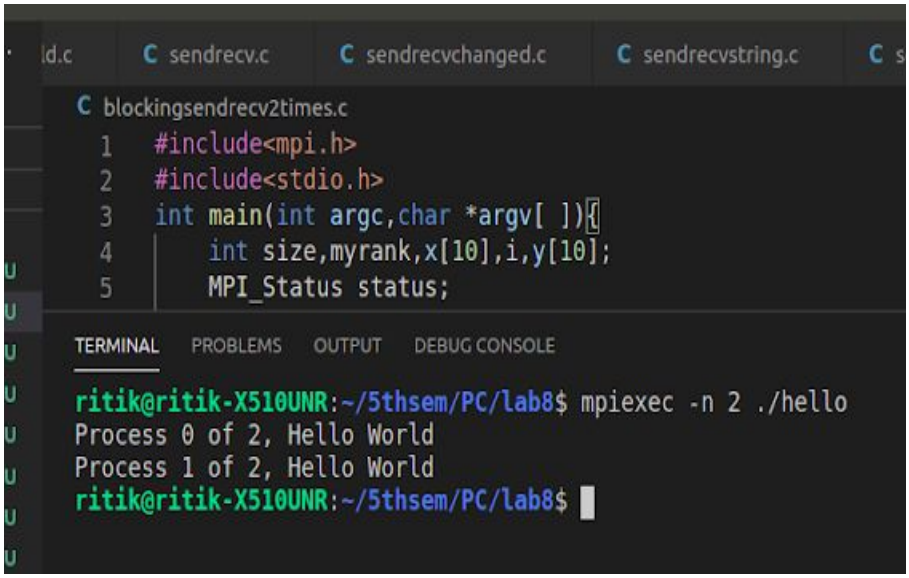
Ritik Pansuriya - 181IT237 - lab 8

1. MPI “Hello World” program:

```c
#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;
}
```

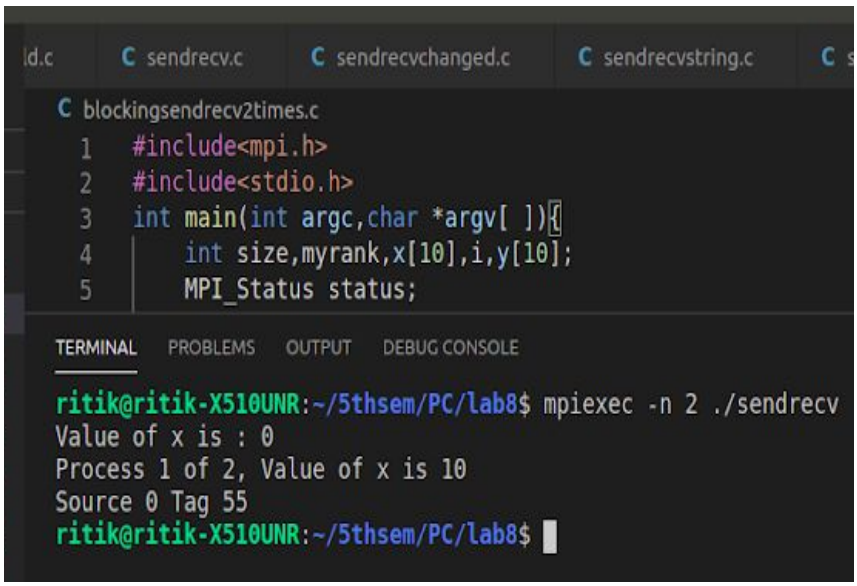
o/p :



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

```c
#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;
}
```

o/p:

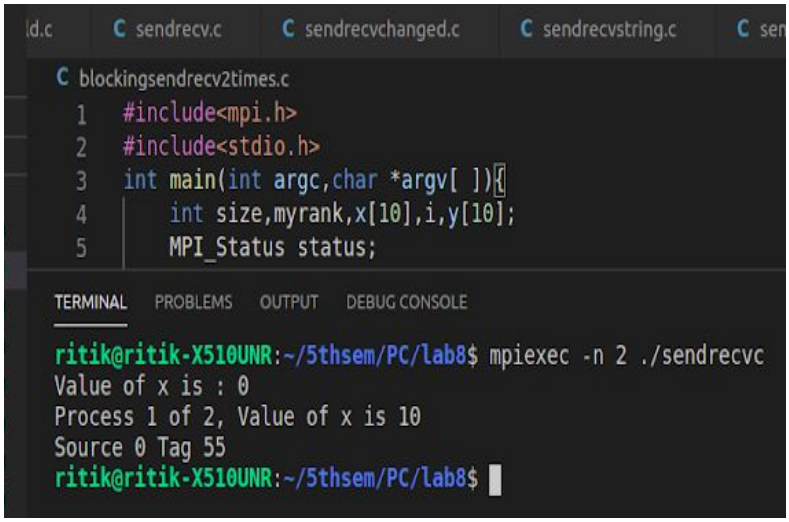


Changed version :

```c
#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;
}
```

