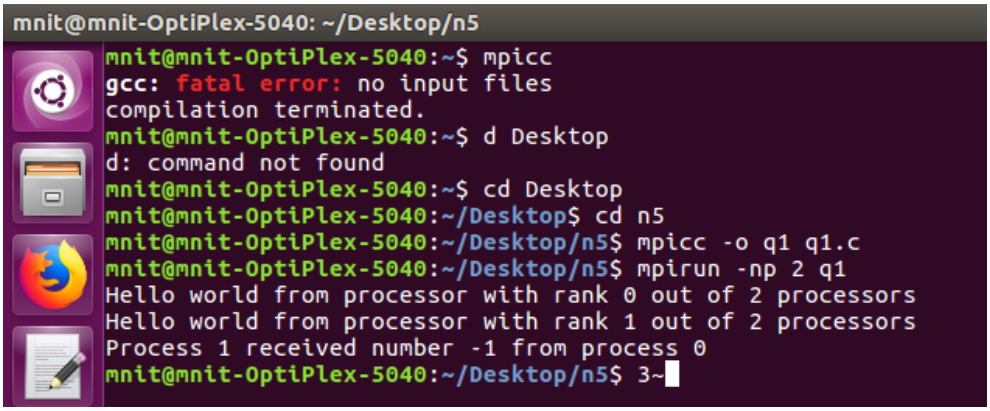
MPI FUNCTIONS

1. **MPI_Init, MPI_Finalize,MPI_Comm_rank, MPI_Comm_size , MPI_Send, MPI_Recv**

Code:

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

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

	MPI_Init(NULL, NULL);
	//float starttime=MPI_Wtime();
	int world_size;//for the size of communicator
	MPI_Comm_size(MPI_COMM_WORLD, &world_size); //MPI_COMM_WORLD: encloses all processes in the job

	int world_rank;// for rank of the process
	MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

	printf("Hello world from processor with rank %d out of %d processors\n",world_rank, world_size);

	if (world_size < 2) {
		printf("World size must be greater than 1 for %s\n", argv[0]);
		MPI_Abort(MPI_COMM_WORLD, 1);
	}
	int number;
	if (world_rank == 0) {
		number = -1;
		MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
	}

	else if (world_rank == 1) {
		MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,MPI_STATUS_IGNORE);
		printf("Process 1 received number %d from process 0\n",number);
	}

	MPI_Finalize();
}
```



2. **MPI_Bcast, MPI_Wtime**

Code:

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv) {
      MPI_Init(NULL, NULL);
      double timetaken;
      int root=0;
      int data=0;
      int world_size;//for the size of communicator
      MPI_Comm_size(MPI_COMM_WORLD, &world_size); //MPI_COMM_WORLD: encloses all
processes in the job

      int world_rank;// for rank of the process
      MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    if(world_rank == root) {
          data = 1;
     }

      printf("process %d: before broadcast data:%d\n",world_rank,data);
      MPI_Bcast(&data,1, MPI_INT, 0, MPI_COMM_WORLD);
      //printf("broadcasted\n");

      printf("process %d: after broadcast data:%d\n",world_rank,data);
      timetaken+=MPI_Wtime();

      printf("time taken:%f\n",timetaken);
      MPI_Finalize();
}
```

```
mnit@mnit-OptiPlex-5040:~/Desktop/n5$ mpirun -np 4 q2
process 0: before broadcast data:1
process 0: after broadcast data:1
process 2: before broadcast data:0
process 2: after broadcast data:1
process 1: before broadcast data:0
process 1: after broadcast data:1
process 3: before broadcast data:0
process 3: after broadcast data:1
```

3. **MPI_Reduce**

Code:

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

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

	MPI_Init(NULL, NULL);
	  int data=0;
	  int reduceddata=0;
	int world_size;//for the size of communicator
	MPI_Comm_size(MPI_COMM_WORLD, &world_size); //MPI_COMM_WORLD: encloses all processes in the job
	int world_rank;// for rank of the process
	MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

	  data=(world_rank+1)*10;
	printf("process:%d with data:%d out of %d processes\n" ,world_rank,data, world_size);

	  MPI_Reduce(&data,&reduceddata,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);

	if (world_rank==0){
		printf("the reduced data is %d\n",reduceddata);
	}
	MPI_Finalize();

}
```

```
mnit@mnit-OptiPlex-5040:~/Desktop/n5$ mpirun -np 4 q4
process:2 with data:30 out of 4 processes
process:0 with data:10 out of 4 processes
process:1 with data:20 out of 4 processes
process:3 with data:40 out of 4 processes
the reduced data is 100
```