SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJ19ITL601 **DATE: 20 / 02 / 24**

COURSE NAME: Parallel and Distributed Computing **CLASS: I2-2**

LAB EXPERIMENT NO. 5

CO/LO: Develop parallel program using CUDA .

AIM / OBJECTIVE: To execute parallel CUDA programs .

DESCRIPTION OF EXPERIMENT:

Implement CUDA program for:

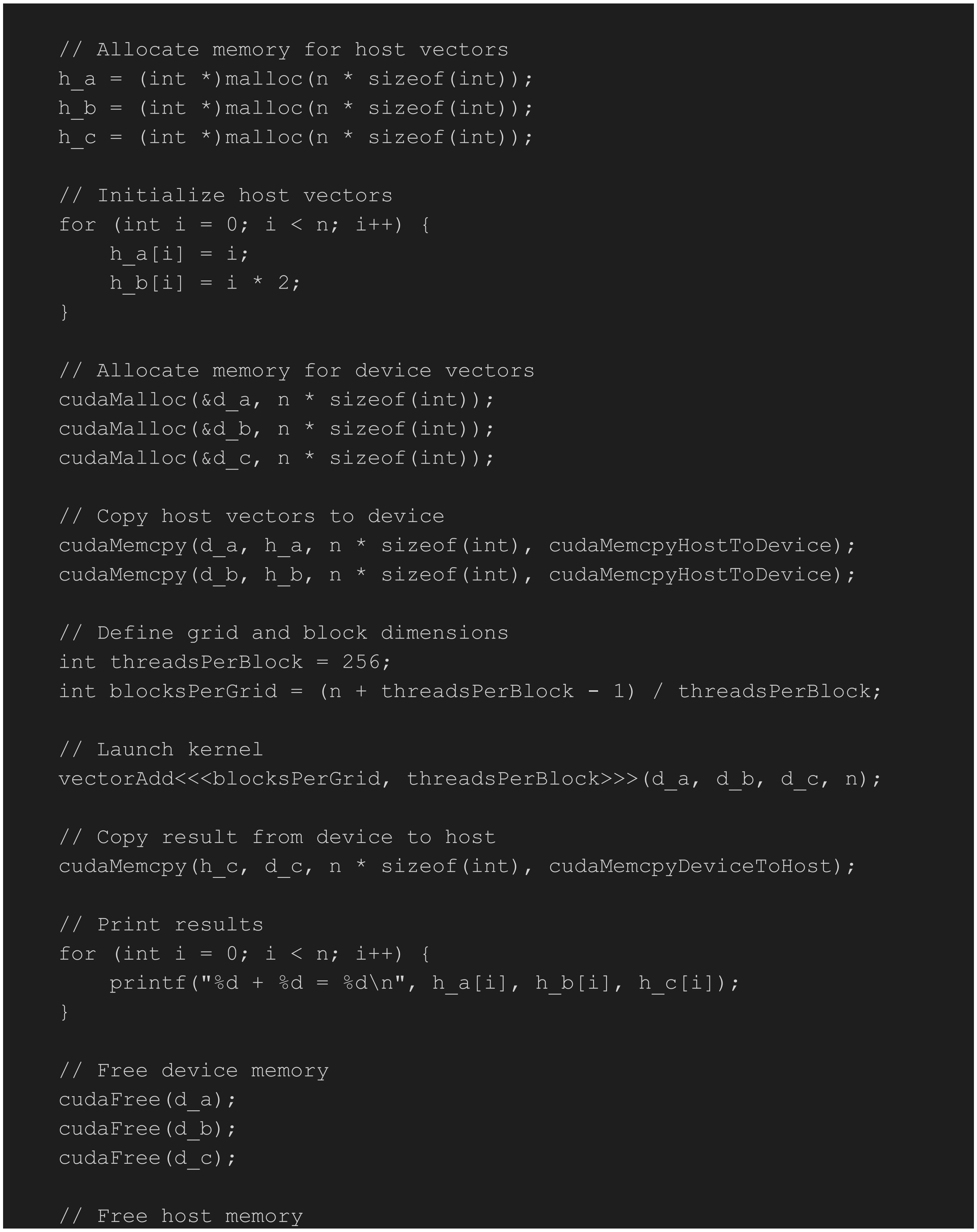
- Vector Addition
- Matrix Multiplication

TECHNOLOGY STACK USED: CUDA C

SOURCE CODE (OPTIONAL):

