  host = socket.gethostname()                    # Get local machine name
4  port = 12346                                  # Reserve a port for your service.
5  conn = socket.socket()                        # Create a socket object
6
7  conn.connect((host, port))
8
9  conn.sendall(b'Connected. Wait for data...')
10
11
12  while 1:
13      intosend = input('message to send:')
14      conn.sendall(intosend.encode('utf-8'))
15      #data received back from seven
16      data = conn.recv(1024)
17      print('Data: ', data.decode('utf-8'))
18      conn.close()                               # Close the socket when done
19
20  while 1

```

The output window shows the following text:

```

C:\Users\Lenovo\Desktop\DCLab\.venv\Scripts\python.exe C:\Users\Lenovo\Desktop\DCLab\Exp_1_client.py
message to send:Hello, client here
Data: Thank you for connecting
message to send:

```


The status bar at the bottom indicates the file is 'Exp\_1\_client.py', the encoding is 'UTF-8', and the Python version is '3.12 (DCLab)'.

```
1 import socket
2 import threading
3 import sys
4
5 s = socket.socket()           # Create a socket object
6 host = socket.gethostname()   # Get local machine name
7
8 port = 12346                  # Reserve a port for your service.
9 s = socket.socket()
10 s.bind((host, port))         # Bind to the port
11
12 s.listen(5)                  # Now wait for client connection.
13
14 !usage
15 def processMessages(conn, addr):
16     while True:
17         try:
18             data = conn.recv(1024)
19             if not data:
20                 conn.close()
```

```
C:\Users\Lenovo\Desktop\DCLab\.venv\Scripts\python.exe C:\Users\Lenovo\Desktop\DCLab\Test.py
Got connection from 192.168.15.107 ( 63142 )
Connected. Wait for data...
Helloo, client here
```

**Conclusion:** The Client-server architecture helps us to communicate data with other computers, devices, and organizations. A networking model includes client-server architecture that enables multi-user updates via a graphical user interface to a shared database. Large and small businesses leverage networking to grow and digitize their operations, advertise their goods, and better understand news and events specific to their sectors.



|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <p style="text-align: center;">JNIESTRT'S</p> <p style="text-align: center;"><b>SMT. INDIRA GANDHI COLLEGE OF ENGINEERING</b></p> <p style="text-align: center;">GHANSOLI, NAVI MUMBAI – 400701</p> <p style="text-align: center;">(Approved by AICTE New Delhi &amp; Govt. of Maharashtra, Affiliated to University of Mumbai)</p> <p style="text-align: center;"><b>COMPUTER ENGINEERING DEPARTMENT</b></p> <p style="text-align: center;"><b>ACADEMIC YEAR: 2020-21</b></p> |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|

## **Experiment No.: 02**

**Aim: Client/Server using RPC/RMI.**

| Date of Performance | Date of Submission | Marks (10)  |   |   |   |   | Sign / Remark |
|---------------------|--------------------|-------------|---|---|---|---|---------------|
|                     |                    | A           | B | C | D | E |               |
|                     |                    | 2           | 3 | 2 | 2 | 1 |               |
|                     |                    |             |   |   |   |   |               |
|                     |                    | Total Marks |   |   |   |   |               |
|                     |                    |             |   |   |   |   |               |

### **EXPERIMENT NO.:02**

**Aim:** Client/Server using RPC/RMI

**Theory:**

**RPC(Remote Procedure Call):**

This is a mechanism where one process interacts with another by means of

procedure calls. One process (client) calls the procedure lying on a remote host. The process on the remote host is said to be Server. Both processes are allocated stubs. This communication happens in the following way:

- The client process calls the client stub. It passes all the parameters pertaining to the program local to it.
- All parameters are then packed (marshalled) and a system call is made to send them to the other side of the network.
- Kernel sends the data over the network and the other end receives it.
- The remote host passes data to the server stub where it is unmarshalled.
- The parameters are passed to the procedure and the procedure is then executed.
- The result is sent back to the client in the same manner.

Three types of RPC are:

#### 1. Callback RPC

This type of RPC enables a P2P paradigm between participating processes. It helps a process to be both client and server services.

Functions of Callback RPC:

- Remotely processed interactive application problems
- Offers server with clients handle
- Callback makes the client process wait
- Manage callback deadlocks
- It facilitates a peer-to-Peer paradigm among participating processes.

#### 2. Broadcast RPC

Broadcast RPC is a client's request that is broadcast on the network, processed by all servers which have the method for processing that request.

Functions of Broadcast RPC:

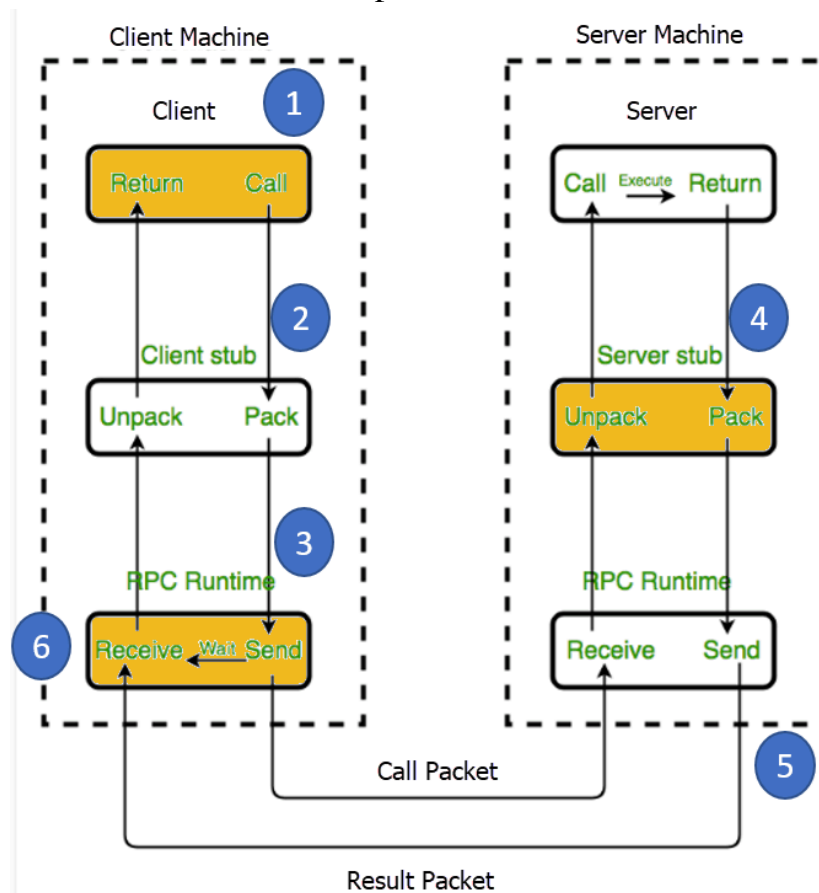
- Allows you to specify that the client's request message has to be broadcasted.
- You can declare broadcast ports.
- It helps to reduce the load on the physical network

#### 3. Batch-mode RPC

Batch-mode RPC helps to queue, separate RPC requests, in a transmission buffer, on the client-side, and then send them on a network in one batch to the server.

Functions of Batch-mode RPC:

- It minimizes overhead involved in sending a request as it sends them over the network in one batch to the server.
- This type of RPC protocol is only efficient for the application that needs lower call rates.
- It needs a reliable transmission protocol.



### RPC Architecture

RPC architecture has mainly five components of the program:

Client  
Client Stub  
RPC Runtime  
Server Stub

Server  
RPC Architecture  
RPC Architecture

## **RMI:**

Remote Method Invocation (RMI) is an API that allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine. Through RMI, an object running in a JVM present on a computer (Client-side) can invoke methods on an object present in another JVM (Server-side). RMI creates a public remote server object that enables client and server-side communications through simple method calls on the server object.

**Stub Object:** The stub object on the client machine builds an information block and sends this information to the server.

The block consists of

- An identifier of the remote object to be used
- Method name which is to be invoked
- Parameters to the remote JVM

**Skeleton Object:** The skeleton object passes the request from the stub object to the remote object. It performs the following tasks

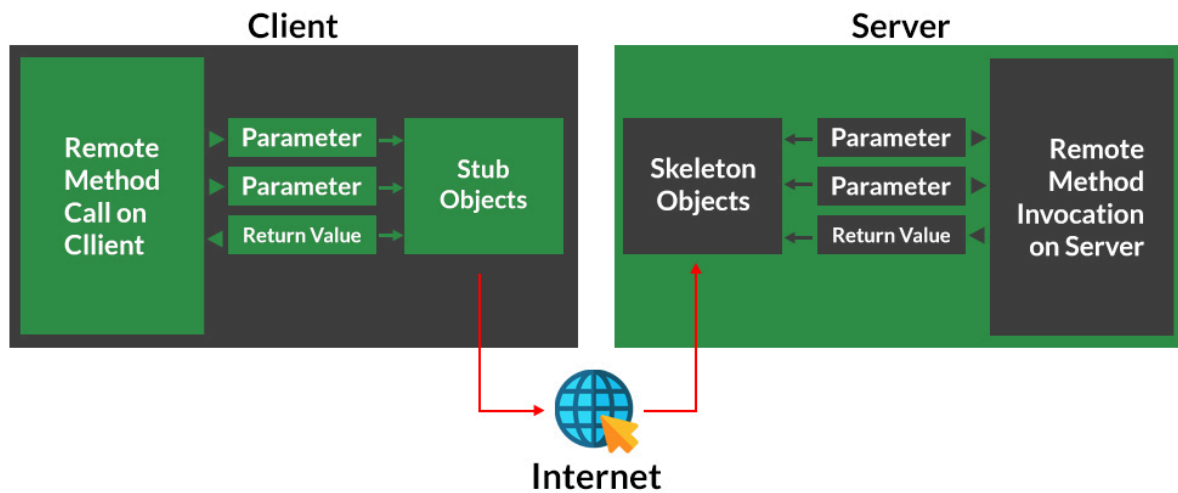
It calls the desired method on the real object present on the server.

It forwards the parameters received from the stub object to the method.

### **Working of RMI**

The communication between client and server is handled by using two intermediate objects: Stub object (on client side) and Skeleton object (on server-side) as also can be depicted from below media as follows:

# Working of RMI



These are the steps to be followed sequentially to implement Interface as defined below as follows:

- Defining a remote interface
- Implementing the remote interface
- Creating Stub and Skeleton objects from the implementation class using `rmic` (RMI compiler)
- Start the `rmiregistry`
- Create and execute the server application program
- Create and execute the client application program.

## Input:

### RPC clinic:

