UNIVERSITY COMPUTER SCIENCE • EXERCISE SHEET

7.3 Synchronisation

Name: _____ Class: _____ Date: _____ **Total: 24 marks**

KEY WORDS mutual exclusion • lock

Objective

Trace races deterministically and break deadlocks by construction.

1 Key knowledge and worked examples

■ **The lost update**

Read-then-write on `counter = 5` by two threads: the interleaving R1 R2 W1 W2 stores 6 twice ✓. With four increments each, every final value from 1 to 8 is reachable ✓.

■ **Worked example: the four conditions**

Deadlock needs mutual exclusion, hold-and-wait, no preemption, and circular wait – break any one; the lock order is the cheap break ✓.

2 Practice

2.1 Two threads run `x = x + 1` on `x = 10`. Give the outcome of the interleaving R1 R2 W2 W1. [3]

2.2 Name the three properties a correct critical-section solution must satisfy. [3]

2.3 Why does a mutex queue beat a spinlock on a single core? [3]

3 Exam-style questions

3.1 Two threads each execute `count = count + 1` three times on `count = 0`, with per-increment registers.

State the maximum final value and the interleaving that achieves it. State the minimum final value and explain the interleaving that achieves it. Give the number of distinct final values reachable, and the one-line fix that pins the result. [8]

3.2 Thread A locks L1 then wants L2; thread B locks L2 then wants L1. Each lock acquisition is 1 ms; each hold lasts 5 ms.

Explain why this deadlocks, naming the condition cycle. State the one-line ordering rule that prevents it, and trace the same run under the rule. Give one other condition that could be broken instead, with its mechanism. [7]

Solutions

2.1

- Both reads see 10; the last write is 11 – final 11 ✓.

2.2

- Mutual exclusion, progress, bounded waiting ✓.

2.3

- The waiter sleeps in the OS queue instead of spinning – a spinning thread on the only core blocks the lock holder itself ✓.

3.1

- Maximum 6: all six increments land in order, each read sees the prior write ✓.
- Minimum 1: all six reads happen before any write (all read 0), then the six writes each store 1 ✓.
- Distinct final values: 6 – every value 1 to 6 is reachable ✓.
- Fix: wrap the increment in a lock (mutual exclusion) – the result is always 6 ✓.

3.2

- A holds L1, waits for L2; B holds L2, waits for L1: each waits forever for the other – circular wait across two holders ✓.
- Rule: always acquire locks in one global order (L1 before L2) ✓.
- Under the rule: B, holding nothing yet, takes L1 first – both threads queue on L1 and proceed one at a time ✓.
- Alternative: break hold-and-wait – request all locks at once (acquire-or-release) so no thread holds while waiting ✓.