Group Name : F

Dated : 2 November 2020

Timing : 2:00 PM - 3:30 PM

Aim :

Write a simple MPI program to print a word 'Greetings' the processes. In order to explore its practical use, you are advised to read and understand the following statements.

~~1. MPI provides~~

Source Code :

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

const int MAX_STRING = 100;

int main (void){

    char greeting[MAX_STRING];
    int my_rank, p, q;
    MPI_Init (NULL, NULL);
    MPI_Comm_size (MPI_COMM_WORLD, &p);
    MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
    if (my_rank == 0) {
        printf ("Greetings from process %d of %d \n", my_rank, p);
        for (q = 1 , q < p, q++){
            MPI_Recv (greeting , MAX_STRING , MPI_CHAR, q , 0,
                      MPI_COMM_WORLD , MPI_STATUS , IGNORE);
            printf ("%s \n", greeting );
        }
}
```

```
    else {
        printf (greeting, "Greeting from process %d of %d",
                my_rank, p);
        MPI_Send (greeting, strlen (greeting) + 1, MPI_CHAR, 0, 0,
                  MPI_COMM_WORLD);
    }
    MPI_Finalize ();
    return 0;
}
```

Execution :

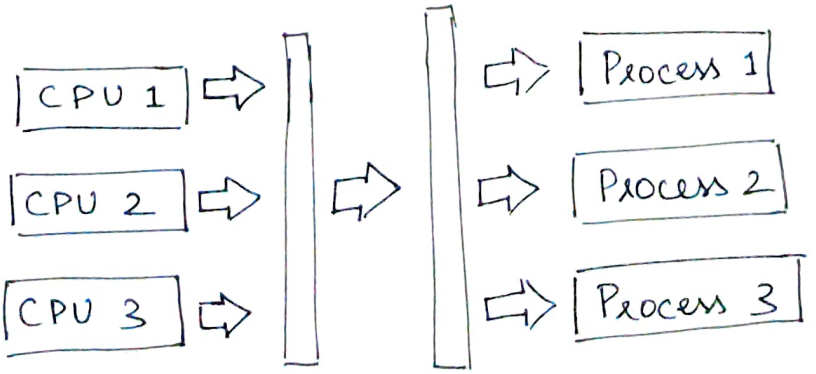
Attached in the file alongside.

Review Question :

Write a suitable example for symmetric and asymmetric multicore processor.

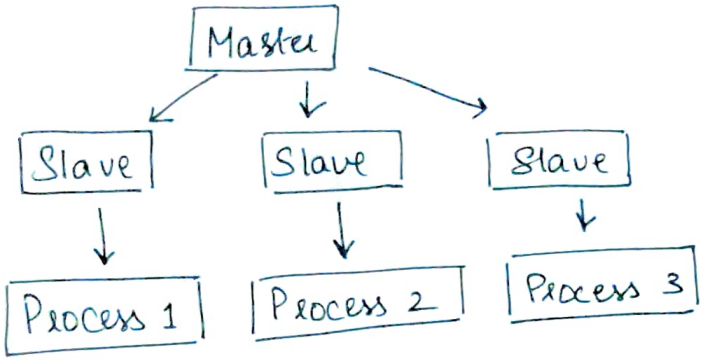
→ Symmetric

- It is one in which all the processor run the tasks in the operating system
- It has no master slave relationship.
- All processors communicate using the shared memory.
- Uses proper load Balancers.



Asymmetric

- It uses master slave relationship among the processors.
- One master ~~slave~~ ~~ed~~ processor controls the other slave processors.
- Master allots processes to all slaves.



Eg: Symmetric

~~A single processor~~

In case of calculating Fibonacci, a ~~sin~~ single CPU calculates the fibonacci upto a specific number & sums it aswell.

Asymmetric:

A single thread generates the fibonacci upto a specific number & the other threads sum it up. All this is managed by the master.

Remarks :

MPI_Send: It is used to perform a blocking send.

Parameters: int MPI_Send (const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

buf: initial address of send buffer.

count: no. of elements in send buffer.

datatype: datatype of each send buffer

dest: rank of destination

tag: message tag

comm: Communicator.

MPI_Receive: It is used to perform blocking receive for a message.

Parameters: MPI_Recv (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

count: max. no. of elements in receive buffer.

datatype: datatype of each receive buffer.

source: rank of source

tag: message tag

comm: communicator.