

ROHIT RAJ

1RV17CS125

7TH CSE C2, PADP

PROGRAM 7

CODE:

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

int main ( int argc, char *argv[] );

void p0_set_input ( int *input1, int *input2 );
void p0_send_input ( int input1, int input2 );
void p0_receive_output ( int *output1, int *output2 );
int p1_receive_input ( );
int p1_compute_output ( int input1 );
void p1_send_output ( int output1 );
int p2_receive_input ( );
int p2_compute_output ( int input2 );
void p2_send_output ( int output2 );
void timestamp ( );

int main ( int argc, char *argv[] )

{
    int id;
    int ierr;
    int input1;
    int input2;
    int output1;
```

```

int output2;

int p;

double wtime;

/*
Process 0 is the "monitor".
It chooses the inputs, and sends them to the workers.
It waits for the outputs.
It plots the outputs.
*/

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 );

/*
Make sure we have enough processes.
*/

if ( p < 3 )
{
    printf ( "\n" );
    printf ( "MPI_MULTITASK - Fatal error!\n" );
    printf ( " Number of available processes must be at least 3!\n" );
    ierr = MPI_Finalize ( );
    exit ( 1 );
}

```

```

    }
/*
    Run program P0 on process 0, and so on.
*/
    if ( id == 0 )
    {
        timestamp ( );

        printf ( "\n" );
        printf ( "MPI_MULTITASK:\n" );
        printf ( "  C / MPI version\n" );

        wtime = MPI_Wtime ( );

        p0_set_input ( &input1, &input2 );
        p0_send_input ( input1, input2 );
        p0_receive_output ( &output1, &output2 );

        wtime = MPI_Wtime ( ) - wtime;
        printf ( "  Process 0 time = %g\n", wtime );

        ierr = MPI_Finalize ( );

        printf ( "\n" );
        printf ( "MPI_MULTITASK:\n" );
        printf ( "  Normal end of execution.\n" );

        timestamp ( );
    }
/*
    Process 1 works on task 1.
    It receives input from process 0.

```

It computes the output.

It sends the output to process 0.

*/

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 = %g\n", wtime);

 ierr = MPI_Finalize ();

}

/*

Process 2 works on task 2.

It receives input from process 0.

It computes the output.

It sends the output to process 0.

*/

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 = %g\n", wtime);

 ierr = MPI_Finalize ();

}

return 0;

}

/******

```
void p0_set_input ( int *input1, int *input2 )
```

```
{  
    *input1 = 10000000;  
    *input2 = 100000;  
  
    printf ( "\n" );  
    printf ( "P0_SET_PARAMETERS:\n" );  
    printf ( " Set INPUT1 = %d\n", *input1 );  
    printf ( "   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;
/*
    Loop until every worker has checked in.
*/
    while ( output_received < 2 )
    {
/*
        Receive the next message that arrives.
*/
        MPI_Recv ( &output, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
            MPI_COMM_WORLD, &status );
/*
        The actual source of the message is saved in STATUS.
*/
        source = status.MPI_SOURCE;
/*
        Save the value in OUTPUT1 or OUTPUT2.
*/
        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 ( &input1, 1, MPI_INT, id, tag, MPI_COMM_WORLD, &status );

    return input1;
}

```

```

int p1_compute_output ( int input1 )

```

```

{
    int i;

```

```
int j;
int k;
int output1;

output1 = 0;

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

    while ( 1 < j )
    {
        if ( ( j % 2 ) == 0 )
        {
            j = j / 2;
        }
        else
        {
            j = 3 * j + 1;
        }
        k = k + 1;
    }
    if ( output1 < k )
    {
        output1 = k;
    }
}
return output1;
}
```



```
void p1_send_output ( int output1 )
```

```
{  
    int id;  
    int 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;
    int j;
    int output2;
    int prime;

    output2 = 0;

    for ( i = 2; i <= input2; i++ )
    {
        prime = 1;
        for ( j = 2; j < i; j++ )
        {
            if ( ( i % j ) == 0 )
            {
                prime = 0;
                break;
            }
        }
        if ( prime )
        {
            output2 = output2 + 1;
        }
    }
    return output2;
}

```

```

void p2_send_output ( int output2 )
{
    int id;
    int 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:
  C / MPI version

P0_SET_PARAMETERS:
  Set INPUT1 = 10000000
  INPUT2 = 100000
  Process 2 time = 2.50949
[Rohit:00081] 2 more processes have sent help message help-btl-vader.txt / cma-permission-denied
[Rohit:00081] Set MCA parameter "orte_base_help_aggregate" to 0 to see all help / error messages

  Process 1 returned OUTPUT1 = 615
  Process 2 returned OUTPUT2 = 9592
  Process 0 time = 8.48878
  Process 1 time = 8.48874

MPI_MULTITASK:
  Normal end of execution.
```