SCC.211 Operating Systems

Session 8

Dr Andrew Scott

a.scott@lancaster.ac.uk

Overview

- Topic 7: Memory Protection (summary/ recap)
- Topic 8: Paging (introduction)

7. Memory Protection

Summary/ Recap

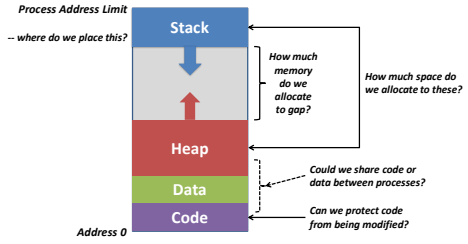
Overview

- Basic approach
- The Memory Management Unit
- Segmentation
- Paging

Memory Protection

- How to split memory between processes
  - Problems and techniques apply more generally
    - e.g. disks and filesystems
- How to protect a process's memory from other processes
- How to limit access to memory-mapped devices
  - They're just seen within/ as part of 'normal' memory
    - In theory, any code can access these locations
- *Remember privileged I/O instructions 'protect' isolated I/O devices*
  - *Normal (user level) code cannot use/ execute these instructions*
    - *We don't generally have to worry about user code accessing these*

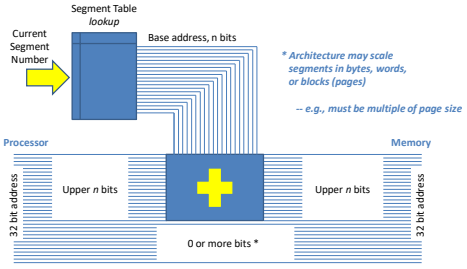
Some Simple Process Questions



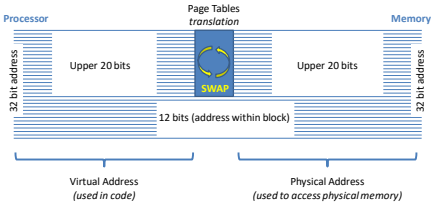
Paging vs. Segmentation

- Segmentation
  - Each segment has own address space: offset from base
  - Works by adding segment base to given address/ offset
- Paging
  - All processes see same address range
    - Translation allows any page to map to any physical frame
  - Works by translating/ switching upper bits of given address, frame for page

Segmentation operation



Paging operation



Varies with address bus width... e.g. 32 vs. 64 bit

Paging vs. Segmentation: *Instructions*

Segmentation

- Two options for segment table(s):
  1. Code & Data share same table
     - Just goes off segment number
  2. Code & Data each have own tables
     - Code → code segment table
     - Data → data segment table

```
LOAD   4, 0x5624    ; 4 is a data seg.
JUMP   3, 0x0088    ; 3 a code seg.
```

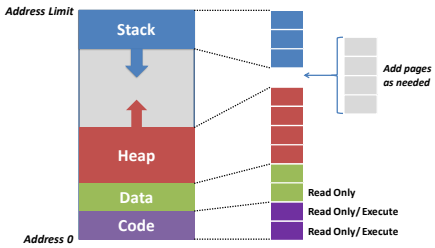
May also have one or more segment registers
-- giving default, if instructions omit explicit segment number

Paging

- Simple linear address
- Just as we're used to:

```
LOAD    0x00fec90b
JUMP    0x04ad8724
```

Fixed Sized Memory Allocation: *Paging*



Segmentation

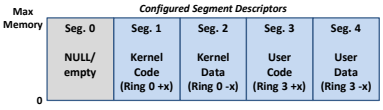
- Now rarely (actively) use segmentation
  - Offers flexible protection (possibly even to object level), but…
  - More difficult to share small amounts of memory (code or data)

    …soon end up with many segments to manage
  - Unlike paging, segmentation visible to code/ programmer

Segmentation: *Common Configuration*

- In mixed/ hybrid systems, typically configure system to rely 'solely' on paging
  - In hybrid system, segmentation still provides User/ System separation, and Code/ Data protection

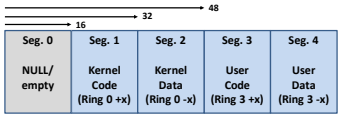
    …user segments configured for least privilege (ring 3), and so 'data' can't be run as code (-x)
  - In this case, all segments configured to span full memory range, as in common x86 set up:



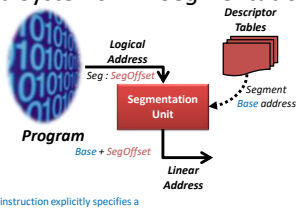
Moving from System to Kernel mode

- For example, an x86 kernel finishes initialisation and initiates the first user process…

```
printk ("\n\n========= Entering x86 user mode ========\n\n");
asm volatile (
                "                    call $48+3, $usrmode;  "
                " usrmode:                                  "
                );
```



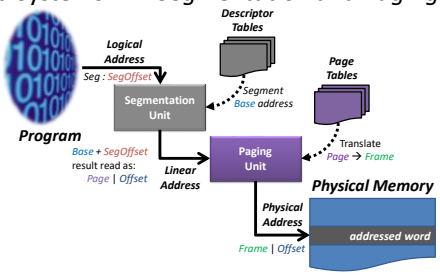
Hybrid Systems: *Segmentation…*



Unless an x86 instruction explicitly specifies a segment* the segment is taken to be that set in the code or data segment register

* In protected mode this is an offset/ index into the Descriptor Table, e.g. JUMP KernelCodeSeg, offset

Hybrid Systems: *Segmentation and Paging*



8. Paging

Introduction

Paging Topics

- Handling large address spaces
  - Translations become increasingly expensive as memory size grows
- Demand paging
  - Growing a process by adding more memory/ pages
- Shared memory
  - Supporting everything from libraries to inter-process communication

Paging Topics II

- Translation Look-aside Buffer
  - Reducing the cost of caching translations
- Speeding up process creation
  - Using paging to reduce cost of *fork( )* and *fork( )...exec( )*