CS3800 Program #2

Due by midnight Sunday October 25, 2015

Implement a solution to the Dining Philosophers problem (as explained in Chapter 6.6) using message passing through MPI. Your solution must utilize a Fork Ombudsman (or Fork Master) who monitors and authorizes the use of the shared resources (forks). The Fork Master must ensure that hungry philosophers eventually get to eat and that two philosophers cannot use the same fork at the same time.

This can be accomplished with only needing the standard MPI Send and Recv commands. These are the “blocking” point-to-point message passing commands outlined in the online MPI tutorials, but note that “blocking send” in this case only requires that the message successfully be processed into the send queue (so it will seem to immediately proceed to execute the next command – but the message will not be “lost” in that when an appropriate Recv is called, that message will be available).

You are responsible for determining implementation details (such as if messages indicate single forks are being “passed” or if a message indicates two forks are “assigned”, how to handle a case when a philosopher is hungry but the forks aren’t immediately available, avoiding deadlock, etc.)

An example of the bare bones structure along with a sending and receiving check-in is included below (this can replace the program.cpp file in the original posting).

Drop-in help sessions are scheduled for Monday and Wednesday evenings from ~4pm – 6pm in EE 107

#include <cstdlib>

#include <iostream>

#include <cerrno>

#include <unistd.h>

using namespace std;

#include "mpi.h"

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

{

int id; //my MPI ID

int p; //total MPI processes

MPI::Status status;

int tag = 1;

// Initialize MPI.

MPI::Init ( argc, argv );

// Get the number of processes.

p = MPI::COMM\_WORLD.Get\_size ( );

// Determine the rank of this process.

id = MPI::COMM\_WORLD.Get\_rank ( );

//Safety check - need at least 2 philosophers to make sense

if (p < 3) {

MPI::Finalize ( );

std::cerr << "Need at least 2 philosophers! Try again" << std::endl;

return 1; //non-normal exit

}

srand(id + time(NULL)); //ensure different seeds...

// Setup Fork Master (Ombudsman) and Philosophers

if ( id == 0 ) //Master

{

int msgIn; //messages are integers

//let the philosophers check in

for (int i = 1; i < p; i++) {

MPI::COMM\_WORLD.Recv ( &msgIn, 1, MPI::INT, MPI::ANY\_SOURCE, tag, status );

std::cout << "Receiving message " << msgIn << " from Philosopher ";

std::cout << status.Get\_source() << std::endl;

}

}

else //I'm a philosopher

{

//pick a number between 0 and the number of processes (one more than # of philosophers)

int msgOut = rand() % p;

std::cout << "This is Philosopher " << id << " sending message of " << msgOut << std::endl;

//check in with master - send a random number (node 0 = master)

MPI::COMM\_WORLD.Send ( &msgOut, 1, MPI::INT, 0, tag );

}

// Terminate MPI.

MPI::Finalize ( );

return 0;

}