11.a IPC: Implement the program IPC/IPS using MPI library. Communication in the processes of users.

Subject:- Unix Operating System
System Lab Class :- TYIT
Name PRN
Aditi Sudhir Ghate 2020BTEIT00044

Objectives:

1. To learn about IPC through MPI.
2. Use of IPC mechanism to write effective application programs.
3. configure cluster and experiment MPI program on it.

Theory:

Inter-process communication or **interprocess communication** (**IPC**) refers specifically to the mechanisms an operating system provides to allow the processes to manage shared data. Typically, applications can use IPC, categorized as clients and servers, where the client requests data and the server responds to client requests.[1] Many applications are both clients and servers, as commonly seen in distributed computing.

IPC is very important to the design process for microkernels and nanokernels, which reduce the number of functionalities provided by the kernel. Those functionalities are then obtained by communicating with servers via IPC, leading to a large increase in communication when compared to a regular monolithic kernel. IPC interfaces generally encompass variable analytic framework structures. These processes ensure compatibility between the multi-vector protocols upon which IPC models rely.[2]

An IPC mechanism is either synchronous or asynchronous. Synchronization primitives may be used to have synchronous behavior with an asynchronous IPC mechanism.

MPI-2 defines three one-sided communications operations, MPI_Put, MPI_Get, and MPI_Accumulate, being a write to remote memory, a read from remote memory, and a reduction operation on the same memory across a number of tasks, respectively. Also defined are three different methods to synchronize this communication (global, pairwise, and remote locks) as the specification does not guarantee that these operations have taken place until a synchronization point.

These types of call can often be useful for algorithms in which synchronization would be inconvenient (e.g. distributed matrix multiplication), or where it is desirable for tasks to be able to balance their load while other processors are operating on data.

Program:

```
/*
  "Hello World" MPI Test Program for IPC
*/
#include <assert.h>
#include <stdio.h>
#include <string.h>
#include <mpi.h>

int main(int argc, char **argv)
{
   char buf[256];
   int my_rank, num_procs;

   /* Initialize the infrastructure necessary for communication */
   MPI_Init(&argc, &argv);

   /* Identify this process */
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

   /* Find out how many total processes are active */
   MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

   /* Until this point, all programs have been doing exactly the same.
     Here, we check the rank to distinguish the roles of the programs */
   if (my_rank == 0) {
      int other_rank;
      printf("We have %i processes.\n", num_procs);

      /* Send messages to all other processes */
      for (other_rank = 1; other_rank < num_procs; other_rank++)
      {
         sprintf(buf, "Hello %i!", other_rank);
         MPI_Send(buf, sizeof(buf), MPI_CHAR, other_rank,
               0, MPI_COMM_WORLD);
      }

      /* Receive messages from all other process */
      for (other_rank = 1; other_rank < num_procs; other_rank++)
      {
```

```
            MPI_Recv(buf, sizeof(buf), MPI_CHAR, other_rank,
                  0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("%s\n", buf);
        }

    } else {

        /* Receive message from process #0 */
        MPI_Recv(buf, sizeof(buf), MPI_CHAR, 0,
              0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        assert(memcmp(buf, "Hello ", 6) == 0);

        /* Send message to process #0 */
        sprintf(buf, "Process %i reporting for duty.", my_rank);
        MPI_Send(buf, sizeof(buf), MPI_CHAR, 0,
              0, MPI_COMM_WORLD);

    }

    /* Tear down the communication infrastructure */
    MPI_Finalize();
    return 0;
}
```

Output:

$ mpicc example.c && mpiexec -n 4 ./a.out

```
We have 4 processes.
Process 1 reporting for duty.
Process 2 reporting for duty.
Process 3 reporting for duty.
```

Conclusion:

1. Implemented the program IPC/IPS using MPI library

References:

https://en.wikipedia.org/wiki/Message_Passing_Interface