Name -Sumit Kumar Jha
Reg -20204212
Sec - CSE C

Motilal Nehru National Institute of Technology Allahabad Prayagraj
Distributed System (CS17201)
B.Tech (CSE) – VII Sem Lab 4

1. Simulate the Distributed Mutual Exclusion

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>

int critical_section = 0; // Shared resource
int num_processes = 5;    // Number of processes

// Mutex variables
pthread_mutex_t mutex;
pthread_cond_t request_cv;

// Function to sleep for a specified number of milliseconds
void delay_ms(int milliseconds)
{
    struct timespec ts;
    ts.tv_sec = milliseconds / 1000;
    ts.tv_nsec = (milliseconds % 1000) * 1000000;
    nanosleep(&ts, NULL);
}

// Function to request access to the critical section
void request_critical_section(int process_id)
{
    pthread_mutex_lock(&mutex);
    // Send request to the centralized server
    // You would typically send a message to the server here
    printf("Process %d requesting access to the critical section\n",
process_id);
    pthread_cond_wait(&request_cv, &mutex);
```

```c
    pthread_mutex_unlock(&mutex);
}

// Function to release access to the critical section
void release_critical_section(int process_id)
{
    pthread_mutex_lock(&mutex);
    // Notify the server that you are done
    // You would typically send a message to the server here
    printf("Process %d releasing critical section\n", process_id);
    pthread_cond_broadcast(&request_cv);
    pthread_mutex_unlock(&mutex);
}

// Simulated process
void *process(void *arg)
{
    int process_id = *(int *)arg;
    while (1)
    {
        request_critical_section(process_id);

        // Critical Section
        printf("Process %d is in the critical section\n", process_id);

        // Simulated work in the critical section
        delay_ms(1000); // Delay for 1 second
        release_critical_section(process_id);

        // Non-critical Section
        printf("Process %d is in the non-critical section\n",
process_id);

        // Simulated work in the non-critical section
        delay_ms(1000); // Delay for 1 second
    }
    pthread_exit(NULL);
}
int main()
```

```c
{
    pthread_t threads[num_processes];
    int process_ids[num_processes];

    // Initialize mutex and condition variable
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&request_cv, NULL);

    // Create and start threads
    for (int i = 0; i < num_processes; i++)
    {
        process_ids[i] = i;
        pthread_create(&threads[i], NULL, process, &process_ids[i]);
    }

    // Wait for threads to finish
    for (int i = 0; i < num_processes; i++)
        pthread_join(threads[i], NULL);

    // Cleanup
    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&request_cv);
    return 0;
}
```