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

Subject:- Unix Operating System System Lab Class :- TYIT

**Name: - SHRENIK JADHAV**

**OM GHARGE**

**PRN: -2020BTEIT00011,2020BTEIT00041**

**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:

*#include <stdio.h>*

#include <mpi.h>

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

int rank, size, value;

MPI\_Status status;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

if (rank == 0) {

value = 42;

MPI\_Send(&value, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

printf("Process %d sent value %d to process %d.\n", rank, value, rank + 1);

} else if (rank == size - 1) {

MPI\_Recv(&value, 1, MPI\_INT, rank - 1, 0, MPI\_COMM\_WORLD, &status);

printf("Process %d received value %d from process %d.\n", rank, value, rank - 1);

} else {

MPI\_Recv(&value, 1, MPI\_INT, rank - 1, 0, MPI\_COMM\_WORLD, &status);

printf("Process %d received value %d from process %d.\n", rank, value, rank - 1);

value \*= 2;

MPI\_Send(&value, 1, MPI\_INT, rank + 1, 0, MPI\_COMM\_WORLD);

printf("Process %d sent value %d to process %d.\n", rank, value, rank + 1);

}

MPI\_Finalize();

return 0;

}

*run--*

*To compile and run the program using MPI on Ubuntu, you'll need to follow these steps:*

1. Ensure that you have the MPI library installed. You can install it by running the following command in the terminal:

arduino

* sudo apt-get install libopenmpi-dev
* Save the code into a file, for example, mpi\_ipc\_ips.c.
* Open a terminal and navigate to the directory where you saved the file.
* Compile the program using the mpicc command:
* mpicc mpi\_ipc\_ips.c -o mpi\_ipc\_ips
* After successful compilation, you can execute the program using the mpirun command:

bash

* mpirun -np <number\_of\_processes> ./mpi\_ipc\_ips

Replace <number\_of\_processes> with the desired number of processes (e.g., 4).

* The program will start running, and you'll see the output in the terminal, showing the communication between processes and the values being passed.

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