5CS022 Distribute and Cloud Systems Programming Week 1 Workshop

Overview

The aim of this workshop is to familiarise you with building, compiling and running MPI programs. You can carry out this workshop on your own Linux system

Tasks

1. Download the sample MPI programs from the drive into your Linux system. Compile and run the program mpi01.c. To compile it, run the following command in the terminal:

   mpicc mpi01.c -o mpi01

   Now run it with the following:

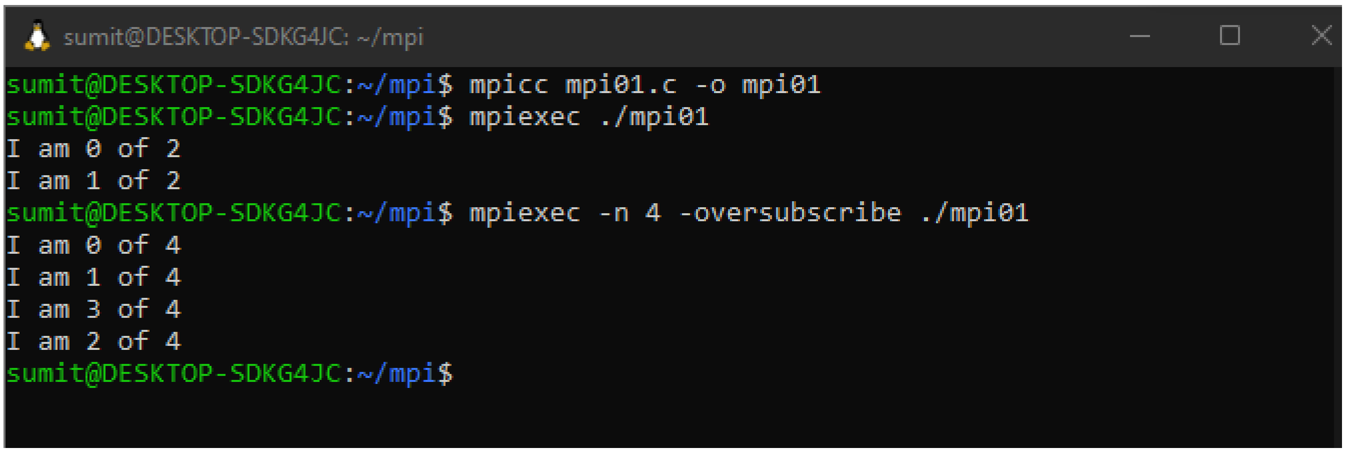
   mpiexec ./mpi01

   This will (probably) only run only one process, which is not very interesting. Run it again with the following command::

   mpiexec -n 4 -oversubscribe ./mpi01

   Note the output this time. It should indicate that 4 processes have run and they all have different process IDs.

   Experiment with higher and higher numbers of processes until it stops running. Then have a look at the error message and try to work out why it stopped working.

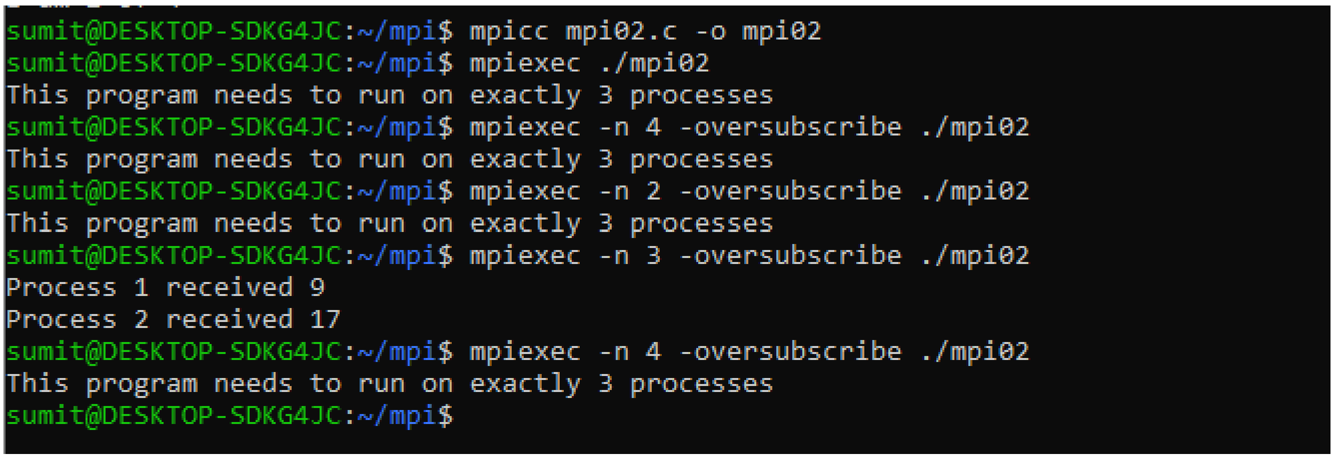


```
sumit@DESKTOP-SDKG4JC:~/mpi$ mpicc mpi01.c -o mpi01
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec ./mpi01
I am 0 of 2
I am 1 of 2
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 4 -oversubscribe ./mpi01
I am 0 of 4
I am 1 of 4
I am 3 of 4
I am 2 of 4
sumit@DESKTOP-SDKG4JC:~/mpi$
```

