NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL
DEPARTMENT OF INFORMATION TECHNOLOGY

IT 301 Parallel Computing LAB 3

26th August 2020

Faculty: Dr. Geetha V and Mrs. Tanmayee

Bhagyashri Bhamare 181IT111

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

2) Steps to execute :

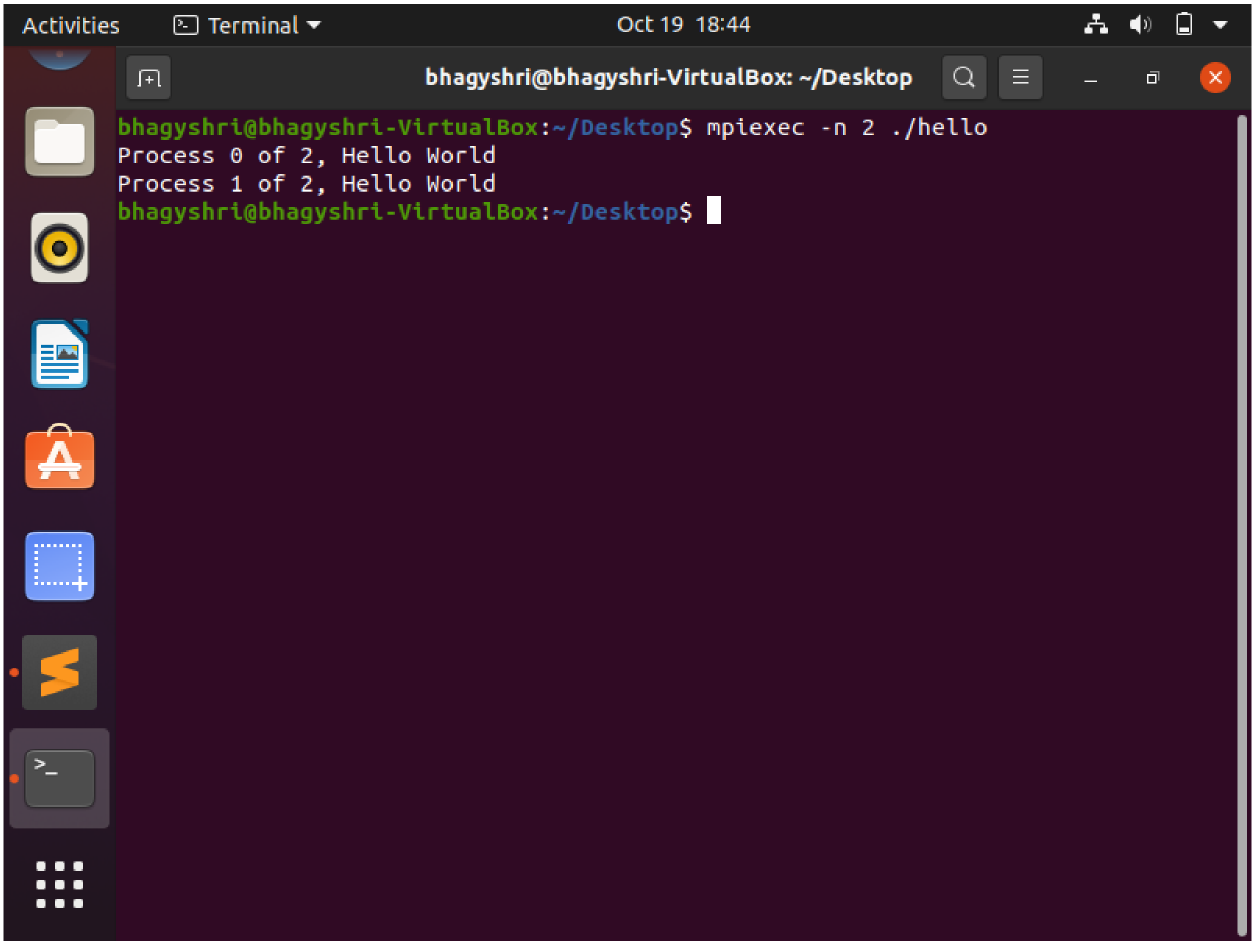
```
mpicc helloworld.c -o hello

mpiexec -n 2 ./hello
```

n is the number of processes to be launched.

