**Lab Practice – MPI**

***19CSE312 – Distributed Systems***

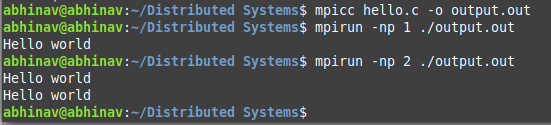
***Date:*** 28-01-2022

1. **Hello World Program with MPI.**

**Code:**

#include <stdio.h>  
#include<mpi.h>  
int main()  
{  
 int commsize;  
 int rank;  
 MPI\_Init(NULL, NULL);   
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &commsize);  
 printf("Hello world\n");  
 MPI\_Finalize();  
 return 0;  
}

**Output:**

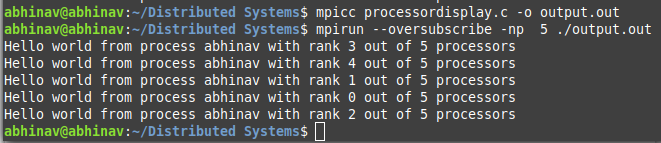
****

1. **Display name of all the processors.**

**Code:**

#include "mpi.h"  
#include <stdio.h>  
  
int main(int argc, char \*\*argv)  
{  
 // Initialize the MPI environment  
 MPI\_Init(&argc, &argv);  
   
 // Get the number of processes ssociated with the communicator  
 int world\_size;  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);  
  
 // Get the rank of the calling process  
 int world\_rank;  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);  
  
 // Get the name of the processor  
 char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];  
 int name\_len;  
 MPI\_Get\_processor\_name(processor\_name, &name\_len);  
  
 printf("Hello world from process %s with rank %d out of %d processors\n", processor\_name, world\_rank, world\_size);  
  
 // Finalize: Any resources allocated for MPI can be freed  
 MPI\_Finalize();  
}

**Output:**

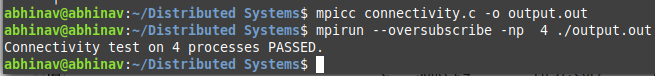
****

1. **Test the connectivity between all processes.**

**Code:**

#include <mpi.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
int main(int argc, char \*\*argv)  
{  
 MPI\_Status status;  
 int verbose = 0;  
 int rank;  
 int np;   
 int peer;  
 int i;  
 int j;  
 int length;  
 char name[MPI\_MAX\_PROCESSOR\_NAME + 1];  
  
 MPI\_Init(&argc, &argv);  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &np);  
  
 if (MPI\_SUCCESS != MPI\_Get\_processor\_name(name, &length)) {  
 strcpy(name, "unknown");  
 }  
  
 if (argc > 1 && strcmp(argv[1], "-v") == 0)  
 verbose = 1;  
  
 for (i = 0; i < np; i++) {  
 if (rank == i) {  
 /\* rank i sends to and receives from each higher rank \*/  
 for (j = i + 1; j < np; j++) {  
 if (verbose)  
 printf("checking connection between rank %d on %s and rank %-4d\n", i, name, j);  
 MPI\_Send(&rank, 1, MPI\_INT, j, rank, MPI\_COMM\_WORLD);  
 MPI\_Recv(&peer, 1, MPI\_INT, j, j, MPI\_COMM\_WORLD, &status);  
 }  
 } else if (rank > i) {  
 /\* receive from and reply to rank i \*/  
 MPI\_Recv(&peer, 1, MPI\_INT, i, i, MPI\_COMM\_WORLD, &status);  
 MPI\_Send(&rank, 1, MPI\_INT, i, rank, MPI\_COMM\_WORLD);  
 }  
 }  
  
 MPI\_Barrier(MPI\_COMM\_WORLD);  
 if (rank == 0)  
 printf("Connectivity test on %d processes PASSED.\n", np);  
  
 MPI\_Finalize();  
 return 0;  
}

**Output:**

****

1. **To practice simple send and receive.**

**Code:**

#include<stdio.h>  
#include<mpi.h>  
int main(int argc, char \*\*argv)  
{  
 int size;  
 int rank;  
 MPI\_Init(NULL, NULL);   
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  
  
 MPI\_Request req,req1;  
 MPI\_Status status;  
  
 int a[4];  
 for (int i=0;i<4;i++)  
 {  
 scanf("%d",&a[i]);  
 }  
 if (size<2)  
 {  
 printf("World size should be greater than 2 but given %d",size);  
 MPI\_Abort(MPI\_COMM\_WORLD, 1);  
 }  
 printf("the array is of size 4 and enter 4 elements:\n");  
   
 if (rank==1)  
 {  
 MPI\_Isend(a,4,MPI\_INT,0,0,MPI\_COMM\_WORLD,&req);  
 printf("Sent to Process 1 Successfully\n");  
  
 }else  
 {  
MPI\_Irecv(a,4,MPI\_INT,1,0,MPI\_COMM\_WORLD,&req1);  
 printf("Process %d received array: [ ", 1);  
 for (int i = 0; i < 4; i++)  
 {  
 if (i==4)  
 {  
 printf("%d ", a[i]);  
 }  
 else{  
 printf("%d, ", a[i]);  
 }  
 }  
 printf("]\n");  
}  
MPI\_Finalize();  
}

**Output:**

****

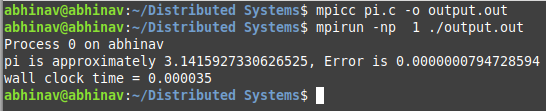
****

1. **To calculate the pi value.**

**Code:**

#include "mpi.h"  
#include <stdio.h>  
#include <math.h>  
  
double f( double );  
double f( double a )  
{  
 return (4.0 / (1.0 + a\*a));  
}  
  
int main( int argc, char \*argv[])  
{  
 int done = 0, n, myid, numprocs, i;  
 double PI25DT = 3.141592653589793238462643;  
 double mypi, pi, h, sum, x;  
 double startwtime = 0.0, endwtime;  
 int namelen;  
 char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];  
  
 MPI\_Init(&argc,&argv);  
 MPI\_Comm\_size(MPI\_COMM\_WORLD,&numprocs);  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD,&myid);  
 MPI\_Get\_processor\_name(processor\_name,&namelen);  
  
 fprintf(stderr,"Process %d on %s\n",  
 myid, processor\_name);  
  
 n = 0;  
 while (!done)  
 {  
 if (myid == 0)  
  
 if (n==0) n=1024\*numprocs; else n=0;  
  
 startwtime = MPI\_Wtime();  
 }  
 MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);  
 if (n == 0)  
 done = 1;  
 else  
 {  
 h = 1.0 / (double) n;  
 sum = 0.0;  
 for (i = myid + 1; i <= n; i += numprocs)  
 {  
 x = h \* ((double)i - 0.5);  
 sum += f(x);  
 }  
 mypi = h \* sum;  
  
 MPI\_Reduce(&mypi, &pi, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);  
  
 if (myid == 0)  
 {  
 printf("pi is approximately %.16f, Error is %.16f\n",  
 pi, fabs(pi - PI25DT));  
 endwtime = MPI\_Wtime();  
 printf("wall clock time = %f\n",  
 endwtime-startwtime);  
 }  
 }  
 }  
 MPI\_Finalize();  
  
 return 0;  
}

**Output:**

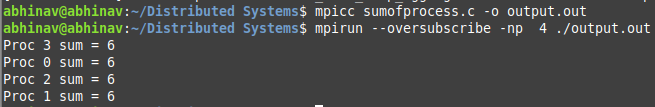
****

1. **To find sum of ranks of all the processes.**

**Code:**

#include <stdio.h>  
#include <mpi.h>  
  
void main(int argc, char \*argv[])  
{  
 int myrank, nprocs, leftid, rightid;  
 int val, sum, tmp;  
 MPI\_Status recv\_status, send\_status;  
 MPI\_Request send\_request;  
   
 MPI\_Init(&argc, &argv);  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myrank);  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &nprocs);  
   
 if ((leftid=(myrank-1)) < 0)   
 leftid = nprocs-1;  
 if ((rightid=(myrank+1)) == nprocs)   
 rightid = 0;  
   
 val = myrank;  
 sum = 0;  
 do {  
 MPI\_Issend(&val,1,MPI\_INT,rightid,99,MPI\_COMM\_WORLD,&send\_request);  
 MPI\_Recv(&tmp,1,MPI\_INT,leftid,99,MPI\_COMM\_WORLD,&recv\_status);  
 MPI\_Wait(&send\_request,&send\_status);  
 val = tmp;  
 sum += val;  
 }   
 while (val != myrank);  
  
 printf("Proc %d sum = %d \n", myrank, sum);  
 MPI\_Finalize();  
}

**Output:**

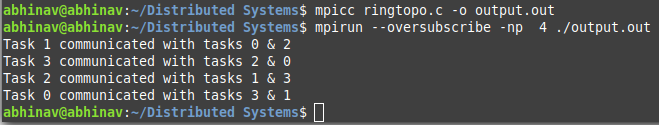
****

1. **Implementing simple ring topology.**

**Code:**

#include "mpi.h"  
#include <stdio.h>  
#include <stdlib.h>  
  
int main (int argc, char \*argv[])  
{  
int numtasks, rank, next, prev, buf[2], tag1=1, tag2=2;  
MPI\_Request reqs[4];  
MPI\_Status stats[4];  
  
