ANSWERS

A2.8 Thinking concurrently

Recursion and computational thinking · A level · OCR H446 2.1.5 · about 25 min

BugBotLab

What this lesson is about

Concurrent and parallel processing, what can happen at once, benefits and trade-offs, and pipelining.

- Concurrent and parallel
- Which parts can happen at the same time?
- Benefits and trade-offs
- Pipelining

Questions

5 marks in all

1. What is the difference between concurrent and parallel processing? [1 mark]

☐ **A** Concurrent tasks are in progress over the same period, possibly taking turns on one core; parallel tasks run at the same instant on separate cores

☐ **B** There is no difference

☐ **C** Parallel tasks take turns; concurrent tasks run at the same instant

☐ **D** Concurrent processing needs a network

> **Answer:** A. Parallel processing is one way to achieve concurrency; time-slicing on one core is another.

2. Which are trade-offs of concurrent processing? [1 mark]

Tick every answer that is true.

☐ **A** Tasks sharing data can give timing-dependent results (race conditions)

☐ **B** Tasks can deadlock, each waiting for a resource the other holds

☐ **C** Coordinating tasks has an overhead

☐ **D** It always makes every task finish sooner

> **Answer:** A, B, C. Only independent parts gain; dependent parts still wait.

3. Jobs: A takes 20 min, B takes 30 min, C takes 10 min and needs A and B to finish first. With unlimited workers, how many minutes until C finishes? [1 mark]

> **Answer:** 40. A and B run at the same time; C starts when the later one, B, finishes at 30 minutes, and ends at 40.

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

```
items, stages, minutes = 10, 4, 3
one_at_a_time = items * stages * minutes
pipelined = stages * minutes + (items - 1) * minutes
print(one_at_a_time, pipelined)
```

> **Answer:**
>
> ```
> 120 39
> ```
>
> One at a time, each item takes 12 minutes. Pipelined, the first item takes 12 and then one more finishes every 3 minutes.

5. 10% of a program must run in sequence and the rest can be split between processors. What is the most it can ever be sped up, with unlimited processors? [1 mark]

- ☐ **A** 10 times
- ☐ **B** 100 times
- ☐ **C** Unlimited
- ☐ **D** 2 times

> **Answer:** A. Even if the other 90% took no time at all, the sequential 10% would remain, so the speed-up is at most 1 / 0.1 = 10.

The task: getting ready at the same time

The jobs before a match are held in a dictionary. Each job maps to a tuple `(minutes, needs)`: how long it takes, and a list of the jobs that must finish before it can start. - `charge` 40 min, needs nothing; `flash` 10 min, needs nothing; `calibrate` 5 min, needs `flash`; - `print` 30 min, needs nothing; `assemble` 15 min, needs `print` and `flash`; - `test` 10 min, needs `assemble`, `charge` and `calibrate`. With as many workers as you like, a job starts the moment the last job it needs has finished. Write a **recursive** function `finish(job)` that takes a job name and returns the earliest time, in minutes from the start, that the job can be finished. Then print: 1. `one worker: <n> min`, the time with one worker doing the jobs one at a time; 2. `test finishes at <n> min`, using `finish("test")`; 3. `unlimited workers: <n> min`, the latest finish of any job; 4. `speed-up: <s>`, one worker's time divided by unlimited workers' time, to 2 decimal places.

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

jobs = {
    "charge": (40, []),
    "flash": (10, []),
    "calibrate": (5, ["flash"]),
    "print": (30, []),
    "assemble": (15, ["print", "flash"]),
    "test": (10, ["assemble", "charge", "calibrate"]),
}

def finish(job):
    minutes, needs = jobs[job]
    return minutes
```

The hint students can ask for: A job cannot start until every job it needs has finished, so its earliest finish is its own time added to the latest finish among the jobs it needs. A job that needs nothing finishes after its own time. That definition refers to itself. The whole project is done when the last job to finish is done.

A solution

```
from bugbot import *
connect()

jobs = {
    "charge": (40, []),
    "flash": (10, []),
    "calibrate": (5, ["flash"]),
    "print": (30, []),
    "assemble": (15, ["print", "flash"]),
    "test": (10, ["assemble", "charge", "calibrate"]),
}

def finish(job):
    minutes, needs = jobs[job]
    start = 0
    for other in needs:
        start = max(start, finish(other))
    return start + minutes

total = 0
for job in jobs:
```

```
    total = total + jobs[job][0]
print("one worker:", total, "min")
print("test finishes at", finish("test"), "min")
longest = 0
for job in jobs:
    longest = max(longest, finish(job))
print("unlimited workers:", longest, "min")
print(f"speed-up: {total / longest:.2f}")
```

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