7

```
#include <math.h>
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void po_send_input (int *input1, int *input2);
void p0_send_input (int input1, int input2);
void p0_receive_output (int *output1, int *output);
int p1_receive_input();
int p1_compute_output (int output1);
void p1_send_output (int output);
int p2_receive_input ();
int p2_compute_output (int input);
void p2_send_output (int output2);
void Hostcomp();
int main (int argc, char *argv[])
{
    int id; ierr, input1, input2, output1, output2, p;
double wtime;
ierr = MPI_Init (&argc, &argv);
if (ierr != 0)
{
    printf("\n");
    printf("\n MPI MULTIMASK - fatal error\n");
    printf("MPI_Init returned non zero IERR\n");
```

```
exit(1);
}
ierr = MPI_Comm_Rank (MPI_COMM_WORLD, &id);
ierr = MPI_Comm_Size (MPI_COMM_WORLD, &p);

if (p<3)
{
printf ("\n");
printf ( MPI_MULTITASK - Fatal err! \n");
printf (" No. of availabl processes must be atleast 3 \n");
ierr = MPI_Finalize();
exit(0);
}
if ( id==0)
{
    timestamp( );
    printf("\n");
    printf ( "MPI_MULTITASK\n");
    printf ("C / MPI_Version\n");

wtime = MPI_Wtime();

po_send_input (&input1, &input2);
po_send_input (input1, input2)
po_receive_output (&output1, &output2);

wtime = MPI_Wtime() - wtime;
printf (" Proces 0 time = %lg\n", wtime);
ierr = MPI_Finalize();
printf("\n" MPI_MULTITASK\n");
printf (" Normal end of execution\n");
timestamp();
```

```
else if (id == 1)
{
    wtime = MPI_Wtime();
    input1 = P1_receive_input();
    output1 = P1_compute_output(input1);
    P1_send_output(output1);
wtime = MPI_Wtime() - wtime;
printf("Process 1 time = %lg\n", wtime);
    ierr = MPI_Finalize();
}
else if (id == 2)
{
    wtime = MPI_Wtime();
    input2 = P2_receive_input();
    output2 = P2_compute_output(input2);
P2_Send_Output(output2);
wtime = MPI_Wtime() - wtime;
printf("Process 2 time = %lg\n", wtime);
ierr = MPI_Finalize();
}
return 0;
}
void P0_set_input(int *input1, int *input2)
{
    *input1 = 10000000;
    *input2 = 100000;
printf("\n Set INPUT1 = %d\n", *input1);
printf("INPUT2 = %d\n", *input2);
return; }
```

```
void p0_Send_Input(int *input1, int *input2)
{
   int id, tag;
   int id = 1, tag = 1;
MPI_Send(&input1, 1, MPI_INT, id, tag, MPI_COMM_WORLD);

   id = 2;
   tag = 2;
MPI_Send(&input1, 1, MPI_INT, id, tag, MPI_COMM_WORLD);

   return;
}
void p0_receive_output(int *output1, int *output2)
{
   int output;
   int output_received;
   int source;
   MPI_Status status;

   output_received = 0;
while(output_received < 2)
{
MPI_Recv(&output, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
      MPI_COMM_WORLD, &status);

source = status.MPI_SOURCE;

if(source == 1)
{
   *output1 = output;
}
else {
```

```
    *output2 = output;
  }
output_received = output_received + 1;
printf("\n Process 1 received OUTPUT 1 = %d \n", *output1);
printf(" Process 2 received OUTPUT 2 = %d \n", *output2);
 return;
}

int p1_receive_input()
{
   int id;
   int input1;
  MPI_Status Status;

   int tag;
   id = 0;
   tag = 1;
  MPI_Recv( &input1, 1, MPI_INT, id, tag, MPI_Comm_WORLD,
            &status);

  return input1;
  }
int p1_write_output(int input1)
  {
  int i, j, k, output1;

  output1 = 0;

for ( i= 2 ; i <= input1; i++)
   {
      j=1;
      k=0;
```

```
while ( 1 < j)
  {
     if ( j % 2 ) == 0)
       {
          j = j / 2 ;
       }
     else
          { j = 3 * j + 1;
          }
          ik = k + 1;
       }
    if (output1 < k)
      { output = k;
      }
    }
  return output1;
}
void P1_Send_output ( int output1)
 {
   int id, tag;

  id = 0; tag = 3;
  MPI_Send ( &output1, 1, MPI_INT, id, tag, MPI_COMM_WORLD);
    return 0;
  }
  int P2- recive - input ( )
  {
     int id, input2;
  MPI_Start status;

   id = 0;
   tag = 2
   MPI_Recv ( &input, 1, MPI_INT, tag, MPI_COMM_WORLD &status);
```

```
return input 2;
}
int P2_Compute_output( int input2)
{
    int i, j, output2,
int prime;
output2 = 0;

for (i=2; i<input; i++)
    {
      prime = 1;

    for (j=2; j<i; j++)
      {
        if (i%j) == 0)

        prime = 0;

        break;

      }
    if (prime)
      { output2 += 1;

      }
  }
  return output2;

}
void P2_Send_output (int output2)

{
  int id, tag;

  id = 0;

  tag = 4;
  MPI_Send ( &output2, 1, MPI_INT, id, tag, MPI_COMM_WORLD);

    return;
```

}

~~void p2_sort_output(int output)~~
~~{~~
~~int id, tag;~~

```
void timestamp()
{
  # define TIME_SIZE 10
  static char time_buffer[TIME_SIZE];
  const struct tm * tm;
  time_t now;
  now = time(NULL);
  tm = localtime(&now);
  strftime(time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p",
       tm);
  printf("%s\n", time_buffer);
  return;
  # undef TIME_SIZE
}
```

```
mpicc p7.c -o p7.
mpirun -np 3 ./p7.
```

Output

```
MPI_MULTITASK
P0_SET_PARAMETERS
SE   INPUT1 = 10000000
     INPUT2 = 100000
Process 2 time = 3.00771
```

```
Process 1 returned OUTPUT1 = 615
Process 2 returned OUTPUT2 = 9592.
Process 0 time = 10.8738
Process 1 time = 10.8793
MPI_MULTITASK
Normal end of execution.
```