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-



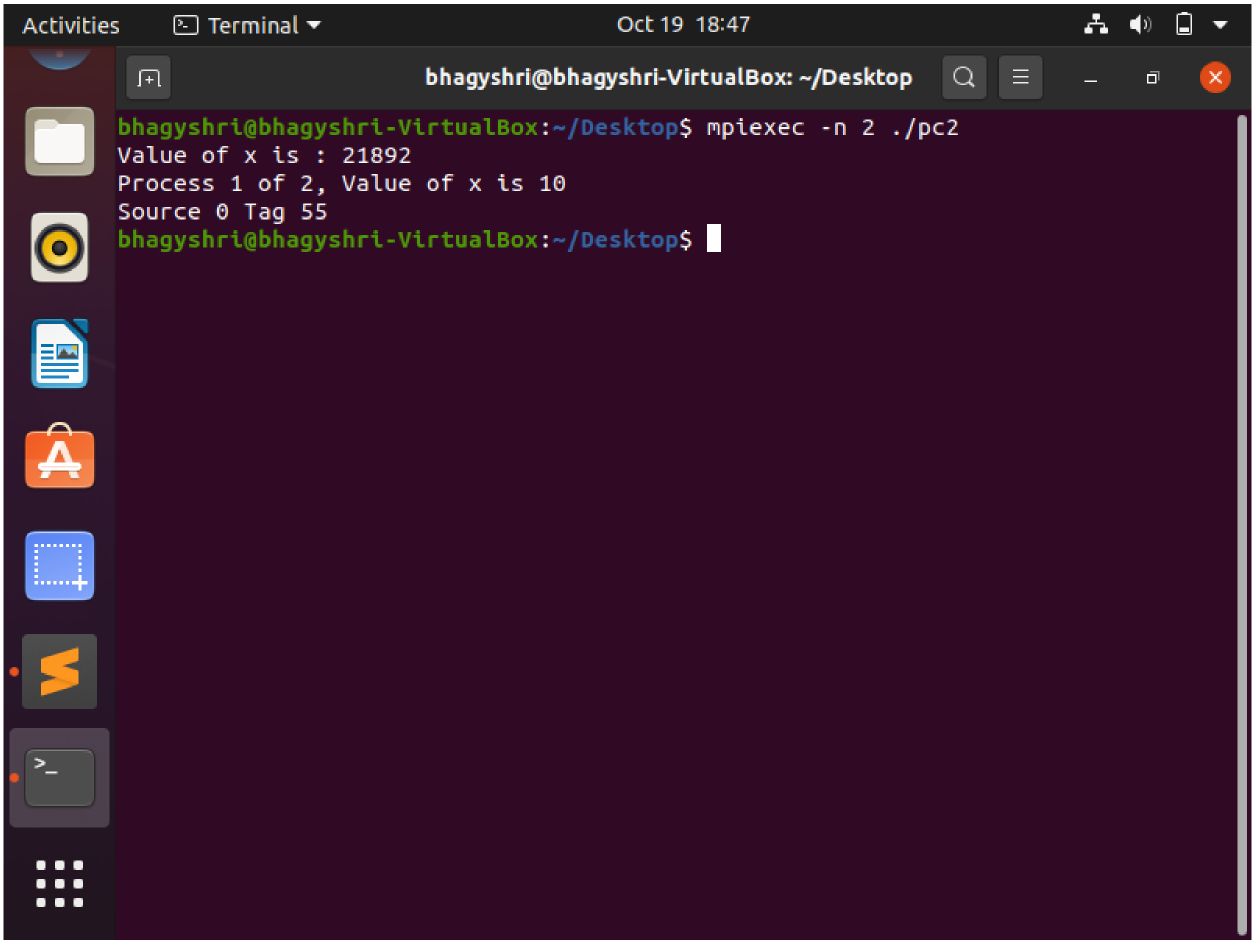
```
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpiexec -n 2 ./hello
Process 0 of 2, Hello World
Process 1 of 2, Hello World
bhagyshri@bhagyshri-VirtualBox:~/Desktop$
```

Observation-Number of times 'Hello World' is printed is equal to the number of processes initialized during the time of execution of the program in the terminal.

