**PARALLEL AND DISTRIBUTED COMPUTING- LAB EXERCISES (MPI)**

**NAME: Aman Saxena**

**REGISTRATION NUMBER: 15BCE0299**

**FACULTY NAME: SRIMATHI.C**

**Question:** Write 2 Simple MPI Programs which demonstrates the following

4. MPI – Collective operation with ‘synchronization’

5. MPI – Collective operation with ‘data movement’

**SYNCHRONIZATION:**

The MPI\_Barrier function can be used to synchronize a group of processes. To synchronize a group of processes, each one must call **MPI\_Barrier** when it has reached a point where it can go no further until it knows that all its cohorts have reached the same point. Once a process has called MPI\_Barrier, it will be blocked until all processes in the group have also called MPI\_Barrier.

**Example 1:**

#include "mpi.h"

#include <stdio.h>

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

{

int rank, nprocs;

MPI\_Init(&argc,&argv);

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

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

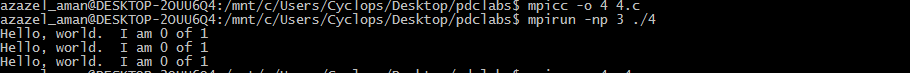
MPI\_Barrier(MPI\_COMM\_WORLD);

printf("Hello, world.  I am %d of %d\n", rank, nprocs);fflush(stdout);

MPI\_Finalize();

return 0;

}



**Example 2:**

#include <unistd.h>

#include <stdlib.h>

#include <mpi.h>

#include <stdio.h>

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

int rank, size;

/\* initialize MPI \*/

MPI\_Init(&argc, &argv);

/\* get the rank (process rank) and size (number of processes) \*/

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

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

/\* we start to work \*/

printf("Process %d starting!\n", rank);

/\* simulate the processes taking slightly different amounts of time by sleeping

\* for our process rank seconds \*/

sleep(rank);

printf("Process %d is done its work!\n", rank);

/\* a barrier \*/

MPI\_Barrier(MPI\_COMM\_WORLD);

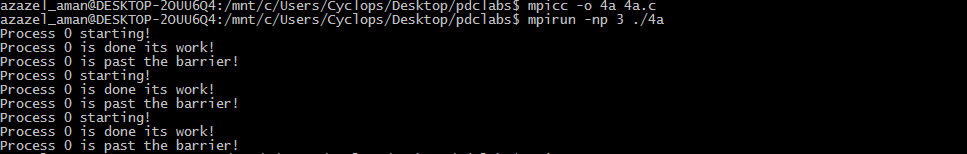
printf("Process %d is past the barrier!\n", rank);

/\* quit MPI \*/

MPI\_Finalize( );

return 0;

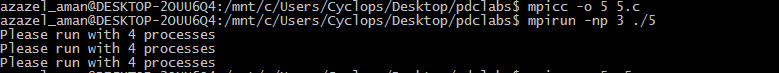
**}**



**DATA MOVEMENT:**

**Example 1:**

#include "mpi.h"  
#include <stdio.h>  
  
int main(int argc, char \*argv[])  
{  
    int buffer[6];  
    int rank, size, i;  
    int receive\_counts[4] = { 0, 1, 2, 3 };  
    int receive\_displacements[4] = { 0, 0, 1, 3 };  
  
