ANSWERS

A10.6 Types of operating system and virtual machines

Operating systems, software and translators · A level · OCR H446 1.2.1, AQA 7517 4.6.3.1, Eduqas A500QS 2.7 · about 40 min

What this lesson is about

Multi-tasking, multi-user, real-time, embedded and distributed systems, and a bytecode virtual machine that drives the robot.

- Five types of operating system
- Virtual machines
- A bytecode machine, twice

Questions

5 marks in all

1. Which type of operating system is needed in a car's anti-lock braking system? [1 mark]

- ☐ **A** Real-time
- ☐ **B** Multi-user
- ☐ **C** Distributed
- ☐ **D** Batch

> **Answer:** A. The brakes must respond within a guaranteed time limit, every time.

2. What is the defining feature of a real-time operating system? [1 mark]

- ☐ **A** It guarantees a response to an input within a fixed time limit
- ☐ **B** It is faster than any other operating system on average
- ☐ **C** It shows the time on the screen
- ☐ **D** It lets many users log in at once

> **Answer:** A. Being fast on average is not enough; it must never miss the deadline.

3. What is a distributed operating system? [1 mark]

- ☐ **A** One operating system running across several networked computers that appear to the user as one
- ☐ **B** An operating system copied to many computers
- ☐ **C** An operating system downloaded from the internet
- ☐ **D** An operating system that lets several users share one computer

> **Answer:** A. The last option is multi-user.

4. Which of these are uses of virtual machines? [1 mark]

Tick every answer that is true.

- ☐ **A** Testing software on several operating systems from one computer
- ☐ **B** Running bytecode on any kind of processor
- ☐ **C** Running several virtual servers on one physical server
- ☐ **D** Making a program run faster than native machine code

> **Answer:** A, B, C. A virtual machine adds a layer of software, so it is slower than running natively.

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

```
stack = []
for op in [('PUSH', 7), ('PUSH', 3), ('SUB',), ('PUSH', 4), ('MUL',)]:
    if op[0] == 'PUSH':
        stack.append(op[1])
    else:
        b = stack.pop()
        a = stack.pop()
        stack.append(a - b if op[0] == 'SUB' else a * b)
print(stack)
```

> **Answer:**
>
> ```
> [16]
> ```
>
> 7 - 3 = 4, then 4 * 4 = 16; the second value popped is the left operand.

The task: a bytecode virtual machine

Finish the virtual machine so it runs `bytecode`, a list of tuples whose first item is the instruction and whose second item, when there is one, is a whole number. `stack` is a Python list whose end is the top, and `pc` is the index of the next instruction. The instructions are: | Instruction | Effect | |---|---| | `PUSH n` | push n | | `DUP` | push a copy of the top value (without popping it) | | `ADD`, `SUB`, `MUL` | pop b, then pop a, then push a + b, a - b or a × b | | `FWD` | pop a distance in cm and drive forward that far at speed 60 | | `TURN` | pop an angle in degrees and turn right that far at speed 30 | | `BEEP` | pop a frequency in Hz and play it for 0.2 seconds | | `JZ a` | pop a value; if it is 0, jump to instruction a | | `JMP a` | jump to instruction a | | `HALT` | stop | When `HALT` is reached, print `stack at halt: <stack>`, printing the list as Python prints it. The program drives a 20 cm square with a beep at each corner, and leaves one value on the stack. Every distance must come off the stack.

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

bytecode = [
    ("PUSH", 4), ("DUP",), ("JZ", 14), ("PUSH", 4), ("PUSH", 5), ("MUL",), ("FWD",),
    ("PUSH", 90), ("TURN",), ("PUSH", 660), ("BEEP",), ("PUSH", 1), ("SUB",), ("JMP", 1),
("HALT",),
]

stack = []
pc = 0
while True:
    instruction = bytecode[pc]
    op = instruction[0]
    if op == "PUSH":
        stack.append(instruction[1])
    elif op == "FWD":
        forward(60, distance=stack.pop())
    elif op == "TURN":
        turn_right(30, angle=stack.pop())
    elif op == "HALT":
        break
    else:
        raise ValueError("unknown instruction " + op)
    pc = pc + 1
```

The hint students can ask for: Every instruction either pushes values, pops values, or changes the program counter. For the arithmetic ones, remember the second value popped was pushed first, which matters for SUB. A jump sets the program counter, so make sure nothing moves it on again afterwards.

A solution

```
from bugbot import *
connect()

bytecode = [
    ("PUSH", 4),
    ("DUP",),
    ("JZ", 14),
    ("PUSH", 4),
    ("PUSH", 5),
    ("MUL",),
```

```
    ("FWD",),
    ("PUSH", 90),
    ("TURN",),
    ("PUSH", 660),
    ("BEEP",),
    ("PUSH", 1),
    ("SUB",),
    ("JMP", 1),
    ("HALT",),
]

stack = []
pc = 0
while True:
    op = bytecode[pc][0]
    pc = pc + 1
    if op == "PUSH":
        stack.append(bytecode[pc - 1][1])
    elif op == "DUP":
        stack.append(stack[-1])
    elif op in ("ADD", "SUB", "MUL"):
        b = stack.pop()
        a = stack.pop()
        if op == "ADD":
            stack.append(a + b)
        elif op == "SUB":
            stack.append(a - b)
        else:
            stack.append(a * b)
    elif op == "FWD":
        forward(60, distance=stack.pop())
    elif op == "TURN":
        turn_right(30, angle=stack.pop())
    elif op == "BEEP":
        tone(stack.pop(), 0.2)
    elif op == "JZ":
        if stack.pop() == 0:
            pc = bytecode[pc - 1][1]
    elif op == "JMP":
        pc = bytecode[pc - 1][1]
    elif op == "HALT":
        break
    else:
        raise ValueError("unknown instruction " + op)
print("stack at halt:", stack)
```

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