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 %d Tag %d \n",status.MPI_SOURCE,status.MPI_TAG);
}
MPI_Finalize();
return 0;
}
```

Output-



```
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpiexec -n 2 ./pc2
Value of x is : 21892
Process 1 of 2, Value of x is 10
Source 0 Tag 55
bhagyshri@bhagyshri-VirtualBox:~/Desktop$
```

Observation-The message, sent to process 1is received by process 0 as the tag is matching. So, there is a successful passing of the message.

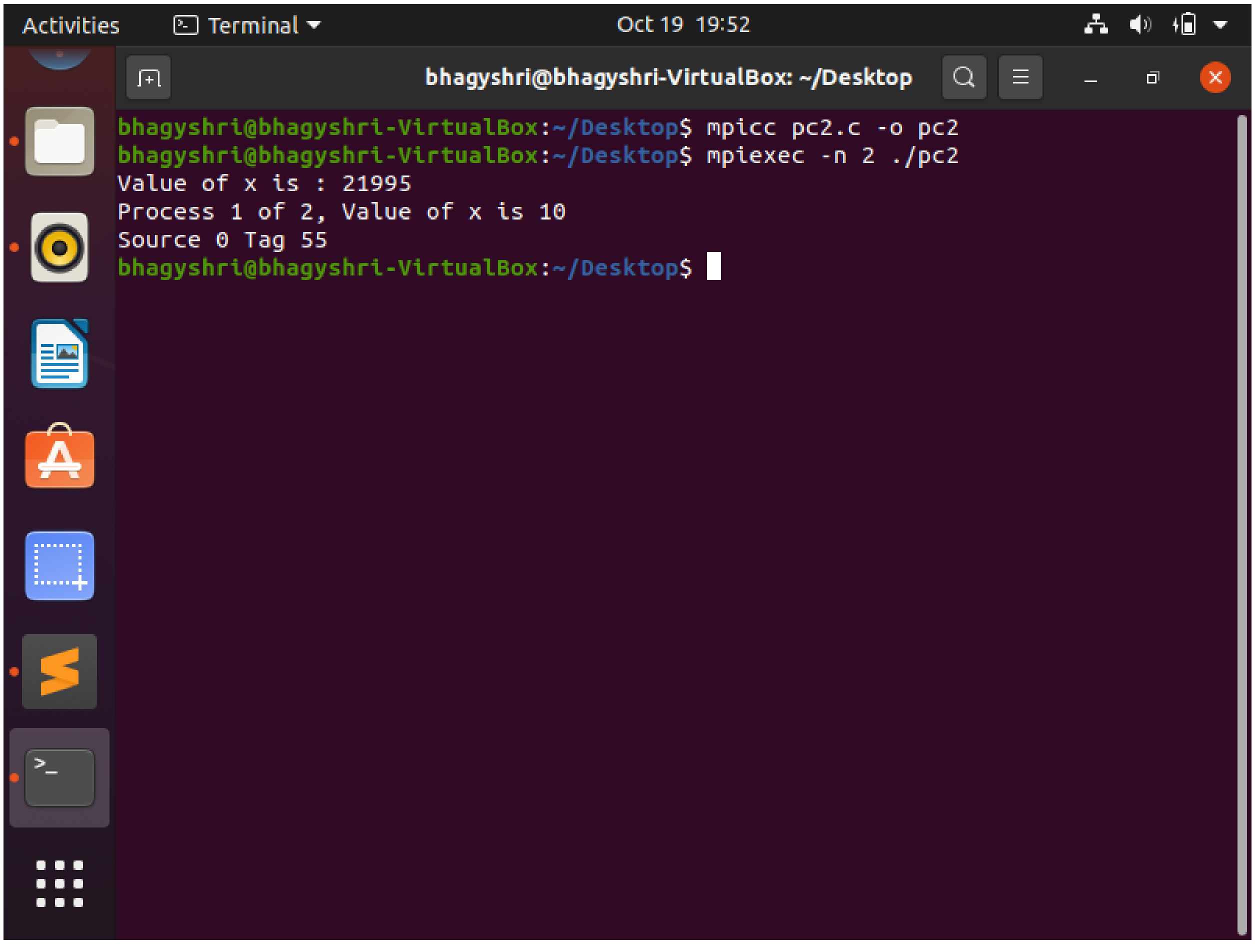
Modifications:

USE wild cards : MPI_ANY_SOURCE, MPI_ANY_TAG in the place of tag and source in

MPI_Recv(). To check the content in status structure.

Note: Add a screenshot of the modified code and output

```
#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 %d Tag %d \n",status.MPI_SOURCE,status.MPI_TAG);

}

MPI_Finalize();

return 0;