2. Compile and run the program mpi02.c. Try running it with 2, 3 and 4 processes. Eg.:

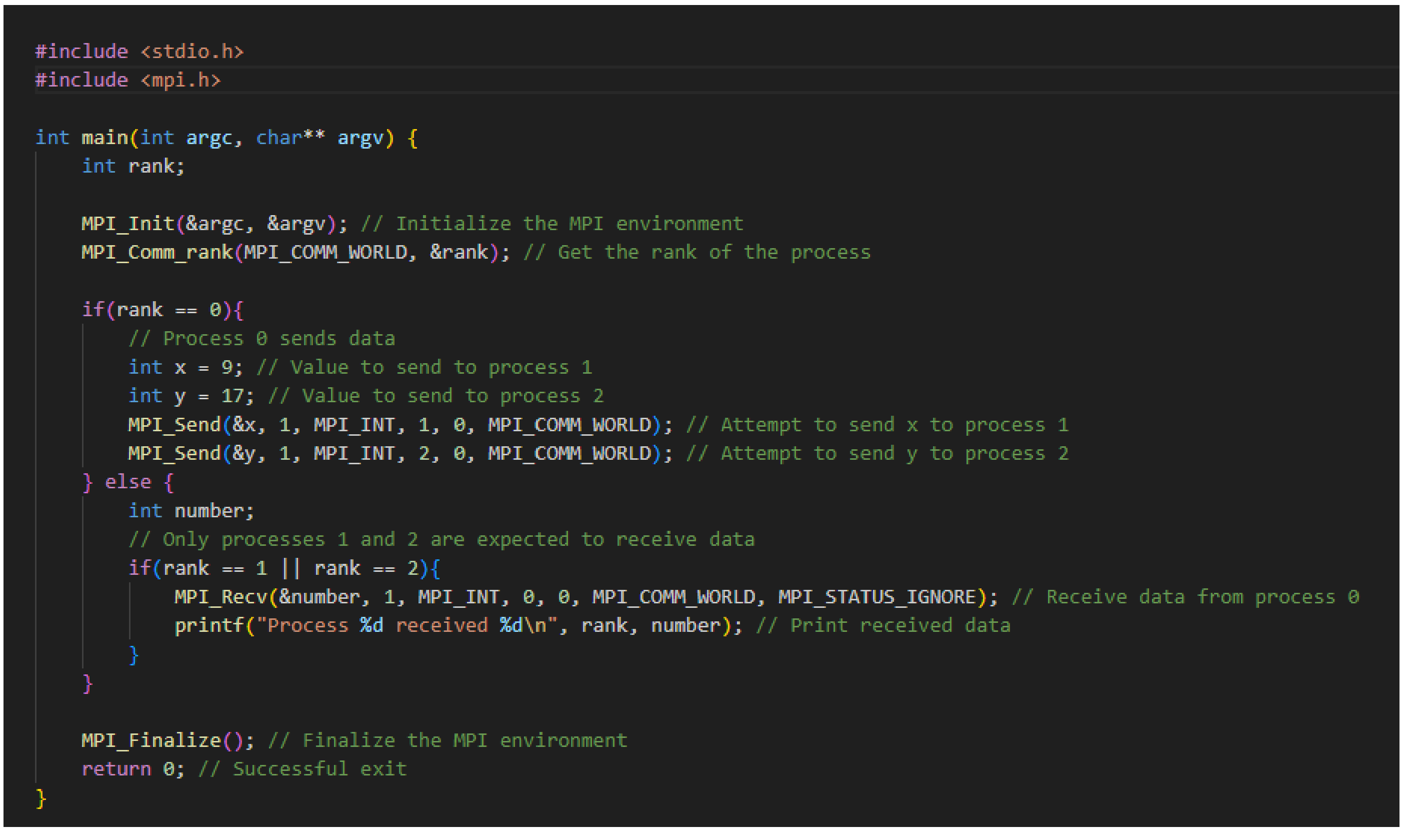
   mpiexec -n 2 -oversubscribe ./mpi02
   mpiexec -n 3 -oversubscribe ./mpi02
   mpiexec -n 4 -oversubscribe ./mpi02

Note what happens. It doesn't let you run the program with anything other than 3 processes.

