**CPP Lab**

**Assignment-5**

**Name-Ashish Goyal**

**Id-2016ucp1100**

**Batch-A (1, 2)**

**Q. Write program(s) implementing use of below MPI functions.**

**MPI\_Init**

**MPI\_Finalize**

**MPI\_Comm\_size**

**MPI\_Comm\_rank**

**MPI\_Send**

**MPI\_Recv**

**MPI\_Bcast**

**MPI\_Reduce**

**MPI\_Wtime**

**Program 1:**

//mnit@mnit-OptiPlex-5040:~/Desktop$ mpicc 1.c

//mnit@mnit-OptiPlex-5040:~/Desktop$ mpirun -np 4 a.out

#include <mpi.h>

#include<stdio.h>

main(int argc, char \*argv[])

{

//current value of time

int npes, myrank;

// Initialize the MPI environment

MPI\_Init(&argc, &argv);

// Get the number of processes

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

// Get the rank of the process

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

float a1=MPI\_Wtime();

printf("start time for process %d is=%f\n\n",myrank,a1);

printf("From process %d out of %d,Hello World!\n\n", myrank, npes);

// Finalize the MPI environment.

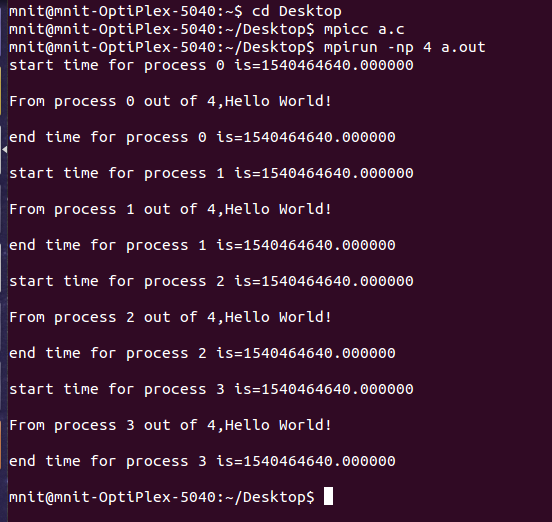
float a2=MPI\_Wtime();

printf("end time for process %d is=%f\n\n",myrank,a2);

MPI\_Finalize();

}

**Output:**

****

**Program 2:**

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

/\* Run with two processes \*/

void main(int argc, char \*argv[])

{

int rank, i, count;

float data[100],value[200];

MPI\_Status status;

MPI\_Init(&argc,&argv);

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

if(rank==1)

{

for(i=0;i<100;++i)

{

data[i]=i;

}

MPI\_Send(data,100,MPI\_FLOAT,0,55,MPI\_COMM\_WORLD);

}

else

{

MPI\_Recv(value,200,MPI\_FLOAT,MPI\_ANY\_SOURCE,55,MPI\_COMM\_WORLD,&status);

printf("P:%d Got data from processor %d \n",rank,status.MPI\_SOURCE);

MPI\_Get\_count(&status,MPI\_FLOAT,&count);

printf("P:%d Got %d elements \n",rank,count);

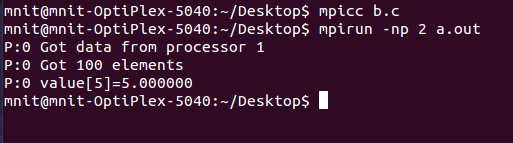
printf("P:%d value[5]=%f \n",rank,value[5]);

}

MPI\_Finalize();

}

**Output:**

****

**Program 3:**

#include <mpi.h>

#include <stdio.h>

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

{

int rank;

int buffer;

const int root=0;

MPI\_Init(&argc, &argv);

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

if(rank == root)

{

buffer=111;

}

printf("[%d]: Before Broadcast, buffer is %d\n", rank, buffer);

/\* everyone calls bcast, data is taken from root and ends up in everyone's buffer \*/

MPI\_Bcast(&buffer, 1, MPI\_INT, root, MPI\_COMM\_WORLD);

printf("[%d]: After Broadcast, buffer is %d\n", rank, buffer);

MPI\_Finalize();

return 0;

}

**Output:**

****

**Program 4:**

#include<stdio.h>

#include <mpi.h>

/\* Run with 16 processes \*/

void main (int argc, char \*argv[])

{

int rank;

struct

{

double value;

int rank;

} in, out;

int root;

MPI\_Init(&argc, &argv);

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

in.value=rank+1;

in.rank=rank;

root=7;

MPI\_Reduce(&in, &out, 1, MPI\_DOUBLE\_INT, MPI\_MAXLOC, root,MPI\_COMM\_WORLD);

if(rank==root)

printf("PE:%d max=%lf at rank %d\n", rank, out.value, out.rank);

MPI\_Reduce(&in, &out, 1, MPI\_DOUBLE\_INT, MPI\_MINLOC, root, MPI\_COMM\_WORLD);

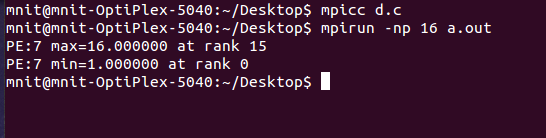
if(rank==root)

printf("PE:%d min=%lf at rank %d\n", rank, out.value, out.rank);

MPI\_Finalize();

}

**Output:**

****