}
```



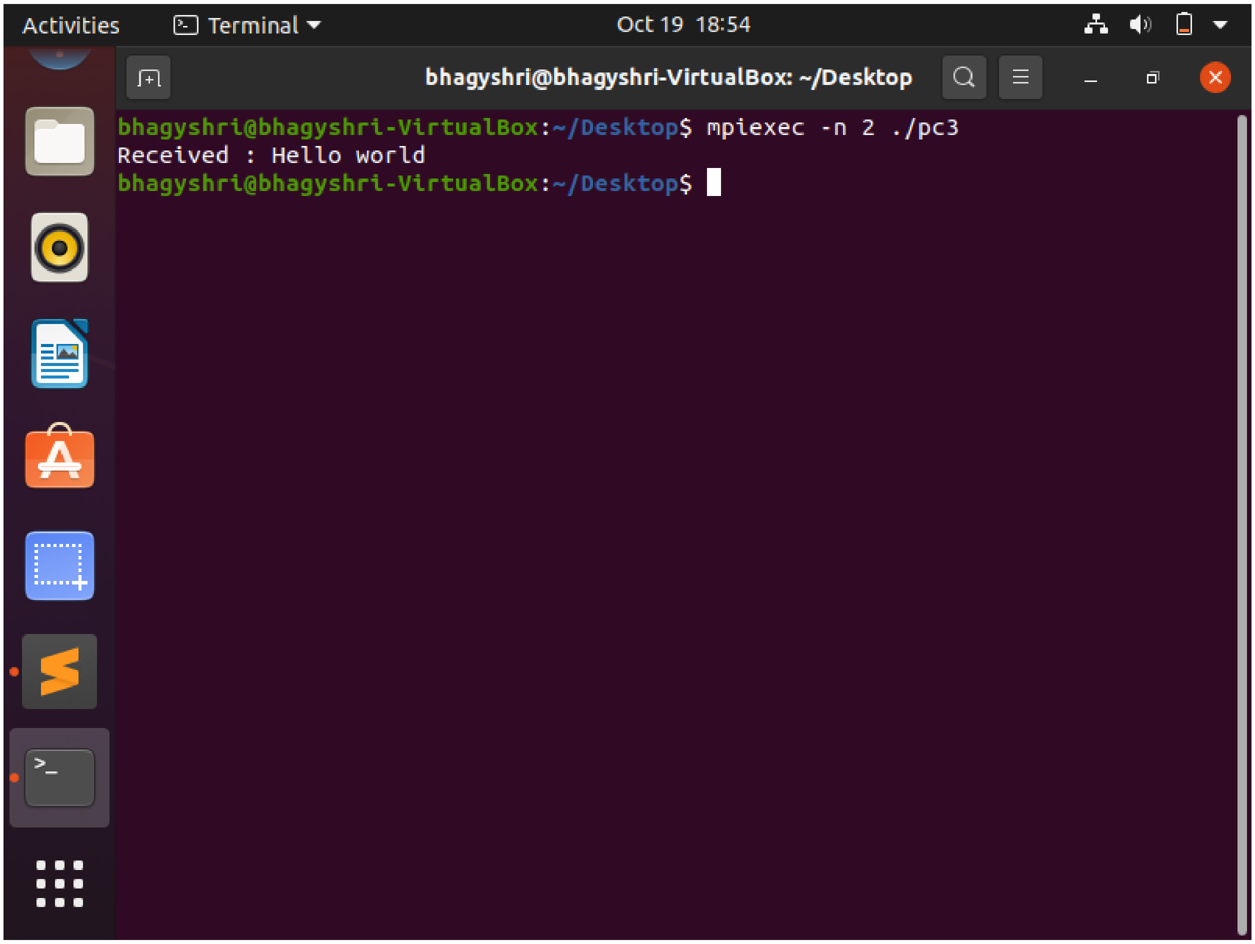
```
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpicc pc2.c -o pc2
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpiexec -n 2 ./pc2
Value of x is : 21995
Process 1 of 2, Value of x is 10
Source 0 Tag 55
bhagyshri@bhagyshri-VirtualBox:~/Desktop$
```

Observation-The message is sent to process 1 and the process 1 receives the message from any of the process and any tag (tag need not be same/matching). So, the data changed by the process 0 sent to the process 1 is successfully received and reflected in the output

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-

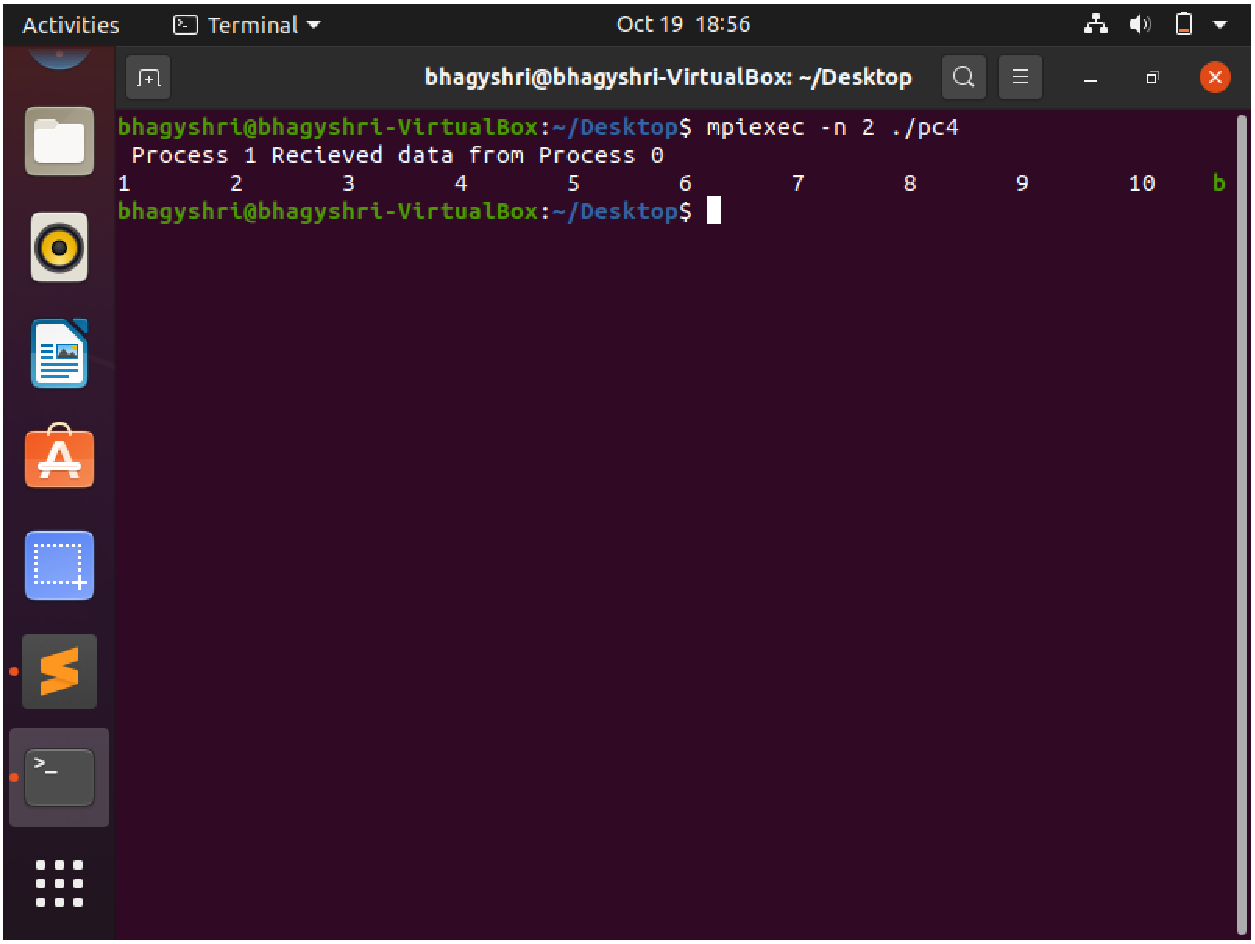


```
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpiexec -n 2 ./pc3
Received : Hello world
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ 
```

Observation-The message sent by the process 0 (to process 1) is successfully received by the process 1 (by process 0) as the tags are matching and the destination and source processes are also mentioned correctly

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]);
}
MPI_Finalize();
return 0;
}
```



```
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpiexec -n 2 ./pc4
 Process 1 Recieved data from Process 0
1       2       3       4       5       6       7       8       9       10      b
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ 
```

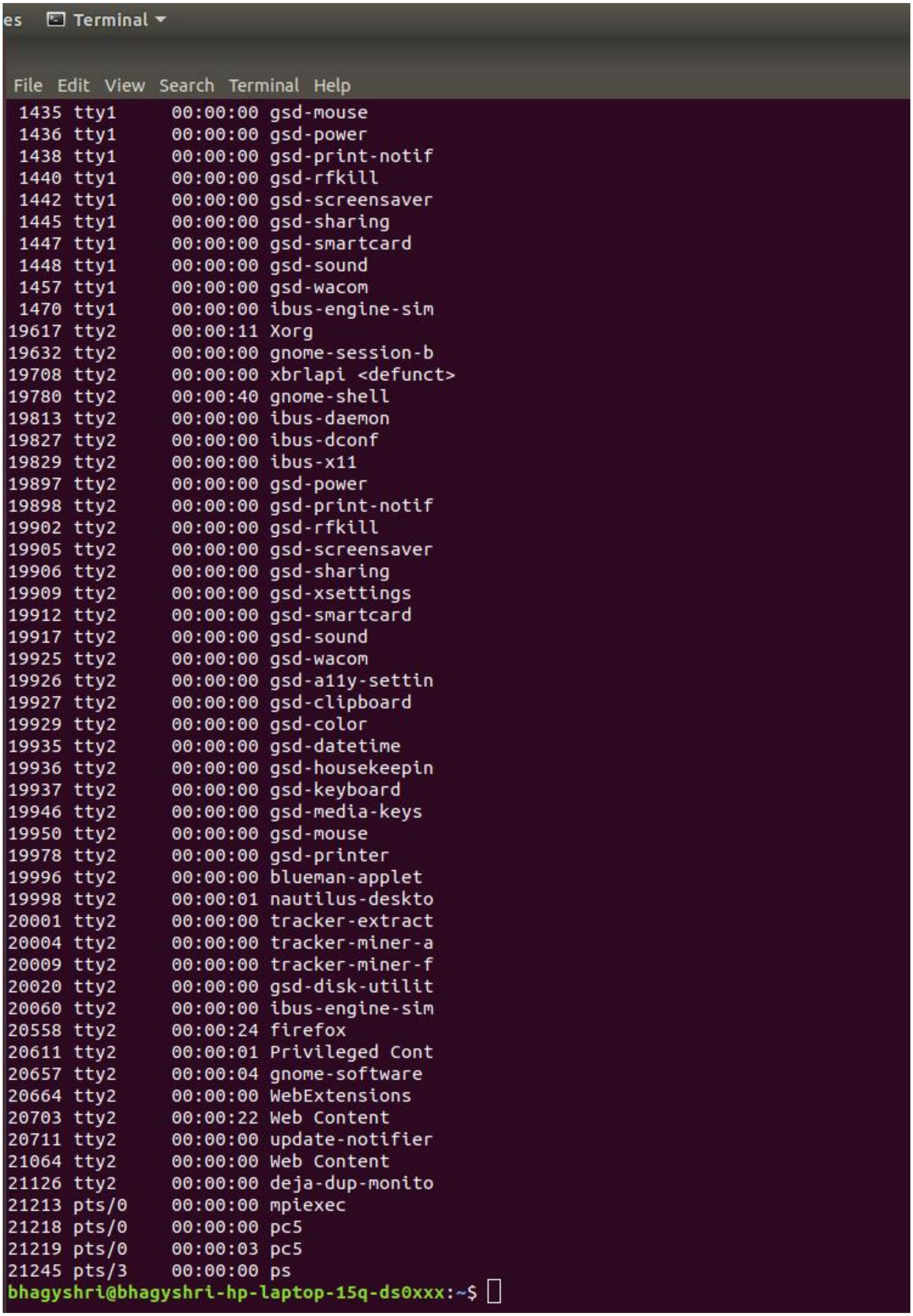
Observation-A count of first 10 messages is sent from process 0 to process 1 and tis is successful as all the attributes I'e', the destination, source are relative to each other and the tags are also matching

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

Here Send and Receive will be posted by Process 0 and Process 1 respectively. The execution will not complete as the Send and Receive does not have matching tags. Basically this is a Standard mode of Send and Receive. In the next program you will learn that Send will buffer the data and continue execution but receive will block if matching send is not posted.

```
#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;
}
```