```
import xmlrpc.client
proxy = xmlrpc.client.ServerProxy("http://localhost:8000/")
print("factorial of 3 is : %s" % str(proxy.factorial_rpc(3)))
print("factorial of 5 is : %s" % str(proxy.factorial_rpc(5)))
```

### RPC server:

```
from xmlrpc.server import SimpleXMLRPCServer
```

```
def factorial(n):
    fact = 1
```

```
for i in range(1,n+1):
    fact=fact*i
return fact
```

```
server = SimpleXMLRPCServer(("localhost", 8000), logRequests=True)
server.register_function(factorial, "factorial_rpc")
```

```
try:
    print("Starting and listening on port 8000...")
    print("Press Ctrl + C to exit.")
    server.serve_forever()
```

```
except:
    print("Exit.")
```

### **RMI clinic:**

```
import Pyro4
```

```
uri = "PYRO:obj_07bd720c7d9542a0b7673fc0abf0b1e8@localhost:51706" #
Replace with actual URI
```

```
with Pyro4.Proxy(uri) as proxy:
    print(proxy.greet("Bard"))
    result = proxy.add(4, 5)
    print(f"4 + 5 = {result}")
```

### **RMI server:**

```
import Pyro4
```

```
@Pyro4.expose
class RemoteObject:
    def greet(self, name):
        return f"Hello, {name}!"

    def add(self, x, y):
        return x + y
```

```
daemon = Pyro4.Daemon()
uri = daemon.register(RemoteObject)
print(f"Server object URI: {uri}")
```

```
daemon.requestLoop()
```

## Output:

### RPC client:

The screenshot shows a PyCharm IDE with two files open: `Exp_2_RPC_serv.py` and `Exp_2_RPC_client.py`. The `Exp_2_RPC_serv.py` file contains the following code:

```
1 from xmlrpc.server import SimpleXMLRPCServer
2
3 1 usage
4
5 def factorial(n):
6     fact = 1
7     for i in range(1,n+1):
8         fact=fact*i
9     return fact
10
11 server = SimpleXMLRPCServer(addr= ("localhost", 8000), logRequests=True)
12 server.register_function(factorial, name= "factorial_rpc")
13
14 try:
15     print("Starting and listening on port 8000...")
16     print("Press Ctrl + C to exit.")
17     server.serve_forever()
18
19 except:
```

The `Exp_2_RPC_client.py` file is empty. The Run console shows the output of the server script:

```
C:\Users\Lenovo\Desktop\DCLab\.venv\Scripts\python.exe C:\Users\Lenovo\Desktop\DCLab\Exp_2_RPC_serv.py
Starting and listening on port 8000...
Press Ctrl + C to exit.
127.0.0.1 - - [16/Jan/2024 12:13:50] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [16/Jan/2024 12:13:52] "POST / HTTP/1.1" 200 -
```

### RPC server:

```
Project
├── DCLab
│   ├── .venv
│   │   ├── library root
│   │   ├── Exp_1_client.py
│   │   ├── Exp_1_server.py
│   │   ├── Exp_2_RPC_client.py
│   │   ├── Exp_2_RPC_serv.py
│   │   └── Test.py
│   └── External Libraries
│       └── Scratches and Consoles
└── ...

Exp_2_RPC_serv.py
1 from xmlrpc.server import SimpleXMLRPCServer
2
3 usage
4 def factorial(n):
5     fact = 1
6     for i in range(1,n+1):
7         fact=fact*i
8     return fact
9
10 server = SimpleXMLRPCServer(addr=('localhost', 8000), logRequests=True)
11 server.register_function(factorial, name='factorial_rpc')
12
13 try:
14     print("Starting and listening on port 8000...")
15     print("Press Ctrl + C to exit.")
16     server.serve_forever()
17 except:
18     print("Exit.")
19
Exp_2_RPC_client.py
1 from xmlrpc.client import proxy
2
3 def main():
4     s = proxy('http://localhost:8000')
5     print(s.factorial(5))
6
7 if __name__ == '__main__':
8     main()

Run
Exp_2_RPC_serv.py
Exp_2_RPC_client.py

C:\Users\Lenovo\Desktop\DCLab\.venv\Scripts\python.exe C:\Users\Lenovo\Desktop\DCLab\Exp_2_RPC_serv.py
Starting and listening on port 8000...
Press Ctrl + C to exit.
127.0.0.1 - - [16/Jan/2024 12:13:50] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [16/Jan/2024 12:13:52] "POST / HTTP/1.1" 200 -

Waiting for process detach
```

## RMI client:

```
PROBLEMS PORTS OUTPUT TERMINAL SEARCH ERROR

PS C:\Users\anish rane> & C:/python/python.exe "c:/Users/anish rane/Desktop/BE/sem 8/dc/exp 2/RMI_client.py"
Hello, man!
4 + 5 = 9
PS C:\Users\anish rane>
```

## RMI server:

```
PROBLEMS PORTS OUTPUT TERMINAL SEARCH ERROR


Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\anish rane\Desktop\BE\sem 8\dc\exp 2> & 'C:\python\python.exe' 'c:\Users\anish rane\.vscode\extensions\ms-python.python-2023.22.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '54898' '--' 'C:\Users\anish rane\Desktop\BE\sem 8\dc\exp 2\RMI_server.py'
Server object URI: PYRO:obj_f91b4b56024e41539276c6651d14b4b7@localhost:54903
[]
```

## Conclusion:

Use RPC if you prioritize portability, lightweight solutions, and platform independence, and are willing to implement additional security measures. Use RMI if you need secure communication, seamless integration with Java, and automatic exception handling, but your application is restricted to the Java ecosystem and can tolerate potentially higher overhead.



|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <p style="text-align: center;">JNIESTRT'S</p> <p style="text-align: center;"><b>SMT. INDIRA GANDHI COLLEGE OF ENGINEERING</b></p> <p style="text-align: center;">GHANSOLI, NAVI MUMBAI – 400701</p> <p style="text-align: center;">(Approved by AICTE New Delhi &amp; Govt. of Maharashtra, Affiliated to University of Mumbai)</p> <p style="text-align: center;"><b>COMPUTER ENGINEERING DEPARTMENT</b></p> <p style="text-align: center;"><b>ACADEMIC YEAR: 2020-21</b></p> |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|

## **Experiment No.: 03**

**Aim: Election Algorithm.**

| Date of Performance | Date of Submission | Marks (10)  |   |   |   |   | Sign / Remark |
|---------------------|--------------------|-------------|---|---|---|---|---------------|
|                     |                    | A           | B | C | D | E |               |
|                     |                    | 2           | 3 | 2 | 2 | 1 |               |
|                     |                    |             |   |   |   |   |               |
|                     |                    | Total Marks |   |   |   |   |               |
|                     |                    |             |   |   |   |   |               |

### **EXPERIMENT NO.:03**

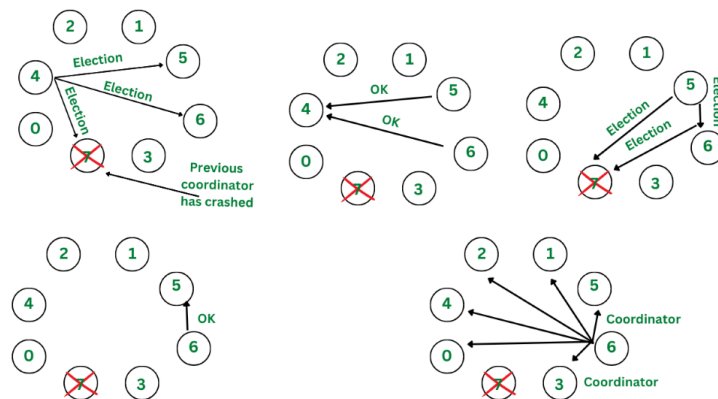
**Aim:**Election Algorithm

**Theory:**

**Bully:**

Bully Algorithm starts its process of selecting a coordinator when a working process notices that the coordinator is not responding to it and then initiates an election. The selection of a coordinator sends an election message to all the other available processes. If none of the process responds to its message the process itself wins the election and becomes the coordinator.

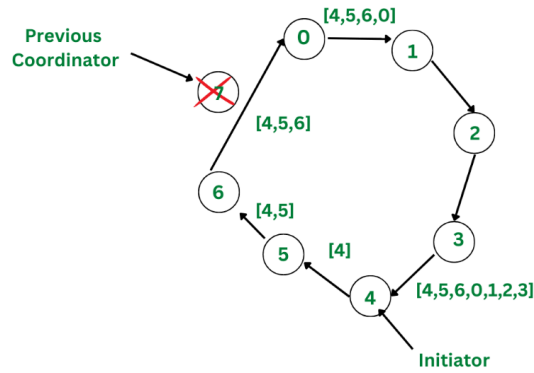
Meanwhile, if any process that has a higher ID than the initiating process answers, it takes charge of initiating the process. In this algorithm, the bully guy that is the participating process that has the highest ID becomes the coordinator. This approach is efficient in terms of fault tolerance but complex to implement.



### Ring:

Ring Algorithm starts its process of selecting of coordinator once any process connected in the ring notices that the coordinator is not functioning. The initiating process then prepares an election message that contains its own process number and then sends this message to its successor in the ring. If the successor is not functioning then it kills that successor and sends it to the next one.

Every next process then adds its own process number to the message and delivers the message to the next process connected in the ring. It continues until the message again reaches the initiating process. Now the message consists of the process number of all the functioning processes. The process that has the highest ID wins the election. The initiating process then sends another message in the ring about the winning process.



Ring algorithm

**Input:**

**Bully:**

class Pro:

```
def __init__(self, id):
    self.id = id
    self.act = True
```

class GFG:

```
def __init__(self):
    self.TotalProcess = 0
    self.process = []
```

```
def initialiseGFG(self, num_processes):
    print(f"No of processes: {num_processes}")
    self.TotalProcess = num_processes
    self.process = [Pro(i) for i in range(self.TotalProcess)]
```

```
def Election(self):
    print("Process no " + str(self.process[self.FetchMaximum()].id) + " fails")
    self.process[self.FetchMaximum()].act = False
    print("Election Initiated by 2")
    initializedProcess = 2
```

```
old = initializedProcess
newer = old + 1
```

```
while (True):
```

```

        if (self.process[newer].act):
            print("Process " + str(self.process[old].id) + " pass Election(" +
str(self.process[old].id) + ") to " + str(self.process[newer].id))
            old = newer
            newer = (newer + 1) % self.TotalProcess
            if (newer == initializedProcess):
                break

    print("Process " + str(self.process[self.FetchMaximum()].id) + " becomes
coordinator")
    coord = self.process[self.FetchMaximum()].id

    old = coord
    newer = (old + 1) % self.TotalProcess
    while (True):
        if (self.process[newer].act):
            print("Process " + str(self.process[old].id) + " pass Coordinator(" +
str(coord) + ") message to process " + str(self.process[newer].id))
            old = newer
            newer = (newer + 1) % self.TotalProcess
            if (newer == coord):
                print("End Of Election ")
                break

def FetchMaximum(self):
    maxId = -9999
    ind = 0
    for i in range(self.TotalProcess):
        if (self.process[i].act and self.process[i].id > maxId):
            maxId = self.process[i].id
            ind = i
    return ind

def main():
    num_processes = int(input("Enter the number of processes: "))
    obj = GFG()
    obj.initialiseGFG(num_processes)
    obj.Election()

```

```
if __name__ == "__main__":  
    main()
```

### **Ring:**

```
class Ring:  
    def __init__(self):  
        self.sc = None  
        self.no_of_processes = 0  
        self.processes = []  
  
    def initialize_ring(self):  
        self.no_of_processes = int(input("Enter no of processes: "))  
        self.processes = [Process(i) for i in range(self.no_of_processes)]  
  
    def get_max(self):  
        max_id = -99  
        max_id_index = 0  
        for i in range(len(self.processes)):  
            if self.processes[i].active and self.processes[i].id > max_id:  
                max_id = self.processes[i].id  
                max_id_index = i  
        return max_id_index  
  
    def perform_election(self):  
        print(f"Process no {self.processes[self.get_max()].id} fails")  
        self.processes[self.get_max()].active = False  
        print("Election Initiated by")  
        initiator_process = int(input())  
        prev = initiator_process  
        next = prev + 1  
        while True:  
            if self.processes[next % self.no_of_processes].active:  
                print(f"Process {self.processes[prev].id} pass  
Election({self.processes[prev].id}) to {self.processes[next %  
self.no_of_processes].id}")  
                prev = next % self.no_of_processes  
                next = (next + 1) % self.no_of_processes
```

```

        if next == initiator_process:
            break

    print(f"Process {self.processes[self.get_max()].id} becomes coordinator")
    coordinator = self.processes[self.get_max()].id
    prev = coordinator
    next = (prev + 1) % self.no_of_processes
    while True:
        if self.processes[next].active:
            print(f"Process {self.processes[prev].id} pass
Coordinator({coordinator}) message to process {self.processes[next].id}")
            prev = next
            next = (next + 1) % self.no_of_processes
        if next == coordinator:
            print("End Of Election")
            break

if __name__ == "__main__":
    r = Ring()
    r.initialize_ring()
    r.perform_election()

```

**Output:**

**Bully:**

```

Enter the number of processes: 10
No of processes: 10
Process no 9 fails
Election Initiated by 2
Process 2 pass Election(2) to 3
Process 3 pass Election(3) to 4
Process 4 pass Election(4) to 5
Process 5 pass Election(5) to 6
Process 6 pass Election(6) to 7
Process 7 pass Election(7) to 8
Process 8 pass Election(8) to 0
Process 0 pass Election(0) to 1
Process 8 becomes coordinator
Process 8 pass Coordinator(8) message to process 0
Process 0 pass Coordinator(8) message to process 1
Process 1 pass Coordinator(8) message to process 2
Process 2 pass Coordinator(8) message to process 3
Process 3 pass Coordinator(8) message to process 4
Process 4 pass Coordinator(8) message to process 5
Process 5 pass Coordinator(8) message to process 6
Process 6 pass Coordinator(8) message to process 7
End Of Election

```

### Ring:

```

Enter no of processes: 3
Process no 2 fails
Election Initiated by
1
Process 1 pass Election(1) to 0
Process 1 becomes coordinator
Process 1 pass Coordinator(1) message to process 0
End Of Election

```

**Conclusion:** Election Algorithms such as ring and Bully algorithms play a very vital role while selecting a coordinating process among all the available processes. This coordinator process is responsible for the proper functioning and execution of the operation. Comparing both the Algorithms the approaches are different but the ring algorithm is less complex as compared to the bully algorithm whereas the bully algorithm is efficient for fault tolerance as the communication takes place from one process to another.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <p style="text-align: center;">JNIESTRT'S</p> <p style="text-align: center;"><b>SMT. INDIRA GANDHI COLLEGE OF ENGINEERING</b></p> <p style="text-align: center;">GHANSOLI, NAVI MUMBAI – 400701</p> <p style="text-align: center;">(Approved by AICTE New Delhi &amp; Govt. of Maharashtra, Affiliated to University of Mumbai)</p> <p style="text-align: center;"><b>COMPUTER ENGINEERING DEPARTMENT</b></p> <p style="text-align: center;"><b>ACADEMIC YEAR: 2020-21</b></p> |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|

## **Experiment No.: 04**

**Aim: Clock Synchronization algorithms**

| Date of Performance | Date of Submission | Marks (10)  |   |   |   |   | Sign / Remark |
|---------------------|--------------------|-------------|---|---|---|---|---------------|
|                     |                    | A           | B | C | D | E |               |
|                     |                    | 2           | 3 | 2 | 2 | 1 |               |
|                     |                    |             |   |   |   |   |               |
|                     |                    | Total Marks |   |   |   |   |               |
|                     |                    |             |   |   |   |   |               |

### **EXPERIMENT NO.:04**

**Aim:** Clock Synchronization algorithms

**Theory:**

The Berkeley Algorithm is a centralized clock synchronization technique used in



distributed systems with no direct access to external time sources. It works by:

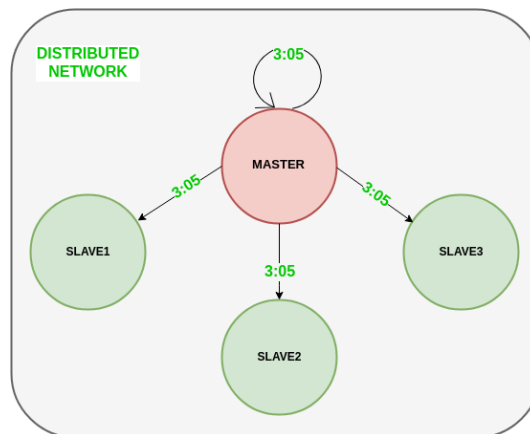
**Electing a master node:** The nodes participate in a leader election algorithm to choose a master node responsible for maintaining global time.

**Time exchange:** The master periodically sends its timestamp to all other nodes (slaves) using Cristian's algorithm, which accounts for message delays.

**Clock adjustment:** Each slave adjusts its clock based on the received timestamp and estimated round-trip delay (average of sending and receiving delays).

**Error correction:** The algorithm iteratively refines the clock adjustments through subsequent rounds of time exchange and adjustment.

Diagram:



- Master sends its timestamp ( $T_m$ ) to all slaves.
- Each slave calculates its local clock deviation from the master ( $\theta_i = T_m - T_i + RTT/2$ ).
- Each slave adjusts its clock by  $\theta_i$ .
- Repeat steps 1-4 until desired accuracy is achieved.

Benefits:

- Simple and efficient.
- Works with imperfect clocks and varying communication delays.

Drawbacks:

- Single point of failure (master node).
- Performance depends on master responsiveness and network overhead.

**Input:**from datetime import datetime

```
class Berkeley:
```

```
    def diff(self, h, m, s, nh, nm, ns):
```

```
        dh = h - nh
```

```
        dm = m - nm
```

```
        ds = s - ns
```

```
        diff = (dh * 60 * 60) + (dm * 60) + ds
```

```
        return diff
```

```
    def average(self, diff, n):
```

```
        _sum = sum(diff)
```

```
        average = _sum / (n + 1)
```

```
        print("The average of all time differences is", average)
```

```
        return average
```

```
    def sync(self, diff, n, h, m, s, nh, nm, ns, average):
```

```
        for i in range(n):
```

```
            diff[i] += average
```

```
            dh = int(diff[i] / (60 * 60))
```

```
            diff[i] %= (60 * 60)
```

```
            dm = int(diff[i] / 60)
```

```
            diff[i] %= 60
```

```
            ds = int(diff[i])
```

```
            nh[i] += dh
```

```
            if nh[i] > 23:
```

```
                nh[i] %= 24
```

```
            nm[i] += dm
```

```
            if nm[i] > 59:
```

```
                nh[i] += 1
```

```
                nm[i] %= 60
```

```
            ns[i] += ds
```

```
            if ns[i] > 59:
```

```
                nm[i] += 1
```

```
                ns[i] %= 60
```

```
            if ns[i] < 0:
```

```
                nm[i] -= 1
```

```
                ns[i] += 60
```

```
    h += int(average / (60 * 60))
```

```

if h > 23:
    h %= 24
m += int(average / (60 * 60 * 60))
if m > 59:
    h += 1
    m %= 60
s += int(average % (60 * 60 * 60))
if s > 59:
    m += 1
    s %= 60
if s < 0:
    m -= 1
    s += 60

print("The synchronized clocks are:\nTime Server -->", h, ":", m, ":", s)
for i in range(n):
    print("Node", (i + 1), "--->", nh[i], ":", nm[i], ":", ns[i])

```

```

def main():
    b = Berkeley()
    date = datetime.now()
    print("Enter number of nodes:")
    n = int(input())

    h = date.hour
    m = date.minute
    s = date.second

    nh = [0] * n
    nm = [0] * n
    ns = [0] * n

    for i in range(n):
        print("Enter time for node", (i + 1), "\nHours:")
        nh[i] = int(input())
        print("Minutes:")
        nm[i] = int(input())
        print("Seconds:")

```

```

ns[i] = int(input())

for i in range(n):
    print("Time Server sent time", h, ":", m, ":", s, "to node", (i + 1))

diff = [b.diff(h, m, s, nh[i], nm[i], ns[i]) for i in range(n)]
for i in range(n):
    print("Node", (i + 1), "sent time difference of", int(diff[i]), "to Time Server.")

average = b.average(diff, n)
b.sync(diff, n, h, m, s, nh, nm, ns, average)

if __name__ == "__main__":
    main()

```

### Output:

```


Enter number of nodes:
3
Enter time for node 1
Hours:
11
Minutes:
15
Seconds:
36
Enter time for node 2
Hours:
19
Minutes:
59
Seconds:
59
Enter time for node 3

```

