University of Wolverhampton
School of Mathematics and Computer Science

5CS022 Distribute and Cloud Systems Programming Week 2 Workshop

Overview

The aim of this workshop is to build on the previous workshop with MPI and to carry out an assessed task. You can carry out this workshop either the university servers: thinlinc.wlv.ac.uk or your own Linux system (if you prefer to use Putty to log in instead of the Thinlinc client, connect to the **server tl-01.wlv.ac.uk** instead).

Tasks

1. The following C program sums up all the values in array "data" and displays the sum total.:

```
#include <stdio.h>

#define NUMDATA 10000
int data[NUMDATA];

void LoadData(int data[])
{
  for(int i = 0; i < NUMDATA; i++){
    data[i] = 1;
  }
}

int AddUp(int data[], int count)
{
  int sum = 0;
  for(int i = 0; i < count; i++){
    sum += data[i];
  }
  return sum;
}

int main(void) {
  int sum;

  LoadData(data);
  sum = AddUp(data, NUMDATA);
  printf("The total sum of data is %d\n", sum);
  return 0;
}
```

   Convert it to MPI to run with any number of nodes including just one.

2. Write an MPI program called pingpong.c to run with exactly 2 process. Process rank 0 is to send an integer variable call "ball" initialised with the value zero to Process rank 1. Process rank 1 will add 1 to the ball and send it back. This will repeat until the ball has a value of 10 in Process rank 0.

3. Write a "Pass-the-parcel" MPI program that will run with 3 or more nodes, such that Process rank 0 will send an integer variable call "parcel" initialised with 1, to Process rank 1 which will add 1 to the parcel and then send it to Process rank 2, and so on until the highest rank process will send it back to Process rank 0, at which point the parcel variable should contain the value of the number of nodes there are.

4. The following program has all the processes with rank greater than 0 send random amounts of data to Process rank 0 :

```
#include <mpi.h>

#include <stdio.h>
#include <stdlib.h>

#define NUMDATA 1000

int main(int argc, char **argv)
{
  int size;
  int rank;
  int tag=0;
  int count;
  MPI_Status status;
  int data[NUMDATA];

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  if (rank == 0){
    for(int i = 0; i < size - 1; i++) {
   MPI_Recv(data, NUMDATA, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
              MPI_COMM_WORLD, &status);
      MPI_Get_count(&status, MPI_INT, &count);
      printf("Node ID: %d; tag: %d; MPI_Get_count: %d; \n",
             status.MPI_SOURCE, status.MPI_TAG, count);
    }
  }
  else {
    MPI_Send(data, rand()%100, MPI_INT, 0, tag, MPI_COMM_WORLD);
  }
  MPI_Finalize();
  return 0;
}
```

Modify it so that it doesn't use a fixed size data buffer but uses "malloc()" to allocate the correct amount of memory to use every time. Note this means that Process rank 0 will need to know what the amount of data it is expecting before it receives it.

Assessed Workshop Task

5. The file "WarAndPeace.txt" on Canvas contains the entire text of the book "War and Peace" by Leo Tolstoy. Write an MPI program to count the number of times each letter of the alphabet occurs in the book. Count both the upper case and the lowercase as the same. Ignore any letter with accents such as " é " and so on.

   Your MPI program should work with any number of processes from 1 to 100 process. Only Process rank 0 (zero) should read in the file and send the **appropriate** chunk of file to each other process. The other processes should not read in the file.

   You should submit this program as "workshoptask1.c" as part of your final portfolio submission. You can also upload it to the formative submission point for formative feedback.