1)

```
%%writefile vectoraddition.cu

#include <stdio.h>

// CUDA kernel to add two vectors
__global__ void vectorAdd(int *a, int *b, int *c, int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) {
        c[i] = a[i] + b[i];
    }
}

int main() {
    int n = 1000; // Size of vectors
    int *h_a, *h_b, *h_c; // Host vectors
    int *d_a, *d_b, *d_c; // Device vectors
```



```
    // Allocate memory for host vectors
    h_a = (int *)malloc(n * sizeof(int));
    h_b = (int *)malloc(n * sizeof(int));
    h_c = (int *)malloc(n * sizeof(int));

    // Initialize host vectors
    for (int i = 0; i < n; i++) {
        h_a[i] = i;
        h_b[i] = i * 2;
    }

    // Allocate memory for device vectors
    cudaMalloc(&d_a, n * sizeof(int));
    cudaMalloc(&d_b, n * sizeof(int));
    cudaMalloc(&d_c, n * sizeof(int));

    // Copy host vectors to device
    cudaMemcpy(d_a, h_a, n * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, n * sizeof(int), cudaMemcpyHostToDevice);

    // Define grid and block dimensions
    int threadsPerBlock = 256;
    int blocksPerGrid = (n + threadsPerBlock - 1) / threadsPerBlock;

    // Launch kernel
    vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_a, d_b, d_c, n);

    // Copy result from device to host
    cudaMemcpy(h_c, d_c, n * sizeof(int), cudaMemcpyDeviceToHost);

    // Print results
    for (int i = 0; i < n; i++) {
        printf("%d + %d = %d\n", h_a[i], h_b[i], h_c[i]);
    }

    // Free device memory
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    // Free host memory
    free(h_a);
    free(h_b);
    free(h_c);

    return 0;
}
```

```
!nvcc -arch=sm_75 -gencode=arch=compute_75,code=sm_75 vectoraddition.cu -o
vectoraddition
```

```
!./vectoraddition
```

2)

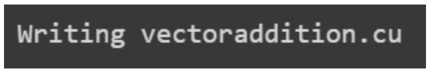
```
%%writefile matrixmultiplication.cu

#include <stdio.h>

#define N 4 // Size of square matrices

// CUDA kernel for matrix multiplication
__global__ void matrixMul(int *a, int *b, int *c) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < N && col < N) {
        int sum = 0;
        for (int i = 0; i < N; i++) {
            sum += a[row * N + i] * b[i * N + col];
        }
        c[row * N + col] = sum;
    }
}

int main() {
    int *h_a, *h_b, *h_c; // Host matrices
    int *d_a, *d_b, *d_c; // Device matrices

    // Allocate memory for host matrices
    h_a = (int *)malloc(N * N * sizeof(int));
    h_b = (int *)malloc(N * N * sizeof(int));
    h_c = (int *)malloc(N * N * sizeof(int));

    // Initialize host matrices
    for (int i = 0; i < N * N; i++) {
        h_a[i] = i;
        h_b[i] = i * 2;
    }

    // Allocate memory for device matrices
    cudaMalloc(&d_a, N * N * sizeof(int));
    cudaMalloc(&d_b, N * N * sizeof(int));
    cudaMalloc(&d_c, N * N * sizeof(int));

    // Copy host matrices to device
    cudaMemcpy(d_a, h_a, N * N * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, N * N * sizeof(int), cudaMemcpyHostToDevice);

    // Define grid and block dimensions
    dim3 threadsPerBlock(16, 16);
    dim3 blocksPerGrid((N + threadsPerBlock.x - 1) / threadsPerBlock.x,
                       (N + threadsPerBlock.y - 1) / threadsPerBlock.y);

    // Launch kernel
    matrixMul<<<blocksPerGrid, threadsPerBlock>>>(d_a, d_b, d_c);

    // Copy result from device to host
    cudaMemcpy(h_c, d_c, N * N * sizeof(int), cudaMemcpyDeviceToHost);

    // Print result
    printf("Matrix A:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d\t", h_a[i * N + j]);
        }
        printf("\n");
    }

    printf("\nMatrix B:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d\t", h_b[i * N + j]);
        }
        printf("\n");
    }

    printf("\nResultant Matrix C:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d\t", h_c[i * N + j]);
        }
        printf("\n");
    }

    // Free device memory
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    // Free host memory
    free(h_a);
    free(h_b);
    free(h_c);

    return 0;
}
```

```
!nvcc -arch=sm_75 -gencode=arch=compute_75,code=sm_75
matrixmultiplication.cu -o matrixmultiplication
```

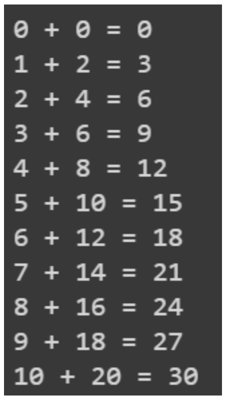
```
!./matrixmultiplication
```

OBSERVATIONS / DISCUSSION OF RESULT:

1)