```

Hours:
23
Minutes:
59
Seconds:
59
Time Server sent time 12 : 22 : 8 to node 1
Time Server sent time 12 : 22 : 8 to node 2
Time Server sent time 12 : 22 : 8 to node 3
Node 1 sent time difference of 3992 to Time Server.
Node 2 sent time difference of -27471 to Time Server.
Node 3 sent time difference of -41871 to Time Server.
The average of all time differences is -16337.5
The synchronized clocks are:
Time Server --> 8 : 23 : 50
Node 1 ---> 8 : 49 : 50
Node 2 ---> 8 : 49 : 50
Node 3 ---> 8 : 49 : 50

```

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                      |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| <p style="text-align: center;">JNIESTRT'S</p> <p style="text-align: center;"><b>SMT. INDIRA GANDHI COLLEGE OF ENGINEERING</b></p> <p style="text-align: center;">GHANSOLI, NAVI MUMBAI – 400701</p> <p style="text-align: center;">(Approved by AICTE New Delhi &amp; Govt. of Maharashtra, Affiliated to University of Mumbai)</p> <p style="text-align: center;"><b>COMPUTER ENGINEERING DEPARTMENT</b></p> <p style="text-align: center;"><b>ACADEMIC YEAR: 2020-21</b></p> |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|

## **Experiment No.: 05**

**Aim: Lamport's Logical Clock.**

| Date of Performance | Date of Submission | Marks (10) |   |   |   |   | Sign / Remark |
|---------------------|--------------------|------------|---|---|---|---|---------------|
|                     |                    | A          | B | C | D | E |               |

|  |  |             |   |   |   |   |  |
|--|--|-------------|---|---|---|---|--|
|  |  | 2           | 3 | 2 | 2 | 1 |  |
|  |  |             |   |   |   |   |  |
|  |  | Total Marks |   |   |   |   |  |
|  |  |             |   |   |   |   |  |

### **EXPERIMENT NO.:05**

**Aim:**Lamport's Logical Clock

**Theory:**

Lamport's Logical Clock was created by Leslie Lamport. It is a procedure to determine the order of events occurring. It provides a basis for the more advanced Vector Clock Algorithm. Due to the absence of a Global Clock in a Distributed Operating System Lamport Logical Clock is needed.

Algorithm:

- Happened before relation( $\rightarrow$ ):  $a \rightarrow b$ , means 'a' happened before 'b'.
- Logical Clock: The criteria for the logical clocks are:
- [C1]:  $C_i(a) < C_i(b)$ , [  $C_i \rightarrow$  Logical Clock, If 'a' happened before 'b', then time of 'a' will be less than 'b' in a particular process. ]
- [C2]:  $C_i(a) < C_j(b)$ , [ Clock value of  $C_i(a)$  is less than  $C_j(b)$  ]

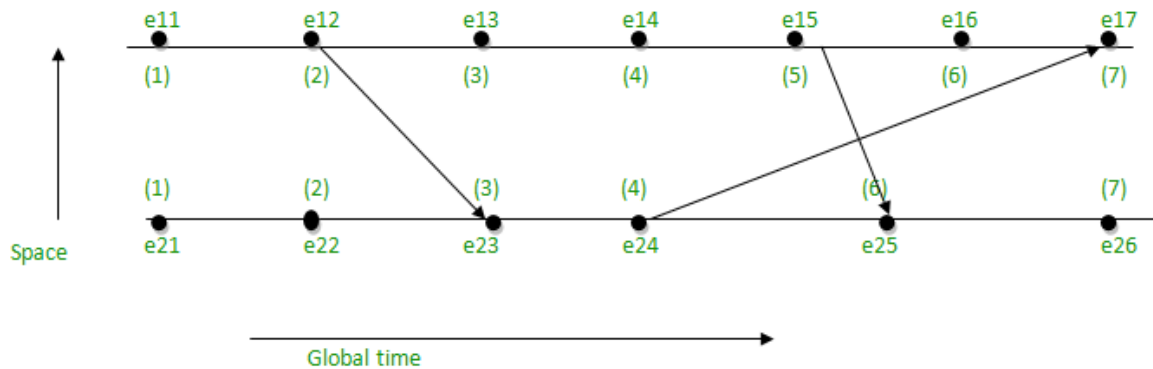
Reference:

- Process:  $P_i$
- Event:  $E_{ij}$ , where i is the process in number and j: jth event in the ith process.
- $t_m$ : vector time span for message m.
- $C_i$  vector clock associated with process  $P_i$ , the jth element is  $C_i[j]$  and contains  $P_i$ 's latest value for the current time in process  $P_j$ .
- d: drift time, generally d is 1.

Implementation Rules[IR]:

- [IR1]: If  $a \rightarrow b$  ['a' happened before 'b' within the same process] then,  $C_i(b) = C_i(a) + d$
- [IR2]:  $C_j = \max(C_j, t_m + d)$  [If there's more number of processes, then  $t_m =$  value of  $C_i(a)$ ,  $C_j = \max$  value between  $C_j$  and  $t_m + d$ ]

For Example:



Take the starting value as 1, since it is the 1st event and there is no incoming value at the starting point:

$$e11 = 1$$

$$e21 = 1$$

The value of the next point will go on increasing by  $d$  ( $d = 1$ ), if there is no incoming value i.e., to follow [IR1].

$$e12 = e11 + d = 1 + 1 = 2$$

$$e13 = e12 + d = 2 + 1 = 3$$

$$e14 = e13 + d = 3 + 1 = 4$$

$$e15 = e14 + d = 4 + 1 = 5$$

$$e16 = e15 + d = 5 + 1 = 6$$

$$e22 = e21 + d = 1 + 1 = 2$$

$$e24 = e23 + d = 3 + 1 = 4$$

$$e26 = e25 + d = 6 + 1 = 7$$

When there will be an incoming value, then follow [IR2] i.e., take the maximum value between  $C_j$  and  $T_m + d$ .

$$e17 = \max(7, 5) = 7, [e16 + d = 6 + 1 = 7, e24 + d = 4 + 1 = 5, \text{maximum among } 7 \text{ and } 5 \text{ is } 7]$$

$$e23 = \max(3, 3) = 3, [e22 + d = 2 + 1 = 3, e12 + d = 2 + 1 = 3, \text{maximum among } 3 \text{ and } 3 \text{ is } 3]$$

$$e25 = \max(5, 6) = 6, [e24 + 1 = 4 + 1 = 5, e15 + d = 5 + 1 = 6, \text{maximum among } 5 \text{ and } 6 \text{ is } 6]$$

Limitation:

- In case of [IR1], if  $a \rightarrow b$ , then  $C(a) < C(b) \rightarrow \text{true}$ .
- In case of [IR2], if  $a \rightarrow b$ , then  $C(a) < C(b) \rightarrow$  May be true or may not be true.

**Input:**

```
def max1(a, b):
```

```
    """Returns the maximum of two values."""
```

```
    return max(a, b)
```

```
def display(e1, e2, p1, p2):
```

```
    """Prints the logical timestamps of events in P1 and P2."""
```

```
    print("\nThe timestamps of events in P1:")
```

```
    print(*p1)
```

```
    print("\nThe timestamps of events in P2:")
```

```
    print(*p2)
```

```
def lamport_logical_clock(e1, e2, m):
```

```
    """Calculates the logical timestamps of events."""
```

```
    p1 = [i + 1 for i in range(e1)] # Initialize p1
```

```
    p2 = [i + 1 for i in range(e2)] # Initialize p2
```

```
    print(" ", *[f"e2{i+1}" for i in range(e2)])
```

```
    for i in range(e1):
```

```
        print(f"e1{i+1}", *m[i])
```

```
    for i in range(e1):
```

```
        for j in range(e2):
```

```
            if m[i][j] == 1: # Message sent
```

```
                p2[j] = max1(p2[j], p1[i] + 1)
```

```
                p2[j + 1:] = [p2[k - 1] + 1 for k in range(j + 1, e2)]
```

```
            elif m[i][j] == -1: # Message received
```

```
                p1[i] = max1(p1[i], p2[j] + 1)
```

```
                p1[i + 1:] = [p1[k - 1] + 1 for k in range(i + 1, e1)]
```

```
    display(e1, e2, p1, p2)
```

```
# Driver code
```



```

if __name__ == "__main__":
    e1 = int(input("Enter the number of events in process 1 (e1): "))
    e2 = int(input("Enter the number of events in process 2 (e2): "))

    m = []
    for i in range(e1):
        row = []
        for j in range(e2):
            value = int(input(f"Enter 1 if message is sent from e{i+1} to e{j+1}, -1 if
received, 0 otherwise: "))
            row.append(value)
        m.append(row)

    lamport_logical_clock(e1, e2, m)

```

### Output:

```

Enter the number of events in process 1 (e1): 5
Enter the number of events in process 2 (e2): 3
Enter 1 if message is sent from e1 to e1, -1 if received, 0 otherwise: 0
Enter 1 if message is sent from e1 to e2, -1 if received, 0 otherwise: 0
Enter 1 if message is sent from e1 to e3, -1 if received, 0 otherwise: 1
Enter 1 if message is sent from e2 to e1, -1 if received, 0 otherwise: 0
Enter 1 if message is sent from e2 to e2, -1 if received, 0 otherwise: 0
Enter 1 if message is sent from e2 to e3, -1 if received, 0 otherwise: 1
Enter 1 if message is sent from e3 to e1, -1 if received, 0 otherwise: 0
Enter 1 if message is sent from e3 to e2, -1 if received, 0 otherwise: 0
Enter 1 if message is sent from e3 to e3, -1 if received, 0 otherwise: 1
Enter 1 if message is sent from e4 to e1, -1 if received, 0 otherwise: 0
Enter 1 if message is sent from e4 to e2, -1 if received, 0 otherwise: 0
Enter 1 if message is sent from e4 to e3, -1 if received, 0 otherwise: 1
Enter 1 if message is sent from e5 to e1, -1 if received, 0 otherwise: 0
Enter 1 if message is sent from e5 to e2, -1 if received, 0 otherwise: 0
Enter 1 if message is sent from e5 to e3, -1 if received, 0 otherwise: 1

```

```
    e21 e22 e23
e11 0 0 1
e12 0 0 1
e13 0 0 1
e14 0 0 1
e15 0 0 1

The timestamps of events in P1:
1 2 3 4 5

The timestamps of events in P2:
1 2 6
```

**Conclusion:** Lamport's Logical Clocks have become a fundamental tool in the realm of distributed systems, providing a powerful way to order events occurring concurrently across multiple processes

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <p style="text-align: center;">JNIESTRT'S</p> <p style="text-align: center;"><b>SMT. INDIRA GANDHI COLLEGE OF ENGINEERING</b></p> <p style="text-align: center;">GHANSOLI, NAVI MUMBAI – 400701</p> <p style="text-align: center;">(Approved by AICTE New Delhi &amp; Govt. of Maharashtra, Affiliated to University of Mumbai)</p> <p style="text-align: center;"><b>COMPUTER ENGINEERING DEPARTMENT</b></p> <p style="text-align: center;"><b>ACADEMIC YEAR: 2020-21</b></p> |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|

## **Experiment No.: 06**

**Aim: Synchronization-Producer-Consumer.**

| Date of Performance | Date of Submission | Marks (10)  |   |   |   |   | Sign / Remark |
|---------------------|--------------------|-------------|---|---|---|---|---------------|
|                     |                    | A           | B | C | D | E |               |
|                     |                    | 2           | 3 | 2 | 2 | 1 |               |
|                     |                    |             |   |   |   |   |               |
|                     |                    | Total Marks |   |   |   |   |               |
|                     |                    |             |   |   |   |   |               |

### **EXPERIMENT NO.:06**

**Aim:** Synchronization-Producer-Consumer

**Theory:**

In computing, the producer-consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The

problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue.

The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.

#### Problem

To make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

#### Solution

The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

An inadequate solution could result in a deadlock where both processes are waiting to be awakened.

#### Input:

##### using semaphores:

```
#semaphores_tut.py
import random, time
from threading import BoundedSemaphore, Thread
max_items = 5
"""
```

Consider 'container' as a container, of course, with a capacity of 5 items. Defaults to 1 item if 'max\_items' is passed.

