Reminders

- Homework 4 & project 2
  - Project 2 code due midnight today
  - Project writeup & homework 4 tomorrow in lecture
- Midterm on Monday, November 8
  - Midterm review tomorrow in lecture
- Today's office hours in 006
- Grading
  - Homework 3 back today (average: 84/100), solutions online
  - I still have some old homework/projects, pick up at the end
- Today:
  - Questions
  - Clear up issues on homework 3
  - Project 3 preview
  - Virtual memory stuff

1

Project 2 – last questions?

2

Homework 4 questions?

3

Clearing up synchronization issues

- Monitors
  - How should we use them?
  - Why is this weird inside a monitor?

```
P(mutex);
account+=balance;
V(mutex);
```

- Smoker problem
  - What's wrong if we had:
    - Agent: V(ingr[i]); V(ingr[j]);
    - Some smoker: P(ingr[a]); P(ingr[b]);
- General note: always init your semaphores!

4

Clearing up synchronization issues

- File sharing problem
  - Recall: processes can share a file as long as $\sum pid < n$
  - What's wrong here:

```
file_open(pid) {
        if(current + pid >= n)
                sum.wait();
        current += pid;
}
```

5

File sharing – (almost) correct solution

```
type file = monitor
   var space_available: condition
       total: integer
   procedure file_open(id)
   begin
     while (total + id >= n)
       space_available.wait();
     total = total + id;
   end
   procedure file_close(id)
   begin
     total = total - id;
     space_available.signal();
   end
```

6

File sharing – correct solution

```
type file = monitor
   var space_available: conditional_wait
       total: integer
   procedure file_open(id)
   begin
     while (total + id >= n)
       space_available.wait(id);
     total = total + id;
     if (total < n - 1)
       space_available.signal();
   end
   procedure file_close(id)
   begin
     total = total - id;
     space_available.signal();
   end
```

7

Project 3 preview

- Out right after midterm, due Nov. 19
- Given: vmtrace
  - Takes a memory trace file (also given)
  - Outputs # of references, # of page faults, compulsory faults, page evictions, pageouts.
- Implement an LRU-like page replacement algorithm
- Design and perform an experiment on some aspect of virtual memory

8

Project 3 experiment

- Have a hypothesis
  - “Big pages are better”
  - “Algorithm y is better”
  - “Prefetching will reduce the number of page faults”
  - “If we understand why x happens, we can fix it”
- Two steps
  - Determine baseline behavior
  - New test
    - Change one aspect of the system, observe differences

9

Some Ideas

- What is the ideal page size for this trace under different amounts of main memory?
- How much better is page replacement algorithm X than LRU
  - “Real” LRU, FIFO, 2Q, ARC, etc
- How close can we come to LRU without doing any work between page faults?
  - No scanning, constant work per page fault
- How important is recency vs. frequency in predicting page re-use?

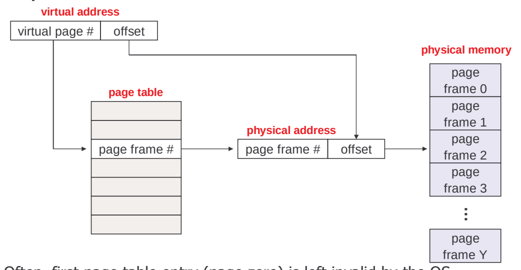
10

Not so good ideas

- What kind of music is made when I convert the address trace to notes?
- Can I make a fractal out of this data?

11

VM stuff

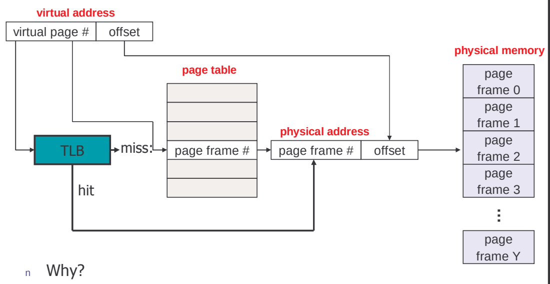


Often, first page table entry (page zero) is left invalid by the OS
- Any ideas why?

- How can we use paging to set up sharing of memory between two processes?

12

TLBs



- Why?
  - No TLB: Average number of memory accesses per virtual addr ref: **2**
  - With a TLB (99% hit rate): 0.99*1 + 0.01*2 = **1.01**

Example Page Sizes

Computer	***Page Size***
Atlas 512	48-bit words
Honeywell-Multics	1024 36-bit words
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 kbytes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
PowerPc	4 Kbytes
IA-64	4 Kbytes to 4 Gbytes

More page table/TLB examples

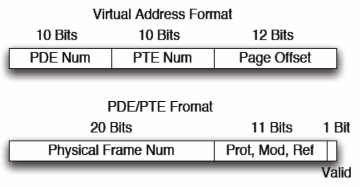
- Intel x86
  - 4K pages (common) or 4M pages (jumbo pages)
  - Two-level page tables
  - Pentium 4: 64-entry TLB
- AMD-64
  - still 4K or 2M pages
  - Four (!) PT levels for 4K pages; three for 2M pages
  - Two-level TLB (40 entries/512 entries)
- Alpha
  - 8K page size
  - Three-level page table, each one page
  - Alpha 21264: 128-entry TLB

Quick VM exercise

- Consider a virtual address space of 8 pages of 1024 words each, mapped onto a physical memory of 32 frames
  - Virtual address size (in bits):
  - Physical address size (in bits):

Intel x86 Memory Architecture

- 2-Level Page Table
- 4KB Page Size
- 32 bit addresses
- PDE/PTE of 32 bits



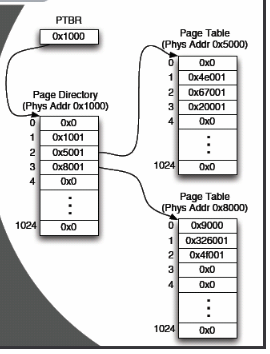
Translation

Describe the result of accessing the following virtual addresses:

0x0
0x00803024
0x00c00136

(2^{22} == 0x400000,
2^{12} == 0x1000)

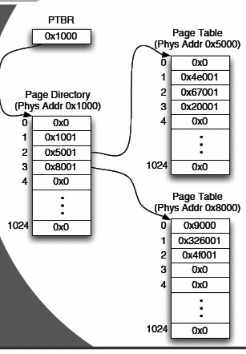
Answers: fault, 0x00020024, fault



Translation

What is the data stored at virtual address 0x00402004?

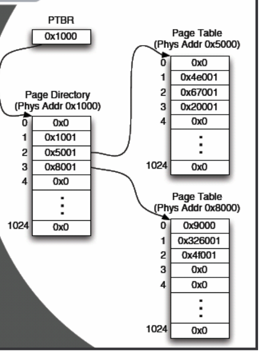
Answer: 0x0004e001



Translation

List the physical frames that this address space has direct access to. Is this address space properly isolated from accessing any other frames?

Answers: 0x1000, 0x5000, 0x8000, 0x326000, 0x4f000, 0x200000, x67000, 0x4e000. Ignoring kernel/user bits and write protection, the page tables have been made accessable to the address space (virtual addresses 0x00400000-0x004ffffff), so a process running in this address space could map-in any physical frame it wanted to.



TLB Hit Rates

Consider a x86 program consisting of 33% load/store instructions. How many extra memory accesses per instruction executed does this program need when the TLB has a 0%, 95%, or 100% hit rate?

Answers: 1.33 base memory accesses per instruction, 100%=0 extra, 0%=2.66 extra, 95%=0.05*2.66 extra