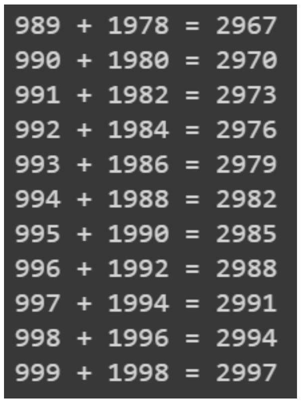


```
Writing vectoraddition.cu
```



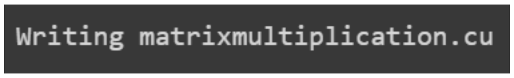
```
0 + 0 = 0
1 + 2 = 3
2 + 4 = 6
3 + 6 = 9
4 + 8 = 12
5 + 10 = 15
6 + 12 = 18
7 + 14 = 21
8 + 16 = 24
9 + 18 = 27
10 + 20 = 30
```

………

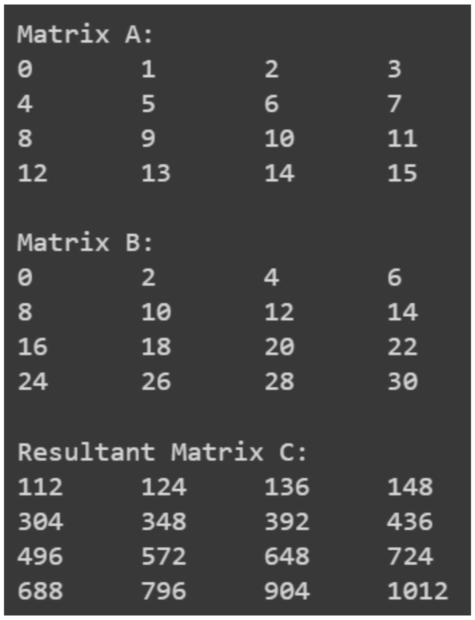


```
989 + 1978 = 2967
990 + 1980 = 2970
991 + 1982 = 2973
992 + 1984 = 2976
993 + 1986 = 2979
994 + 1988 = 2982
995 + 1990 = 2985
996 + 1992 = 2988
997 + 1994 = 2991
998 + 1996 = 2994
999 + 1998 = 2997
```

2)



```
Writing matrixmultiplication.cu
```



```
Matrix A:
0       1       2       3
4       5       6       7
8       9       10      11
12      13      14      15

Matrix B:
0       2       4       6
8       10      12      14
16      18      20      22
24      26      28      30

Resultant Matrix C:
112     124     136     148
304     348     392     436
496     572     648     724
688     796     904     1012
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

CONCLUSION:

Thus in this experiment we learnt about how executing parallel CUDA programs offers significant performance benefits through performing various programs like vector addition and matrix multiplication.

REFERENCES:

(List the references as per format given below and citations to be included the document)

[1] Ponniah P., “Data Warehousing: Fundamentals for IT Professionals”, 2nd Edition, Wiley India, 2013.

[2] Ageed, Z. S., Zeebaree, S. R., Sadeeq, M. M., Kak, S. F., Yahia, H. S., Mahmood, M. R., & Ibrahim, I. M. (2021), “Comprehensive survey of big data mining approaches in cloud systems”, Qubahan Academic Journal, 1(2), 29-38.

Website References:

Author's Last Name, First Initial. Middle Initial. (Date of Publication or Update). Title of work. Site name. Retrieved Month Day, Year, from URL from Homepage

[3] U.S. Census Bureau. U.S. and world population clock. U.S. Department of Commerce. Retrieved July 3, 2019, from https://www.census.gov/popclock.

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJ19ITL601 **DATE: 27/02/2024**

COURSE NAME: Parallel and Distributed Computing **CLASS: T Y B. TECH IT I2-2**

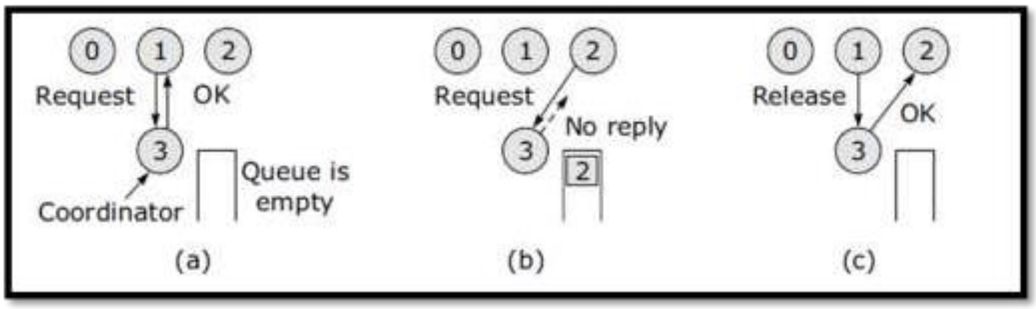
LAB EXPERIMENT NO. 6

CO/LO: Implementation of Mutual Exclusion for Distributed environment.

AIM / OBJECTIVE: To implement Centralized and Distributed mutual exclusion algorithms.

INSTRUCTIONS:

i. **Centralized Algorithm**



(a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted.
(b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply.
(c) When Process 1 exits the critical region, it tells the coordinator, which then replies to Process 2.

CODE:

Server:

