(Q7). Write an MPI-C program which demonstrates how to multi-task, to execute several unrelated & distinct tasks simultaneously

→ Code:

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

int main( int argc , char* argv[] )
  { int id; int ierr; int input1, input2, output1, output2;
    int p; double wtime;

   ierr = MPI_Init (&argc, &argv);
   if (ierr != 0)
     {      printf( "\n");
            printf ( " MPI_multitask - Fatal Error!\n " );
            printf ( " MPI_Init returned nonzero 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-Multi task - Error\n" );
           printf( " No. of available must be atleast 3 \n" );
           ierr = MPI_Finalize();
```

```
    exit(1); }
if (id==0)
    {       timestamp();
            Printf("\n");
            Printf("MPI_Multi task \n");
            Printf("  (/mpi version \n");

        wtime = MPI_Wtime();
       PO_set_input( &input1, &input2);
       PO_send_input(input1, input2);
       PO_receive_output( &output1, &output2);

  wtime =   MPI_Wtime() - wtime;
   Printf("  Process 0 time = %lg \n", wtime);
    ierror =    MPI_Finalize();
   printf("\n");
   Printf("  MPI_Multi task \n Normal End of  execution \n");

     timestamp();

else if (id==1)
   {     wtime =   MPI_Wtime();
        input1 =   PI_receive_input();
        output1=   PI_Compute_output(input1);
        PI_Send_Output( output1);

    wtime =   MPI_Wtime() - wtime;
    wtime   printf("  Process  1 time =  %lg \n", wtime);
      ierror =  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 = 10000;
        printf("\n");
       printf(" P0_SET_PARAMETERS: \n");
       printf("Set INPUT1 = %d \n", *input1);
       printf("Set INPUT2 = %d \n", *input2);
      return;
    }
Void p0_send_input( int * input1, int input2 )
  {   int id; int tag;
    id=1; tag=1;
    MPI_Send( &input1, 1, MPI_INT, id, tag, MPI_Comm_WORLD);
    id=2; tag=2;
   MPI_Send( &input2, 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");
Printf(" Process 1 returned OUTPUT1 = %d\n", *output1);
Printf(" Process 2 returned OUTPUT2 = %d\n", *Output2);

   return;
    }

int P1_receive_input()
   {    int id; int input1;
        MPI_Status status;
         int tag;

   id=0;   tag=1;
    MPI_Recv( &input2, 1, MPI_INT, id, tag, MPI_Comm_WORLD,
                                          &status);
```

```
    return input1;
}
int p1_compute_output ( int input1 )
{       int i, j, k, int output1;
        Output1 = 0;
    for ( i = 2;  i <= input1;  i++ );
        {       j = i;    k = 0;
            while (i<j)
                {   if ((j%2) == 0)
                        {   j = j/2;   }
                    else {
                            j = 3*j+1;
                         }
                    k = k+1;
                    if (output2< k)
                        {   output1 = k;   }
                }
    return Output1;
}
Void   p1_Send_Input ( int output1 )
    {       int id; tag;
            id = 0;   tag = 3;
        MPI_SEND ( &output1, 1, MPI_INT, id, tag , MPI_Comm_WORLD);
        return;
    }
int  p2_receive_input( )
    {           int id;  int input2;
            MPI_Status   status;
```

```
    int tag;
    id = 0;   tag = 2;
    MPI_Recv( &input2, 1, MPI_INT, id, tag, MPI_COMM_WORLD, &Status);

    return input2;
}

int p2_compute_output( int input2)
{       int i, j, k, output2, prime;

        output2 = 0;

    for (i=2; i<=input2; i++)
      {     prime = 1;
          for (j=2; j<i; j++)
            {   if (~~j%2 == 0~~ (i%j) == 0)
                {   prime = 0;
                    break; } }

          if (prime)
            {   output2 = 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 timestamp()
{
    #define TIME_SIZE 40
    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
}
```

OUTPUT : ~~MPI_MULTITASK~~

- mpicc prog7.c
  mpirun -np2 ./a.out

MPI_MULTITASK - fatal error!
Number of available process must be atleast 3!

- mpicc prog7.c
  mpirun -np ./a.out

MPI_MULTITASK -
C / MPI Version

PO_SET_PARAMETERS :

SET INPUT1 = 10000000

SET INPUT2 = 100000.

Process 2 time = 4.53377

Process 1 time = 12.9537

Process 1 returned OUTPUT1 = 615

Process 2 returned OUTPUT2 = 9592

Process 0 time = 12.9523

MPI_MULTITASK:

Normal End of Execution.