8 Concurrent and Distributed Systems (RNW)

(a) Monitors are a programming primitive linking data with two synchronization types: mutual exclusion and condition synchronisation. Which is provided implicitly; which is provided explicitly? [1 mark]

(b) Describe two ways in which Monitors and Conditional Critical Regions differ. [2 marks]

(c) The object-oriented programming style encouraged by Monitors has many benefits as the number of data types and locks increases in the system.

(i) Placing all data in a single monitor may improve program correctness. Explain why this might have undesirable performance effects. [1 mark]

(ii) One problem that can arise when using multiple locks is deadlock, which can be prevented by imposing a partial order on locks. Describe the implications this has for code structure when using Monitors. [2 marks]

(iii) Explain why Java’s Monitor feature does not necessarily impose this code structure. [2 marks]

(d) Condition variables allow condition satisfaction to be signalled between threads. Explain the difference between Hoare’s signal-and-wait and Mesa’s signal-and-continue in terms of mutual exclusion and scheduling. [4 marks]

(e) Consider the (incorrect) pseudocode on the next page:

(i) Describe and justify minimal modifications to this code, referencing line numbers, in order to make it correct in the presence of Hoare signal-and-wait semantics. [4 marks]

(ii) Describe and justify minimal modifications to this code, referencing line numbers, in order to make it correct in the presence of Mesa signal-and-continue semantics. [4 marks]
1: monitor ProducerConsumer {
2:   int in, out, buf[N];
3:   condition notfull, notempty;
4:   
5:   procedure produce(item) {
6:     if ((in-out) == N)
7:       wait(notfull);
8:     buf[in % N] = item;
9:     if ((in-out) == 0)
10:       signal(notempty);
11:     in = in + 1;
12:   }
13:   
14:   procedure int consume() {
15:     if ((in-out) == 0)
16:       wait(notempty);
17:     item = buf[out % N];
18:     if ((in-out) == N)
19:       signal(notfull);
20:     out = out + 1;
21:   }
22:   
23:   /* init */ { in = out = 0; }
24: }