Exam 1: Review

Questions answered in this lecture:

What are some useful things to remember about virtualization?

Announcements

P1: Graded in Learn@UW; If major surprises see your TA (or 537-help@cs)

P2:
  • Pace: Good to have finished Shell (Part A) by now
  • Spend more time on Scheduler (Part B)
    • Purpose of graph is to demonstrate all aspects of scheduler are working correctly

Exam
  • Two hours – 7:15 – 9:15 pm in 272 Bascom Hall
  • Bring #2 pencils and student id
  • All multiple choice
  • Covers everything so far in course:
    • Lectures + Reading + Homework + Project 1
    • Chapters 1 - 24, excluding 10 (Multiprocessor Scheduling), 17 (Free-Space Management), and 23 (VAX/VMS Virtual Memory System)
    • Look over sample exams
REVIEW: EASY PIECE 1

Virtualization
  - CPU
    - Process
  - Context Switch
  - Schedulers

Memory
  - Address Space
  - Dynamic Relocation
  - Segmentation
  - Paging
    - TLBs
      - Multilevel
      - Swapping

WHAT QUESTIONS DID YOU ASK?
P can only see its own memory because of **user mode** (other areas, including kernel, are hidden)
P wants to call read() but no way to call it directly

OS is not part of P’s address space

System Call

**read()**:  
\texttt{movl} $6$, \%eax; \texttt{int} $64$
SYSTEM CALL

Kernel mode: we can do anything!
SYSTEM CALL

Process P

movl $6, %eax;   int $64

syscall-table index
trap-table index

Follow entries to correct system call code

RAM

SYSTEM CALL

Kernel can access user memory to fill in user buffer

movl $6, %eax;   int $64

syscall-table index
trap-table index

buf

Kernel can access user memory to fill in user buffer

return-from-trap at end to return to Process P
HW, OS, OR USER PROCESS?

Create entry for process list  OS  HW  USER
Allocate memory for program  OS  HW  USER
Load program into memory  OS  HW  USER
Setup user stack with argv  OS  HW  USER
Fill kernel stack with esp/pc  OS  HW  USER
execute return-from-trap instruction  OS  HW  USER
restore regs from kernel stack  OS  HW  USER
switch to user mode  OS  HW  USER
set PC to main()  OS  HW  USER
Start running in main()  OS  HW  USER
Call a system call  OS  HW  USER
execute trap instruction  OS  HW  USER
save regs to kernel stack  OS  HW  USER
switch to kernel mode  OS  HW  USER
set PC to OS trap handler  OS  HW  USER
Handle trap  OS  HW  USER
Do work of syscall  OS  HW  USER
execute return-from-trap instruction  OS  HW  USER
restore regs from kernel stack  OS  HW  USER
switch to user mode  OS  HW  USER
set PC to instruction after earlier trap  OS  HW  USER
Call wait() system call  OS  HW  USER

PROCESS API: HW IN BOOK

Write a program using fork(). The child process should print "hello"; the parent process should print "goodbye". You should try to ensure that the child process always prints first; can you do this without calling wait() in the parent?

- Waitpid, sleep, other synchronization primitives such as condition variables and semaphores (next topic!)

Is it possible for child process to wait for a parent or does it always have to be the other way around?

- Wait() and waitpid() apply to children processes

Typical workflow of creating a new process is to call exec in child after forking. Would there ever be a reason to create a child and call exec in the parent instead?

- No good reason I can think of
**PROCESS API**

If a parent and a child can access the same file descriptor, why does closing a file descriptor in a child not effect the parent process? Is it just because the file descriptor table is unique for each, but each entry references the same file?

![File Descriptor Diagram]

**MULTI-LEVEL FEEDBACK QUEUE (MLFQ) RULES**

Rule 1: If priority(A) > Priority(B),
A runs

Rule 2: If priority(A) == Priority(B),
A & B run in RR

More rules:
- Q3 → A  Rule 3: Processes start at top priority
- Q2 → B  Rule 4: If job uses whole slice, demote process
  (longer time slices at lower priorities)
- Q1
- Q0 → C → D
**JOB THAT PERFORMS I/O PERIODICALLY**

Stays in Q1 queue as long as doesn’t use entire Q1 timeslice

---

**PREVENT GAMING**

Problem: High priority job could trick scheduler and get more CPU by performing I/O right before time-slice ends

Fix: Account for process’s total run time at priority level downgrade when exceed threshold
**HOW ARE VIRTUAL ADDRESSES GENERATED?**

- What do addresses look like from the program’s perspective? (from the user process’s perspective)
- Generated by compiler and contents of registers

**QUIZ: MEMORY ACCESSES?**

Initial \%rip = 0x10  
\%rbp = 0x200

1) Fetch instruction at addr 0x10  
   Exec:

2) load from addr 0x208

3) Fetch instruction at addr 0x13  
   Exec:

4) Fetch instruction at addr 0x19  
   Exec:

5) store to addr 0x208

\%rbp is the base pointer:  
points to base of current stack frame  
\%rip is instruction pointer (or program counter, PC)

Memory Accesses to what virtual addresses?
### QUIZ: ADDRESS FORMAT

Given known page size, how many bits are needed in address to specify *offset* in page?

