WORKSHEET

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

Name _____ Class _____ Date _____

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]

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

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

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]

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)
```

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

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
```

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

Do it on the robot

www.bugbotlab.com/learn/a10-3-memory-management/

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

Challenges

1. Work out by hand where logical address 1023 goes with this page table, then check with your program.

2. When `free_frames` is empty, a page must be swapped out. Choose the page that was loaded first, set its entry back to `None`, and reuse its frame. Add enough addresses to cause it.
3. Count the page faults for a list of 20 addresses with only 2 frames, then with 4. What would thrashing look like in these numbers?