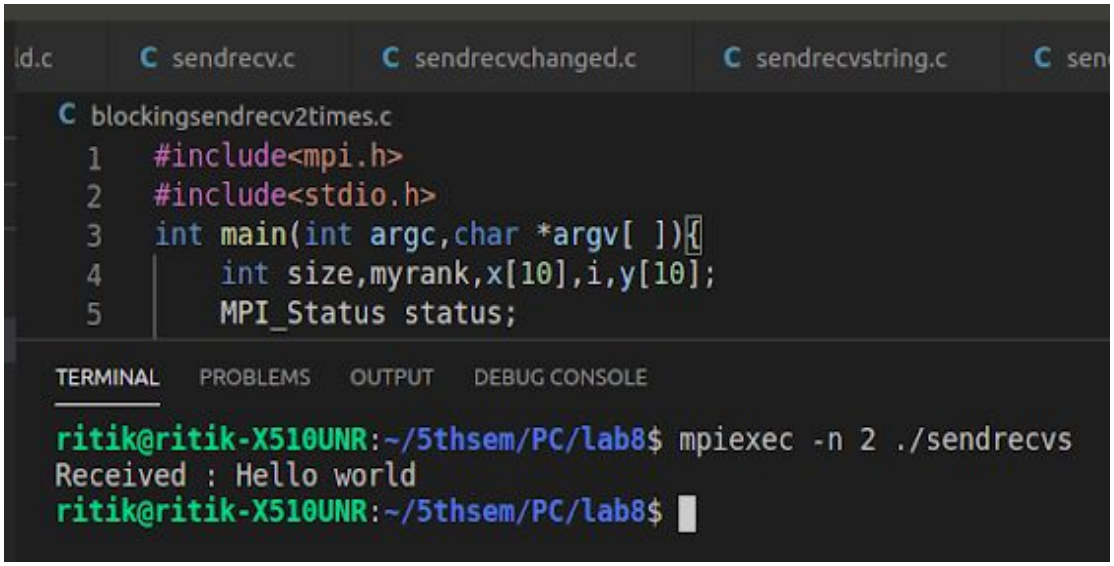
o/p:



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

```c
#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;
}
```

o/p:



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

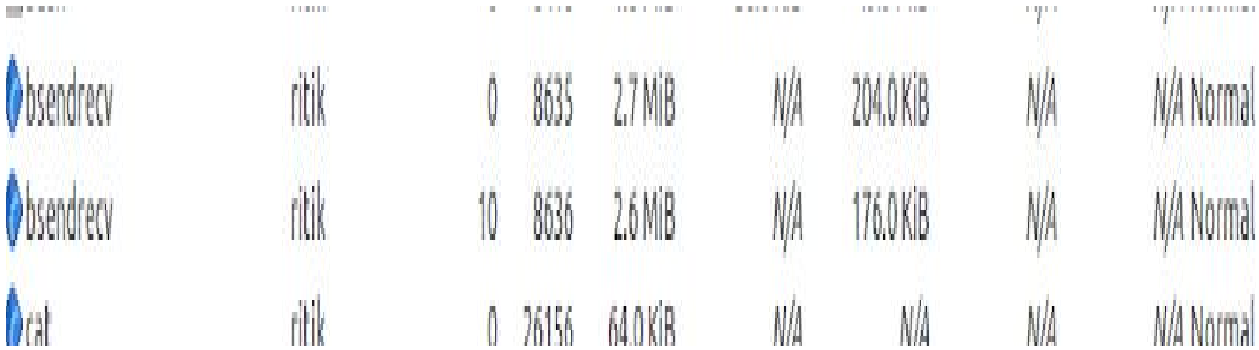
```c
#include<mpi.h>
#include<stdio.h>
int main(int argc,char *argv[ ]){
   int size,myrank,x[50],y[50],i;
   MPI_Status status;
   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&size);
   MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
   if(myrank==0){
       for(i=0;i<50;i++)
       x[i]=i+1;
       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;
}
```

o/p:

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;
}
```

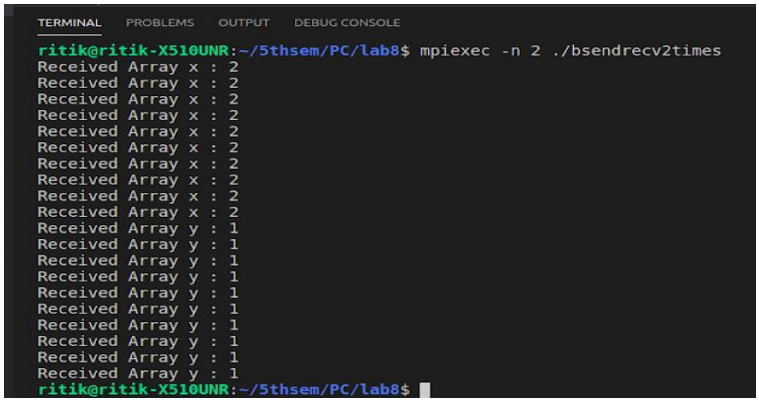
o/p:



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

```c
#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;
}
```

o/p:



A) here array x and y is sent by CPU0 to CPU1 which was received in reverse order because a matching tag was found for the second MPI_send and as it is standard send so it is non blocking and thus code executed without any deadlock. Finally content of array y was printed first and then was of x.

b) Tag matching is very important as we need to make sure that CPU must accept the correct information which will be made sure by tag so if tag matches that means that is the data which It was looking for.
c) Yeah it didn't go to deadlock and that is the biggest advantage for blocking send and receive so if the send statement of some data is before some needed data then also the system will not face any error.

d) The advantage of using the non-blocking send occurs when the system buffer is full. In this case, a blocking send would have to wait until the receiving task pulled some message data out of the buffer. If a non-blocking call is used, computation can be done during this interval.