```
import socket
import threading


class Coordinator:
    def__init_(self):
        self.lock = threading.Lock()
        self.in_critical_region = False
        self.pending_request = None
    def handle_request(self, conn, addr):
        while True:
            data = conn.recv(1024).decode()
            if not data:
                print("Connection closed by", addr)
                break
            if data == "request":
                with self.lock:
                    if self.in_critical_region:
                        self.pending_request = conn
                        print("Pending request from", addr)
                    else:
                        self.in_critical_region = True
                        conn.sendall("granted".encode())
                        print("Granted access to", addr)
            elif data == "release":
                with self.lock:
                    self.in_critical_region = False
                    if self.pending_request:
                        self.pending_request.sendall("granted".encode())
                        self.in_critical_region = True
                        self.pending_request = None
                        print("Granted access to pending request")
            else:
                print("Invalid message from", addr)
        conn.close()


def server():
    coordinator = Coordinator()
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

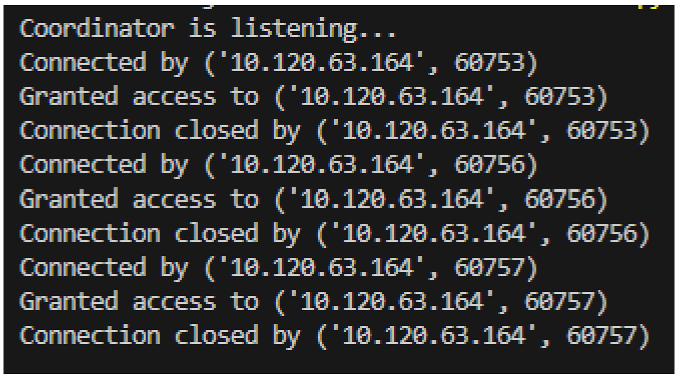
```
    server_socket.bind(('10.120.63.164', 12345))
    server_socket.listen(5)
    print("Coordinator is listening...")

    while True:
        conn, addr = server_socket.accept()
        print("Connected by", addr)
        threading.Thread(target=coordinator.handle_request, args=(conn, addr)).start()

if__name__== "_main_":
    server()
```

Client:

```
import socket
import time

def client(process_id):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('10.120.63.164', 12345))
    print("Connected to server as Process", process_id)

    # Request access to critical region
    client_socket.sendall("request".encode())

    # Receive response from server
    response = client_socket.recv(1024).decode()
    if response == "granted":
        print("Process", process_id, "entered the critical region")
        time.sleep(3) # Simulate critical section work
        print("Process", process_id, "exited the critical region")
        # Notify server that critical section is exited
        client_socket.sendall("release".encode())
    else:
        print("Process", process_id, "is waiting for access")

    client_socket.close()

if__name__== "_main__":
    # Start three processes (clients)
    for i in range(3):
        time.sleep(1) # Ensure staggered start
        client(i + 1)
```

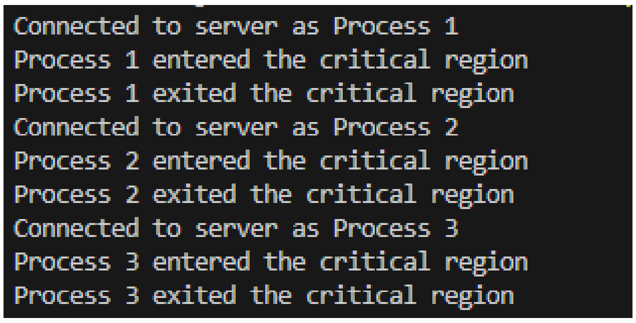
OUTPUT:

Server:



```
Coordinator is listening...
Connected by ('10.120.63.164', 60753)
Granted access to ('10.120.63.164', 60753)
Connection closed by ('10.120.63.164', 60753)
Connected by ('10.120.63.164', 60756)
Granted access to ('10.120.63.164', 60756)
Connection closed by ('10.120.63.164', 60756)
Connected by ('10.120.63.164', 60757)
Granted access to ('10.120.63.164', 60757)
Connection closed by ('10.120.63.164', 60757)
```

Client:

