WORKSHEET

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

Name _____ Class _____ Date _____

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

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

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

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

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)
```

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
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a10-6-types-of-os-and-virtual-machines/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Trace the stack for the first time round the loop, instruction by instruction. What is on the stack just before `JZ` each time?
2. Change the bytecode so the robot draws a triangle instead. Did you need to change the virtual machine?
3. Write a second virtual machine for the same bytecode that only prints what it would do, like machine 2 above. Why is this useful for testing?