PPL LAB 4

Name: Abhinav Singh Bhagtana

Section: A

Roll no: 29

Reg no: 230905225

Q1.Write a MPI program using N processes to find 1! + 2! + ... + N! Use scan. Also handle different errors using error handling routines.

#include <stdio.h> #include <mpi.h> #include <stdlib.h>

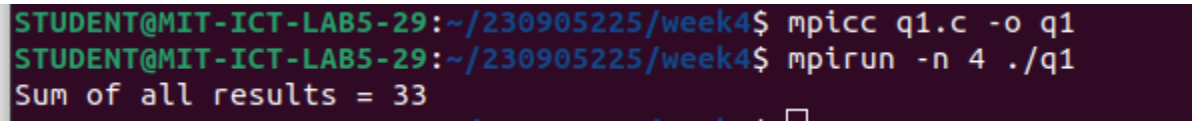
// Helper function to handle and print MPI errors void handle_mpi_error(int err_code, const char *func_name) { if (err_code != MPI_SUCCESS) { char err_string[MPI_MAX_ERROR_STRING]; int result_len; MPI_Error_string(err_code, err_string, &result_len); fprintf(stderr, "MPI Error in %s: %s\n", func_name, err_string); MPI_Abort(MPI_COMM_WORLD, err_code); } }

int main(int argc, char **argv) { int rank, err;

```
err = MPI_Init(&argc, &argv);
if (err != MPI_SUCCESS) {
   fprintf(stderr, "Failed to initialize MPI\n");
   return err;
}

// Set error handler so the program doesn't just die on error
MPI_Comm_set_errhandler(MPI_COMM_WORLD, MPI_ERRORS_RETURN);

err = MPI_Comm_rank(MPI_COMM_WORLD, &rank);
handle_mpi_error(err, "MPI_Comm_rank");

int fact = 1;
for (int i = 2; i <= rank + 1; i++)
   fact *= rank;

int factsum = 0;
err = MPI_Reduce(&fact, &factsum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
handle_mpi_error(err, "MPI_Reduce");
```

```
if (rank == 0) {
   printf("Sum of all results = %d\n", factsum);
}

MPI_Finalize();
return 0;


}
```

OUTPUT:



Q2.Write a MPI program to read a 3x3 matrix. Enter an element to be searched in the root process. Find the number of occurrences of this element in the matrix using three processes.

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

int main(int argc,char **argv) { int rank = 0; MPI_Init(&argc,&argv); MPI_Comm_rank(MPI_COMM_WORLD,&rank); int mat[3][3] = {0}; int destarr[3] = {0}; int count = 0; int find; int totalcount = 0;

```
if(rank == 0){
   printf("enter a 3x3 matrix\n");
   for(int i = 0;i < 3;i++)
      for(int j = 0;j<3;j++)
         scanf("%d",&mat[i][j]);
   printf("enter an element to find:\n");
   scanf("%d",&find);
}

//sending the number to find
MPI_Bcast(&find,1,MPI_INT,0,MPI_COMM_WORLD);
//sending the rows
MPI_Scatter(mat,3,MPI_INT,destarr,3,MPI_INT,0,MPI_COMM_WORLD);
//finding the element
```

```
for(int i = 0;i < 3;i++)
   if(destarr[i] == find)
      count++;
MPI_Reduce(&count,&totalcount,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
if(rank == 0)
   printf("total occurences of %d = %d\n",find,totalcount);
MPI_Finalize();
return 0;


}
```

OUTPUT:

```
STUDENT@MIT-ICT-LAB5-29:~/230905225/week4$ mpicc q2.c -o q2
STUDENT@MIT-ICT-LAB5-29:~/230905225/week4$ mpirun -n 3 ./q2
enter a 3x3 matrix
1 2 3
1 1 2
1 1 1
enter an element to find:
1
total occurences of 1 = 6
```

Q3.write a MPI program to read a 4x4 matrix and display the following output using four processes.