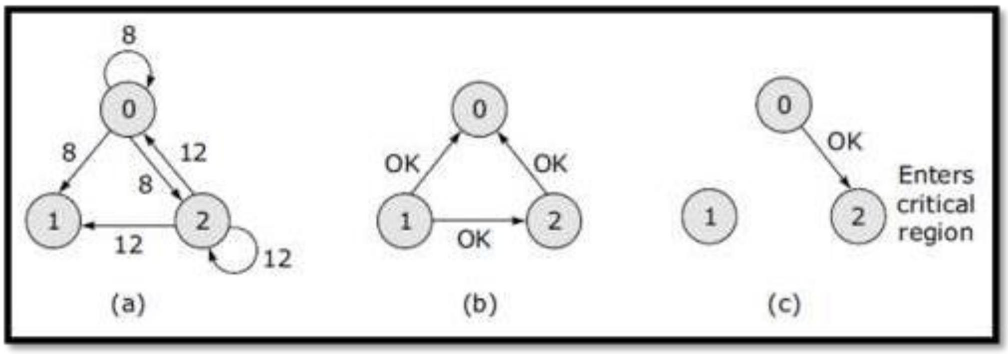


```
Connected to server as Process 1
Process 1 entered the critical region
Process 1 exited the critical region
Connected to server as Process 2
Process 2 entered the critical region
Process 2 exited the critical region
Connected to server as Process 3
Process 3 entered the critical region
Process 3 exited the critical region
```

ii. Distributed Algorithm (Ricart Agrawala Algorithm)

1. Two processes want to enter the same critical region at the same moment.
2. Process 0 has the lowest timestamp, so it wins.
3. When Process 0 is done, it sends an OK, so Process 2 can now enter the critical region.

Example:



CODE:

Server:

```
import socket
import threading
import time

class Server:
    def__init_(self):
        self.lock = threading.Lock()
        self.timestamp = 0
        self.pending_requests = []
        self.in_critical_region = False

    def handle_request(self, conn, addr, timestamp):
        with self.lock:
            print("Received request from Process", addr[1], "with timestamp", timestamp)
            if self.in_critical_region or (self.pending_requests and (timestamp, addr[1]) <
self.pending_requests[0]):
                self.pending_requests.append((timestamp, addr[1]))
                print("Process", addr[1], "added to pending requests")
                conn.sendall("deferred".encode())
            else:
                conn.sendall("granted".encode())
                self.in_critical_region = True
                print("Granted access to Process", addr[1])

    def release_request(self):
        with self.lock:
            self.in_critical_region = False
            if self.pending_requests:
                timestamp, pid = self.pending_requests.pop(0)
                print("Granted access to pending request from Process", pid)

def server():
    server_obj = Server()
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('127.0.0.1', 12345))
    server_socket.listen(5)
    print("Server is listening...")

    while True:
        conn, addr = server_socket.accept()
        print("Connected by", addr)
        data = conn.recv(1024).decode()
        timestamp = int(data)
        threading.Thread(target=server_obj.handle_request, args=(conn, addr, timestamp)).start()

if__name__== "_main_":
    server()
```

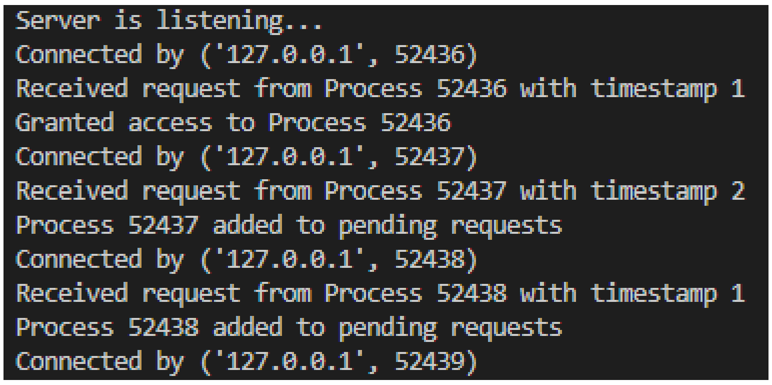
Client:

```
import socket
import time
import threading
import random

class Client:
    def__init_(self, pid):
        self.pid = pid
        self.timestamp = 0
        self.server_address = ('127.0.0.1', 12345)
        self.response = None
        self.lock = threading.Lock()

    def request_critical_section(self):
        with self.lock:
            self.timestamp += 1
            client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            client_socket.connect(self.server_address)
            client_socket.sendall(str(self.timestamp).encode())
            response = client_socket.recv(1024).decode()
            self.response = response
            print("Received response from server for Process", self.pid, ":", response)
            client_socket.close()
```

```
def client_process(client_obj):
    while True:
        time.sleep(random.randint(1, 5)) # Random delay before requesting critical section
        client_obj.request_critical_section()

def client(pid):
    client_obj = Client(pid)
    threading.Thread(target=client_process, args=(client_obj,)).start()

if__name__== "__main__":
    # Start three client processes
    for i in range(3):
        client(i + 1)
```