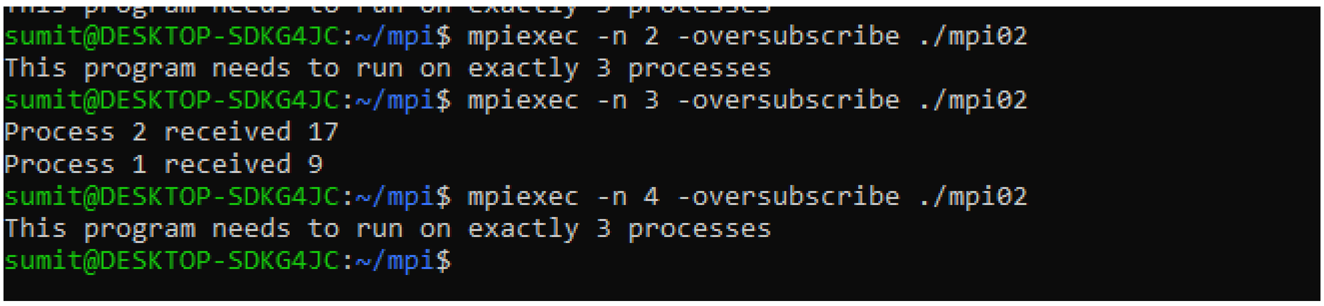


```
sumit@DESKTOP-SDKG4JC:~/mpi$ mpicc mpi02.c -o mpi02
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec ./mpi02
This program needs to run on exactly 3 processes
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 4 -oversubscribe ./mpi02
This program needs to run on exactly 3 processes
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 2 -oversubscribe ./mpi02
This program needs to run on exactly 3 processes
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 3 -oversubscribe ./mpi02
Process 1 received 9
Process 2 received 17
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 4 -oversubscribe ./mpi02
This program needs to run on exactly 3 processes
sumit@DESKTOP-SDKG4JC:~/mpi$
```

3. Now change the code so that you remove the check for only 3 processes. Now run it with 2, then 3 , then 4 and then more processes.



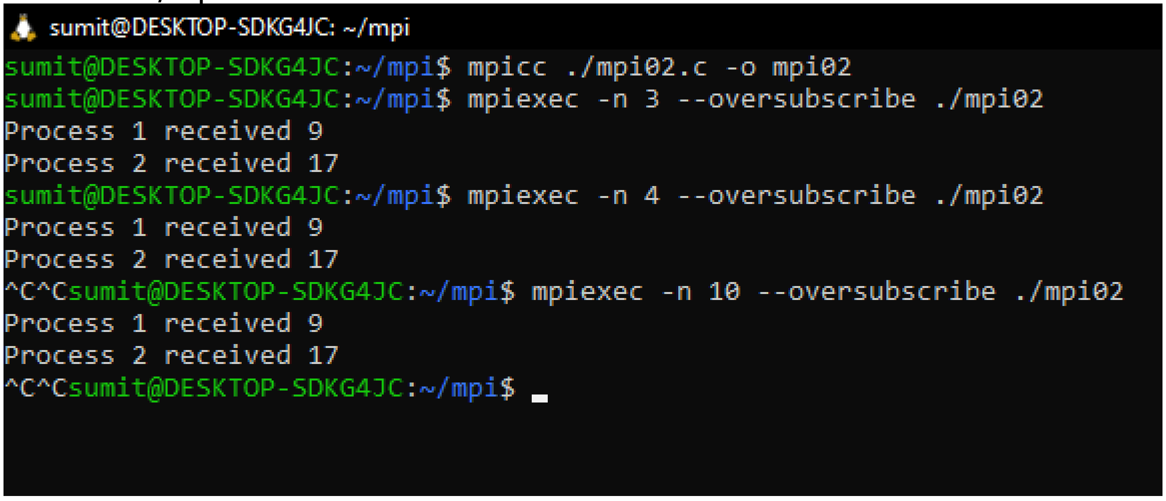
```c
#include <stdio.h>
#include <mpi.h>

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

    MPI_Init(&argc, &argv); // Initialize the MPI environment
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Get the rank of the process

    if(rank == 0){
        // Process 0 sends data
        int x = 9; // Value to send to process 1
        int y = 17; // Value to send to process 2
        MPI_Send(&x, 1, MPI_INT, 1, 0, MPI_COMM_WORLD); // Attempt to send x to process 1
        MPI_Send(&y, 1, MPI_INT, 2, 0, MPI_COMM_WORLD); // Attempt to send y to process 2
    } else {
        int number;
        // Only processes 1 and 2 are expected to receive data
        if(rank == 1 || rank == 2){
            MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE); // Receive data from process 0
            printf("Process %d received %d\n", rank, number); // Print received data
        }
    }

    MPI_Finalize(); // Finalize the MPI environment
    return 0; // Successful exit
}
```



```
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 2 -oversubscribe ./mpi02
This program needs to run on exactly 3 processes
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 3 -oversubscribe ./mpi02
Process 2 received 17
Process 1 received 9
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 4 -oversubscribe ./mpi02
This program needs to run on exactly 3 processes
sumit@DESKTOP-SDKG4JC:~/mpi$
```

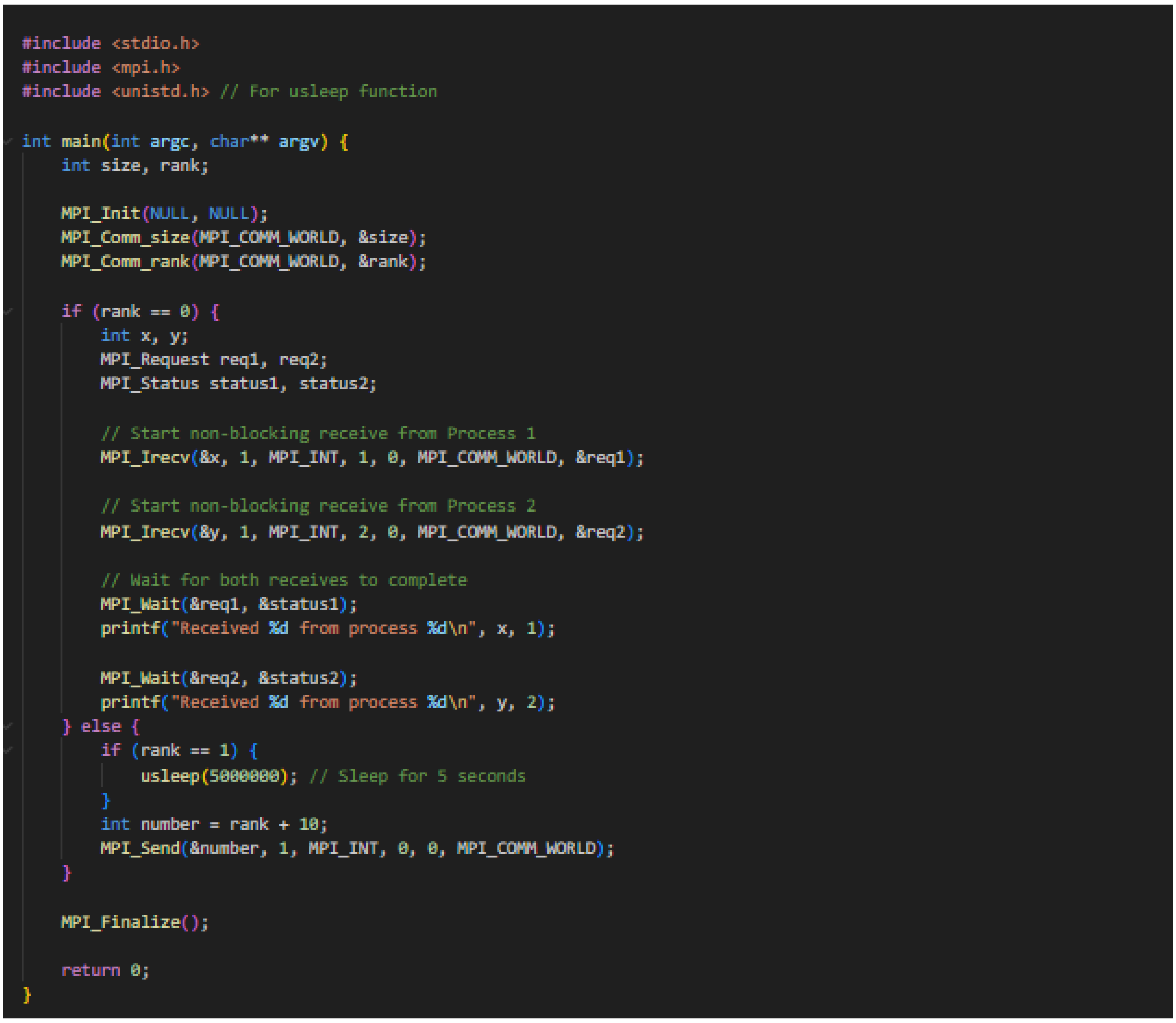
4. When you try to run it with 4 or more processes, it probably runs and appears to work, but never ends. You will have to end with "Ctrl-C". Why do you think it doesn't end when you run it with more than 3 processes? Change it so that it will work with any number of processes.

