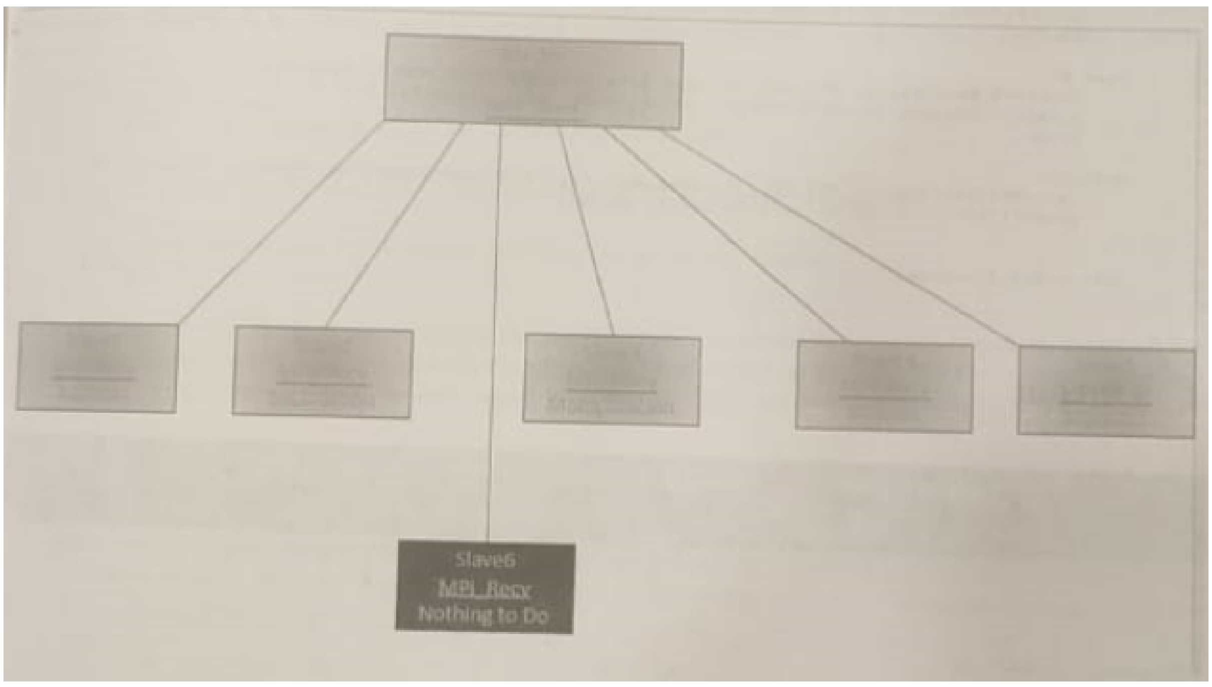
Shourya Kothari
60004190103
B2

Experiment No. 6

Aim: Implement a parallel programming for calculator application using directives of MPI/OpenMP.

Theory:

MPI is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementers, and users. The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in C, C++, and Fortran. There are several well-tested and efficient implementations of MPI, many of which are open source or in the public domain. These fostered the development of a parallel software industry, and encouraged development of portable and scalable large-scale parallel applications. One of the uses of parallel programming is in developing calculators. We have used MPI to implement parallel programming for designing a calculator. Our program involves 1 master class and 6 slave classes (5 for operations and 1 for default case). Master class takes input from the user and sends it to the receiver. We have used MPI.Send and MPI.Recv for communication between master and slave class. The calculator will perform all the operations on input variables in parallel and generate their respective output along with process id.

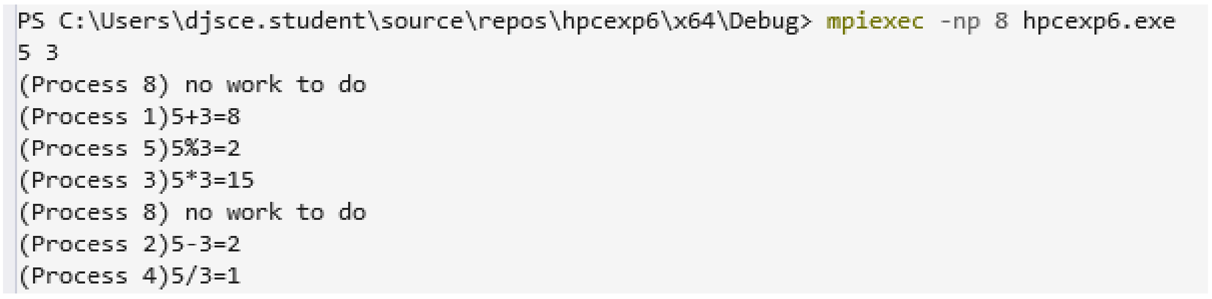


Code:

```
#include<stdio.h>
#include<mpi.h>
#define send_data_tag 2001
#define return_data_tag 2002

int main(int argc, char** argv) {
MPI_Status status;
int ierr, my_id, num_procs;
int a[2];
ierr = MPI_Init(&argc, &argv);
ierr = MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
ierr = MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
switch (my_id) {
case 0:
scanf_s("%d %d", &a[0], &a[1]);
for (int i = 1; i < num_procs; i++) {
ierr = MPI_Send(&a, 2, MPI_INT, i, send_data_tag, MPI_COMM_WORLD);
}
break;
case 1:
ierr = MPI_Recv(&a, 2, MPI_INT, 0, send_data_tag, MPI_COMM_WORLD, &status);
printf("(Process %d)%d+%d=%d\n", my_id, a[0], a[1], (a[0] + a[1]));
break;
case 2:
ierr = MPI_Recv(&a, 2, MPI_INT, 0, send_data_tag, MPI_COMM_WORLD, &status);
printf("(Process %d)%d-%d=%d\n", my_id, a[0], a[1], (a[0] - a[1]));
break;
case 3:
ierr = MPI_Recv(&a, 2, MPI_INT, 0, send_data_tag, MPI_COMM_WORLD, &status);
printf("(Process %d)%d*%d=%d\n", my_id, a[0], a[1], (a[0] * a[1]));
break;
case 4:
ierr = MPI_Recv(&a, 2, MPI_INT, 0, send_data_tag, MPI_COMM_WORLD, &status);
printf("(Process %d)%d/%d=%d\n", my_id, a[0], a[1], (a[0] / a[1]));
break;
case 5:
ierr = MPI_Recv(&a, 2, MPI_INT, 0, send_data_tag, MPI_COMM_WORLD, &status);
printf("(Process %d)%d%%%d=%d\n", my_id, a[0], a[1], (a[0] % a[1]));
break;
default:
ierr = MPI_Recv(&a, 2, MPI_INT, 0, send_data_tag, MPI_COMM_WORLD, &status);
printf("(Process %d) no work to do\n");
break;
}
ierr = MPI_Finalize();
}
```

Output:



```
PS C:\Users\djsce.student\source\repos\hpcexp6\x64\Debug> mpiexec -np 8 hpcexp6.exe
5 3
(Process 8) no work to do
(Process 1)5+3=8
(Process 5)5%3=2
(Process 3)5*3=15
(Process 8) no work to do
(Process 2)5-3=2
(Process 4)5/3=1
```

Conclusion: From this experiment, we studied how to use MPI in C Program to construct parallel programming for calculator application using directives of MPI/OpenMP.