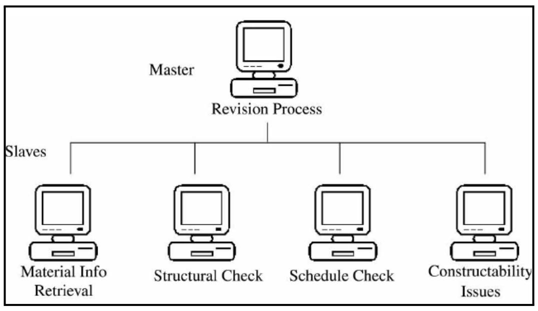
EXPERIMENT 6

AIM: To implement parallel programming for calculator application using directives of MPI.

THEORY:
Master/slave is a model of asymmetric communication or control where one device or process (the "master") controls one or more other devices or processes (the "slaves") and serves as their communication hub. In some systems, a master is selected from a group of eligible devices, with the other devices acting in the role of slaves.
Key Features:

1. The system comprises a master controller generating commands and receiving status signals and slave devices associated with the master controller.
2. Each slave receives commands, executes local commands responsive to the commands and generates status signals for the master controller.
3. Each slave has a communication arrangement for signals transmitted between it and the master controller.
4. The arrangement comprises a communication controller associated with the master controller.
5. The communication controller receives commands, transmits commands to each slave, receives status signals and provides information relating to the status signals to the master controller.
6. The controller has a communication link which transmits commands to each slave and the status signals to the controller.
7. The system allows local commands executed by the slaves to replace other commands directed by the master controller to the slave.
8. Further, each slave communicates independently with the master controller.



CODE:

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

#define send_data_tag 2001

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) performs %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) performs %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) performs %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) performs %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) performs %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) has no work to do \n", my_id);
    }
    ierr = MPI_Finalize();
}
```

OUTPUT:

```
C:\Users\JARVIS\source\repos\Experiment 6\x64\Debug>mpiexec -n 8 "Experiment
6.exe"
5 9
(Process 3) performs 5 * 9 = 45
(Process 1) performs 5 + 9 = 14
(Process 4) performs 5 / 9 = 0
(Process 2) performs 5 - 9 = -4
(Process 6) has no work to do
(Process 7) has no work to do
(Process 5) performs 5 % 9 = 5

C:\Users\JARVIS\source\repos\Experiment 6\x64\Debug>mpiexec -n 8 "Experiment
6.exe"
2 4
```

```
(Process 1) performs 2 + 4 = 6
(Process 3) performs 2 * 4 = 8
(Process 6) has no work to do
(Process 5) performs 2 % 4 = 2
(Process 2) performs 2 - 4 = -2
(Process 7) has no work to do
(Process 4) performs 2 / 4 = 0
```

CONCLUSION: Master slave architectures allow a distributed application with a single (or a few) servers handling request management. The rest of the network acts as slaves to the master server and executes commands on request. P2P links can be established between each master and slave using MPI. Thus, each node can act as a processing unit for the master executing different operations. This drastically reduces the processing and response time for a request and is directly correlated to the number of processors assigned per request.