ANSWERS

A10.5 Scheduling algorithms

Operating systems, software and translators · A level · OCR H446 1.2.1, AQA 7517 4.6.1.4, Eduqas A500QS 2.6 · about 45 min

BugBotLab

What this lesson is about

First come first served, round robin, shortest job first, shortest remaining time and multi-level feedback queues, compared on one set of processes.

- What a scheduler is trying to do
- One set of processes, five algorithms
- Comparing them

Questions

5 marks in all

1. Which scheduling algorithm is pre-emptive and chooses the process with the least time left to run? [1 mark]

- ☐ **A** Shortest remaining time
- ☐ **B** Shortest job first
- ☐ **C** First come first served
- ☐ **D** Round robin

> **Answer:** A. SJF chooses by burst time too, but only when the processor becomes free.

2. Processes P (arrives 0, burst 6), Q (arrives 1, burst 3) and R (arrives 2, burst 1) are scheduled first come first served. What is the average waiting time? [1 mark]

> **Answer:** 4 (accept within 0.01). P waits 0, Q waits 6 - 1 = 5, R waits 9 - 2 = 7: a total of 12, so the average is 12 / 3 = 4.

3. A long process never gets the processor because short processes keep arriving. What is this called, and which algorithm can cause it? [1 mark]

- ☐ **A** Starvation; shortest job first
- ☐ **B** Thrashing; round robin
- ☐ **C** Deadlock; first come first served
- ☐ **D** Starvation; round robin

> **Answer:** A. Round robin and FCFS always reach every process eventually; SJF and SRT can keep pushing a long job back.

4. Which statements about multi-level feedback queues are true? [1 mark]

Tick every answer that is true.

- ☐ **A** A process that uses its whole time slice moves to a lower-priority queue
- ☐ **B** It does not need to know burst times in advance
- ☐ **C** A process that has waited a long time can be moved up to prevent starvation
- ☐ **D** Every process stays in the queue it started in

> **Answer:** A, B, C. MLFQ learns how each process behaves and moves it between queues.

5. What does this program print? [1 mark]

```
queue = [('A', 5), ('B', 2), ('C', 4)]
slice = 3
time = 0
while queue:
    name, left = queue.pop(0)
    run = min(slice, left)
    time = time + run
    if left > run:
        queue.append((name, left - run))
    else:
        print(name, 'finishes at', time)
```

Answer:

```
B finishes at 5
A finishes at 10
C finishes at 11
```

B finishes in its first slice at 5; C and A each need a second turn.

The task: waiting times

Write `sjf(processes)` and `srt(processes)`. `processes` is a list of tuples `(name, arrival, burst)`: `name` is a string, and `arrival` and `burst` are whole numbers of time units (arrival 0 or more, burst 1 or more). Each function returns the **average waiting time** as a float, where a process's waiting time is its finish time minus its arrival time minus its burst time. - `sjf`: non-pre-emptive. Whenever the processor is free, run the arrived process with the smallest burst time to completion. If nothing has arrived, move the clock on to the next arrival. - `srt`: pre-emptive. Move the clock one unit at a time; each unit, run the arrived, unfinished process with the least time left. - In both, break a tie by choosing the process that arrived first. Keep the `fcfs` line and print all three, to 2 decimal places, one per line: `FCFS average wait: <x>`, `SJF average wait: <x>` and `SRT average wait: <x>`. The robot stays still.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

processes = [("A", 0, 8), ("B", 1, 4), ("C", 2, 9), ("D", 3, 5)]

def fcfs(processes):
    time = 0
    total_wait = 0
    for name, arrival, burst in sorted(processes, key=lambda p: p[1]):
        time = max(time, arrival)
        total_wait = total_wait + time - arrival
        time = time + burst
    return total_wait / len(processes)

print(f"FCFS average wait: {fcfs(processes):.2f}")
```

The hint students can ask for: For SJF, whenever the processor is free, choose only from the processes that have already arrived, and run the chosen one to the end. For SRT, step the clock one unit at a time and choose again every unit, using the time each process has left. A process's waiting time is its finish time minus its arrival time minus its burst time.

A solution

```
from bugbot import *
connect()

processes = [("A", 0, 8), ("B", 1, 4), ("C", 2, 9), ("D", 3, 5)]

def fcfs(processes):
    time = 0
    total_wait = 0
    for name, arrival, burst in sorted(processes, key=lambda p: p[1]):
        time = max(time, arrival)
        total_wait = total_wait + time - arrival
        time = time + burst
    return total_wait / len(processes)

def sjf(processes):
    waiting = list(processes)
    time = 0
    total_wait = 0
    while waiting:
        ready = [p for p in waiting if p[1] <= time]
        if not ready:
```

```
            time = min(p[1] for p in waiting)
            continue
        job = min(ready, key=lambda p: (p[2], p[1]))
        waiting.remove(job)
        total_wait = total_wait + time - job[1]
        time = time + job[2]
    return total_wait / len(processes)

def srt(processes):
    left = {name: burst for name, arrival, burst in processes}
    finish = {}
    time = 0
    while left:
        ready = [p for p in processes if p[0] in left and p[1] <= time]
        if ready:
            job = min(ready, key=lambda p: (left[p[0]], p[1]))
            left[job[0]] = left[job[0]] - 1
            if left[job[0]] == 0:
                del left[job[0]]
                finish[job[0]] = time + 1
        time = time + 1
    total_wait = sum(finish[name] - arrival - burst for name, arrival, burst in processes)
    return total_wait / len(processes)

print(f"FCFS average wait: {fcfs(processes):.2f}")
print(f"SJF average wait: {sjf(processes):.2f}")
print(f"SRT average wait: {srt(processes):.2f}")
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.