Amrita School of Engineering
Department of Computer Science and Engineering
19CSE312: Distributed Systems

Lab-Evaluation-1
Set -2

Date: 04/02/2022 **Topic: MPI** **Time: 2hrs**

1. **Implement a MPI C program that**
   a. **Computes factorial of a number in process 1**
   b. **Generate sum of the series from 1 to n in process 2 given a common value of n sent by process 0. Display the results computed by each process and show the output in Process 0.**

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mpi.h>
#define ROOT 0
#define SUM 1
#define FACT 2

void sum(int num, int sum)
{
    sum = 0;
    for (int i = 0; i <= num; i++)
    {
        sum = sum + i;
    }
}
int fact(int n)
{
    int mul = 1;
    for (int i = 1; i <= n; i++)
        mul *= i;
    return mul;
}
int main(int argc, char **argv)
{
    MPI_Init(NULL, NULL);
    int my_rank, comm_size;
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
    if (comm_size != 3)
    {
        printf("This aplication is designed for 3 processes.\n");
        MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
    }
    if (my_rank == ROOT)
    {
        int n = atoi(argv[1]); // CLA input
```

```
        int fibo_buffer;
        int fact_buf;
        // send n to both 1 and 2
        MPI_Send(&n, 1, MPI_INT, SUM, 0, MPI_COMM_WORLD);
        MPI_Send(&n, 1, MPI_INT, FACT, 0, MPI_COMM_WORLD);
        // recieve from P[1]
        MPI_Recv(&fibo_buffer, n, MPI_INT, SUM, 1, MPI_COMM_WORLD,
                 MPI_SUCCESS);
        // recieve from P[2]
        MPI_Recv(&fact_buf, 1, MPI_INT, FACT, 2, MPI_COMM_WORLD,
                 MPI_SUCCESS);
        //printf("P[%d] -> sum(%d) : ", my_rank, n);
        for (int i = 0; i < n; i++)
            //printf("%d ", fibo_buffer);
        printf("\nP[%d] -> fact(%d) : %d\n", my_rank, n, fact_buf);
    }
    if (my_rank == SUM || my_rank == FACT)
    {
        // Either FIBO or FACT should recieve n from ROOT
        int n; // CLA input
        MPI_Recv(&n, 1, MPI_INT, ROOT, 0, MPI_COMM_WORLD,
                 MPI_SUCCESS);
        int fibo_buffer;
        if (my_rank == SUM)
        {
            sum(n, fibo_buffer);
            MPI_Send(&fibo_buffer, n, MPI_INT, ROOT, 1,
                     MPI_COMM_WORLD);
            printf("P[%d] sent factorial result to P[%d]\n",
                   my_rank, ROOT);
        }
        if (my_rank == FACT)
        {
            int fact_buf;
            fact_buf = fact(n);
            MPI_Send(&fact_buf, 1, MPI_INT, ROOT, 2,
                     MPI_COMM_WORLD);
            printf("P[%d] sent factorial result to P[%d]\n",
                   my_rank, ROOT);
        }
    }
    MPI_Finalize();
    return EXIT_SUCCESS;
}
```

Output:

```
abhinav@abhinav:~/Distributed Systems$ mpicc evalq1.c -o output.out
abhinav@abhinav:~/Distributed Systems$ mpirun --oversubscribe -np 3 ./output.out 8
P[1] sent factorial result to P[0]
P[2] sent sum to P[0]

P[0] -> fact(8) : 40320
```

1b)
Code:

```c
#include <stdio.h>
#include <mpi.h>

int main(int argc, char** argv) {
    int Rank, size, n,sum=0;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &Rank);

    if(Rank == 0){
        //n = 8;
        scanf("%d",&n);
        printf("P[%d] sent %d \n", Rank,n);
        MPI_Send(&n, 1, MPI_INT, 2, 1, MPI_COMM_WORLD);
        MPI_Recv(&sum, 1, MPI_INT, 2, 1, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        printf("sum %d in rank %d\n ",sum,Rank);

    }

    else if(Rank == 2){
        MPI_Recv(&n, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        sum = (n*(n+1))/2;
        MPI_Send(&sum, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);

        printf("P[2] sent sum result to P[0]\n");
    }

    MPI_Finalize();
    return 0;
}
```

Output:

```
abhinav@abhinav:~/Distributed Systems$ mpicc dummy.c -o output.out
abhinav@abhinav:~/Distributed Systems$ mpirun --oversubscribe -np 3 ./output.out
7
P[0] sent 7
P[2] sent sum result to P[0]
sum 28 in rank 0
 abhinav@abhinav:~/Distributed Systems$
```

2. **Write a collective communication-based program in which the broadcaster initializes the vectors X and slave processes to compute the squares of their share of the vector. Collect the output from each process and print the output in Process 0.**

   Code:

   Output: