

**IT 301 Parallel Computing LAB 9**

21<sup>st</sup> October 2020

**Faculty: Dr. Geetha V and Mrs. Thanmayee**

---

**Bhagyashri Nilesh Bhmare-181IT111**

**1. Non-Blocking Send and Receive.**

```
#include<mpi.h> #include<stdio.h> int main(intargc,char *argv[ ])
{
int size,myrank,x,i; 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)

{ x=10;

MPI_Isend(&x,1,MPI_INT,1,20,MPI_COMM_WORLD,&request); // Tag is different at receiver.

printf("Send returned immediately\n");

} else

if(myrank==1)

{

printf("Value of x is : %d\n",x);

MPI_Irecv(&x,1,MPI_INT,0,25,MPI_COMM_WORLD,&request);

printf("Receive returned immediately\n");

}

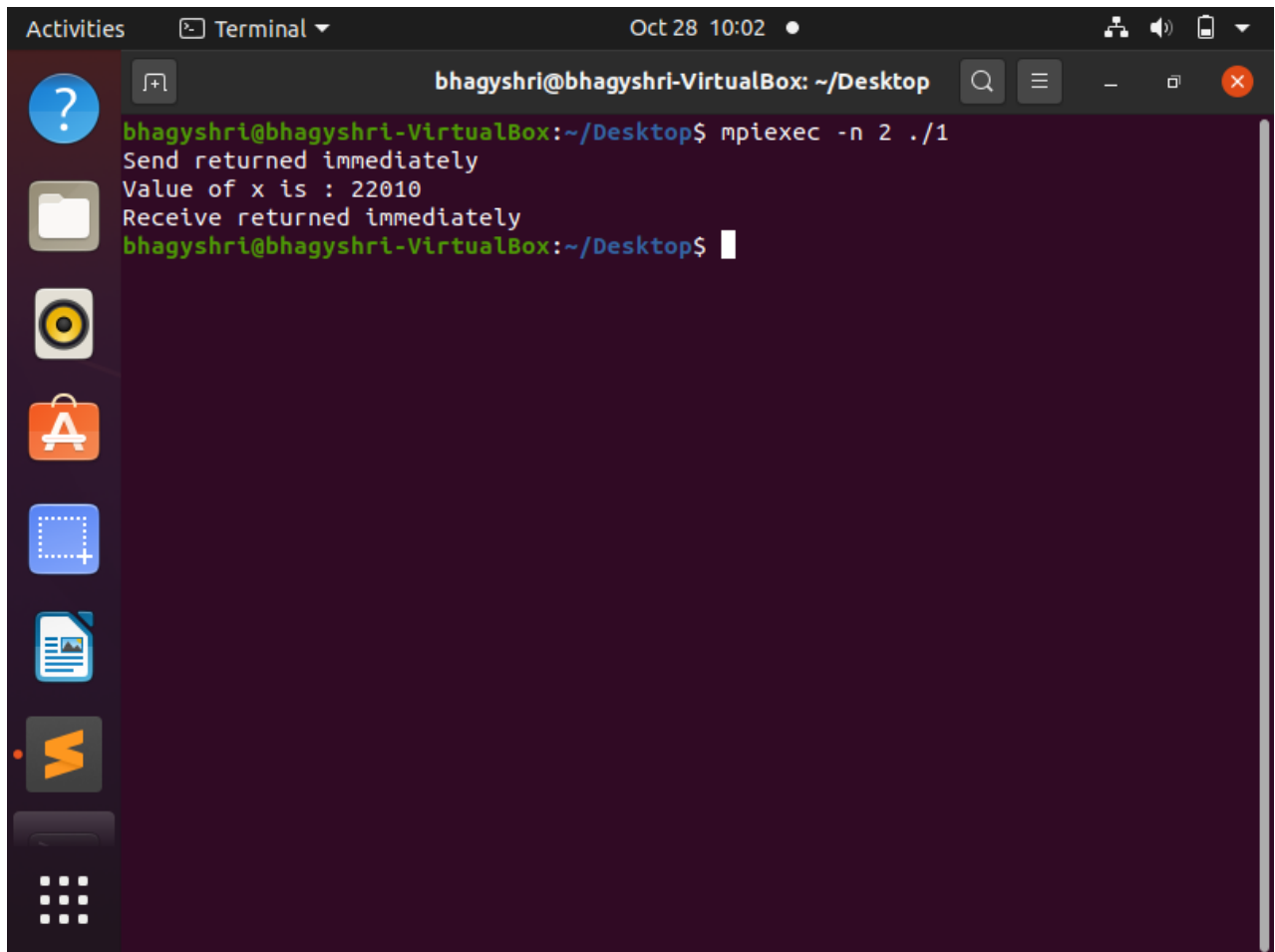
MPI_Finalize(); return

0;
```

```
}
```

### Observation:

a) Note the difference between standard mode send and non blocking send.

A screenshot of a terminal window titled "bhagyshri@bhagyshri-VirtualBox: ~/Desktop". The terminal shows the command `mpiexec -n 2 ./1` being executed. The output is: `Send returned immediately`, `Value of x is : 22010`, and `Receive returned immediately`. The prompt `bhagyshri@bhagyshri-VirtualBox:~/Desktop$` is visible at the end of the output. The terminal window has a sidebar with various application icons on the left and a top bar with system status icons on the right.

Standard mode send returns only after the value is copied out of send buffer but here at non blocking send, without waiting for that the next print statement is executed

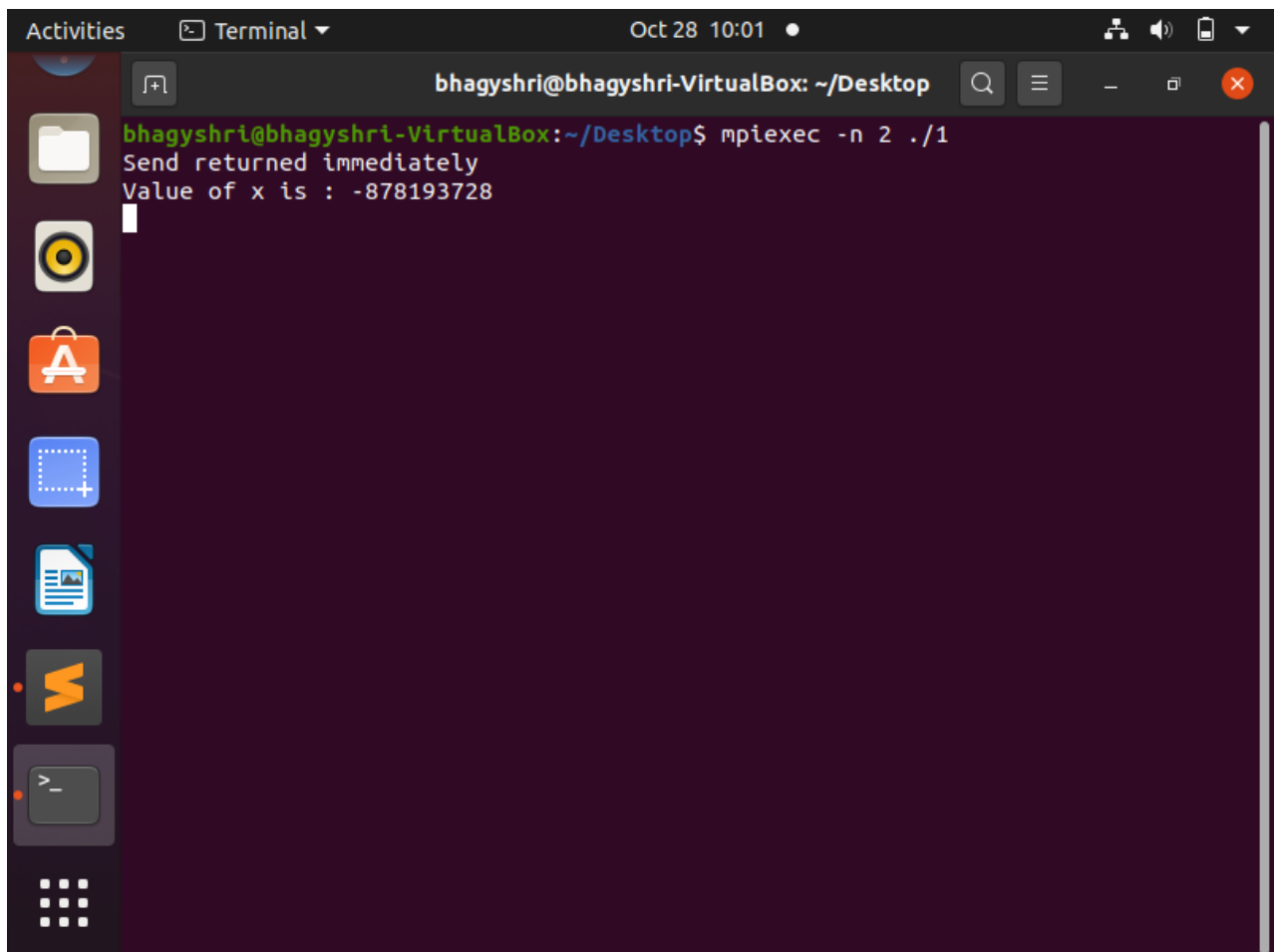
b) Note the observation by placing `MPI_Wait()` routine after `MPI_Isend()` and `MPI_Irecv()`.

```
#include<mpi.h>
#include<stdio.h>
int main(int argc,char *argv[ ])
{
int size,myrank,x,i;
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)
{
```

```

x=10;
MPI_Isend(&x,1,MPI_INT,1,20,MPI_COMM_WORLD,&request);
MPI_Wait(&request,&status);
printf("Send returned immediately \n");
}
else if(myrank==1)
{
printf("Value of x is : %d \n",x);
MPI_Irecv(&x,1,MPI_INT,0,25,MPI_COMM_WORLD,&request);
MPI_Wait(&request,&status);
printf("Receive returned immediately \n");}
MPI_Finalize();
return 0;
}

```



The screenshot shows a terminal window titled "bhagyshri@bhagyshri-VirtualBox: ~/Desktop". The user has executed the command `mpiexec -n 2 ./1`. The output of the program is displayed as follows:

```

bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpiexec -n 2 ./1
Send returned immediately
Value of x is : -878193728

```

Observation-Once `MPI_Wait()` is placed for `MPI_Isend()`, it will return only when the value is copied out of send buffer, for `MPI_Irecv()`, it will return only when some value is copied to receiving buffer. Therefore here after executing `MPI_Wait()` of `MPI_Send()`, the next print statement is also executed. But it will wait indefinitely at `MPI_Wait()` of `MPI_Irecv()` till a send with matching tag is made without executing the next print statement.

## 2. Demonstration of Broadcast operation : MPI\_Bcast().

```
#include<mpi.h>

#include<stdio.h> int main(int

argc,char *argv[ ])

{ int

size,myrank,x;

MPI_Init(&argc,&argv);

MPI_Comm_size(MPI_COMM_WORLD,&size); MPI_Comm_rank(MPI_COMM_WORLD,&myrank);

if(myrank==0)

{ scanf("%d",&x);

}

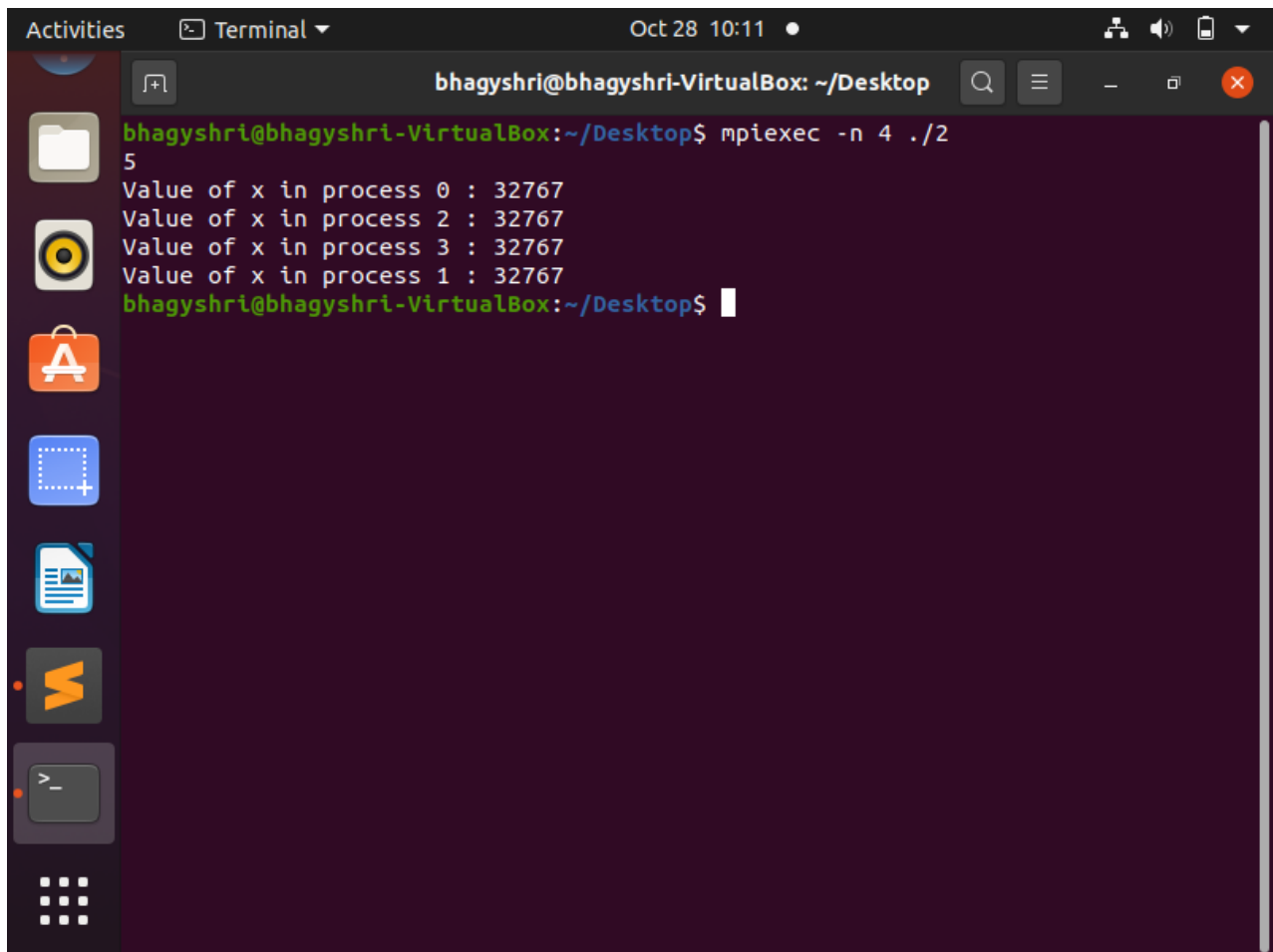
MPI_Bcast(&x,1,MPI_INT,1,MPI_COMM_WORLD);

printf("Value of x in process %d : %d\n",myrank,x);

MPI_Finalize(); return 0;

}
```

Output-

A terminal window titled 'bhagyshri@bhagyshri-VirtualBox: ~/Desktop' with a search icon, menu icon, and window control buttons. The terminal shows the command 'mpirun -n 4 ./2' being executed. The output is: '5', 'Value of x in process 0 : 32767', 'Value of x in process 2 : 32767', 'Value of x in process 3 : 32767', 'Value of x in process 1 : 32767'. The prompt 'bhagyshri@bhagyshri-VirtualBox:~/Desktop\$' is visible at the end of the output.

```
bhagyshri@bhagyshri-VirtualBox: ~/Desktop
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpirun -n 4 ./2
5
Value of x in process 0 : 32767
Value of x in process 2 : 32767
Value of x in process 3 : 32767
Value of x in process 1 : 32767
bhagyshri@bhagyshri-VirtualBox:~/Desktop$
```

Here process with id 0, reads in value of x as 5 and broadcasts it into every process in MPI\_COMM\_WORLD.

### 3. Demonstration of MPI\_Reduce with Sum Operation

You may use MPI\_PROD to get product of elements in each process.

You may also try using array of elements instead of single element x.

Try to understand the working of Reduce.

```
#include<mpi.h>

#include<stdio.h> int main(int
argc,char *argv[ ])
{
int size,myrank,i,x,y;
MPI_Init(&argc,&argv);
```

```
MPI_Comm_size(MPI_COMM_WORLD,&size); MPI_Comm_rank(MPI_COMM_WORLD,&myrank);

x=myrank; // Note the value of x in each process.

MPI_Reduce(&x,&y,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);

if(myrank==0) {

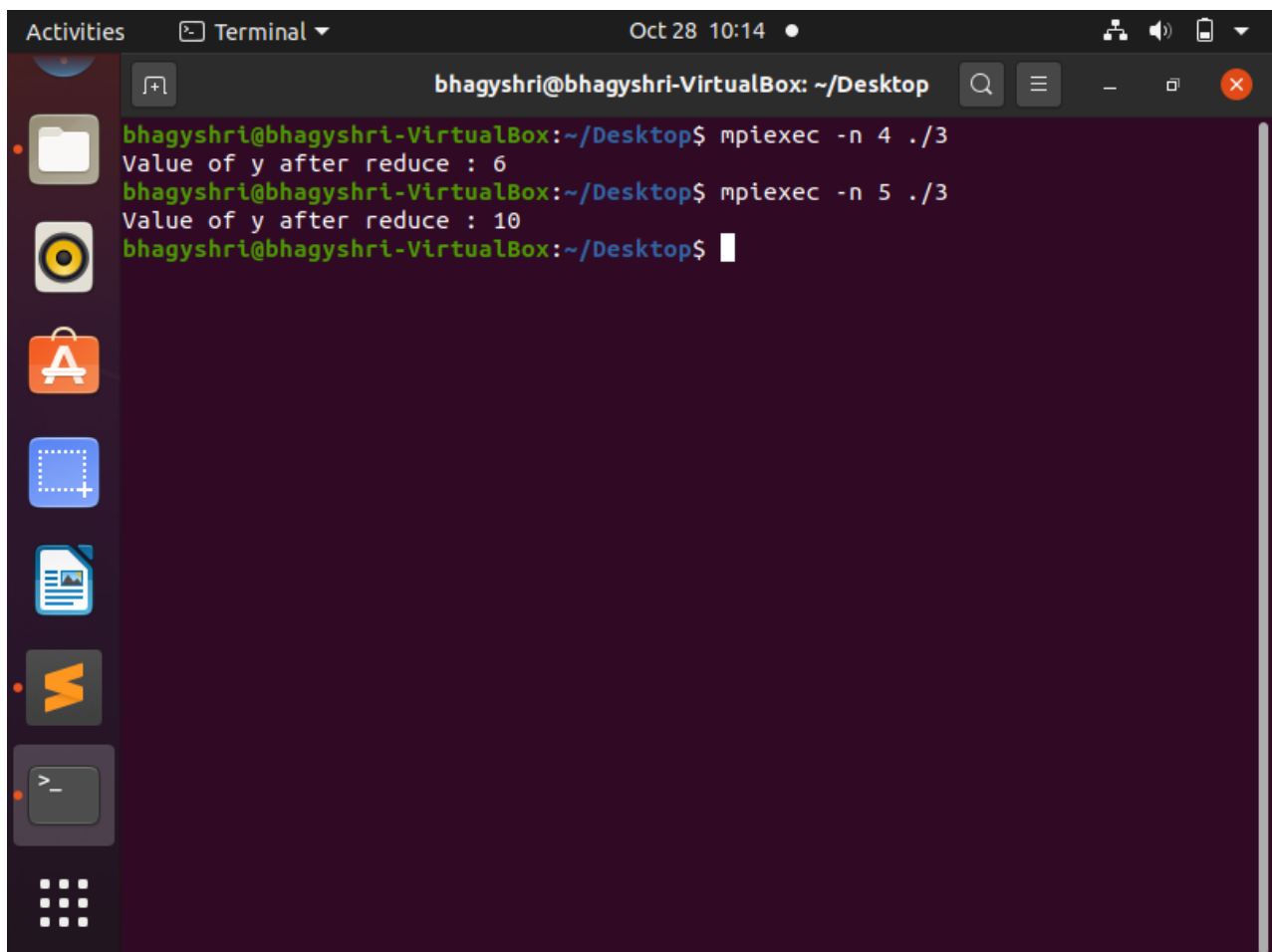
printf("Value of y after reduce : %d\n",y);

} MPI_Finalize();

return 0;

}
```

Output-



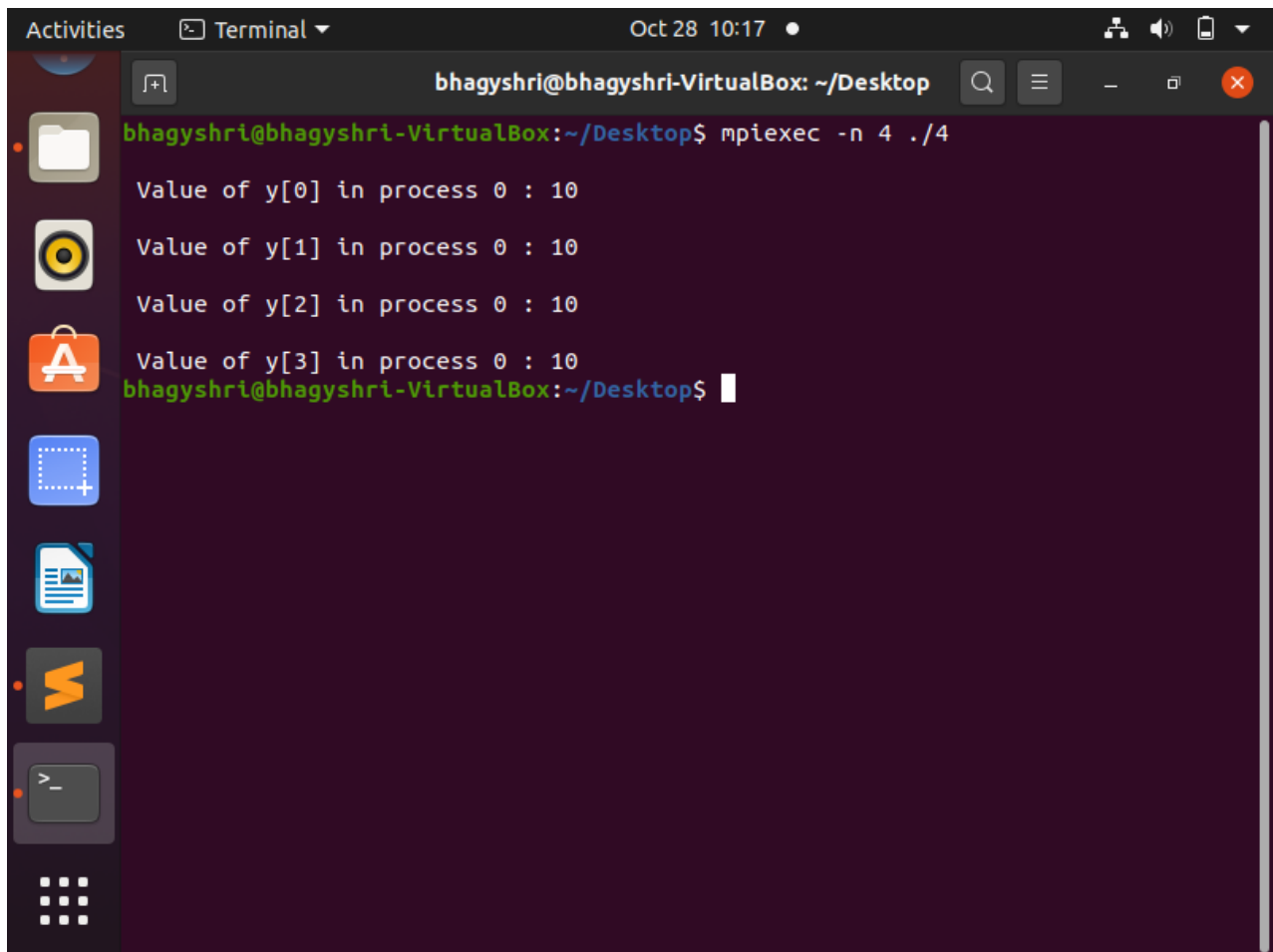
```
Oct 28 10:14
bhagyshri@bhagyshri-VirtualBox: ~/Desktop
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpiexec -n 4 ./3
Value of y after reduce : 6
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpiexec -n 5 ./3
Value of y after reduce : 10
bhagyshri@bhagyshri-VirtualBox:~/Desktop$
```

Here in every process x value is the respective process id. All this values are added using reduce and stored in y buffer of 0 th process For total 4 no of process,  $y = 0+1+2+3=6$  For total 5 no of process,  $y = 0+1+2+3+4=10$

#### 4. Demonstration of MPI\_Gather():

```
#include<mpi.h>
#include<stdio.h> int main(int
argc,char *argv[ ])
{
int size,myrank,x=10,y[5],i;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
MPI_Gather(&x,1,MPI_INT,y,1,MPI_INT,0,MPI_COMM_WORLD); // Value of x at each process is
copied to array y in Process 0 if(myrank==0)
{
for(i=0;i<size;i++) printf("\nValue of y[%d] in process %d :
%d\n",i,myrank,y[i]);
}
MPI_Finalize();
return 0;

}
```

A terminal window titled 'bhagyshri@bhagyshri-VirtualBox: ~/Desktop' showing the execution of an MPI program. The command 'mpirun -n 4 ./4' has been run, and the output shows four lines of values for y[0], y[1], y[2], and y[3], all of which are 10, indicating that process 0 has gathered data from all other processes. The prompt 'bhagyshri@bhagyshri-VirtualBox: ~/Desktop\$' is visible at the bottom.

```
bhagyshri@bhagyshri-VirtualBox: ~/Desktop$ mpirun -n 4 ./4
Value of y[0] in process 0 : 10
Value of y[1] in process 0 : 10
Value of y[2] in process 0 : 10
Value of y[3] in process 0 : 10
bhagyshri@bhagyshri-VirtualBox: ~/Desktop$
```

Observation -Process 0 gathers the value stored in the buffer from all other processes and stores in y

## 5. Demonstration of MPI\_Scatter()

- Note that the program is hard coded to work with 4 processes receiving two chunks from the array.
- You may change according to what you want to explore.

```
#include<mpi.h>
#include<stdio.h> int main(int
argc,char *argv[ ])
{
int size,myrank,x[8],y[2],i;
MPI_Init(&argc,&argv);
```



```

MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank); if(myrank==0)
{
printf("Enter 8 values into array x:\n");
for(i=0;i<8;i++) scanf("%d",&x[i]);
}
MPI_Scatter(x,2,MPI_INT,y,2,MPI_INT,0,MPI_COMM_WORLD);
for(i=0;i<2;i++)
printf("\nValue of y in process %d : %d\n",myrank,y[i]);
MPI_Finalize(); return 0; }

```

```

bhagyshri@bhagyshri-VirtualBox: ~/Desktop
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpirun -n 4 ./5
Enter 8 values into array x:
1
2
3
4
5
6
7
8
Value of y in process 0 : 1
Value of y in process 0 : 2
Value of y in process 2 : 5
Value of y in process 2 : 6
Value of y in process 3 : 7
Value of y in process 3 : 8
Value of y in process 1 : 3
Value of y in process 1 : 4
bhagyshri@bhagyshri-VirtualBox:~/Desktop$

```

Process 0 reads 8 values and is scattered among the 4 processes. Each process prints the 2 values it received

## 6. Write an MPI program to find the smallest element in a given array of size N.

- Try to find out how many processes you may need for parallel computation based on N.

- Use MPI\_Reduce routine. Identify which routine you would use to find the minimum number in a given array.

```
#include<mpi.h>
#include<stdio.h>

int main(int argc,char *argv[ ])
{
    int size,myrank;
    int n=20;
    int x[20],y[20];
    int Min=n+1;
    int GlobalMin;
    int indSize;
    double s;
    int sendcount[20],displacement[20];
    double start,end,time,avgtime;
    MPI_Init(&argc,&argv);
    start = MPI_Wtime();
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
    indSize=n/size;
    s=size;
    if(myrank==0)
    {
        printf("\n The array elements are:");
        for(int i=0;i<n;i++)
        {
            x[i] =1+ rand()%n;
            printf("%d ",x[i]);
        }
    }
    displacement[0]=0;
    for(int i=0;i<size-1;i++)
    {
```

```

sendcount[i]=indSize;
displacement[i+1]+=indSize;
}
sendcount[size-1]=n-(size-1)*indSize;
MPI_Scatterv(&x,sendcount,displacement,MPI_INT,&y,5,MPI_INT,0,MPI_COMM_WORLD
);
for(int i=0;i<sendcount[myrank];i++)
{
if(y[i]<Min)
Min=y[i];
}
printf("\nSmallest element in process %d is %d\n",myrank,Min);
MPI_Reduce(&Min,&GlobalMin,1,MPI_INT,MPI_MIN,0,MPI_COMM_WORLD);
if(myrank ==0)
{
printf("\nSmallest element is %d\n",GlobalMin );
}
MPI_Barrier(MPI_COMM_WORLD);
end=MPI_Wtime();
time=end-start;
MPI_Reduce(&time,&avgtime,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
avgtime= avgtime/s;
if(myrank==0)
printf("\n Average taken with %d processes is %f\n",size,avgtime);
MPI_Finalize();
return 0;
}

```

```
Activities Terminal Oct 28 11:55
bhagyshri@bhagyshri-VirtualBox: ~/Desktop
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpiexec -n 4 ./6
The array elements are:4 7 18 16 14 16 7 13 10 2 3 8 11 20 4 7 1 7 13 17
Smallest element in process 0 is 4
Smallest element in process 1 is 2
Smallest element in process 2 is 3
Smallest element in process 3 is 1
Smallest element is 1
Average taken with 4 processes is 0.000436
bhagyshri@bhagyshri-VirtualBox:~/Desktop$
```

Process 0 assigns random values to the array elements, and scatter the array to different processes. Each process finds the local minimum and the minimum is found using reduce

MPI\_Scatter() and MPI\_Reduce() routines are used here

No of processes	Time taken
1	0.000038
2	0.000070
3	0.000196
4	0.000236
5	0.035423
6	0.042532