ANSWERS

A10.3 Memory management: paging, segmentation and virtual memory

Operating systems, software and translators · A level · OCR H446 1.2.1, AQA 7517 4.6.1.4, Eduqas A500QS 2.6 · about 40 min

What this lesson is about

Logical and physical addresses, page tables, segments, page faults and disk thrashing.

- Logical and physical addresses
- Paging
- Segmentation
- Virtual memory

Questions

6 marks in all

1. With a page size of 1000 bytes, what is the physical address of logical address 2345, if page 2 is stored in frame 7? [1 mark]

> **Answer:** 7345. Page 2345 DIV 1000 = 2, offset 345; frame 7 starts at 7000, so 7000 + 345 = 7345.

2. How does segmentation differ from paging? [1 mark]

- ☐ **A** Segments are variable-size logical divisions of a program; pages are fixed-size physical divisions
- ☐ **B** Segments are fixed-size; pages vary in size
- ☐ **C** Segmentation uses secondary storage; paging does not
- ☐ **D** Paging divides a program into code, data and stack

> **Answer:** A. Pages ignore the program's structure; segments follow it.

3. What happens when a process uses an address in a page that is currently on disk? [1 mark]

- ☐ **A** A page fault: the OS loads the page into a frame, updates the page table, and the process continues
- ☐ **B** The process is terminated with a syntax error
- ☐ **C** The processor reads the data directly from the disk
- ☐ **D** The page table is deleted and rebuilt

> **Answer:** A. Virtual memory makes the swap invisible to the process, at the cost of time.

4. What is the term for the state where a computer spends more time swapping pages between memory and disk than running processes? [1 mark]

> **Answer:** disk thrashing. Thrashing happens when RAM is far too small for the processes loaded.

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

```
PAGE = 256
table = {0: 4, 1: 9}
for address in [10, 300]:
    page, offset = address // PAGE, address % PAGE
    print(address, page, offset, table[page] * PAGE + offset)
```

> **Answer:**
>
> ```
> 10 0 10 1034
> 300 1 44 2348
> ```
>
> 300 is page 1, offset 44, and frame 9 starts at 2304.

6. Which problem does virtual memory solve? [1 mark]

- ☐ **A** The processes running need more memory than the RAM installed
- ☐ **B** Programs run too slowly because of the cache
- ☐ **C** Files are fragmented on the disk
- ☐ **D** Two processes want the same printer

> **Answer:** A. Secondary storage is used as an extension of RAM, so more, and larger, processes can run.

The task: paging with page faults

Write `translate(address)` for a paging system. The inputs are: - `PAGE_SIZE`, the size of a page and a frame in bytes (256); - `page_table`, a dictionary from page number to frame number, where `None` means the page is on disk; - `free_frames`, a list of frame numbers that are free, used from the front; - `addresses`, the logical addresses to translate, each a whole number from 0 to 1023. For each address, work out the page number and offset. If the page's entry is `None`, handle the page fault: take the first frame out of `free_frames`, store it in the page table, and print `page fault: page <page> loaded into frame <frame>`. Then print `<address> -> page <page> offset <offset> -> <physical address>`, and return the physical address. Call `translate` for every address in `addresses`, in order. A page that was loaded by an earlier fault is in memory now, so it does not fault again. The robot stays still.

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

PAGE_SIZE = 256
page_table = {0: 5, 1: 2, 2: None, 3: 7}
free_frames = [1, 4]
addresses = [300, 12, 600, 1000, 700]

def translate(address):
    pass
```

The hint students can ask for: The page number is how many whole pages fit before the address, and the offset is what is left over. A page fault is a page whose table entry is empty: fill the entry with the first free frame, then carry on as normal.

A solution

```
from bugbot import *
connect()

PAGE_SIZE = 256
page_table = {0: 5, 1: 2, 2: None, 3: 7}
free_frames = [1, 4]
addresses = [300, 12, 600, 1000, 700]

def translate(address):
    page = address // PAGE_SIZE
    offset = address % PAGE_SIZE
    if page_table[page] is None:
        frame = free_frames.pop(0)
        page_table[page] = frame
        print(f"page fault: page {page} loaded into frame {frame}")
    physical = page_table[page] * PAGE_SIZE + offset
    print(f"{address} -> page {page} offset {offset} -> {physical}")
    return physical

for address in addresses:
    translate(address)
```

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