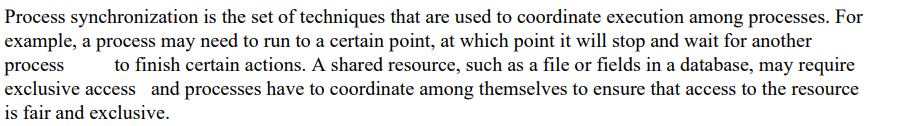
DS A5



mutual exclusion: semaphores or locking mechanism

Critical Section refers to the segment of code or the program which tries to access or modify the value of the variables in a shared resource.

Lock on resource and permission to access that resource exclusively.

P1 finished using resource, release the lock, p2 gets resource, locks it.

A picture containing text, font, screenshot, algebra

Description automatically generated

A picture containing text, screenshot, font, letter

Description automatically generated

TOKEN RING ALGO

a logical ring is constructed in software. Each process is assigned a position in the ring and each process must know who is next to it in the ring.

DS A3

MPI (Message Passing Interface) is a standardized communication protocol and library for developing distributed and parallel computing applications.

To develop a distributed system in Java using either MPI or OpenMP for finding the sum of N elements in an array, we'll choose MPI (Message Passing Interface) as the communication framework. Here's an example implementation using the MPJ Express library, which is a Java implementation of MPI:

import mpi.\*;

public class DistributedSum {

public static void main(String[] args) throws MPIException {

MPI.Init(args);

int rank = MPI.COMM\_WORLD.Rank(); // Get the rank of the current process

int size = MPI.COMM\_WORLD.Size(); // Get the total number of processes

int N = 100; // Total number of elements in the array

int n = size; // Number of processors (assumed equal to the number of processes)

// Calculate the number of elements to distribute to each process

int elementsPerProcess = N / n;

// Create the array and initialize it

int[] array = new int[N];

for (int i = 0; i < N; i++) {

array[i] = i + 1;

}

// Calculate the starting and ending indices for each process

int startIndex = rank \* elementsPerProcess;

int endIndex = startIndex + elementsPerProcess - 1;

// Calculate the local sum for each process

int localSum = 0;

for (int i = startIndex; i <= endIndex; i++) {

localSum += array[i];

}

// Display the intermediate sum calculated at each process

System.out.println("Process " + rank + " - Local Sum: " + localSum);

// Perform global sum reduction using MPI

int[] globalSum = new int[1];

MPI.COMM\_WORLD.Reduce(new int[]{localSum}, 0, globalSum, 0, 1, MPI.INT, MPI.SUM, 0);

// Display the final sum calculated at the root process

if (rank == 0) {

System.out.println("Final Sum: " + globalSum[0]);

}

MPI.Finalize();

}

}

Once you have MPJ Express set up, you can compile and run the code using the following commands:

bashCopy code

mpjrun DistributedSum.java -np <number\_of\_processes>

Replace **<number\_of\_processes>** with the desired number of processes for your distributed system.

A screenshot of a computer program

Description automatically generated with medium confidence