Solutions

Marking: **M1** a correct method, **A1** the correct answer, **B1** an independent mark.

Each answer shows the steps, then the **result**.

2.1

3 enter

2 block

2.2

mutual exclusion; hold and wait

no pre-emption; circular wait

a fixed order breaks circular wait

2.3

yes

T1 → B → T2 → C → T3 → A → T1 is a cycle, and each resource has one copy

2.4

spinning wastes processor time while it waits

blocking costs two context switches

spin when the wait is shorter than a context switch, on a multicore machine

3.1

(a)

```
empty = Semaphore(N); full = Semaphore(0); mutex = Lock()

def produce(item):
    empty.wait()
    with mutex:
        buffer.put(item)
    full.signal()

def consume():
    full.wait()
    with mutex:
        item = buffer.get()
    empty.signal()
    return item
```

`empty` starts at N and `full` at 0; producer waits on `empty`, signals `full`; consumer the reverse; the mutex only around the buffer itself

(b) after 2 items: `empty = 0`, `full = 2`; the third `empty.wait()` blocks until the consumer takes an item

(c) the semaphores count slots, but two threads could still change the buffer's indexes at the same moment

3.2

(a) all five pick up their left fork at once; each waits for a right fork that a neighbour holds: a cycle

(b) number the forks 0 to 4; each philosopher picks up the lower-numbered of their two forks first

(c) the last philosopher now reaches first for the fork the first one wants, so a full cycle cannot form

3.3

(a) the first locks `x`, the second locks `y`; each then waits for the lock the other holds, for ever

(b) lock the two accounts in a fixed order, e.g. by account number: `first, second = sorted([a, b], key=lambda c: c.id)` then `with first.lock: with second.lock: ...`

(c) only one transfer at a time, each 1 ms: at most **1000** a second

(d) per-account locks taken in id order: transfers between different accounts run in parallel, so 50 000 a second is possible, while a global lock caps it at 1000

exercises.lol exercises.com.cn