```
"""
```

```
container = BoundedSemaphore(max_items)
def producer(nloops):
    for i in range(nloops):
        time.sleep(random.randrange(2, 5))
        print(time.ctime(), end=": ")
        try:
            container.release()
            print("Produced an item.")
        except ValueError:
```

```

        print("Full, skipping.")
def consumer(nloops):
    for i in range(nloops):
        time.sleep(random.randrange(2, 5))
        print(time.ctime(), end=": ")
        """
        In the following if statement we disable the default
        blocking behaviour by passing False for the blocking flag.
        """
        if container.acquire(False):
            print("Consumed an item.")
        else:
            print("Empty, skipping.")
threads = []
nloops = random.randrange(3, 6)
print("Starting with %s items." % max_items)
threads.append(Thread(target=producer, args=(nloops,)))
threads.append(Thread(target=consumer, args=(random.randrange(nloops,
nloops+max_items+2),)))
for thread in threads: # Starts all the threads.
    thread.start()
for thread in threads: # Waits for threads to complete before moving on with the
main script.
    thread.join()
print("All done.")

```

### **using Thread:**

```

import threading
import time

```

```

# Shared Memory variables
CAPACITY = 10
buffer = [-1 for i in range(CAPACITY)]
in_index = 0
out_index = 0

```

```

# Declaring Semaphores
mutex = threading.Semaphore()

```

```
empty = threading.Semaphore(CAPACITY)
full = threading.Semaphore(0)
```

```
# Producer Thread Class
```

```
class Producer(threading.Thread):
```

```
    def run(self):
```

```
        global CAPACITY, buffer, in_index, out_index
        global mutex, empty, full
```

```
        items_produced = 0
```

```
        counter = 0
```

```
        while items_produced < 20:
```

```
            empty.acquire()
```

```
            mutex.acquire()
```

```
            counter += 1
```

```
            buffer[in_index] = counter
```

```
            in_index = (in_index + 1)%CAPACITY
```

```
            print("Producer produced : ", counter)
```

```
            mutex.release()
```

```
            full.release()
```

```
            time.sleep(1)
```

```
            items_produced += 1
```

```
# Consumer Thread Class
```

```
class Consumer(threading.Thread):
```

```
    def run(self):
```

```
        global CAPACITY, buffer, in_index, out_index, counter
```

```
        global mutex, empty, full
```

```
        items_consumed = 0
```

```
while items_consumed < 20:
    full.acquire()
    mutex.acquire()

    item = buffer[out_index]
    out_index = (out_index + 1)%CAPACITY
    print("Consumer consumed item : ", item)

    mutex.release()
    empty.release()

    time.sleep(2.5)

    items_consumed += 1
```

```
# Creating Threads
producer = Producer()
consumer = Consumer()
```

```
# Starting Threads
consumer.start()
producer.start()
```

```
# Waiting for threads to complete
producer.join()
consumer.join()
```

**Output:**  
**using semaphores:**

```

Starting with 5 items.
Sun Jan 21 13:18:28 2024: Full, skipping.
Sun Jan 21 13:18:28 2024: Consumed an item.
Sun Jan 21 13:18:32 2024: Produced an item.
Sun Jan 21 13:18:32 2024: Consumed an item.
Sun Jan 21 13:18:34 2024: Produced an item.
Sun Jan 21 13:18:36 2024: Consumed an item.
Sun Jan 21 13:18:39 2024: Consumed an item.
Sun Jan 21 13:18:42 2024: Consumed an item.
Sun Jan 21 13:18:44 2024: Consumed an item.
Sun Jan 21 13:18:48 2024: Consumed an item.
All done.

```

### using Thread:

```

Producer produced : 1
Consumer consumed item : 1
Producer produced : 2
Producer produced : 3
Consumer consumed item : 2
Producer produced : 4
Producer produced : 5
Consumer consumed item : 3
Producer produced : 6
Producer produced : 7
Producer produced : 8
Consumer consumed item : 4
Producer produced : 9
Producer produced : 10
Consumer consumed item : 5
Producer produced : 11
Producer produced : 12
Producer produced : 13
Consumer consumed item : 6
Producer produced : 14
Producer produced : 15

```

```

Producer produced : 15
Consumer consumed item : 7
Producer produced : 16
Producer produced : 17
Consumer consumed item : 8
Producer produced : 18
Consumer consumed item : 9
Producer produced : 19
Consumer consumed item : 10
Producer produced : 20
Consumer consumed item : 11
Consumer consumed item : 12
Consumer consumed item : 13
Consumer consumed item : 14
Consumer consumed item : 15
Consumer consumed item : 16
Consumer consumed item : 17
Consumer consumed item : 18
Consumer consumed item : 19
Consumer consumed item : 20

```

**Conclusion:** The Producer-Consumer problem is a classic concurrency problem in computer science. It involves two processes: a producer that generates data and a consumer that consumes it. The challenge is to synchronize these two processes to ensure that the buffer never overflows or underflows.



|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <p style="text-align: center;">JNIESTRT'S</p> <p style="text-align: center;"><b>SMT. INDIRA GANDHI COLLEGE OF ENGINEERING</b></p> <p style="text-align: center;">GHANSOLI, NAVI MUMBAI – 400701</p> <p style="text-align: center;">(Approved by AICTE New Delhi &amp; Govt. of Maharashtra, Affiliated to University of Mumbai)</p> <p style="text-align: center;"><b>COMPUTER ENGINEERING DEPARTMENT</b></p> <p style="text-align: center;"><b>ACADEMIC YEAR: 2020-21</b></p> |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|

## **Experiment No.: 07**

**Aim: Stateful Server.**

| Date of Performance | Date of Submission | Marks (10)  |   |   |   |   | Sign / Remark |
|---------------------|--------------------|-------------|---|---|---|---|---------------|
|                     |                    | A           | B | C | D | E |               |
|                     |                    | 2           | 3 | 2 | 2 | 1 |               |
|                     |                    |             |   |   |   |   |               |
|                     |                    | Total Marks |   |   |   |   |               |
|                     |                    |             |   |   |   |   |               |

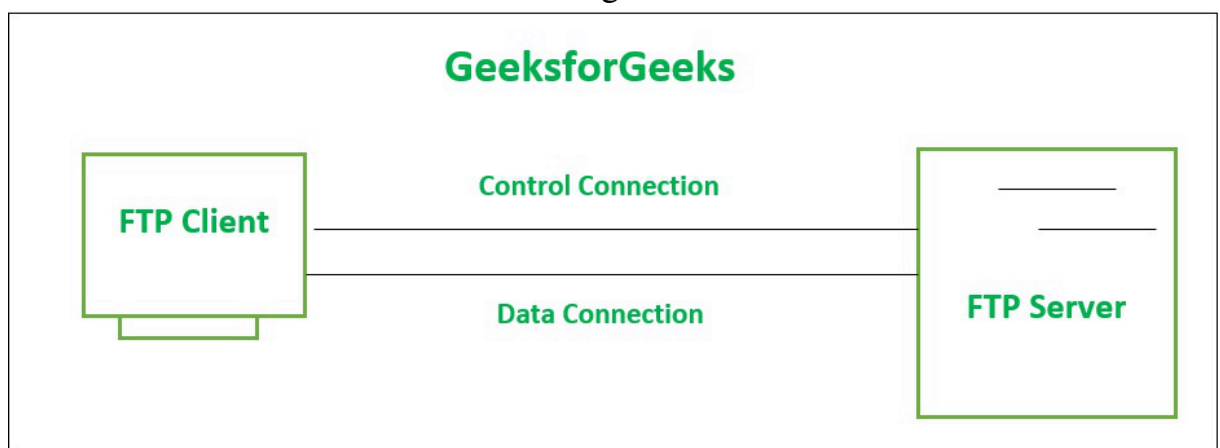
### **EXPERIMENT NO.:07**

**Aim:**Stateful IDFEWFIEGHHFUIEGGSEGFISGFUISGFGFUIGFUISEFUI Server  
**Theory:**

In Stateful Protocol If a client sends a request to the server then it expects some kind of response, if it does not get any response then it resends the request. FTP (File Transfer Protocol), TCP, and Telnet are examples of Stateful Protocol.

Salient features of Stateful Protocol:

- Stateful Protocols provide better performance to the client by keeping track of the connection information.
- Stateful Applications require Backing storage.
- Stateful requests are always dependent on the server-side state.
- TCP sessions follow stateful protocol because both systems maintain information about the session itself during its life.



Maintains the state of the client from one Remote Procedure Call (RPC), which can then be utilized to execute other calls. Stateful servers are able to give a higher performance to clients than stateless servers. The size of messages to and from the server can be greatly reduced because clients do not have to send entire file metadata every time they execute an operation. In addition, the client also gets features like file locking and remembering read and write positions. The advantage of stateful architecture is that a security layer is added to systems due to which online banking systems prefer to use it for managing transactions. On the other side, its complex server design, poor crash recovery, and a lot of dependency between client and server make developers move on to a stateless approach. Examples- FTP (File Transfer Protocol), Telnet, TCP (Transmission Control Protocol).

**Input:**  
**Client:**

```
import socket
```

```
class FTPClient:
```

```
    def __init__(self):
```

```
        self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
        self.client_socket.connect(("127.0.0.1", 5217))
```

