CSCI 624 DISTRIBUTED SYSTEMS - PROBLEMS

All programs should be implemented using either MPI or PC. Program deliverables are code, screenshots of tests and input/output data. References to text are to “Scientific Parallel Programming, Scott, Clark and Bagheri.

1. Hello world (program)

Write a program in PC or MPI that writes "Hello world from P" from each processor, where P is the number of the process, counting from 0. Modify the code so that it writes these messages in order by processor number (hint: look at baton passing algorithms).

2. Parallel prefix (program)

Implement and test a parallel prefix algorithm (see chapter 6.5 of text). Use MPI or PC for your implementation.

3. Baton passing (program)

Exercise 7.3 from text.

4. Leader election (theory)

Consider a set of \( P \) processes, each of which has a randomly generated (integer) number \( X_p \). We will assume \( X_p \neq X_q \) if \( p \neq q \) (we can dispense with this assumption by using process i.d. as a tiebreaker, but this would make the algorithm more complicated to understand how many messages are required, and what dependencies there are that force one message to appear before another message.

5. Distributed sort (theory)

Design a distributed sort algorithm based on a leader election algorithm. The algorithm should be suitable for implementation on an SPMD system in which you initially know the number of processes \( N \) and process identifiers are 0 to \( N - 1 \). Consider two possibilities:

1. You will use \( N \) processes to sort \( N \) integers in the range 0…\( M \), \( M \gg N \). Initial data consists of one integer at each process. Final data is a sorted list at some process.
2. You will use \( N \) processes to sort \( K \) integers in the range 0…\( M \), \( M \gg K \gg N \). Initial data is a disordered list of \( \frac{K}{N} \) integers at each process. Final data is a sorted list of \( K \) integers at some process.

Implement the algorithm above. Let \( 8 \leq N \leq 16 \). Use MPI, PC or any other system you can get to work.
6. Broadcast: (theory)

Write a routine to efficiently perform a broadcast on a ring of nodes numbered 0...N, from a root node R in the ring. (We will consider a broadcast perfectly efficient if no message has to wait because sender or receiver is busy handling other messages, and if the number of steps required is less than N).

Assume:
- The ring is bidirectional
- A node can send or receive a message on each link at each step
- Assume your only communication function is \( M(\text{source}, \text{destination}, \text{variable}) \), which sends the contents of \( \text{variable} \) from \( \text{source} \), receives it at \( \text{destination} \), and does nothing if executed on a node that is neither \( \text{source} \) nor \( \text{destination} \).
- Assume the underlying communication system buffers if necessary.
- You have variables \( ME = (\text{number of local node}) \), and \( NODES = (\text{total number of nodes}) \).

Estimate the number of steps required and the total number of messages sent.

7. Synchronization (theory)

A barrier is a synchronization statement that forces every one of a group of processes that executes the statement to wait until they know that all other processes in the group have reached the barrier. Barrier statements are sometimes, but not always, provided in hardware.

Describe an algorithm that uses point to point messages (such as the \( M \) function in problem 2) to implement a barrier. How does your algorithm scale with number of processes \( N \)? (That is, how does the time it takes increase as a function of \( N \)). Justify your answer.

8. Deadlock (theory)

A standard requirement for deadlock to occur is for there to be a cycle of dependencies. Consider dependencies of the following kinds:

A message sent from process \( p \) to process \( q \) creates a dependence \( q \leftarrow p \) (that is, \( q \) depends on \( p \); \( q \) must happen before \( p \).)

A statement \( p_1 \) that appears textually before a statement \( p_2 \), both at process \( p \), creates a dependence \( p_1 \leftarrow p_2 \) in the sense that \( p_1 \) must happen before \( p_2 \).

Therefore we get deadlock if, for example, we have \( p_1 \leftarrow p_2, q_1 \leftarrow q_2, p_2 \leftarrow q_1 \) and \( q_2 \leftarrow p_1 \) (i.e., the statements \( p_i \) and \( q_i \) appear in index order in their respective processes, and we have send from \( q_2 \) to receive at \( p_1 \) and send from \( p_2 \) to receive at \( q_1 \).

Suppose we have \( N \) communicating concurrent processes, each of which has \( M \) send and/or receive statements to or from the other processes. What is the complexity (in terms of \( M \) and \( N \) of determining if there is a deadlock cycle?"