    [MPI\_Init](http://mpi.deino.net/mpi_functions/MPI_Init.html)(&argc, &argv);  
    [MPI\_Comm\_size](http://mpi.deino.net/mpi_functions/MPI_Comm_size.html)(MPI\_COMM\_WORLD, &size);  
    [MPI\_Comm\_rank](http://mpi.deino.net/mpi_functions/MPI_Comm_rank.html)(MPI\_COMM\_WORLD, &rank);  
    if (size != 4)  
    {  
        if (rank == 0)  
        {  
            printf("Please run with 4 processes\n");fflush(stdout);  
        }  
        [MPI\_Finalize](http://mpi.deino.net/mpi_functions/MPI_Finalize.html)();  
        return 0;  
    }  
    for (i=0; i<rank; i++)  
    {  
        buffer[i] = rank;  
    }  
    [MPI\_Gatherv](http://mpi.deino.net/mpi_functions/MPI_Gatherv.html)(buffer, rank, MPI\_INT, buffer, receive\_counts, receive\_displacements, MPI\_INT, 0, MPI\_COMM\_WORLD);  
    if (rank == 0)  
    {  
        for (i=0; i<6; i++)  
        {  
            printf("[%d]", buffer[i]);  
        }  
        printf("\n");  
        fflush(stdout);  
    }  
    [MPI\_Finalize](http://mpi.deino.net/mpi_functions/MPI_Finalize.html)();  
    return 0;  
}



**Example 2:**

|  |  |
| --- | --- |
| #include "mpi.h" #include <stdlib.h> #include <stdio.h> #include <string.h>  #define ROOT 0 #define NUM\_REPS 5 #define NUM\_SIZES 3  int main( int argc, char \*\*argv) {     int \*buf;     int i, rank, reps, n;     int bVerify = 1;     int sizes[NUM\_SIZES] = { 100, 64\*1024, 128\*1024 };     int num\_errors=0, tot\_errors;      [MPI\_Init](http://mpi.deino.net/mpi_functions/MPI_Init.html)(&argc, &argv);     [MPI\_Comm\_rank](http://mpi.deino.net/mpi_functions/MPI_Comm_rank.html)(MPI\_COMM\_WORLD, &rank);     if (argc > 1)     {         if (strcmp(argv[1], "-novalidate") == 0 || strcmp(argv[1], "-noverify") == 0)             bVerify = 0;     }     buf = (int \*) malloc(sizes[NUM\_SIZES-1]\*sizeof(int));     memset(buf, 0, sizes[NUM\_SIZES-1]\*sizeof(int));     for (n=0; n<NUM\_SIZES; n++)     {         if (rank == ROOT)         {             printf("bcasting %d MPI\_INTs %d times\n", sizes[n], NUM\_REPS);             fflush(stdout);         }         for (reps=0; reps < NUM\_REPS; reps++)         {             if (bVerify)             {                 if (rank == ROOT)                 {                     for (i=0; i<sizes[n]; i++)                     {                         buf[i] = 1000000 \* (n \* NUM\_REPS + reps) + i;                     }                 }                 else                 {                     for (i=0; i<sizes[n]; i++)                     {                         buf[i] = -1 - (n \* NUM\_REPS + reps);                     }                 }             }             [MPI\_Bcast](http://mpi.deino.net/mpi_functions/MPI_Bcast.html)(buf, sizes[n], MPI\_INT, ROOT, MPI\_COMM\_WORLD);             if (bVerify)             {                 num\_errors = 0;                 for (i=0; i<sizes[n]; i++)                 {                     if (buf[i] != 1000000 \* (n \* NUM\_REPS + reps) + i)                     {                         num\_errors++;                         if (num\_errors < 10)                         {                             printf("Error: Rank=%d, n=%d, reps=%d, i=%d, buf[i]=%d expected=%d\n", rank, n, reps, i, buf[i],                                     1000000 \* (n \* NUM\_REPS + reps) +i);                             fflush(stdout);                         }                     }                 }                 if (num\_errors >= 10)                 {                     printf("Error: Rank=%d, num\_errors = %d\n", rank, num\_errors);                     fflush(stdout);                 }             }         }     }      [MPI\_Reduce](http://mpi.deino.net/mpi_functions/MPI_Reduce.html)( &num\_errors, &tot\_errors, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD );     if (rank == 0 && tot\_errors == 0)          printf(" No Errors\n");     fflush(stdout);     free(buf);     [MPI\_Finalize](http://mpi.deino.net/mpi_functions/MPI_Finalize.html)();     return 0; } | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |