**LAB EVALUATION 1**

**Distributed systems**

**Set 1**

**NITHIN KM  
CB.EN.U4CSE20342**

Question 1:

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#include <time.h>

#define MAX\_CLOCK 10

*int* main(*int* *argc*, *char* \*\**argv*) {

    MPI\_Init(&*argc*, &*argv*);

*int* loser=10;

*int* np, rank;

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &np);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

*int* bomb = 0;

*int* clock\_value;

    if (rank == 0) {

        // Initialize the countdown value

*int* N = 45;

        bomb = N;

        printf("Process %d initialized bomb with value %d\n", rank, bomb);

        // Send the bomb to the next process

        MPI\_Send(&bomb, 1, MPI\_INT, (rank+1)%np, 0, MPI\_COMM\_WORLD);

    }

    while (1) {

        MPI\_Barrier(MPI\_COMM\_WORLD);

        // Receive the bomb

        MPI\_Recv(&bomb, 1, MPI\_INT, (rank-1+np)%np, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

        // Decrement the bomb by a random number (clock value)

        clock\_value = rand() % MAX\_CLOCK;

        bomb -= clock\_value;

        printf("Process %d has received the bomb (%d on the clock) and has decremented it to %d\n", rank, clock\_value, bomb);

        // Check if the bomb has reached zero or less

        if (bomb <= 0) {

            printf("Process %d lost\n", rank);

            // Notify all other processes of the loser's rank

            loser = rank;

            MPI\_Bcast(&loser, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

            break;

        }

        // Pass the bomb to the next process

        MPI\_Send(&bomb, 1, MPI\_INT, (rank+1)%np, 0, MPI\_COMM\_WORLD);

        // MPI\_Barrier(MPI\_COMM\_WORLD);

    }

    // Display the loser's rank for each process

    // MPI\_Barrier(MPI\_COMM\_WORLD);

    MPI\_Bcast(&loser, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

    printf("I am process %d and %d is the loser\n", rank, loser);

    if(rank != loser)

    {

        printf("I am process %d and %d is the loser\n", rank, loser);

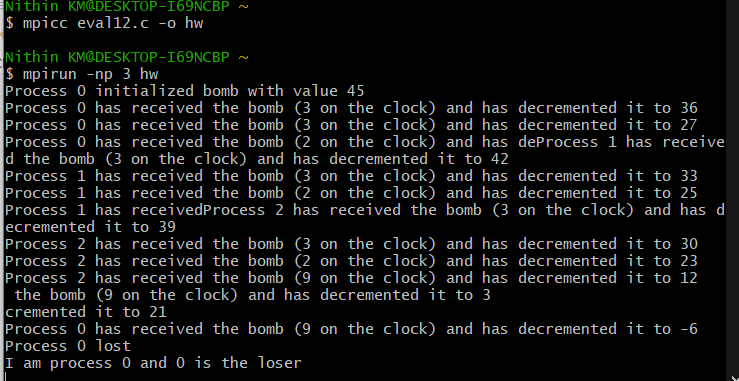
    }

    MPI\_Finalize();

    return 0;

}

Output:



Question 2:

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#define ARRAY\_SIZE 1000

*int* main(*int* *argc*, *char*\*\* *argv*) {

*int* rank, size, chunk\_size, i;

*int* TARGET\_VALUE = 42;

*int* array[ARRAY\_SIZE];

*int* local\_count = 0, total\_count = 0;

*int*\* recv\_counts = NULL, \* recv\_displs = NULL;

    MPI\_Init(&*argc*, &*argv*);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

    // Generate array and target value in the master process

    if (rank == 0) {

        srand(12345);

        for (i = 0; i < ARRAY\_SIZE; i++) {

            array[i] = rand() % 100;

        }

    }

    // Broadcast the target value to all processes

    MPI\_Bcast(&TARGET\_VALUE, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

    // Determine chunk size for each process

    chunk\_size = ARRAY\_SIZE / size;

    // Allocate memory for recv\_counts and recv\_displs arrays

    recv\_counts = (*int*\*)malloc(size \* sizeof(*int*));

    recv\_displs = (*int*\*)malloc(size \* sizeof(*int*));

    // Scatter the array to all processes

    MPI\_Scatter(array, chunk\_size, MPI\_INT, array + rank \* chunk\_size, chunk\_size, MPI\_INT, 0, MPI\_COMM\_WORLD);

    // Search for the target value in the local array

    for (i = 0; i < chunk\_size; i++) {

        if (array[rank \* chunk\_size + i] == TARGET\_VALUE) {

            local\_count++;

        }

    }

    // Gather the local counts to the master process

    MPI\_Gather(&local\_count, 1, MPI\_INT, recv\_counts, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

    // Display the results in the master process

    if (rank == 0) {

        printf("Total occurrences of target value %d: ", TARGET\_VALUE);

        for (i = 0; i < size; i++) {

            printf("%d ", recv\_counts[i]);

            total\_count += recv\_counts[i];

            recv\_displs[i] = (i == 0) ? 0 : (recv\_displs[i-1] + recv\_counts[i-1]);

        }

        printf("Total: %d",total\_count);

        printf("\n");

        printf("Occurrences from each worker: ");

        for (i = 0; i < size; i++) {

            printf("%d ", recv\_counts[i]);

        }

        printf("\n");

    }

    // Free memory

    free(recv\_counts);

    free(recv\_displs);

    MPI\_Finalize();

    return 0;

}

