**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**

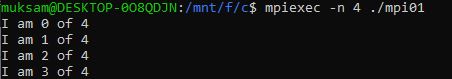
1.JPG

Now run it with the following:

**mpiexec ./mpi01**

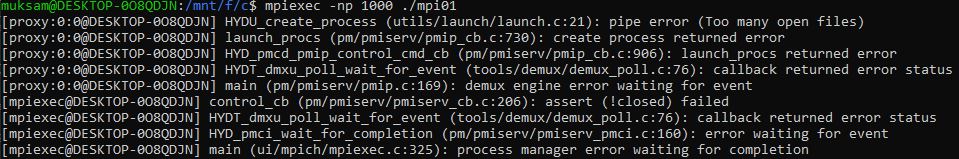
1.JPG  
  
This will (probably) only run only one process, which is not very interesting. Run it again with the following command:

**mpiexec -n 4 ./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.



Suppose, if you type this command ‘**mpiexec –np 1000 ./mpi01**’. It will fetch error. Because it has some resource limitation. The main reason behind it that, there is not enough memory spaces. This error occurs because each MPI process required a certain amount of memory to run, and when you try to run 1000 processes, the memory requirements exceed the available memory on the system.

1. 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**

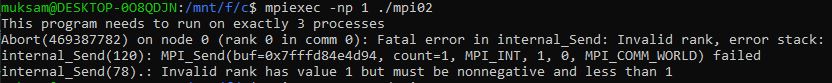
**mpiexec -n 4 -oversubscribe ./mpi02**

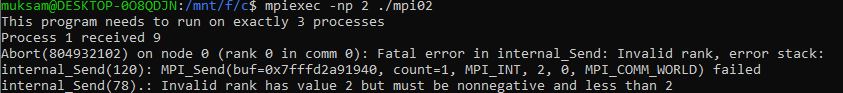
1.JPG

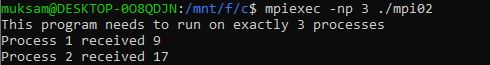
Note what happens. It doesn't let you run the program with anything other than 3 processes.

1.JPG

1. 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.







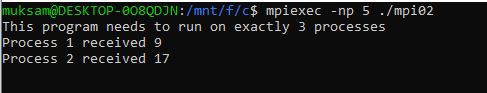
When the program is run with more than 3 processes, it enters an issue related to the blocking communication between processes.

Process 0 sends messages to Processes 1 and 2.

Processes 1 and 2 wait to receive the messages from Process 0 using MPI\_Recv.

However, if there are more than 3 processes, additional processes beyond 0, 1, and 2 may not receive any messages, causing the program to hang, leading to a deadlock.

1. 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.



#include <stdio.h>

#include <mpi.h>

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){

      for(int i = 1; i < size; i ++){

        int x = 10;

        MPI\_Send(&x, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

      }

    } else {

      int number;

      MPI\_Recv(&number, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

      printf("Process %d received %d\n", rank, number);

    }

  MPI\_Finalize();

  return 0;

}

// Communication Logic:

// Process 0:

// Enters the loop (for (int i = 1; i < size; ++i)) to send messages to all other processes.

// Creates an integer x with the value 10.

// Uses MPI\_Send to send x to each process (i) with a tag of 0.

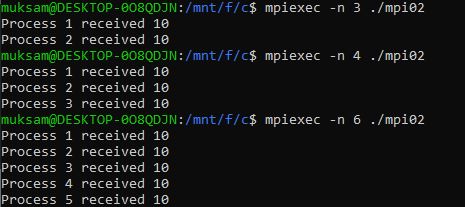
// Other Processes (non-zero ranks):

// Wait to receive a message from Process 0 using MPI\_Recv.

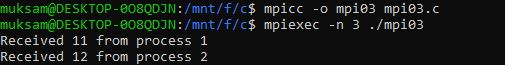
// The message is expected to have the tag 0 and comes from Process 0.

// The received value is stored in the number variable.

