ANSWERS

A10.2 The operating system, BIOS and device drivers

Operating systems, software and translators · A level · OCR H446 1.2.1, AQA 7517 4.6.1.4, Eduqas A500QS 2.6 · about 35 min

What this lesson is about

Hiding the hardware, managing resources, booting from the BIOS, and a driver table for the robot.

- Hiding the hardware
- Managing resources
- Device drivers
- The BIOS and booting

Questions

5 marks in all

1. What does it mean to say the operating system hides the complexity of the hardware? [1 mark]

☐ **A** Programs use simple services such as opening a file, and the OS deals with the details of the actual devices

☐ **B** The operating system stops users seeing inside the computer case

☐ **C** The operating system encrypts the hardware so it cannot be attacked

☐ **D** The operating system removes hardware that is not being used

> **Answer:** A. Programs make system calls; the OS and its drivers turn them into device-specific operations.

2. Why are device drivers needed? [1 mark]

☐ **A** They translate the operating system's general requests into the commands a particular device understands

☐ **B** They move data between RAM and the processor

☐ **C** They start the computer when the power comes on

☐ **D** They schedule processes onto the processor

> **Answer:** A. Each device model has its own commands, so the OS cannot contain code for every one.

3. Put these stages of starting a computer in order. [1 mark]

Number the lines 1 to 4 to put them in the right order.

___ `The BIOS loads the bootstrap loader from the boot device into RAM`

___ `The bootloader loads the operating system kernel into RAM and starts it`

___ `The power-on self-test checks the essential hardware`

___ `The processor starts running the BIOS from non-volatile memory`

Answer:

```
The processor starts running the BIOS from non-volatile memory
The power-on self-test checks the essential hardware
The BIOS loads the bootstrap loader from the boot device into RAM
The bootloader loads the operating system kernel into RAM and starts it
```

RAM is empty at power-on, so the chain starts in non-volatile memory and ends with the OS in RAM.

4. Why must the BIOS be stored in non-volatile memory? [1 mark]

- ☐ **A** RAM is empty when the power comes on, so the start-up instructions must survive with the power off
- ☐ **B** Non-volatile memory is faster than RAM
- ☐ **C** The BIOS is too large to fit in RAM
- ☐ **D** So that users can edit it easily

Answer: A. Volatile memory loses its contents without power; the very first instructions must already be there.

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

```
drivers = {"lamp": lambda v: f"lamp at {v}%", "fan": lambda v: f"fan at {v} rpm"}
for device, value in [("fan", 900), ("lamp", 40), ("door", 1)]:
    if device in drivers:
        print(drivers[device](value))
    else:
        print("no driver:", device)
```

Answer:

```
fan at 900 rpm
lamp at 40%
no driver: door
```

Each request is looked up in the driver table; a device with no driver is reported rather than crashing.

The task: the driver table

Build a tiny operating system layer for the robot. Write three driver functions, each taking one argument `value`: - a driver for `"led"`: `value` is a colour name string; it sets the LED to that colour; - a driver for `"piezo"`: `value` is a frequency in hertz (an integer from 100 to 10000); it plays that note for 0.2 seconds; - a driver for `"motor"`: `value` is a distance in centimetres (a positive integer); it drives forward that far at speed 50. Put them in a dictionary called `drivers`, keyed by the device names `"led"`, `"piezo"` and `"motor"`. Then write the system call `syscall(device, value)`. If `device` is a key in `drivers`, it calls that driver with `value`, prints `ok: <device> <value>`, and returns `True`. Otherwise it prints `error: no driver for <device>`, does nothing else, and returns `False`. Do not test the device name with `==`: the table is the whole point. Finally, call `syscall` for each `(device, value)` pair in `requests`, in order. The robot should end 20 cm ahead with a green LED.

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

requests = [("led", "blue"), ("piezo", 880), ("motor", 20), ("camera", "on"), ("led",
"green")]
```

The hint students can ask for: Store each driver function in a dictionary under its device name. The system call only has to check the name is a key, then call whatever function it finds there with the value.

A solution

```
from bugbot import *
connect()

requests = [("led", "blue"), ("piezo", 880), ("motor", 20), ("camera", "on"), ("led",
"green")]

def led_driver(value):
    led(value)

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

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

drivers = {"led": led_driver, "piezo": piezo_driver, "motor": motor_driver}

def syscall(device, value):
    if device not in drivers:
        print(f"error: no driver for {device}")
        return False
    drivers[device](value)
    print(f"ok: {device} {value}")
    return True

for device, value in requests:
    syscall(device, value)
```

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