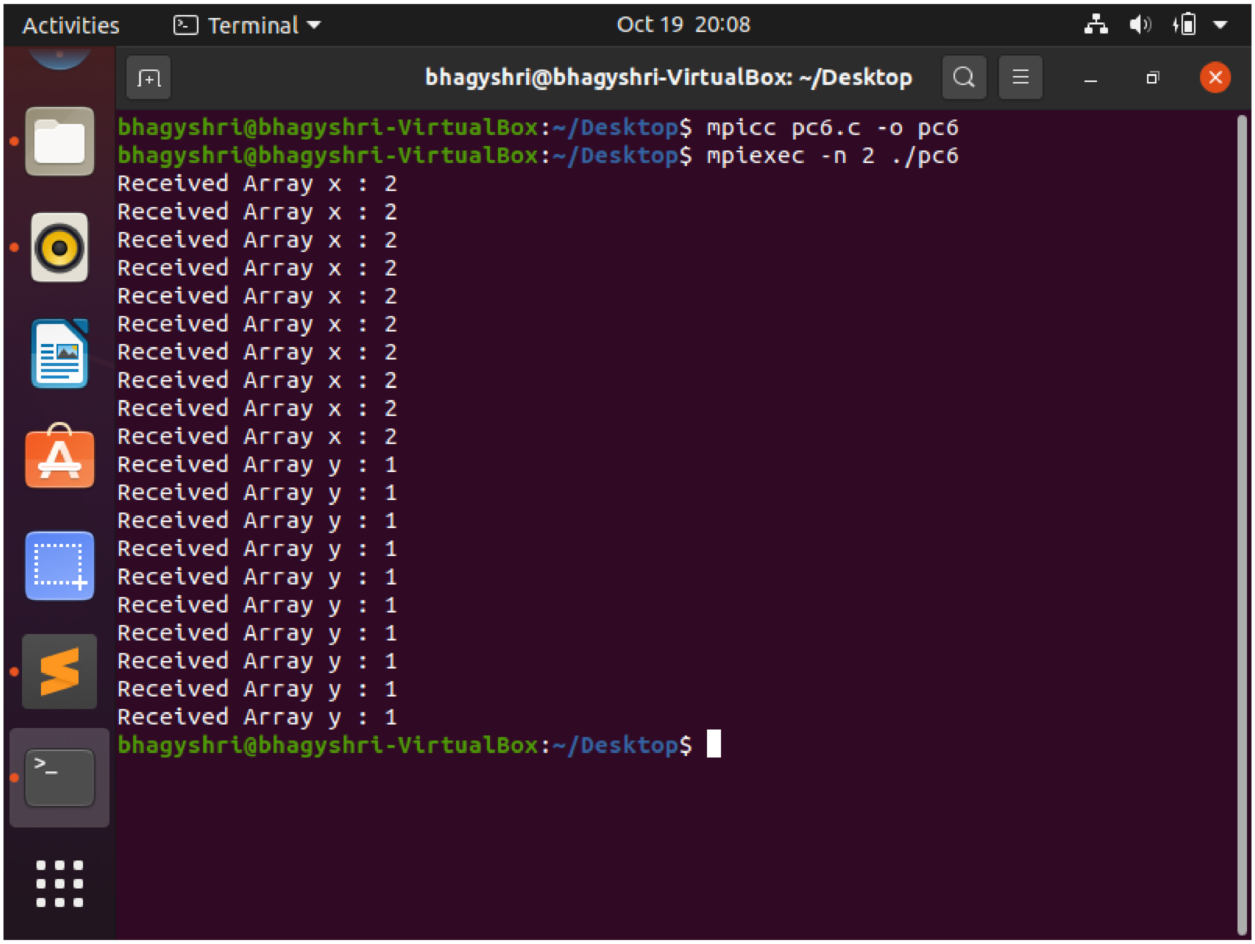
```
 1435 tty1     00:00:00 gsd-mouse
 1436 tty1     00:00:00 gsd-power
 1438 tty1     00:00:00 gsd-print-notif
 1440 tty1     00:00:00 gsd-rfkill
 1442 tty1     00:00:00 gsd-screensaver
 1445 tty1     00:00:00 gsd-sharing
 1447 tty1     00:00:00 gsd-smartcard
 1448 tty1     00:00:00 gsd-sound
 1457 tty1     00:00:00 gsd-wacom
 1470 tty1     00:00:00 ibus-engine-sim
19617 tty2     00:00:11 Xorg
19632 tty2     00:00:00 gnome-session-b
19708 tty2     00:00:00 xbrlapi <defunct>
19780 tty2     00:00:40 gnome-shell
19813 tty2     00:00:00 ibus-daemon
19827 tty2     00:00:00 ibus-dconf
19829 tty2     00:00:00 ibus-x11
19897 tty2     00:00:00 gsd-power
19898 tty2     00:00:00 gsd-print-notif
19902 tty2     00:00:00 gsd-rfkill
19905 tty2     00:00:00 gsd-screensaver
19906 tty2     00:00:00 gsd-sharing
19909 tty2     00:00:00 gsd-xsettings
19912 tty2     00:00:00 gsd-smartcard
19917 tty2     00:00:00 gsd-sound
19925 tty2     00:00:00 gsd-wacom
19926 tty2     00:00:00 gsd-a11y-settin
19927 tty2     00:00:00 gsd-clipboard
19929 tty2     00:00:00 gsd-color
19935 tty2     00:00:00 gsd-datetime
19936 tty2     00:00:00 gsd-housekeepin
19937 tty2     00:00:00 gsd-keyboard
19946 tty2     00:00:00 gsd-media-keys
19950 tty2     00:00:00 gsd-mouse
19978 tty2     00:00:00 gsd-printer
19996 tty2     00:00:00 blueman-applet
19998 tty2     00:00:01 nautilus-deskto
20001 tty2     00:00:00 tracker-extract
20004 tty2     00:00:00 tracker-miner-a
20009 tty2     00:00:00 tracker-miner-f
20020 tty2     00:00:00 gsd-disk-utilit
20060 tty2     00:00:00 ibus-engine-sim
20558 tty2     00:00:24 firefox
20611 tty2     00:00:01 Privileged Cont
20657 tty2     00:00:04 gnome-software
20664 tty2     00:00:00 WebExtensions
20703 tty2     00:00:22 Web Content
20711 tty2     00:00:00 update-notifier
21064 tty2     00:00:00 Web Content
21126 tty2     00:00:00 deja-dup-monito
21213 pts/0    00:00:00 mpiexec
21218 pts/0    00:00:00 pc5
21219 pts/0    00:00:03 pc5
21245 pts/3    00:00:00 ps
bhagyshri@bhagyshri-hp-laptop-15q-ds0xxx:~$ 
```