```
        self.din = self.client_socket.makefile('rb')
```

```
        self.dout = self.client_socket.makefile('wb')
```

```
    def display_menu(self):
```

```
        while True:
```

```
            print("[ MENU ]")
```

```
            print("1. Send File")
```

```
            print("2. Receive File")
```

```
            print("3. Exit")
```

```
        try:
```

```
            choice = int(input("\nEnter Choice: "))
```

```
            if choice == 1:
```

```
                self.dout.write(b"SEND\n")
```

```
                self.dout.flush()
```

```
                self.send_file()
```

```
            elif choice == 2:
```

```
                self.dout.write(b"GET\n")
```

```
                self.dout.flush()
```

```
                self.receive_file()
```

```
            elif choice == 3:
```

```
                self.dout.write(b"DISCONNECT\n")
```

```
                self.dout.flush()
```

```
                self.client_socket.close()
```

```
                break
```

```
            else:
```

```
                print("Invalid choice. Please try again.")
```

```
        except ValueError:
```

```
            print("Invalid input. Please enter a number.")
```

```
    def send_file(self):
```

```

try:
    filename = input("Enter File Name: ")
    file = open(filename, 'rb')

    if file:
        self.dout.write(filename.encode() + b"\n")
        self.dout.flush()

        msg_from_server = self.din.readline().strip().decode()
        if msg_from_server == "File Already Exists":
            option = input("File Already Exists. Want to OverWrite (Y/N)? ")
            if option.upper() == "Y":
                self.dout.write(b"Y\n")
                self.dout.flush()
            else:
                return

        print("Sending File ...")
        while True:
            data = file.read(1024)
            if not data:
                break
            self.dout.write(data)

        file.close()
        print(self.din.readline().strip().decode()) # Receive server response

    else:
        print("File not Exists...")
        self.dout.write(b"File not found\n")
        self.dout.flush()

except Exception as ex:
    print(f"Error sending file: {ex}")

def receive_file(self):
    try:
        filename = input("Enter File Name: ")

```

```

self.dout.write(filename.encode() + b"\n")
self.dout.flush()

msg_from_server = self.din.readline().strip().decode()
if msg_from_server == "File Not Found":
    print("\nFile not found on Server ...")
    return
elif msg_from_server == "READY":
    print("Receiving File ...")

    file = open(filename, 'wb')
    if file.exists():
        option = input("\nFile Already Exists. Want to OverWrite (Y/N)? ")
        if option.upper() == "N":
            self.dout.flush()
            return

    while True:
        data = self.din.readline().strip()
        if not data:
            break
        file.write(data.encode())

    file.close()
    print(self.din.readline().strip().decode()) # Receive server response

except Exception as ex:
    print(f"Error receiving file: {ex}")

if __name__ == "__main__":
    client = FTPClient()
    client.display_menu()

```

### **Server:**

```

import socket
import threading

class FTPServer:

```

```

def __init__(self):
    self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.server_socket.bind(('0.0.0.0', 5217)) # Bind to all interfaces
    self.server_socket.listen(5)
    print("FTP Server Started on Port Number 5217")

def start(self):
    while True:
        print("Waiting for Connection ...")
        client_socket, client_address = self.server_socket.accept()
        transferfile_thread = threading.Thread(target=self.handle_client,
args=(client_socket,))
        transferfile_thread.start()

def handle_client(self, client_socket):
    try:
        print("FTP Client Connected ...")
        din = client_socket.makefile('rb')
        dout = client_socket.makefile('wb')

        while True:
            print("Waiting for Command ...")
            command = din.readline().strip().decode()

            if command == "GET":
                print("\tGET Command Received ...")
                self.send_file(din, dout)

            elif command == "SEND":
                print("\tSEND Command Received ...")
                self.receive_file(din, dout)

            elif command == "DISCONNECT":
                print("\tDisconnect Command Received ...")
                break

    except Exception as ex:
        print(f"Error handling client: {ex}")

```

```

finally:
    client_socket.close()

def send_file(self, din, dout):
    try:
        filename = din.readline().strip().decode()
        file = open(filename, 'rb')

        if file:
            dout.write(b"READY\n")
            dout.flush()

            while True:
                data = file.read(1024)
                if not data:
                    break
                dout.write(data)

            file.close()
            dout.write(b"File Receive Successfully\n")
            dout.flush()

        else:
            dout.write(b"File Not Found\n")
            dout.flush()

    except Exception as ex:
        print(f"Error sending file: {ex}")

def receive_file(self, din, dout):
    try:
        filename = din.readline().strip().decode()

        if filename == "\nFile not found":
            return

        file = open(filename, 'wb')

```

```

if file.exists():
    dout.write(b"\nFile Already Exists\n")
    dout.flush()
    option = din.readline().strip().decode()
else:
    dout.write(b"SendFile\n")
    dout.flush()
    option = "Y"

if option == "Y":
    while True:
        data = din.readline().strip()
        if not data:
            break
        file.write(data.encode())

    file.close()
    dout.write(b"File Send Successfully\n")
    dout.flush()

except Exception as ex:
    print(f"Error receiving file: {ex}")

if __name__ == "__main__":
    server = FTPServer()
    server.start()

```

**Output:**

**Client:**



```
PROBLEMS  PORTS  OUTPUT  TERMINAL  SEARCH ERROR

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\anish rane\Desktop\BE\sem 8\dc\statefull server> & 'C:\python\python.exe' 'c:\Users\anish rane\.vscode\extensions\ms-python.python-2023.22.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '55565' '--' 'C:\Users\anish rane\Desktop\BE\sem 8\dc\statefull server\client.py'
[ MENU ]
1. Send File
2. Receive File
3. Exit

Enter Choice: 1
Enter File Name: HII.txt
█
```


## Server:

```
PROBLEMS  PORTS  OUTPUT  TERMINAL  SEARCH ERROR

PS C:\Users\anish rane\Desktop\BE\sem 8\dc\statefull server> & C:/python/python.exe "c:/Users/anish rane/Desktop/BE/sem 8/dc/statefull server/server.py"
FTP Server Started on Port Number 5217
Waiting for Connection ...
FTP Client Connected ...
Waiting for Connection ...
Waiting for Command ...
SEND Command Received ...
Error receiving file: '_io.BufferedReader' object has no attribute 'exists'
Waiting for Command ...
```

are Code Link Explain Code Comment Code Code Chat Search Error Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Go Live Blackbox

**Conclusion:** Stateful servers have their upsides and downsides, ultimately making them a choice suitable for specific scenarios.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <p style="text-align: center;">JNIESTRT'S</p> <p style="text-align: center;"><b>SMT. INDIRA GANDHI COLLEGE OF ENGINEERING</b></p> <p style="text-align: center;">GHANSOLI, NAVI MUMBAI – 400701</p> <p style="text-align: center;">(Approved by AICTE New Delhi &amp; Govt. of Maharashtra, Affiliated to University of Mumbai)</p> <p style="text-align: center;"><b>COMPUTER ENGINEERING DEPARTMENT</b></p> <p style="text-align: center;"><b>ACADEMIC YEAR: 2020-21</b></p> |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|

## **Experiment No.: 08**

**Aim: Group Communication.**

| Date of Performance | Date of Submission | Marks (10)  |   |   |   |   | Sign / Remark |
|---------------------|--------------------|-------------|---|---|---|---|---------------|
|                     |                    | A           | B | C | D | E |               |
|                     |                    | 2           | 3 | 2 | 2 | 1 |               |
|                     |                    |             |   |   |   |   |               |
|                     |                    | Total Marks |   |   |   |   |               |
|                     |                    |             |   |   |   |   |               |

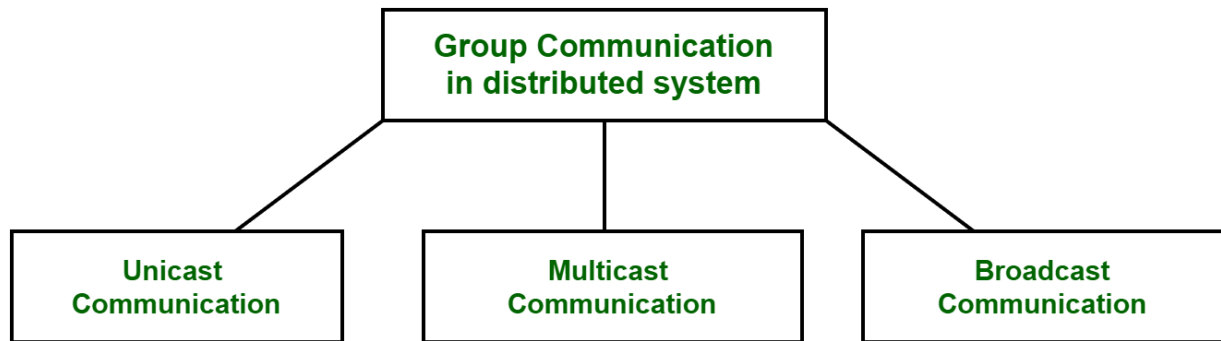
### **EXPERIMENT NO.:08**

**Aim:**Group Communication

**Theory:**

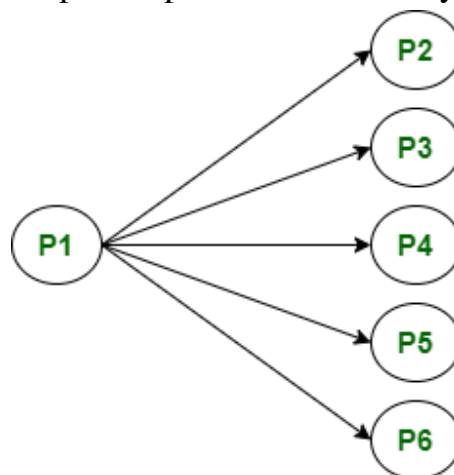
Communication between two processes in a distributed system is required to exchange various data, such as code or a file, between the processes. When one

source process tries to communicate with multiple processes at once, it is called Group Communication. A group is a collection of interconnected processes with abstraction. This abstraction is to hide the message passing so that the communication looks like a normal procedure call. Group communication also helps the processes from different hosts to work together and perform operations in a synchronized manner, therefore increasing the overall performance of the system.



Types of Group Communication in a Distributed System:

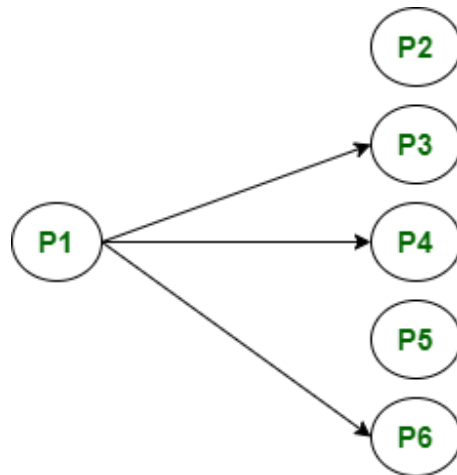
**Broadcast Communication :** When the host process tries to communicate with every process in a distributed system at same time. Broadcast communication comes in handy when a common stream of information is to be delivered to each and every process in the most efficient manner possible. Since it does not require any processing whatsoever, communication is very fast in comparison to other modes of communication. However, it does not support a large number of processes and cannot treat a specific process individually.



A broadcast Communication: P1 process communicating with every process in the system

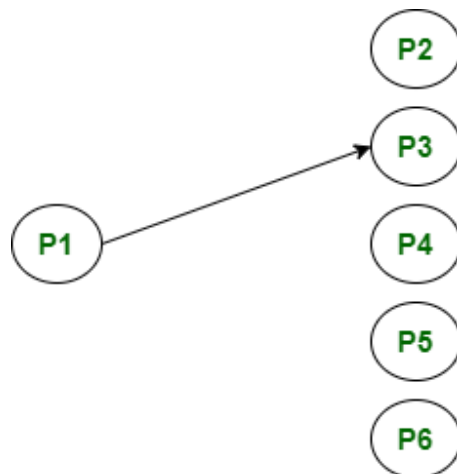
**Multicast Communication :** When the host process tries to communicate with a

designated group of processes in a distributed system at the same time. This technique is mainly used to find a way to address the problem of a high workload on the host system and redundant information from processes in the system. Multitasking can significantly decrease time taken for message handling.



A multicast Communication: P1 process communicating with only a group of the process in the system

Unicast Communication : When the host process tries to communicate with a single process in a distributed system at the same time. Although, the same information may be passed to multiple processes. This works best for two processes communicating as only it has to treat a specific process only. However, it leads to overheads as it has to find the exact process and then exchange information/data.



A unicast Communication: P1 process communicating with only P3 process

**Input:**