// Prints a message indicating the received value and the process rank.



1. 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. How would you change the code so that Process 1 does not block Process 2, even if it does sleep for 5 seconds?



#include <stdio.h>

#include <mpi.h>

#include <unistd.h>

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(size != 3) {

    if(rank == 0) {

      printf("This program needs to run on exactly 3 processes\n");

    }

  } else {

    if(rank ==0){

      int x, y;

      MPI\_Recv(&x, 1, MPI\_INT, 2, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

      printf("Received %d from process %d\n", x, 2);

      MPI\_Recv(&y, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

      printf("Received %d from process %d\n", y, 1);

    } else {

      if(rank == 1){

        usleep(5000000);

      }

      int number = rank + 10;

      MPI\_Send(&number, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

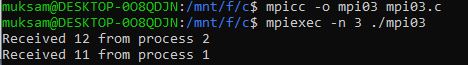
    }

  }

  MPI\_Finalize();

  return 0;

}



MPI\_Irecv is a function in the Message Passing Interface (MPI) library that initiates a non-blocking receive operation. Non-blocking receive operations allow the program to continue executing while waiting for a message to arrive, which can improve performance in some cases.

Here I changed rank 1 to 2 and rank 2 to 1 on MPI\_Irecv function which reverses the receiving end.

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

**#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.

#include <stdio.h>

#include <mpi.h>

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

    int i, c;

    int size, rank;

    int nstart = 1, nfinish = 1000;

    MPI\_Init(NULL, NULL); // If you dont' want any command-line argument simply pass null

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    printf("Prime numbers between %d and %d are: \n", nstart, nfinish);

    // syntax for(initialize, condition, increment)

    for(i = nstart; i <= nfinish; i++){

        for (c = 2; c <= i; c++){

            if (i % c == 0){

                break;

            }

            if (c = i){

                printf("%s: %d\n", argv[0], i);

            }

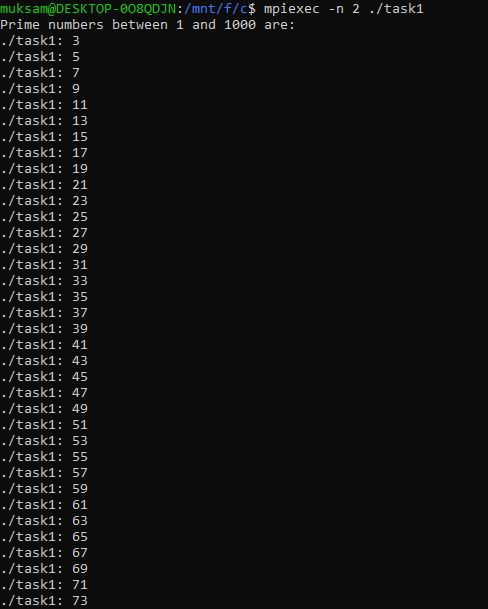
        }

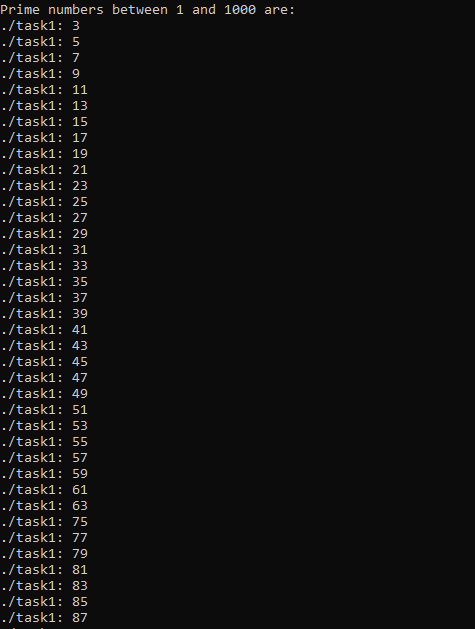
    }

    MPI\_Finalize();

    return 0;

}

****

****