CS3211 Parallel and Concurrent Programming – Week 11 tutorial

Sample Exercises:
[Please conduct these as an interactive discussion, rather than an evaluation. Please also make it clear to the students that they are not being evaluated for their performance in these exercises, so that they are not afraid to make mistakes while answering.]

MPI usage instructions - Posted in IVLE see Lesson Plan – week 10.

```
MPI program running over Ethernet (MPICH)
[user@access0]$ /opt/mpich/bin/mpicc -c cpi.c]
[user@access0]$ /opt/mpich/bin/mpicc -o cpi cpi.o]
```

For MPICH, create a machine file that looks like this:

```
# cat mynodes
access0
access1
access2
access3
access4
access5
access6
access7
access8
access9
```

```
Run binary MPI program (MPICH)
[user@access0]$ /opt/mpich/bin/mpirun -machinefile mynodes -np 8 /home/user/cpi
```

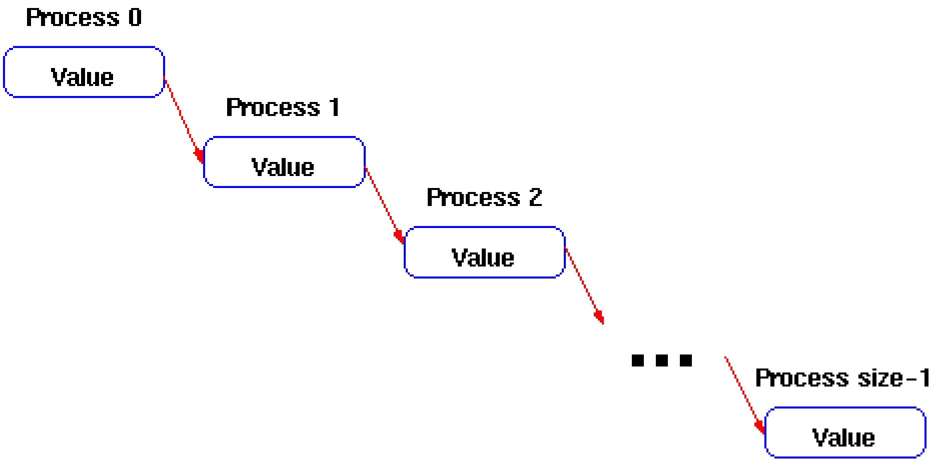
1. Hello world program – discussed in class. [Practice the above steps]

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

int main(int argc,  char *argv[])
{
int rank, size;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
printf("Hello world from processor %d of %d\n", rank, size);
printf("Hello world\n");
MPI_Finalize();
return 0;
}
```

2. Write a program that takes data from process zero and sends it to all of the other processes by sending it in sequence. That is, process i should receive the data and send it to process i+1, until the last process is reached.



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

int main( argc, argv )
int argc;
char **argv;
{
    int rank, value, size;
    MPI_Status status;

    MPI_Init( &argc, &argv );

    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    do {
        if (rank == 0) {
            scanf( "%d", &value );
            MPI_Send( &value, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD );
        }
        else {
            MPI_Recv( &value, 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD,
                      &status );
            if (rank < size - 1)
                MPI_Send( &value, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD );
        }
        printf( "Process %d got %d\n", rank, value );
       fflush(stdout);
    } while (value >= 0);

    MPI_Finalize( );
    return 0;
}
```

Sample output

```
% mpicc -o ring ring.c
% mpirun -np 4 ring
   -    Enter numbers repeatedly until a negative number is entered.

%
```

3. Write a program to test how fair the message passing implementation is. To do this, have all processes except process 0 send 100 messages to process 0. Have process 0 print out the messages as it receives them, using MPI_ANY_SOURCE and MPI_ANY_TAG in MPI_Recv. Is the MPI implementation fair? How will you judge this?

Answer:

```
#include "mpi.h"
#include <stdio.h>
int main(argc, argv)
int argc;
char **argv;
{
    int rank, size, i, buf[1];
    MPI_Status status;

    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    if (rank == 0) {
        for (i=0; i<100*(size-1); i++) {
            MPI_Recv( buf, 1, MPI_INT, MPI_ANY_SOURCE,
                      MPI_ANY_TAG, MPI_COMM_WORLD, &status );
            printf( "Msg from %d with tag %d\n",
                    status.MPI_SOURCE, status.MPI_TAG );
        }
    }
    else {
        for (i=0; i<100; i++)
            MPI_Send( buf, 1, MPI_INT, 0, i, MPI_COMM_WORLD );
    }
    MPI_Finalize();
    return 0;
}
```

Comments

MPI makes no guarantees about fairness in the handling of communication.

Suppose a send operation is executed. It is possible that the destination process repeatedly posts a receive operation that matches this send (via tag, source), and yet the message is never received. This could be because it is repeatedly overtaken by other messages from other sources.