NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL
DEPARTMENT OF INFORMATION TECHNOLOGY
IT 301 Parallel Computing LAB 8

P Akshara - 181IT132

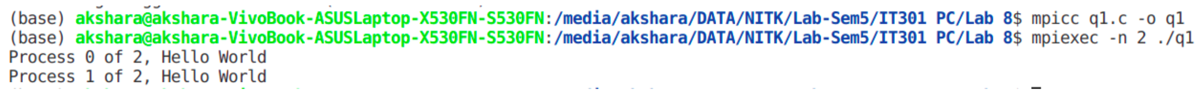
For each program, you must add a screenshot of the output. Write analysis for each observation.

1. MPI “Hello World” program

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
int size, myrank;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
printf("Process %d of %d, Hello World\n", myrank, size);
MPI_Finalize();
return 0;
}
```

Output

2 processes are spawned using n = 2. MPI_Comm_size gives the total no. of processes, MPI_Comm_rank gives the rank(process id) of current process in the default communication world MPI_COMM_WORLD.



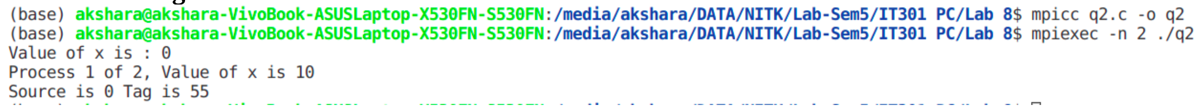
```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 8$ mpicc q1.c -o q1
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 8$ mpiexec -n 2 ./q1
Process 0 of 2, Hello World
Process 1 of 2, Hello World
```

2. Demonstration of MPI_Send() and MPI_Recv(). Sending an Integer.

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
int size, myrank, x, i;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0)
{
x = 10;
MPI_Send(&x, 1, MPI_INT, 1, 55, MPI_COMM_WORLD);
}
else if (myrank == 1)
{
printf("Value of x is : %d\n", x);
MPI_Recv(&x, 1, MPI_INT, 0, 55, MPI_COMM_WORLD, &status);
printf("Process %d of %d, Value of x is %d\n", myrank, size, x);
printf("Source is %d Tag is %d \n", status.MPI_SOURCE, status.MPI_TAG);
}
MPI_Finalize();
return 0;
}
```

Output

Process with rank = 0 sets x=10 and sends it to process with rank=1 in standard mode. For process 1, initial value of x=0, after receiving from process 0, updates x=10. Using status structure we can get the source and tag of the received message.



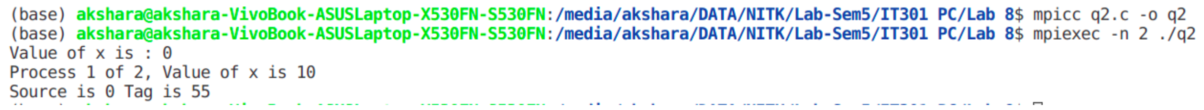
```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 8$ mpicc q2.c -o q2
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 8$ mpiexec -n 2 ./q2
Value of x is : 0
Process 1 of 2, Value of x is 10
Source is 0 Tag is 55
```

Modified

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
int size, myrank, x, i;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0)
{
x = 10;
MPI_Send(&x, 1, MPI_INT, 1, 55, MPI_COMM_WORLD);
}
else if (myrank == 1)
{
printf("Value of x is : %d\n", x);
MPI_Recv(&x, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
printf("Process %d of %d, Value of x is %d\n", myrank, size, x);
printf("Source is %d Tag is %d \n", status.MPI_SOURCE, status.MPI_TAG);
}
MPI_Finalize();
return 0;
}
```

Output

Receives x value from any process having any tag.



```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 8$ mpicc q2.c -o q2
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 8$ mpiexec -n 2 ./q2
Value of x is : 0
Process 1 of 2, Value of x is 10
Source is 0 Tag is 55
```

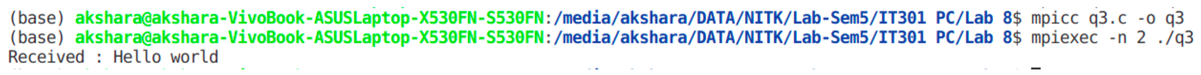
3. Demonstration of MPI_Send() and MPI_Recv(). Sending a string.

```
#include <mpi.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[])
{
char message[20];
int myrank;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0) /* code for process zero */
{
strcpy(message, "Hello world");
MPI_Send(message, strlen(message) + 1, MPI_CHAR, 1, 10, MPI_COMM_WORLD);
}
else if (myrank == 1) /* code for process one */
{
```

```
MPI_Recv(message, 20, MPI_CHAR, 0, 10, MPI_COMM_WORLD, &status);
printf("Received : %s\n", message);
}
MPI_Finalize();
return 0;
}
```

Output

Process 0 sends a string to process 2 in standard mode. No. of data elements sent is len(string) + 1.



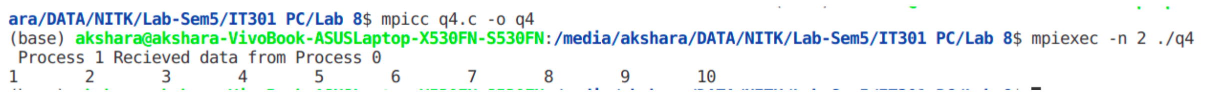
```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 8$ mpicc q3.c -o q3
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 8$ mpiexec -n 2 ./q3
Received : Hello world
```

4. Demonstration of MPI_Send() and MPI_Recv(). Sending elements of an array.

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
int size, myrank, x[50], y[50], i;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0)
{
for (i = 0; i < 50; i++)
x[i] = i + 1;
MPI_Send(x, 10, MPI_INT, 1, 20, MPI_COMM_WORLD);
}
else if (myrank == 1)
{
MPI_Recv(y, 10, MPI_INT, 0, 20, MPI_COMM_WORLD, &status);
printf(" Process %d Recieved data from Process %d\n", myrank, status.MPI_SOURCE);
for (i = 0; i < 10; i++)
printf("%d\t", y[i]);
printf("\n");
}
MPI_Finalize();
return 0;
}
```

Output

Process 0 sends an array x to process 2 in standard mode, with no. of data elements = 10. So 1st 10 elements of y are set equal to those of x, rest 40 are garbage values.



```
ara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 8$ mpicc q4.c -o q4
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 8$ mpiexec -n 2 ./q4
 Process 1 Recieved data from Process 0
1       2       3       4       5       6       7       8       9       10
```

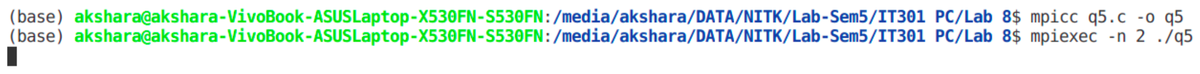
5. Demonstration of Blocking Send and Receive with mismatched tags.

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
int size, myrank, x[50], y[50], i;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0)
{
```

```
for (i = 0; i < 50; i++)
x[i] = i + 1;
MPI_Send(x, 10, MPI_INT, 1, 10, MPI_COMM_WORLD);
}
else if (myrank == 1)
{
MPI_Recv(y, 10, MPI_INT, 0, 1, MPI_COMM_WORLD, &status);
printf(" Process %d Recieved data from Process %d\n", myrank, status.MPI_SOURCE);
for (i = 0; i < 10; i++)
printf("%d\t", y[i]);
}
MPI_Finalize();
return 0;
}
```

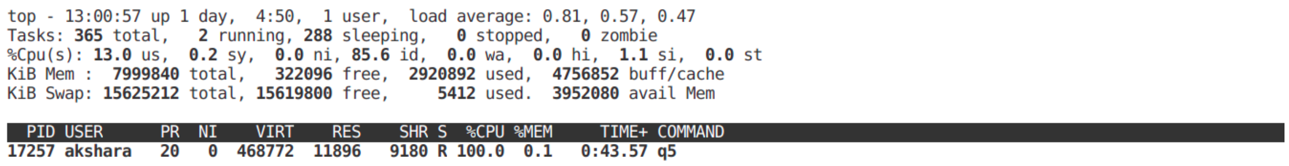
Output

This is standard mode with mismatched tags, so the execution does not proceed.



```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 8$ mpicc q5.c -o q5
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 8$ mpiexec -n 2 ./q5
█
```

Top command shows that the program ‘q5’ is still running in the background.



```
top - 13:00:57 up 1 day,  4:50,  1 user,  load average: 0.81, 0.57, 0.47
Tasks: 365 total,   2 running, 288 sleeping,   0 stopped,   0 zombie
%Cpu(s): 13.0 us,  0.2 sy,  0.0 ni, 85.6 id,  0.0 wa,  0.0 hi,  1.1 si,  0.0 st
KiB Mem :  7999840 total,   322096 free,  2920892 used,  4756852 buff/cache
KiB Swap: 15625212 total, 15619800 free,     5412 used.  3952080 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
17257 akshara   20   0  468772  11896   9180 R 100.0  0.1   0:43.57 q5
```

6. MPI_Send() and MPI_Recv() standard mode

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
int size, myrank, x[10], i, y[10];
MPI_Status status;
MPI_Request request;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0)
{
for (i = 0; i < 10; i++)
{
x[i] = 1;
y[i] = 2;
}
MPI_Send(x, 10, MPI_INT, 1, 1, MPI_COMM_WORLD);
MPI_Send(y, 10, MPI_INT, 1, 2, MPI_COMM_WORLD);
}
else if (myrank == 1)
{
MPI_Recv(x, 10, MPI_INT, 0, 2, MPI_COMM_WORLD, &status);
for (i = 0; i < 10; i++)
printf("Received Array x : %d\n", x[i]);
MPI_Recv(y, 10, MPI_INT, 0, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
for (i = 0; i < 10; i++)
printf("Received Array y : %d\n", y[i]);
}
MPI_Finalize();
return 0;
}
```

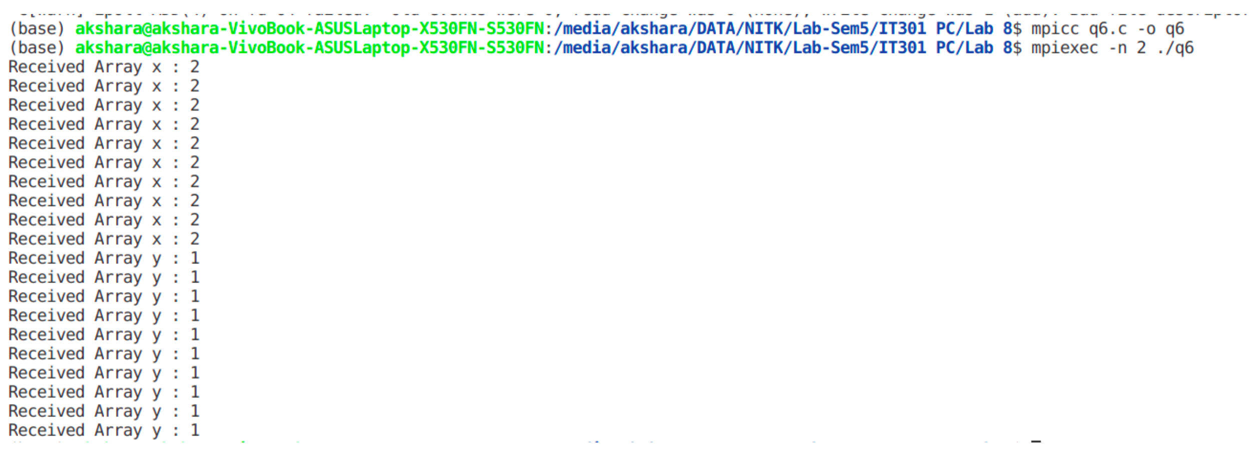
Output

a) In standard mode, when P1 posts MPI_Recv(), it blocks until x is available in its application buffer. But since the tags of send and recv are mismatched, P1 enters into deadlock. The 1st MPI_Send() in P0, returns as the contents of x would reside in the system buffer. Then it posts the 2nd Send(). This tag matches with 1st receive in P1. So x in P1 would recieve y values and Recv() returns and P1 resumes with 2nd Recv(). This tag matches with the 1st Send() from P0. So this also executes. Y in P1 has contents of x in P0 and x in P1 Has contents of y in P0.

b) Tag is a non-negative integer assigned by the programmer to uniquely identify a message. Send and receive operations should have matching message tags. Mismatch could lead to corrupt data, deadlock.

c) Blocking send will return only after it is safe to modify the application buffer. The sending data might have been received or would be in the system buffer. In synchronous case, it is strictly blocking, it won't return until the data has been received. Blocking receive returns only after the data has arrived and is present in the application buffer.
In blocking calls, the computations are halted until the blocked buffer is freed (if system buffer is not available, it may wait for long time, and in case of mismatched tags could go into deadlock). This usually leads to wastage of computational resources as Send/Recv is usually copying data from one memory location to another memory location, while the registers in CPU remain idle as computation resumes only after successful transfer. But it is used when it is sufficient, since it is easier to use.

d) Non-blocking send and recv are used to overlap computation with communication to achieve performance gains. They just request MPI library to perform the operation, but do not wait till it does (copy data from application to system buffer etc). So user buffer cannot be modified without confirming the completion of the operation. Done using routines like Wait(), Test().



```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 8$ mpicc q6.c -o q6
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 8$ mpiexec -n 2 ./q6
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array y : 1
Received Array y : 1
Received Array y : 1
Received Array y : 1
Received Array y : 1
Received Array y : 1
Received Array y : 1
Received Array y : 1
Received Array y : 1
Received Array y : 1
```