**Program 7**

**Write MPI- C program which demonstrates how to "multitask", that is, to execute several unrelated and distinct tasks simultaneously, using MPI for parallel execution.**

# 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

}









