Solutions

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

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

2.1

waits for a key: running → blocked

time slice ends: running → ready

disk read finishes: blocked → ready

2.2

each `fork` doubles the number: 2^3

$= \mathbf{8}$

2.3

shared: heap, global variables, open files

own: stack, program counter, registers

2.4

zombie processes

each keeps an entry in the process table, which can fill up so no new process can start

the parent calls `wait` (e.g. when a child-ended signal arrives)

3.1

(a) $0.1/(10 + 0.1) \approx \mathbf{0.99}\%$

(b) $0.1/(1 + 0.1) \approx \mathbf{9.1}\%$

(c) each process gets a turn sooner, so interactive programs respond quickly

(d) any four: registers, program counter, stack pointer, process state, memory map (page table pointer), open files

3.2

(a) processes: 2000×1 ms $= \mathbf{2}$ **s** a second, so two whole cores; threads: $2000 \times 0.05 = \mathbf{0.1}$ **s** a second

(b) a crash in one thread ends the whole server; threads share memory, so data races are possible

(c) it makes the threads once and reuses them, so no creation cost per request

3.3

(a) 3 processes, joined by 2 pipes: one program's output becomes the next one's input

(b) `fork` a child, connect its input or output to the pipe (`dup2`), then `exec` the command; the shell `wait`s for them

exercises.lol exercises.com.cn

(c) when the pipe's buffer is full, `cat`'s write blocks until `grep` reads; this flow control stops a fast writer from using unlimited memory

(d) for: a crashing or hacked tab cannot harm the others, so it is safer and more reliable; against: each process needs its own memory, so many tabs use far more RAM; browsers accept this for security