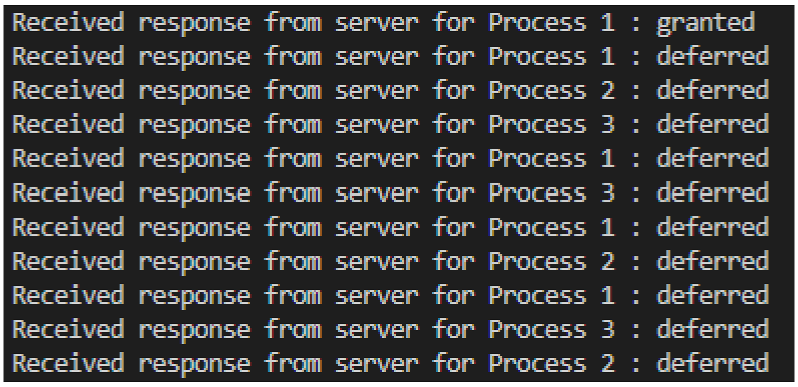
OUTPUT:

Server:



```
Server is listening...
Connected by ('127.0.0.1', 52436)
Received request from Process 52436 with timestamp 1
Granted access to Process 52436
Connected by ('127.0.0.1', 52437)
Received request from Process 52437 with timestamp 2
Process 52437 added to pending requests
Connected by ('127.0.0.1', 52438)
Received request from Process 52438 with timestamp 1
Process 52438 added to pending requests
Connected by ('127.0.0.1', 52439)
```

Client:



```
Received response from server for Process 1 : granted
Received response from server for Process 1 : deferred
Received response from server for Process 2 : deferred
Received response from server for Process 3 : deferred
Received response from server for Process 1 : deferred
Received response from server for Process 3 : deferred
Received response from server for Process 1 : deferred
Received response from server for Process 2 : deferred
Received response from server for Process 1 : deferred
Received response from server for Process 3 : deferred
Received response from server for Process 2 : deferred
```

Conclusion:

Thus in this experiment, we performed to implement Centralized and Distributed mutual exclusion algorithms successfully using client server communication.

Web Reference:

[1] https://github.com/trunc8/ricart-agrawala-algorithm

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJ19ITL601 **DATE: 27/02/2024**

COURSE NAME: Parallel and Distributed Computing **CLASS: T Y B. TECH IT I2-2**

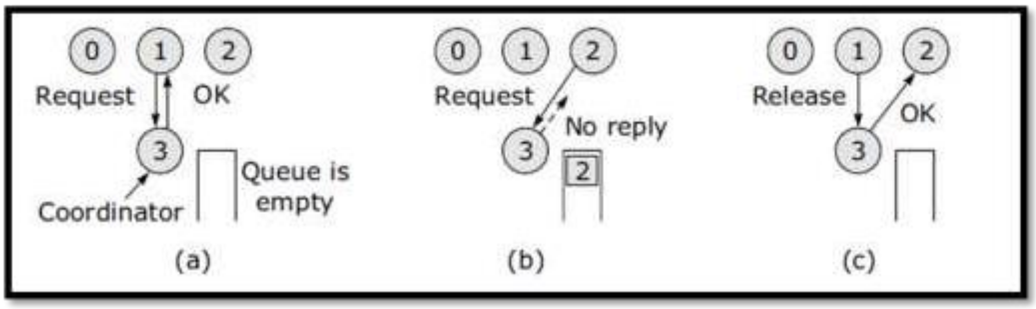
LAB EXPERIMENT NO. 6

CO/LO: Implementation of Mutual Exclusion for Distributed environment.

AIM / OBJECTIVE: To implement Centralized and Distributed mutual exclusion algorithms.

INSTRUCTIONS:

i. Centralized Algorithm



(a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted.
(b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply.
(c) When Process 1 exits the critical region, it tells the coordinator, which then replies to Process 2.

CODE:

Server:

```
import socket
import threading


class Coordinator:
    def__init_(self):
        self.lock = threading.Lock()
        self.in_critical_region = False
        self.pending_request = None
    def handle_request(self, conn, addr):
        while True:
            data = conn.recv(1024).decode()
            if not data:
                print("Connection closed by", addr)
                break
            if data == "request":
                with self.lock:
                    if self.in_critical_region:
                        self.pending_request = conn
                        print("Pending request from", addr)
                    else:
                        self.in_critical_region = True
                        conn.sendall("granted".encode())
                        print("Granted access to", addr)
            elif data == "release":
                with self.lock:
                    self.in_critical_region = False
                    if self.pending_request:
                        self.pending_request.sendall("granted".encode())
                        self.in_critical_region = True
                        self.pending_request = None
                        print("Granted access to pending request")
            else:
                print("Invalid message from", addr)
        conn.close()


def server():
    coordinator = Coordinator()
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('10.120.63.164', 12345))
    server_socket.listen(5)
    print("Coordinator is listening...")

    while True:
        conn, addr = server_socket.accept()
        print("Connected by", addr)
        threading.Thread(target=coordinator.handle_request, args=(conn, addr)).start()

if__name__== "_main_":
    server()
```

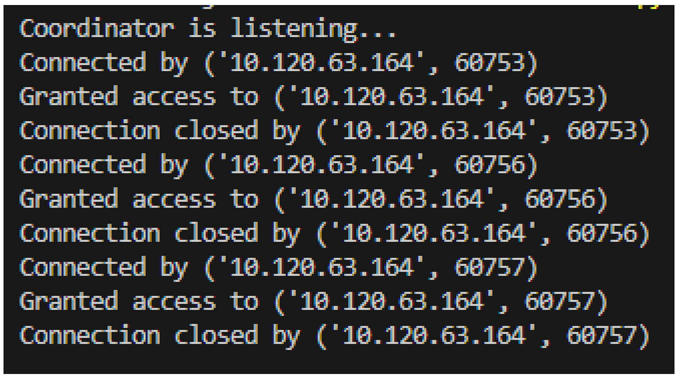
Client:

```
import socket
import time

def client(process_id):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('10.120.63.164', 12345))
    print("Connected to server as Process", process_id)

    # Request access to critical region
    client_socket.sendall("request".encode())

    # Receive response from server
    response = client_socket.recv(1024).decode()
    if response == "granted":
        print("Process", process_id, "entered the critical region")
        time.sleep(3) # Simulate critical section work
        print("Process", process_id, "exited the critical region")
        # Notify server that critical section is exited
```

```
            client_socket.sendall("release".encode())
        else:
            print("Process", process_id, "is waiting for access")

        client_socket.close()

if__name__== "_main__":
    # Start three processes (clients)
    for i in range(3):
        time.sleep(1) # Ensure staggered start
        client(i + 1)
```

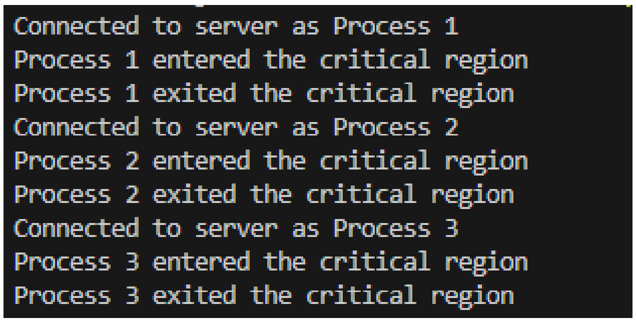
OUTPUT:

Server:



```
Coordinator is listening...
Connected by ('10.120.63.164', 60753)
Granted access to ('10.120.63.164', 60753)
Connection closed by ('10.120.63.164', 60753)
Connected by ('10.120.63.164', 60756)
Granted access to ('10.120.63.164', 60756)
Connection closed by ('10.120.63.164', 60756)
Connected by ('10.120.63.164', 60757)
Granted access to ('10.120.63.164', 60757)
Connection closed by ('10.120.63.164', 60757)
```

Client:

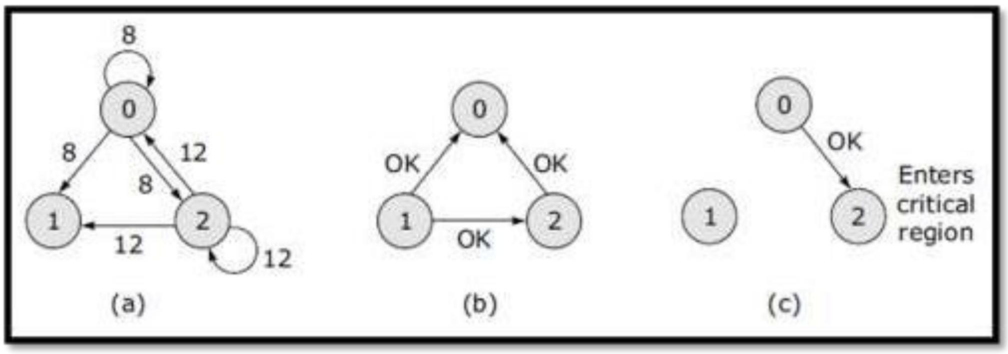


```
Connected to server as Process 1
Process 1 entered the critical region
Process 1 exited the critical region
Connected to server as Process 2
Process 2 entered the critical region
Process 2 exited the critical region
Connected to server as Process 3
Process 3 entered the critical region
Process 3 exited the critical region
```

ii. Distributed Algorithm (Ricart Agrawala Algorithm)

1. Two processes want to enter the same critical region at the same moment.
2. Process 0 has the lowest timestamp, so it wins.
3. When Process 0 is done, it sends an OK, so Process 2 can now enter the critical region.

