Operating System: Main Memory

Alex Chi

Update: April 9, 2020

Contents

1 Background

- process must be brought to memory somewhere
- CPU cannot directly access disk

1.1 MMU

- memory management unit
- base + limit (like 8 086)
- CPU issue logical address, and MMU remaps the address to physical one
- what is logical address space larger than physical one?

2 Memory Management

- contiguous memory allocation
- non-contiguous memory allocation
    - paging

2.1 Contiguous Allocation

- multiple-partition allocation
- a process arrives, allocate memory from a hole
- exit frees partition
- simple, but there might be fragmentation
- strategies
    - first-fit
    - given N blocks allocated, another 0.5N lost
        - 1/3 may be unusable: 50-percent rule
    - best-fit
    - worst-fit

2.2 Fragmentation

- external: total memory space enough, but not contiguous
- internal: allocate memory larger than requested

- reduce external fragmentation by compaction
  - shuffle memory contents
  - compaction (dynamic relocation + pause execution)
- or paging + segmentation

2.3 Paging

- divide physical memory into fix-sized blocks called frames
  - power of 2, 512B - 16MB
- divide logical memory into blocks of same size called pages
- need to set up page table

2.3.1 Address Translation

- `aaaabbbbccccd|ddd`
- `ddd` is page offset
- `aaaabbbbccccd` is page number
- page size is $2^12 = 4KB$
- logical address space 2^64B
- page table: lookup page number in page table, find physical address base (frame number)
- e.g. `aaaabbbbccccd` maps to 233, then physical address is `233ddd`.

2.3.2 Fragmentation

- worst case = pgsize - 1
- average size = pgsize / 2

2.3.3 Free Frames

- before setting up page table, allocate some region for saving free-frame list

2.3.4 Implementation

- page table in memory
  - page-table base register (PTBR)
  - example: RISC-V satp, x86 CR3
  - Page-table length register (PTLR)

- lookup cache (associative memory / TLB or translation look-aside buffers)
    - TLB is small (64-1024 entries)
    - on TLB miss, value is loaded to TLB
        - replacement policy
        - entries can be wired down
        - hardware / software management
    - we need to flush TLB in context switching

2.3.5 Effective Access Time

- refer to slides

2.3.6 Structure of Page Table

- page table requires large amount of memory to store!
- hierarchical page table
    - example: RISC-V Sv39 (39-bit virtual address
        - virtual addr `|9bit|9bit|9bit|12bit|`
        - three-level page table
    - two-level paging
        - virtual addr `|10b level1|10b level2|12b offset|`
        - first lookup level 2 page table address in outer page table with level1 offset
        - then look up physical address in inner table
        - a.k.a. forward-mapped page table
        - example: `|0ab|0cd|233|`, page table at P
        - lv2 = `M[P+0ab]`
        - base = `M[lv2+0cd]`
        - phy = `M[base+233]`
- hashed page table
    - common in addr space > 32bits
    - VPN hashed page table
    - chain of elements hashed to same location
- inverted page table
    - track all physical pages
    - trade page table overhead for more searching time
    - `|pid|p|d|` $\longrightarrow$ `|idx|d|`

2.3.7 Q&A

1. Can page table be put behind another page table?
   For some implementations yes, for others (inverted page table) not.
2. TLB doesn't solve the problem that process switch requires reload the page table?
   It doesn't. But we can re-use some TLB entries.
3. There exists frame that has no page mapped to?
   It's possible.
4. Linear search takes multiple access to memory?
   Yes
5. Can p it self be the index?
   Hash table might be smaller than all possible VPN. Therefore we must use a hash function.
6. How to implement inter-process shared memory in inverted page table?
   It requires an extension of page table.
7. Why we need pid in inverted page table?
   One process owns an address space. We must tell from each other.

2.4 Segmentation

- just like 8086
- refer to slides