7 Concurrent and Distributed Systems (RNW)

(*a*) Deadlock is a classic problem in concurrent systems.

(*i*) What are the four necessary conditions for deadlock? [4 marks]

(*ii*) Deadlock is often explained using the Dining Philosopher's Problem. In this pseudo-code, each fork is represented by a lock:

```
Lock forks[] = new Lock[5];

// Code for each philosopher (i)
while (true) {
    think();
    lock(fork[i]);
    lock(fork[(i + 1) % 5]);
    eat();
    unlock(fork[i]);
    unlock(fork[(i + 1) % 5]);
}
```

Partial ordering is a common deadlock prevention scheme. Describe modifications to the above code, changing only array indices, such that philosophers can be fed not only safely, but also deadlock-free, using a partial order. [4 marks]

(*b*) Priority inversion can occur when threads of differing priorities synchronise on access to common resources – threads of greater priority may end up waiting on threads of lesser priority, leading to undesirable realtime properties.

(*i*) Describe how this problem can be solved for mutexes using priority inheritance. [2 marks]

(*ii*) Describe how priority inheritance would need to be modified to handle reader-writer locks. [2 marks]

(*iii*) Priority inversion can also arise between two threads involved in process synchronisation – for example, when one thread uses a semaphore to signal completion of work. Why might implementing priority inversion be more difficult with process synchronisation than with mutual exclusion? [4 marks]

(*iv*) What could we do to solve the problem in (*b*)(*iii*)? [4 marks]