MPI\_Init(&argc,&argv);  
MPI\_Comm\_size(MPI\_COMM\_WORLD, &numtasks);  
MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  
  
prev = rank-1;  
next = rank+1;  
if (rank == 0) prev = numtasks - 1;  
if (rank == (numtasks - 1)) next = 0;  
  
MPI\_Irecv(&buf[0], 1, MPI\_INT, prev, tag1, MPI\_COMM\_WORLD, &reqs[0]);  
MPI\_Irecv(&buf[1], 1, MPI\_INT, next, tag2, MPI\_COMM\_WORLD, &reqs[1]);  
  
MPI\_Isend(&rank, 1, MPI\_INT, prev, tag2, MPI\_COMM\_WORLD, &reqs[2]);  
MPI\_Isend(&rank, 1, MPI\_INT, next, tag1, MPI\_COMM\_WORLD, &reqs[3]);  
  
MPI\_Waitall(4, reqs, stats);  
printf("Task %d communicated with tasks %d & %d\n",rank,prev,next);  
  
MPI\_Finalize();  
}

**Output:**

****

1. **To practice with reduce operation.**

**Code:**

#include <stdio.h>  
#include <stdlib.h>  
#include <mpi.h>  
  
int main(int argc, char\* argv[])  
{  
 MPI\_Init(&argc, &argv);  
  
 // Determine root's rank  
 int root\_rank = 0;  
  
 // Get the size of the communicator  
 int size = 0;  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  
 if(size != 4)  
 {  
 printf("This application is meant to be run with 4 MPI processes.\n");  
 MPI\_Abort(MPI\_COMM\_WORLD, EXIT\_FAILURE);  
 }  
  
 // Get my rank  
 int my\_rank;  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_rank);  
  
 // Each MPI process sends its rank to reduction, root MPI process collects the result  
 int reduction\_result = 0;  
 MPI\_Reduce(&my\_rank, &reduction\_result, 1, MPI\_INT, MPI\_SUM, root\_rank, MPI\_COMM\_WORLD);  
  
 if(my\_rank == root\_rank)  
 {  
 printf("The sum of all ranks is %d.\n", reduction\_result);  
 }  
  
 MPI\_Finalize();  
  
 return EXIT\_SUCCESS;  
}

**Output:**

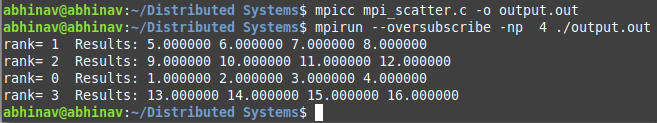
****

1. **To Scatter the data among the processes.**

**Code:**

#include "mpi.h"  
#include <stdio.h>  
#include <stdlib.h>  
#define SIZE 4  
  
int main (int argc, char \*argv[])  
{  
int numtasks, rank, sendcount, recvcount, source;  
float sendbuf[SIZE][SIZE] = {  
 {1.0, 2.0, 3.0, 4.0},  
 {5.0, 6.0, 7.0, 8.0},  
 {9.0, 10.0, 11.0, 12.0},  
 {13.0, 14.0, 15.0, 16.0} };  
float recvbuf[SIZE];  
  
MPI\_Init(&argc,&argv);  
MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  
MPI\_Comm\_size(MPI\_COMM\_WORLD, &numtasks);  
  
if (numtasks == SIZE) {  
 source = 1;  
 sendcount = SIZE;  
 recvcount = SIZE;  
 MPI\_Scatter(sendbuf,sendcount,MPI\_FLOAT,recvbuf,recvcount,  
 MPI\_FLOAT,source,MPI\_COMM\_WORLD);  
  
 printf("rank= %d Results: %f %f %f %f\n",rank,recvbuf[0],  
 recvbuf[1],recvbuf[2],recvbuf[3]);  
 }  
else  
 printf("Must specify %d processors. Terminating.\n",SIZE);  
  
MPI\_Finalize();  
}

**Output:**

****