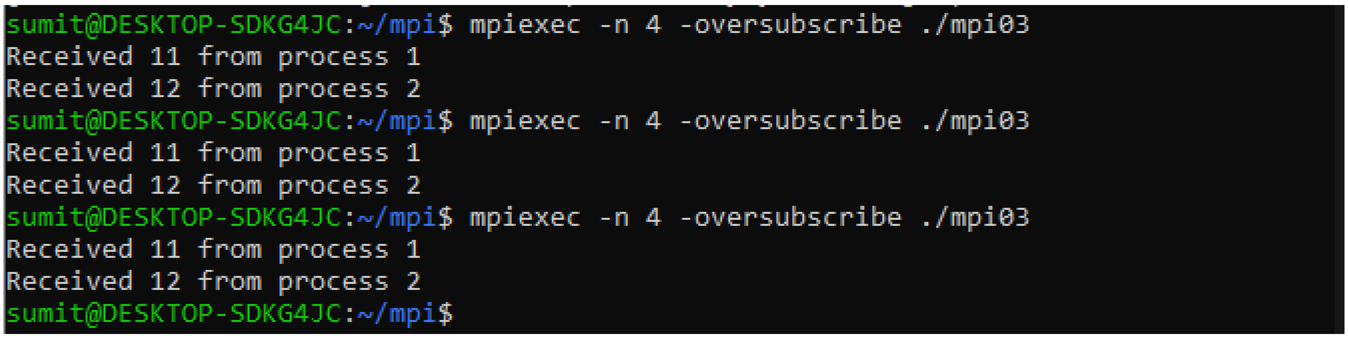
mpiexec -n 2 ./mpi02
mpiexec -n 3 ./mpi02
mpiexec -n 4 ./mpi02



```
sumit@DESKTOP-SDKG4JC: ~/mpi
sumit@DESKTOP-SDKG4JC:~/mpi$ mpicc ./mpi02.c -o mpi02
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 3 --oversubscribe ./mpi02
Process 1 received 9
Process 2 received 17
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 4 --oversubscribe ./mpi02
Process 1 received 9
Process 2 received 17
^C^Csumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 10 --oversubscribe ./mpi02
Process 1 received 9
Process 2 received 17
^C^Csumit@DESKTOP-SDKG4JC:~/mpi$ _
```

5. Build and run the program mpi03.c. In this program Process 0 will wait for messages from Process 1 and Process 2. However, Process 1 ends up blocking Process 2 because it sleeps for 5 seconds.


```c
#include <stdio.h>
#include <mpi.h>
#include <unistd.h> // For usleep function

int main(int argc, char** argv) {
    int size, rank;

    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (rank == 0) {
        int x, y;
        MPI_Request req1, req2;
        MPI_Status status1, status2;

        // Start non-blocking receive from Process 1
        MPI_Irecv(&x, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &req1);

        // Start non-blocking receive from Process 2
        MPI_Irecv(&y, 1, MPI_INT, 2, 0, MPI_COMM_WORLD, &req2);

        // Wait for both receives to complete
        MPI_Wait(&req1, &status1);
        printf("Received %d from process %d\n", x, 1);

        MPI_Wait(&req2, &status2);
        printf("Received %d from process %d\n", y, 2);
    } else {
        if (rank == 1) {
            usleep(5000000); // Sleep for 5 seconds
        }
        int number = rank + 10;
        MPI_Send(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }

    MPI_Finalize();

    return 0;
}
```


```
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 4 -oversubscribe ./mpi03
Received 11 from process 1
Received 12 from process 2
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 4 -oversubscribe ./mpi03
Received 11 from process 1
Received 12 from process 2
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 4 -oversubscribe ./mpi03
Received 11 from process 1
Received 12 from process 2
sumit@DESKTOP-SDKG4JC:~/mpi$
```

