

Virtual Memory

- **Process must be in memory**
  - In order to execute instructions
- **But maybe not all of process**
  - Only part currently executing
- **Choose in-memory portion**
  - Automatically?

---

---

---

---

---

---

---

---

Demand Paging

- **Like swapping**
  - Process written to disk
  - Read again into memory
- **But not the whole process**
  - Transfer one or more pages
  - Read in only when needed

---

---

---

---

---

---

---

---

Hardware Support

- **Page “invalid” bit**
  - Read and write enable bits off
  - Or “illegal” physical page (frame)
  - In page table entry for virtual page
- **Instruction interrupt support**
  - When invalid page referenced
  - Restart or continue instruction later

---

---

---

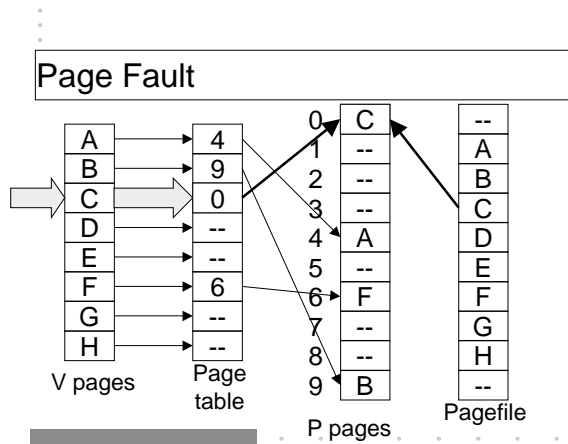
---

---

---

---

---




---

---

---

---

---

---

---

---

---

---

---

---

**Page Fault Handling**

Virtual reference generates page fault

- Check if virtual address is OK
- Find free physical page (frame)
- Read from disk to frame
- Map virtual page to frame
- Restart/continue instruction

---

---

---

---

---

---

---

---

---

---

---

---

**Paging Performance**

p: probability of page fault  
 $t_{ma}$ : memory access time  
 $t_{pf}$ : page fault handling time  
 $t_{ea}$ : effective memory access time

$$t_{ea} = (1 - p)t_{ma} + pt_{pf}$$

---

---

---

---

---

---

---

---

---

---

---

---

Paging Performance Example

$$t_{ea} = (1 - p)t_{ma} + pt_{pf}$$

$$t_{ma} = 100 \cdot 10^{-9}$$

$$t_{pf} = 25 \cdot 10^{-3}$$

$$p = 0.01$$

$$t_{ea} = (0.99)(100 \cdot 10^{-9}) + (0.01)(25 \cdot 10^{-3})$$

$$= 250.1 \cdot 10^{-6} = 250.1t_{ma}$$

---

---

---

---

---

---

---

---

Desired Paging Performance

$$t_{ea} = (1 - p)t_{ma} + pt_{pf}$$

$$t_{ea} = t_{ma} - pt_{ma} + pt_{pf}$$

$$t_{ea} = t_{ma} + p(t_{pf} - t_{ma})$$

$$p = \frac{t_{ea} - t_{ma}}{t_{pf} - t_{ma}}$$

For:  $t_{ea} = 1.1t_{ma}$   
 $t_{ma} = 100\text{ns}$   
 $t_{pf} = 25\text{ms}$   
 $p = 4 \cdot 10^{-7}$

---

---

---

---

---

---

---

---

Page Size Effects

- **Page fault on single location**
  - Access of single byte?
- **At least one page is read**
  - Page is many bytes
- **Tends to reduce future faults**
  - If other needed bytes are nearby

---

---

---

---

---

---

---

---

Page Boundaries

- **What if multi-byte reference?**
  - Data may cross page boundary
  - May cause multiple page faults
- **Data alignment**
  - Relationship between size and address
  - Guaranteed not to cross boundary?

---

---

---

---

---

---

---

---

Instruction Issues

- **One fault per instruction?**
  - RISC versus CISC ?
  - Indirect addressing?
- **Restart or resume**
  - Save state in middle of instruction?
  - Restart from beginning?

---

---

---

---

---

---

---

---

Page Replacement

- **Must allocate new frame**
  - To handle page fault
- **What if none available?**
  - All physical pages in use
- **Deallocate existing frame**
  - Write old page to pagefile

---

---

---

---

---

---

---

---

Selective Page Writes

- **Normally write replaced page**
  - Or modifications would be lost
- **But what if read-only page?**
  - Code, read-only data
- **Or if not modified?**
  - Page table “modified” bit

---

---

---

---

---

---

---

---

Page Replacement

- **Disk I/O requirements**
  - Modified page: read/write
  - Unmodified or R-O: read new only
- **Needed algorithms**
  - Page-replacement (victim choice)
  - Physical page (frame) allocation

---

---

---

---

---

---

---

---

Initial Program Loading

- **Load all of program?**
  - Allocate all physical pages ???
- **Load nothing**
  - Initialize PC
  - Let page faults begin
- **Load something**
  - Initial application pages?

---

---

---

---

---

---

---

---

⋮  
Program Pagefile Loading

- Executable program in file
- Running program in pagefile
- Copy all program pages?
  - From program file to pagefile
  - Pagefile access faster?
    - No file overhead?
  - Copy unneeded pages? R-O pages?

---

---

---

---

---

---

---

---