```

import socket
import struct
import threading

TERMINATE = "Exit"
name = ""
finished = False

def main():
    global name, finished

    # Default multicast group and port
    multicast_group = "224.1.1.1"
    port = 5007

    try:
        group = socket.inet_aton(multicast_group)
        socket_address = (multicast_group, port)

        name = input("Enter your name: ")

        # Create a UDP socket
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.settimeout(0.2)

        # Set the time-to-live for messages to 0 (only on localhost)
        sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL,
            struct.pack('b', 0))

        # Join the multicast group
        sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP,
            struct.pack('4sL', group, socket.INADDR_ANY))

        # Start a thread for reading messages
        read_thread = threading.Thread(target=read_messages, args=(sock, group,
            port))
        read_thread.start()

```

```

print("Start typing messages...\n")

while True:
    message = input()
    if message.lower() == TERMINATE.lower():
        finished = True
        sock.sendto(TERMINATE.encode('utf-8'), socket_address)
        sock.close()
        break

    message = f"{name}: {message}"
    sock.sendto(message.encode('utf-8'), socket_address)

except (socket.error, KeyboardInterrupt) as e:
    print("Error: {}".format(str(e)))
finally:
    finished = True

def read_messages(sock, group, port):
    global finished

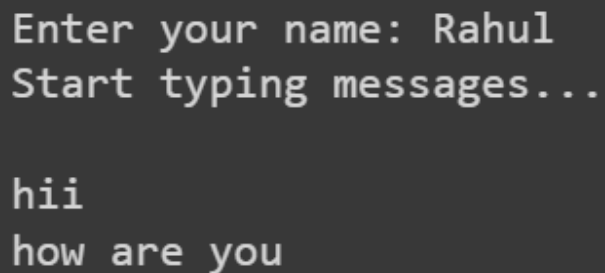
    while not finished:
        try:
            data, addr = sock.recvfrom(1024)
            message = data.decode('utf-8')

            if not message.startswith(name):
                print(message)

        except socket.timeout:
            pass
        except socket.error as e:
            print("Socket closed!")
            finished = True

```

```
if __name__ == "__main__":  
    main()
```

**Output:**A screenshot of a terminal window with a dark background and light gray text. The text shows the program's execution: it prompts for a name, which is 'Rahul', then prompts to start typing messages. The user then enters 'hii' and 'how are you' on separate lines.


```
Enter your name: Rahul  
Start typing messages...  
  
hii  
how are you
```

**Conclusion:**

Group communication enables collaboration and information flow within distributed systems. It empowers a group to:

- Co-decide and solve problems together, crucial for consensus algorithms and collaborative work.
- Broadcast and receive updates efficiently, keeping members informed and responsive.
- Maintain a level of fault tolerance, mitigating single points of failure.

However, scalability, synchronization, and security pose challenges. Choosing the right approach requires balancing these factors with your specific needs.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <p style="text-align: center;">JNIESTRT'S</p> <p style="text-align: center;"><b>SMT. INDIRA GANDHI COLLEGE OF ENGINEERING</b></p> <p style="text-align: center;">GHANSOLI, NAVI MUMBAI – 400701</p> <p style="text-align: center;">(Approved by AICTE New Delhi &amp; Govt. of Maharashtra, Affiliated to University of Mumbai)</p> <p style="text-align: center;"><b>COMPUTER ENGINEERING DEPARTMENT</b></p> <p style="text-align: center;"><b>ACADEMIC YEAR: 2020-21</b></p> |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|

## **Experiment No.: 09**

**Aim: Load Balancing Algorithm.**

| Date of Performance | Date of Submission | Marks (10)  |   |   |   |   | Sign / Remark |
|---------------------|--------------------|-------------|---|---|---|---|---------------|
|                     |                    | A           | B | C | D | E |               |
|                     |                    | 2           | 3 | 2 | 2 | 1 |               |
|                     |                    |             |   |   |   |   |               |
|                     |                    | Total Marks |   |   |   |   |               |
|                     |                    |             |   |   |   |   |               |

### **EXPERIMENT NO.:09**

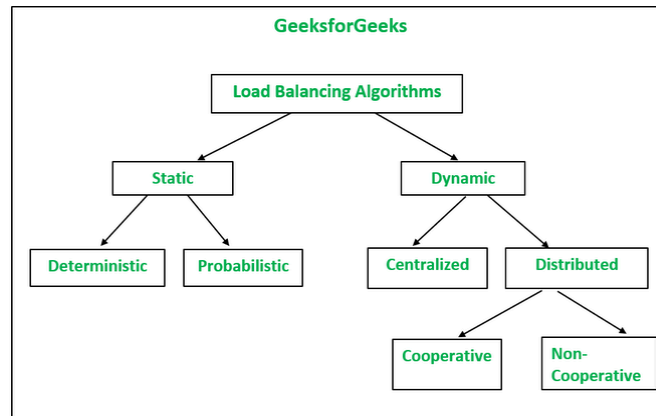
**Aim:** Load Balancing Algorithm

**Theory:**

The Load Balancing approach refers to the division of load among the processing elements of a distributed system. The excess load of one processing element is



distributed to other processing elements that have less load according to the defined limits. In other words, the load is maintained at each processing element in such a manner that neither it gets overloaded nor idle during the execution of a program to maximize the system throughput which is the ultimate goal of distributed systems. This approach makes all processing elements equally busy thus speeding up the entire task leads to the completion of the task by all processors approximately at the same time.



Types of Load Balancing Algorithms:

- **Static Load Balancing Algorithm:** In the Static Load Balancing Algorithm, while distributing load the current state of the system is not taken into account. These algorithms are simpler in comparison to dynamic load balancing algorithms. Types of Static Load Balancing Algorithms are as follows:
  - **Deterministic:** In Deterministic Algorithms, the properties of nodes and processes are taken into account for the allocation of processes to nodes. Because of the deterministic characteristic of the algorithm, it is difficult to optimize to give better results and also costs more to implement.
  - **Probabilistic:** In Probabilistic Algorithms, Statistical attributes of the system are taken into account such as several nodes, topology, etc. to make process placement rules. It does not give better performance.
- **Dynamic Load Balancing Algorithm:** Dynamic Load Balancing Algorithm takes into account the current load of each node or computing unit in the system, allowing for faster processing by dynamically redistributing workloads away from overloaded nodes and toward underloaded nodes. Dynamic algorithms are significantly more difficult to design, but they can give superior results, especially when execution durations for distinct jobs vary greatly. Furthermore, because dedicated nodes for task distribution are

not required, a dynamic load balancing architecture is frequently more modular. Types of Dynamic Load Balancing Algorithms are as follows:

- Centralized: In Centralized Load Balancing Algorithms, the task of handling requests for process scheduling is carried out by a centralized server node. The benefit of this approach is efficiency as all the information is held at a single node but it suffers from the reliability problem because of the lower fault tolerance. Moreover, there is another problem with the increasing number of requests.
- Distributed: In Distributed Load Balancing Algorithms, the decision task of assigning processes is distributed physically to the individual nodes of the system. Unlike Centralized Load Balancing Algorithms, there is no need to hold state information. Hence, speed is fast.
- Types of Distributed Load Balancing Algorithms:
- Cooperative In Cooperative Load Balancing Algorithms, as the name implies, scheduling decisions are taken with the cooperation of entities in the system. The benefit lies in the stability of this approach. The drawback is the complexity involved which leads to more overhead than Non-cooperative algorithms.
- Non-cooperative: In Non-cooperative Load Balancing Algorithms, scheduling decisions are taken by the individual entities of the system as they act as autonomous entities. The benefit is that minor overheads are involved due to the basic nature of non-cooperation. The drawback is that these algorithms might be less stable than Cooperative algorithms.

**Input:**

```
def print_load(servers, processes):  
    each = processes // servers  
    extra = processes % servers  
  
    total = 0  
  
    i = 0  
    for i in range(extra):  
        print(f"Server {i+1} has {each+1} Processes")  
  
    for i in range(i, servers):  
        print(f"Server {i+1} has {each} Processes")
```

```

def main():
    servers = int(input("Enter the number of Servers: "))
    processes = int(input("Enter the number of Processes: "))

    while True:
        print_load(servers, processes)

        print("1. Add Servers 2. Remove Servers 3. Add Processes 4. Remove
Processes 5. Exit")
        choice = int(input())

        if choice == 1:
            servers += int(input("How many more servers to add? "))
        elif choice == 2:
            servers -= int(input("How many servers to remove? "))
        elif choice == 3:
            processes += int(input("How many more processes to add? "))
        elif choice == 4:
            processes -= int(input("How many processes to remove? "))
        elif choice == 5:
            return
        else:
            print("Invalid choice. Please enter a valid option.")

