**CPP Lab**

**Assignment-6**

**Name-Ashish Goyal**

**Id-2016ucp1100**

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

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

**MPI\_Gather()**

**MPI\_Allgather()**

**MPI\_Scatter()**

**MPI\_Alltoall(****)**

**MPI\_Alltoallv()**

**MPI\_Get\_count()**

**MPI\_Issend()**

**MPI\_Irecv()**

**MPI\_Test()**

**Program 1:**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

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

{

int size, rank;

MPI\_Init(&argc, &argv);

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

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

int \*globaldata=NULL;

int localdata;

int i;

if (rank == 0)

{

globaldata = malloc(size \* sizeof(int) );

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

globaldata[i] = 2\*i+1;

printf("Processor %d has value: ", rank);

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

printf("%d ", globaldata[i]);

printf("\n");

}

MPI\_Scatter(globaldata, 1, MPI\_INT, &localdata, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

printf("Processor %d has value %d\n", rank, localdata);

localdata \*= 2;

printf("Processor %d doubled the value, updated value now is %d\n", rank, localdata);

MPI\_Gather(&localdata, 1, MPI\_INT, globaldata, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (rank == 0)

{

printf("Processor %d has value: ", rank);

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

printf("%d ", globaldata[i]);

printf("\n");

}

if (rank == 0)

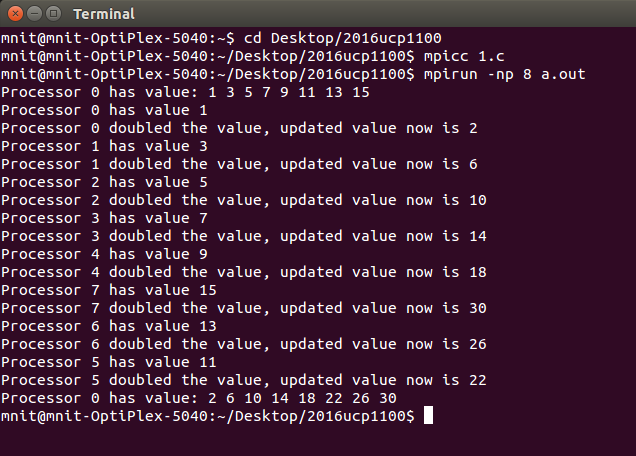
free(globaldata);

MPI\_Finalize();

return 0;

}

**Output:**

****

**Program 2:**

#include <stdio.h>

#include <stdlib.h>

#include "mpi.h"

struct mystruct{

int sendarray[4];

int a;

char array2[4];

};

typedef struct mystruct struct\_t;

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

{

int rank, size;

struct\_t fd;

struct\_t recv;

int i, j;

MPI\_Init(&argc, &argv);

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

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

// init

for (i=0;i<4;i++){

fd.sendarray[i] = 0;

fd.array2[i] = 0;

recv.sendarray[i] =999;

recv.array2[i] = 99;

}

recv.a =999;

// put some stuff in your array

fd.sendarray[rank] = rank\*10;

fd.array2[rank] = (char)(rank\*20);

fd.a = rank;

printf("rank : %d, fd.sendarray[%d] : %d\n", rank, rank, fd.sendarray[rank]);

// gather data from all now.. send the int:

MPI\_Allgather (&(fd.sendarray[rank]), 1, MPI\_INT, recv.sendarray, 1, MPI\_INT, MPI\_COMM\_WORLD);

// then the char

MPI\_Allgather (&(fd.array2[rank]), 1, MPI\_CHAR, recv.array2, 1, MPI\_CHAR, MPI\_COMM\_WORLD);

// check if correct data has been received

if (rank == 0) {

printf("Received:\n");

printf("---\n");

printf("int array: ");

for (j=0; j<4; j++) printf("%3d ", recv.sendarray[j]);

printf("\nchar array: ");

for (j=0; j<4; j++) printf("%3d ", (int)(recv.array2[j]));

printf("\n");

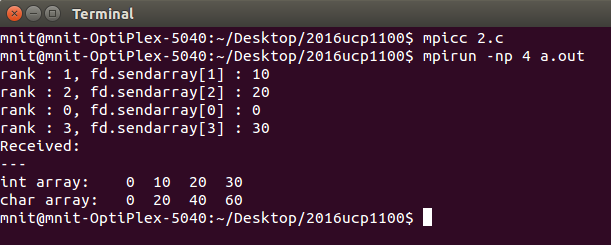
}

MPI\_Finalize();

return 0;

}

**Output:**



**Program 3:**

#include <stdio.h>

#include "mpi.h"

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

{

int send[4], recv[3];

int rank, size, k;

MPI\_Init( &argc, &argv );

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

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

if (size != 4) {

printf("Error,Number of processors must be equal to 4");

printf("Program aborting");

MPI\_Abort(MPI\_COMM\_WORLD, 1);

}

for (k=0;k<size;k++)

send[k] = (k+1) + rank\*size;

printf("Rank %d : send = %d %d %d %d\n", rank, send[0], send[1], send[2], send[3]);

MPI\_Alltoall(&send, 1, MPI\_FLOAT, &recv, 1, MPI\_INT, MPI\_COMM\_WORLD);

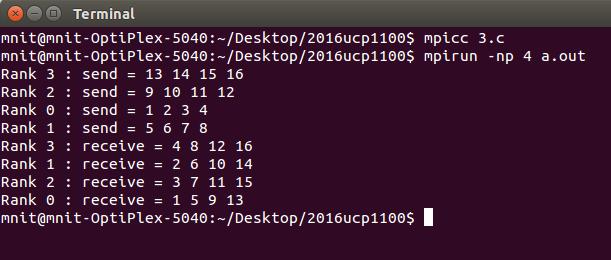
printf("Rank %d : receive = %d %d %d %d\n", rank, recv[0], recv[1], recv[2], recv[3]);

MPI\_Finalize();

return 0;

}

**Output:**



**Program 4:**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

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

{

MPI\_Init(NULL, NULL);

int world\_size;

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

int world\_rank;

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

int number\_amount;

if (world\_rank == 0)

{

const int MAX\_NUMBERS = 100;

int numbers[MAX\_NUMBERS];

srand(time(NULL));

number\_amount = (rand() / (float)RAND\_MAX) \* MAX\_NUMBERS;

MPI\_Send(numbers, number\_amount, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

printf("Rank 0 sent %d numbers to Rank 1\n", number\_amount);

}

else if (world\_rank == 1)

{

MPI\_Status status;

MPI\_Probe(0, 0, MPI\_COMM\_WORLD, &status);

MPI\_Get\_count(&status, MPI\_INT, &number\_amount);

int\* number\_buf = (int\*)malloc(sizeof(int) \*number\_amount);

MPI\_Recv(number\_buf, number\_amount, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE);

printf("Rank 1 dynamically received %d numbers from Rank 0.\n",number\_amount);

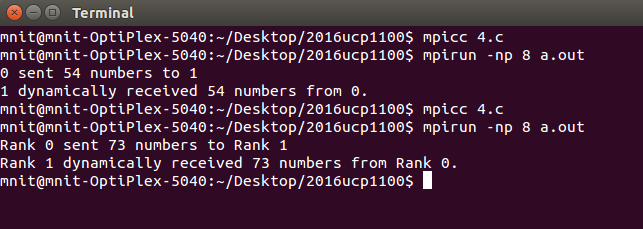
free(number\_buf);

}

MPI\_Finalize();

}

**Output:**

****

**Program 5:**

#include "mpi.h"

#include <stdio.h>

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

{

int myid, numprocs, i, j;

int buffer[10], buffer2[10];

MPI\_Request req, req2;

MPI\_Status status;

MPI\_Init(NULL,NULL);

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

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

j = (myid + 1) % numprocs;

i = myid - 1;

if (i < 0)

i = numprocs - 1;

if(MPI\_Issend(buffer2, 10, MPI\_INT,j, 123, MPI\_COMM\_WORLD, &req2)==0)

{

printf("Success in Issend()\n");

}

if(MPI\_Irecv(buffer, 10, MPI\_INT,i, 123, MPI\_COMM\_WORLD, &req)==0)

{

printf("Success in Irecv()\n");

}

MPI\_Wait(&req, &status);

MPI\_Wait(&req2, &status);

MPI\_Finalize();

return 0;

}

**Output:**

****

**Program 6:**

#include "mpi.h"

#include <stdio.h>

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

{

int myid, numprocs, left, right, flag=0;

int buffer[10], buffer2[10];

MPI\_Request request;

MPI\_Status status;

MPI\_Init(&argc,&argv);

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

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

right = (myid + 1) % numprocs;

left = myid - 1;

if (left < 0)

left = numprocs - 1;

MPI\_Irecv(buffer, 10, MPI\_INT, left, 123, MPI\_COMM\_WORLD, &request);

MPI\_Send(buffer2, 10, MPI\_INT, right, 123, MPI\_COMM\_WORLD);

MPI\_Test(&request, &flag, &status);

if(flag)

printf("Test Passed request Accepted\n");

else

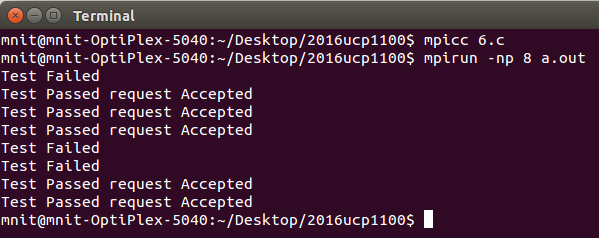
printf("Test Failed\n");

MPI\_Finalize();

return 0;

}

**Output:**

****