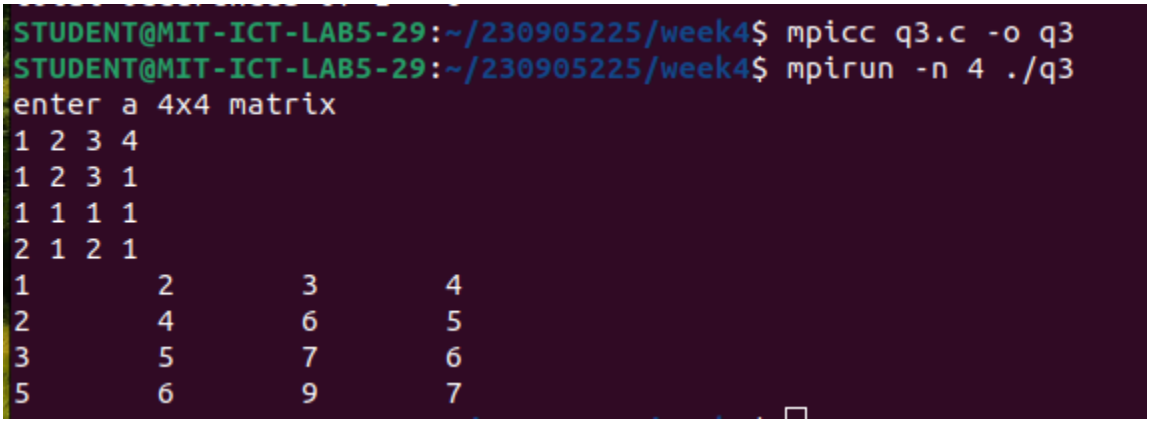
#include <mpi.h> #include <stdio.h>

int main(int argc, char **argv) { int rank; MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);

```
int matrix[4][4] = {0};
int row[4] = {0};
int prefixSum[4] = {0};

if(rank == 0){
   printf("enter a 4x4 matrix\n");
   for(int i = 0;i < 4;i++)
      for(int j = 0; j < 4;j++)
         scanf("%d",&matrix[i][j]);
}
```

```
MPI_Scatter(matrix,4,MPI_INT,row,4,MPI_INT,0,MPI_COMM_WORLD);
MPI_Scan(row,prefixSum,4,MPI_INT,MPI_SUM,MPI_COMM_WORLD);
for(int i = 0;i < 4;i++)
   printf("%d\t",prefixSum[i]);
printf("\n");
MPI_Finalize();
return 0;

}
```

OUTPUT:



Q4.write a mpi program to read a word of length N. Using N processes including the root get output word with the pattern as shown in example. Display the resultant output word in the root.

Inp: PCAP

Output: PCCAAAPPPP

#include <stdio.h> #include <mpi.h> #include <string.h>

int main(int argc, char **argv) { int rank, size; char word[100]; char my_char;

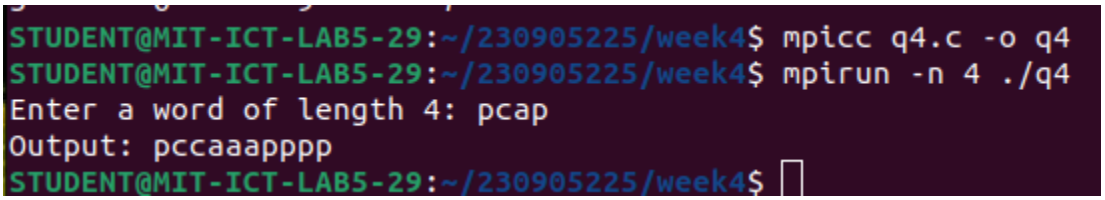
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

int N = size; // The problem states N processes for N length word

if (rank == 0) {

```
  printf("Enter a word of length %d: ", N);
  fflush(stdout);
  scanf("%s", word);
}

// 1. Distribute one character to each process
MPI_Scatter(word, 1, MPI_CHAR, &my_char, 1, MPI_CHAR, 0, MPI_COMM_WORLD);

// 2. Each process creates a local buffer of length N (padding with 0)
char local_buffer[N];
memset(local_buffer, 0, N); // Clear buffer

// Fill the buffer with 'rank + 1' occurrences of the character
for (int i = 0; i <= rank; i++) {
  local_buffer[i] = my_char;
}

// 3. Gather all buffers back to root.
char gathered_matrix[size * N];
MPI_Gather(local_buffer, N, MPI_CHAR, gathered_matrix, N, MPI_CHAR, 0,
MPI_COMM_WORLD);

// 4. Root displays the result
if (rank == 0) {
  printf("Output: ");
  for (int i = 0; i < size * N; i++) {
    if (gathered_matrix[i] != 0) {
      printf("%c", gathered_matrix[i]);
    }
  }
  printf("\n");
}

MPI_Finalize();
return 0;


}
```

OUTPUT:


STUDENT@MIT-ICT-LAB5-29:~/230905225/week4$ mpicc q4.c -o q4
STUDENT@MIT-ICT-LAB5-29:~/230905225/week4$ mpirun -n 4 ./q4
Enter a word of length 4: pcap
Output: pccaaapppp
STUDENT@MIT-ICT-LAB5-29:~/230905225/week4$