Observation-The message is sent by process 0 but not received by process 1 as the tags are mismatching

6. MPI_Send() and MPI_Recv() standard mode:

```
/* Demonstration of Blocking send and receive.*/

#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);

//Blocking send will expect matching receive at the destination

//In Standard mode, Send will return after copying the data to the system buffer. The

call will block if the buffer is not available or buffer space is not sufficient.
MPI_Send(y,10,MPI_INT,1,2,MPI_COMM_WORLD);
// This send will be initiated and matching receive is already there so the program will
not lead to deadlock
}
else if(myrank==1)
{
MPI_Recv(x,10,MPI_INT,0,2,MPI_COMM_WORLD,&status);
//P1 will block as it has not received a matching send with tag 2
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-



```
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpicc pc6.c -o pc6
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpiexec -n 2 ./pc6
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
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ 
```

a) Note down your observation on the content of x and y at Process 1.

Content of x: The value of x is not received (hence the values of x[i] are not changed to 2) as the tags of the send and receive of process 0 and process 1 respectively doesn't match.

Content of y: The value of y is not received (hence the values of y[i] are not changed to 2) as the tags of the send and receive of process 0 and process 1 respectively doesn't match

b) Explain the importance of tag.

Messages are sent with an accompanying user-defined integer tag, in order to assist the receiving process in identifying the message. Messages are screened at the receiving end by specifying a specific tag, or not screened by specifying MPI_ANY_TAG (which receives from any sender though the tag is not matching) as the tag in a receive**.**

c) Write your analysis about Blocking Send and Receive. Whether it is advantageous?

Function MPI_Send does not return until either the message has been copied into a system buffer or the message has been sent. In either case, we can overwrite the message buffer as soon as the function returns. Function MPI_Recv does not return until the message has been received into the buffer specified by the user and we may access the message values as soon as the function returns. Its not advantageous as ti may limit the performance of parallel program.

d) What is the need for Non blocking Send and Receive.

Posting a receive before the arrival of a message can save time, because the system can save a copy operation by transferring the contents of the incoming message directly into the destination buffer rather than a temporary system buffer. It is difficult to do this with MPl_Recv because, if the function is called too soon, the calling process blocks until the message arrives. If the function is called too late, the incoming message would have been copied into a system buffer and must be copied again. The message buffer may not be accessed by the user process until it explicitly completes the communication with a call to MPC_Wait.