if __name__ == "__main__":
    main()

```

**Output:**


```
Enter the number of Servers: 4
Enter the number of Processes: 5
Server 1 has 2 Processes
Server 1 has 1 Processes
Server 2 has 1 Processes
Server 3 has 1 Processes
Server 4 has 1 Processes
1. Add Servers 2. Remove Servers 3. Add Processes 4. Remove Processes 5. Exit
1
How many more servers to add? 10
Server 1 has 1 Processes
Server 2 has 1 Processes
Server 3 has 1 Processes
Server 4 has 1 Processes
Server 5 has 1 Processes
Server 5 has 0 Processes
```

```
Server 6 has 0 Processes
Server 7 has 0 Processes
Server 8 has 0 Processes
Server 9 has 0 Processes
Server 10 has 0 Processes
Server 11 has 0 Processes
Server 12 has 0 Processes
Server 13 has 0 Processes
Server 14 has 0 Processes
1. Add Servers 2. Remove Servers 3. Add Processes 4. Remove Processes 5. Exit
3
How many more processes to add? 100
Server 1 has 8 Processes
Server 2 has 8 Processes
Server 3 has 8 Processes
Server 4 has 8 Processes
Server 5 has 8 Processes
Server 6 has 8 Processes
Server 7 has 8 Processes
Server 7 has 7 Processes
Server 8 has 7 Processes
Server 9 has 7 Processes
```

```
Server 5 has 8 Processes
Server 6 has 8 Processes
Server 7 has 8 Processes
Server 7 has 7 Processes
Server 8 has 7 Processes
Server 9 has 7 Processes
Server 10 has 7 Processes
Server 11 has 7 Processes
Server 12 has 7 Processes
Server 13 has 7 Processes
Server 14 has 7 Processes
1. Add Servers 2. Remove Servers 3. Add Processes 4. Remove Processes 5. Exit
5
```

**Conclusion:** Load balancing aims to distribute workload evenly across different resources (servers, processes) in a distributed system. By achieving this, we:

- Optimize resource utilization: No single resource becomes overloaded while others sit idle, maximizing overall system efficiency.
- Minimize response times: Faster processing due to balanced workload leads to quicker responses for users or applications.
- Enhance scalability and fault tolerance: The system can handle increased load by spreading it across resources, and can maintain functionality even if individual resources fail.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <p style="text-align: center;">JNIESTRT'S</p> <p style="text-align: center;"><b>SMT. INDIRA GANDHI COLLEGE OF ENGINEERING</b></p> <p style="text-align: center;">GHANSOLI, NAVI MUMBAI – 400701</p> <p style="text-align: center;">(Approved by AICTE New Delhi &amp; Govt. of Maharashtra, Affiliated to University of Mumbai)</p> <p style="text-align: center;"><b>COMPUTER ENGINEERING DEPARTMENT</b></p> <p style="text-align: center;"><b>ACADEMIC YEAR: 2020-21</b></p> |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|

## **Experiment No.: 10**

**Aim: Mutual Exclusion Algorithm.**

| Date of Performance | Date of Submission | Marks (10)  |   |   |   |   | Sign / Remark |
|---------------------|--------------------|-------------|---|---|---|---|---------------|
|                     |                    | A           | B | C | D | E |               |
|                     |                    | 2           | 3 | 2 | 2 | 1 |               |
|                     |                    |             |   |   |   |   |               |
|                     |                    | Total Marks |   |   |   |   |               |
|                     |                    |             |   |   |   |   |               |

### **EXPERIMENT NO.:10**

**Aim:**Mutual Exclusion Algorithm

**Theory:**

Mutual exclusion is a concurrency control property which is introduced to prevent race conditions. It is the requirement that a process can not enter its critical section

while another concurrent process is currently present or executing in its critical section i.e only one process is allowed to execute the critical section at any given instance of time.

Mutual exclusion in single computer system Vs. distributed system: In single computer system, memory and other resources are shared between different processes. The status of shared resources and the status of users is easily available in the shared memory so with the help of shared variables (For example: Semaphores) mutual exclusion problem can be easily solved. In Distributed systems, we neither have shared memory nor a common physical clock and therefore we can not solve mutual exclusion problems using shared variables. To eliminate the mutual exclusion problem in distributed systems, an approach based on message passing is used. A site in a distributed system does not have complete information of the state of the system due to lack of shared memory and a common physical clock.

Requirements of Mutual exclusion Algorithm:

- No Deadlock: Two or more sites should not endlessly wait for any message that will never arrive.
- No Starvation: Every site who wants to execute a critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other site are repeatedly executing critical section
- Fairness: Each site should get a fair chance to execute a critical section. Any request to execute a critical section must be executed in the order they are made i.e Critical section execution requests should be executed in the order of their arrival in the system.
- Fault Tolerance: In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

**Input:**

```
import math
import tkinter as tk
```

```
class Lamport:
    def __init__(self):
        # Arrays for events and timestamps
```

```

self.e = [[0 for _ in range(10)] for _ in range(10)]
self.en = [[0 for _ in range(10)] for _ in range(10)]
self.ev = [0] * 10

# Number of processes
self.p = 0

# HashMap for relationships
self.hm = {}

def calc(self):
    # Get input for number of processes and events
    self.p = int(input("Enter the number of processes: "))
    for i in range(1, self.p + 1):
        self.ev[i] = int(input(f"Enter the no of events per process {i}: "))

    # Get input for relationships
    for i in range(1, self.p + 1):
        print(f"For process: {i}")
        for j in range(1, self.ev[i] + 1):
            input_val = int(input(f"For event: {j} "))
            k = i * 10 + j
            self.hm[k] = input_val
            if j == 1:
                self.en[i][j] = 1

    # Calculate timestamps
    for i in range(1, self.p + 1):
        for j in range(2, self.ev[i] + 1):
            k = i * 10 + j
            if self.hm.get(k) == 0:
                self.en[i][j] = self.en[i][j - 1] + 1
            else:
                a = self.hm.get(k)
                p1 = a // 10
                e1 = a % 10
                self.en[i][j] = max(self.en[p1][e1], self.en[i][j - 1]) + 1

```



```

# Print timestamps
for i in range(1, self.p + 1):
    for j in range(1, self.ev[i] + 1):
        print(self.en[i][j])

# Create the GUI
self.draw_gui()

def draw_gui(self):
    root = tk.Tk()
    root.title("Lamport Clock")

    canvas = tk.Canvas(root, width=500, height=500)
    canvas.pack()

    # Draw lines for processes
    for i in range(1, self.p + 1):
        canvas.create_line(50, 100 * i, 450, 100 * i, fill="black")

    # Draw events and arrows
    for i in range(1, self.p + 1):
        for j in range(1, self.ev[i] + 1):
            k = i * 10 + j
            x = 50 * j
            y = 100 * i - 3
            canvas.create_oval(x - 2, y - 2, x + 2, y + 2, fill="blue")
            canvas.create_text(x, y - 5, text=f"e{i}{j}({self.en[i][j]})")
            h1 = self.hm.get(k)
            if h1 != 0:
                h11 = h1 // 10
                h12 = h1 % 10
                canvas.create_line(50 * h12 + 2, 100 * h11, 50 * j + 2, 100 * i,
                                   arrow=tk.LAST, fill="red")

    root.mainloop()

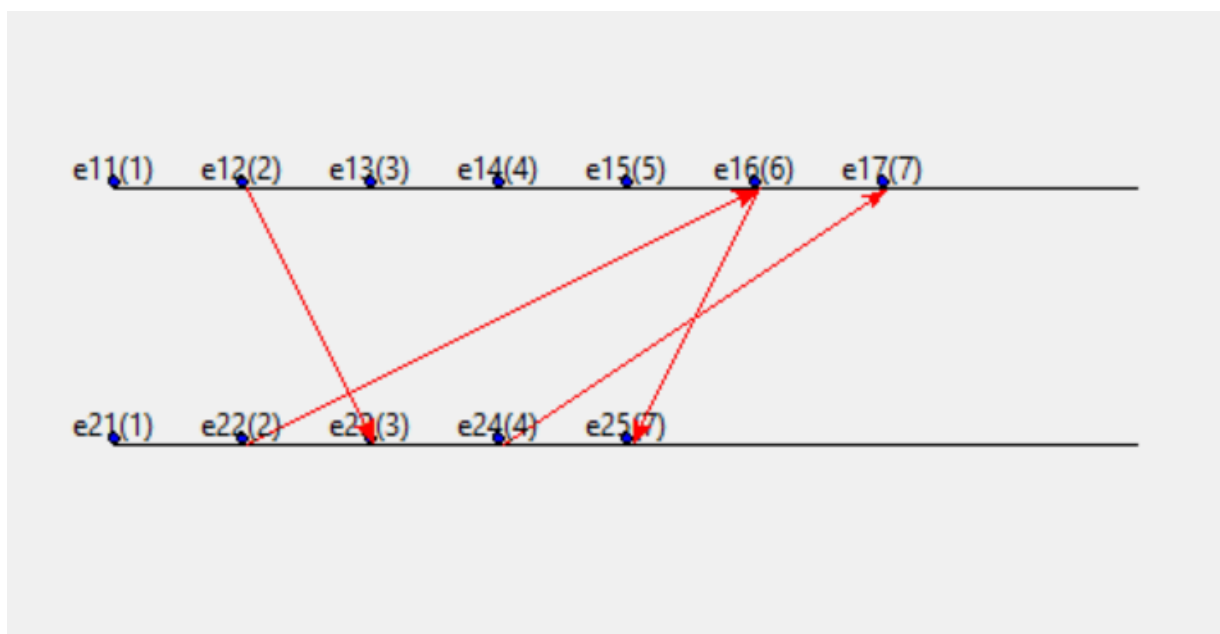
if __name__ == "__main__":
    lamport = Lamport()

```

lamport.calc()

## Output:

```
PROBLEMS  PORTS  OUTPUT  TERMINAL  SEARCH ERROR
sktop\BE\sem 8\dc\exp 10 mutual exclusion\exp10.py'
Enter the number of processes: 2
Enter the no of events per process 1: 7
Enter the no of events per process 2: 5
For process: 1
For event: 1 0
For event: 2 0
For event: 3 0
For event: 4 0
For event: 5 0
For event: 6 22
For event: 7 24
For process: 2
For event: 1 0
For event: 2 0
For event: 3 12
For event: 4 0
For event: 5 16
1
2
3
4
5
6
7
1
2
3
4
7
[]
```



**Conclusion:** Mutual exclusion algorithms ensure that only one process can access a critical section (shared resource) at a time, preventing data corruption and race conditions.

In brief:

- Prevents conflicts: Guarantees exclusive access to critical sections, avoiding unexpected interactions and ensuring data integrity.
- Enables concurrency: Allows multiple processes to run concurrently outside critical sections, maximizing efficiency.
- Trade-offs exist: Different algorithms offer varying levels of efficiency, complexity, and fault tolerance. Choosing the right one depends on your specific needs.