ANSWERS

A10.9 Project: a tiny robot operating system

Operating systems, software and translators · A level · OCR H446 1.2.1, AQA 7517 4.6.1.4, Eduqas A500QS 2.6 · about 50 min

What this lesson is about

Share the robot between three tasks with round robin, drive it through device drivers, and stop the patrol with an interrupt.

Questions

5 marks in all

1. In the tiny robot OS, what does the check after every step model? [1 mark]

- ☐ **A** The check for interrupts at the end of each fetch-decode-execute cycle
- ☐ **B** Paging
- ☐ **C** Lexical analysis
- ☐ **D** The BIOS power-on self-test

> **Answer:** A. The OS checks for waiting interrupts between steps, as a processor does between cycles.

2. Why does the project run every step through a driver table instead of calling forward, led and tone directly? [1 mark]

- ☐ **A** The scheduler does not need to know how any device works, and a new device only needs a new driver
- ☐ **B** Driver tables make the robot drive faster
- ☐ **C** Python cannot call robot commands from inside a loop
- ☐ **D** It stops the interrupt from firing

> **Answer:** A. That is the job of device drivers in a real operating system.

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

```
queue = ['patrol', 'lights', 'music']
left = {'patrol': 3, 'lights': 1, 'music': 2}
order = []
while queue:
    name = queue.pop(0)
    left[name] = left[name] - 1
    order.append(name[0])
    if left[name] > 0:
        queue.append(name)
print(''.join(order))
```

> **Answer:**
>
> ```
> plmpmp
> ```
>
> A time slice of one step: each task takes a turn until it has nothing left.

4. BugBot's motor loop must correct the robot's heading within a few milliseconds every time. Which two types of operating system describe FreeRTOS on the robot? [1 mark]

- ☐ **A** Embedded and real-time
- ☐ **B** Distributed and multi-user
- ☐ **C** Multi-user and real-time
- ☐ **D** Distributed and embedded

> **Answer:** A. It runs inside a dedicated device with limited resources, and must meet deadlines.

5. In the project, the patrol task is stopped by the interrupt with one step still to run. Which scheduling idea does this show? [1 mark]

- ☐ **A** Pre-emption: a running task loses the processor before it has finished its slice
- ☐ **B** First come first served
- ☐ **C** Shortest job first
- ☐ **D** Disk thrashing

> **Answer:** A. The handler takes the processor away from the patrol, the way a pre-emptive OS can.

The task: a tiny robot OS

Build the operating system from the brief. The inputs are: - `tasks`, a list of `(name, steps)` tuples in their starting queue order, where `steps` is a list of `(command, value)` pairs; `command` is `"forward"` (value: cm, a positive integer), `"led"` (value: a colour name) or `"tone"` (value: hertz, 100 to 10000); - `TIME_SLICE`, the most steps a task runs before the next task starts (2); - `SAFE_CM`, the interrupt threshold: the obstacle interrupt fires when `distance()` is less than this (25). Write a `drivers` dictionary with a driver for each command: `"forward"` drives forward at speed 50, `"led"` sets the LED colour, and `"tone"` plays the note for 0.2 seconds. Run every step through `drivers`. Print exactly these lines, built from your variables: `<name>: <command> <value>` after each step, `interrupt: obstacle` and `patrol stopped` from the handler (with the 220 Hz, 0.2 second tone between them), and `<name> finished` when a task runs its last step. The wall is 51 cm ahead of the robot.

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

tasks = [
    ("patrol", [("forward", 10), ("forward", 10), ("forward", 10), ("forward", 10)]),
    ("lights", [("led", "blue"), ("led", "yellow"), ("led", "green")]),
    ("music", [("tone", 523), ("tone", 659), ("tone", 784)]),
]
TIME_SLICE = 2
SAFE_CM = 25
```

The hint students can ask for: Build it in layers: first the drivers table, then the round-robin loop with no interrupt, and check the order of the lines. Only then add the check after every step. The handler must stop the patrol whichever task happens to be running.

A solution

```
from bugbot import *
connect()

tasks = [
    ("patrol", [("forward", 10), ("forward", 10), ("forward", 10), ("forward", 10)]),
    ("lights", [("led", "blue"), ("led", "yellow"), ("led", "green")]),
    ("music", [("tone", 523), ("tone", 659), ("tone", 784)]),
]
TIME_SLICE = 2
SAFE_CM = 25

def motor_driver(cm):
    forward(50, distance=cm)

def led_driver(colour):
    led(colour)

def piezo_driver(hz):
    tone(hz, 0.2)

drivers = {"forward": motor_driver, "led": led_driver, "tone": piezo_driver}

jobs = {}
queue = []
for name, steps in tasks:
```

```
    jobs[name] = list(steps)
    queue.append(name)

stopped = []
while queue:
    name = queue.pop(0)
    for turn in range(TIME_SLICE):
        if not jobs[name]:
            break
        command, value = jobs[name].pop(0)
        drivers[command](value)
        print(f"{name}: {command} {value}")
        # the interrupt check at the end of every step
        if "patrol" not in stopped and distance() < SAFE_CM:
            print("interrupt: obstacle")
            tone(220, 0.2)
            jobs["patrol"].clear()
            stopped.append("patrol")
            print("patrol stopped")
    if name in stopped:
        continue
    if jobs[name]:
        queue.append(name)
    else:
        print(f"{name} finished")
```

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