

Solutions

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

2.1

- any two of: CPU (time), memory/RAM, disk/storage, I/O devices

2.2

- gives each process a **fixed time slice** of CPU in turn, then moves it to the **back of the queue**

2.3

- New, Ready, Running, Blocked, Terminated**

2.4

- saving the state of one process (to its PCB) and **restoring** another's, so the CPU can switch between them

2.5

- any two:
 - first come first served (FCFS); shortest job first (SJF); shortest remaining time (SRT); round robin

2.6

- any one:
 - paging uses **fixed-size** pages; segmentation uses **variable-size** segments
 - pages are **physical** divisions; segments are **logical** units of the program

2.7

- A then B: A waits 0, B waits 5; average = $\frac{0+5}{2} = 2.5$ ms
- B then A: B waits 0, A waits 1; average = $\frac{0+1}{2} = 0.5$ ms

3.1

- New → Ready (admit)
- Ready ⇌ Running (dispatch / time-out)
- Running → Blocked (I/O request) and Blocked → Ready (I/O done)
- Running → Terminated (exit)

3.2

- the process moves from **Running** to **Blocked**
- the scheduler then **dispatches another ready process** to the CPU while it waits

3.3

- each process has a **virtual address space** split into **pages**
- only the pages in use are kept in RAM **frames**, the rest stay in the **swap file** on disk

- on a **page fault** the OS swaps a page in (evicting one if needed), so the program can be larger than physical RAM

3.4

- too little RAM for the active pages, so the OS spends most of its time **swapping pages in and out** instead of doing useful work

3.5

- SJF runs the ready process with the **shortest expected run time** next
- benefit: the **lowest average waiting time** – short jobs are not stuck behind long ones

3.6

- any two:
 - a **user interface** (GUI) lets the user work without touching the hardware
 - device drivers** let programs use devices without knowing how they work
 - system calls / an API** carry out hardware tasks for programs
 - file management** shows named files and folders, not physical storage locations

3.7

- each process gets a **fixed time slice** of the processor, **in turn**
- when its slice ends, the process is **pre-empted** and goes to the **back of the ready queue**
- benefit: it is **fair** – every process gets processor time / no process **starves** / good **response times**

3.8

(a) any three:

- memory is divided into **variable-size** blocks called **segments**
- each segment is a **logical unit** of the program, e.g. a procedure or a data block
- a **segment table** stores each segment's **start address** and **length**
- an address is given as a **segment number** and an **offset** within the segment

(b)

- pages are **swapped** between RAM and disk **so often**
- that the processor spends more time **transferring pages** than running the program, so performance **collapses**

3.9

(a)

- FCFS waits: W 0, X 8, Y 12, Z 21
- average = $\frac{0+8+12+21}{4} = 10.25$ ms

(b)

- SJF order X, Z, W, Y; waits: X 0, Z 4, W 9, Y 17
- average = $\frac{0 + 4 + 9 + 17}{4} = \mathbf{7.5\ ms}$

(c) any three:

- the **timer** sends an **interrupt** when the slice ends
- the kernel **saves the state** (registers, PC) of the running process in its PCB
- it runs the **ISR** for the timer interrupt, found from the **interrupt vector table**
- the **scheduler** moves the process to the ready queue and **dispatches** the next one

(d)

- SJF needs each burst time **in advance**, which is usually not known
- round robin is **fair**: long processes are **not starved**, and every process **responds** quickly