6. The following is a simple program that looks for prime numbers between 1 to 10000:

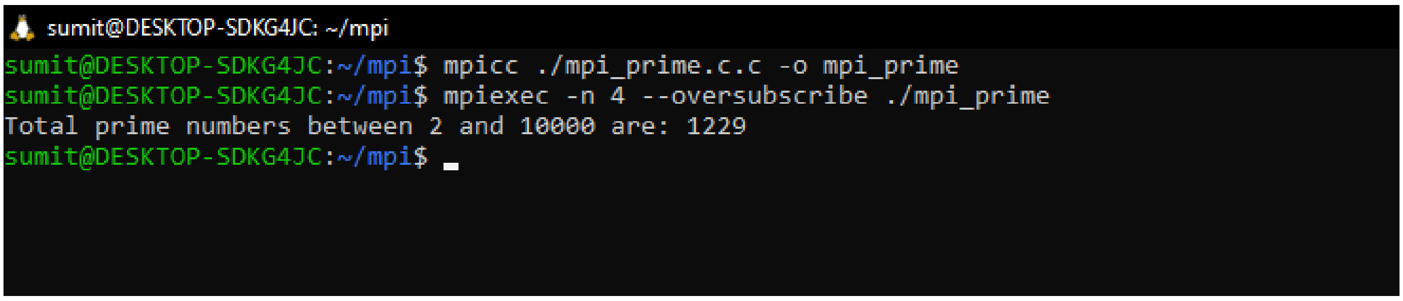
```c
#include <stdio.h>

int main(int argc, char **argv)
{
  int i, c;
  int nstart=1, nfinish=10000;
  printf("%s : Prime numbers between %d and %d are :\n",
        argv[0], nstart, nfinish);
  for(i=nstart; i<=nfinish; i++)
  {
```

```
  for(c=2; c<=i-1; c++)
  {
   if ( i%c==0 )
    break;
  }
  if ( c==i )
   printf("%s : %d\n",argv[0], i);
 }
 return 0;
}
```

Convert it to MPI so that it can run with different numbers of processes including just one process.

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

// Function to check if a number is prime
int is_prime(int n) {
    if (n <= 1) return 0; // 0 and 1 are not prime
    if (n <= 3) return 1; // 2 and 3 are prime
    if (n % 2 == 0 || n % 3 == 0) return 0; // Exclude multiples of 2 and 3
    for (int i = 5; i * i <= n; i += 6) {
        if (n % i == 0 || n % (i + 2) == 0) return 0; // Exclude multiples of other primes
    }
    return 1; // It's prime
}

int main(int argc, char **argv) {
    int rank, size;
    int nstart = 2, nfinish = 10000; // Start and end of the range
    int local_start, local_end;
    int range, local_range;
    int local_primes = 0, total_primes = 0;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    range = nfinish - nstart + 1;
    local_range = range / size;
    local_start = rank * local_range + nstart;
    local_end = (rank + 1) * local_range + nstart - 1;
    if (rank == size - 1) {
        local_end = nfinish;
    }

    // Each process counts primes in its range
    for (int i = local_start; i <= local_end; i++) {
        if (is_prime(i)) {
            local_primes++;
        }
    }

    // Reduce local counts to the root process
    MPI_Reduce(&local_primes, &total_primes, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Print prime numbers by the root process
    if (rank == 0) {
        printf("Total prime numbers between %d and %d are: %d\n", nstart, nfinish,
total_primes);
    }

    MPI_Finalize();
    return 0;
}
```

mpicc -o mpi_prime mpi_prime.c -lm
mpiexec -n <num_processes> --oversubscribe ./mpi_prime



```
sumit@DESKTOP-SDKG4JC:~/mpi$ mpicc ./mpi_prime.c.c -o mpi_prime
sumit@DESKTOP-SDKG4JC:~/mpi$ mpiexec -n 4 --oversubscribe ./mpi_prime
Total prime numbers between 2 and 10000 are: 1229
sumit@DESKTOP-SDKG4JC:~/mpi$
```