Example:



CODE:

Server:

```
import socket
import threading
import time

class Server:
    def__init_(self):
        self.lock = threading.Lock()
        self.timestamp = 0
        self.pending_requests = []
        self.in_critical_region = False

    def handle_request(self, conn, addr, timestamp):
        with self.lock:
            print("Received request from Process", addr[1], "with timestamp", timestamp)
            if self.in_critical_region or (self.pending_requests and (timestamp, addr[1]) <
self.pending_requests[0]):
                self.pending_requests.append((timestamp, addr[1]))
                print("Process", addr[1], "added to pending requests")
                conn.sendall("deferred".encode())
            else:
                conn.sendall("granted".encode())
                self.in_critical_region = True
                print("Granted access to Process", addr[1])

    def release_request(self):
        with self.lock:
            self.in_critical_region = False
            if self.pending_requests:
                timestamp, pid = self.pending_requests.pop(0)
                print("Granted access to pending request from Process", pid)

def server():
    server_obj = Server()
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('127.0.0.1', 12345))
    server_socket.listen(5)
    print("Server is listening...")

    while True:
        conn, addr = server_socket.accept()
        print("Connected by", addr)
        data = conn.recv(1024).decode()
        timestamp = int(data)
        threading.Thread(target=server_obj.handle_request, args=(conn, addr, timestamp)).start()

if__name__== "_main_":
    server()
```

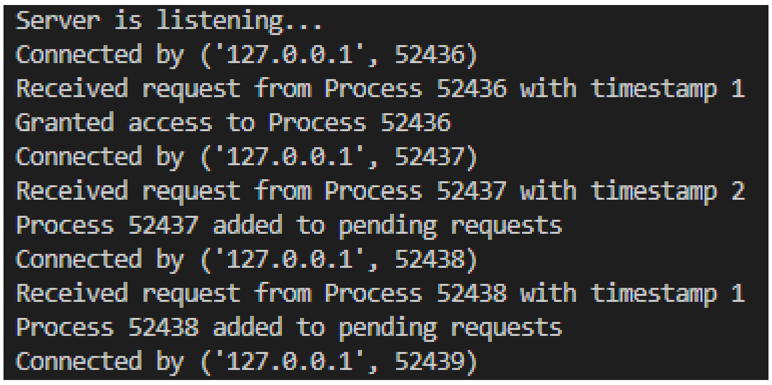
Client:

```
import socket
import time
import threading
import random

class Client:
    def__init_(self, pid):
        self.pid = pid
        self.timestamp = 0
        self.server_address = ('127.0.0.1', 12345)
        self.response = None
        self.lock = threading.Lock()

    def request_critical_section(self):
        with self.lock:
            self.timestamp += 1
            client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            client_socket.connect(self.server_address)
            client_socket.sendall(str(self.timestamp).encode())
            response = client_socket.recv(1024).decode()
            self.response = response
            print("Received response from server for Process", self.pid, ":", response)
            client_socket.close()
```

```
def client_process(client_obj):
    while True:
        time.sleep(random.randint(1, 5)) # Random delay before requesting critical section
        client_obj.request_critical_section()

def client(pid):
    client_obj = Client(pid)
    threading.Thread(target=client_process, args=(client_obj,)).start()

if__name__==  "__main__":
    # Start three client processes
    for i in range(3):
        client(i + 1)
```

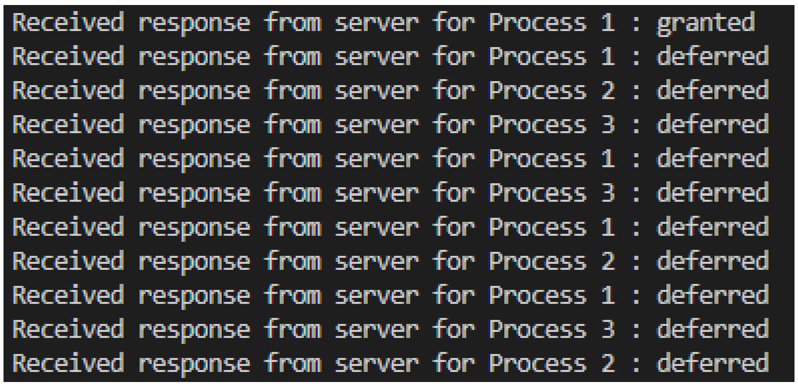
OUTPUT:

Server:



```
Server is listening...
Connected by ('127.0.0.1', 52436)
Received request from Process 52436 with timestamp 1
Granted access to Process 52436
Connected by ('127.0.0.1', 52437)
Received request from Process 52437 with timestamp 2
Process 52437 added to pending requests
Connected by ('127.0.0.1', 52438)
Received request from Process 52438 with timestamp 1
Process 52438 added to pending requests
Connected by ('127.0.0.1', 52439)
```

Client:



```
Received response from server for Process 1 : granted
Received response from server for Process 1 : deferred
Received response from server for Process 2 : deferred
Received response from server for Process 3 : deferred
Received response from server for Process 1 : deferred
Received response from server for Process 3 : deferred
Received response from server for Process 1 : deferred
Received response from server for Process 2 : deferred
Received response from server for Process 1 : deferred
Received response from server for Process 3 : deferred
Received response from server for Process 2 : deferred
```

Web Reference:

[1] https://github.com/trunc8/ricart-agrawala-algorithm