<table>
<thead>
<tr>
<th>Page Size</th>
<th>Low Bits (offset)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bytes</td>
<td>4</td>
</tr>
<tr>
<td>1 KB</td>
<td>10</td>
</tr>
<tr>
<td>1 MB</td>
<td>20</td>
</tr>
<tr>
<td>512 bytes</td>
<td>9</td>
</tr>
<tr>
<td>4 KB</td>
<td>12</td>
</tr>
</tbody>
</table>

Assuming byte addressable architecture

### QUIZ: ADDRESS FORMAT

Given number of bits in virtual address and bits for offset, how many bits for virtual page number?

<table>
<thead>
<tr>
<th>Page Size</th>
<th>Low Bits (offset)</th>
<th>Virt Addr Bits</th>
<th>High Bits (vpn)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bytes</td>
<td>4</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>1 KB</td>
<td>10</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>1 MB</td>
<td>20</td>
<td>32</td>
<td>12</td>
</tr>
<tr>
<td>512 bytes</td>
<td>9</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>4 KB</td>
<td>12</td>
<td>32</td>
<td>20</td>
</tr>
</tbody>
</table>

Correct?
**QUIZ: ADDRESS FORMAT**

Given number of bits for vpn, how many virtual pages can there be in an address space?

<table>
<thead>
<tr>
<th>Page Size</th>
<th>Low Bits (offset)</th>
<th>Virt Addr Bits</th>
<th>High Bits (vpn)</th>
<th>Virt Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bytes</td>
<td>4</td>
<td>10</td>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td>1 KB</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>1 K</td>
</tr>
<tr>
<td>1 MB</td>
<td>20</td>
<td>32</td>
<td>12</td>
<td>4 K</td>
</tr>
<tr>
<td>512 bytes</td>
<td>9</td>
<td>16</td>
<td>7</td>
<td>128</td>
</tr>
<tr>
<td>4 KB</td>
<td>12</td>
<td>32</td>
<td>20</td>
<td>1 M</td>
</tr>
</tbody>
</table>

Tells you how many entries are needed in page tables!

**VIRTUAL => PHYSICAL PAGE MAPPING**

Number of bits in virtual address format does not need to equal number of bits in physical address format.

How should OS translate VPN to PPN?

For segmentation, OS used a formula (e.g., phys_addr = virt_offset + base_reg)

For paging, OS needs more general mapping mechanism

What data structure is good?

Big array: pagetable
WHERE ARE PAGETABLES STORED?

How big is a typical page table?
- assume 32-bit address space
- assume 4 KB pages
- assume 4 byte page table entries (PTEs)

Final answer: $2^{(32 - \log(4KB))} \times 4 = 4 \text{ MB}$
  - Page table size = Num entries * size of each entry
  - Num entries = num virtual pages = $2^{\text{bits for vpn}}$
  - Bits for vpn = 32 – number of bits for page offset
  $= 32 - \log(4KB) = 32 - 12 = 20$
  - Num entries = $2^{20} = 1 \text{ MB}$
  - Page table size = Num entries * 4 bytes = 4 MB

Implication: Store each page table in memory
  - Hardware finds page table base with register (e.g., CR3 on x86)

What happens on a context-switch?
  - Change contents of page table base register to newly scheduled process
  - Save old page table base register in PCB of descheduled process

QUIZ: HOW BIG ARE PAGE TABLES?

How big is each page table?

1. PTE’s are 2 bytes, and 32 possible virtual page numbers
   \[32 \times 2 \text{ bytes} = 64 \text{ bytes}\]

2. PTE’s are 2 bytes, virtual addr are 24 bits, pages are 16 bytes
   \[2 \text{ bytes} \times 2^{(24 - \log(16))} = 2^{21} \text{ bytes} (2 \text{ MB})\]

3. PTE’s are 4 bytes, virtual addr are 32 bits, and pages are 4 KB
   \[4 \text{ bytes} \times 2^{(32 - \log(4K))} = 2^{22} \text{ bytes} (4 \text{ MB})\]

4. PTE’s are 4 bytes, virtual addr are 64 bits, and pages are 4 KB
   \[4 \text{ bytes} \times 2^{(64 - \log(4K))} = 2^{54} \text{ bytes}\]
3) MULTILEVEL PAGE TABLES

Goal: Allow each page table to be allocated non-contiguously

Idea: Page the page tables
  • Creates multiple levels of page tables; outer level “page directory”
  • Only allocate page tables for pages in use
  • Used in x86 architectures (hardware can walk known structure)

30-bit address:

outer page (8 bits)  inner page (10 bits)  page offset (12 bits)

Quiz: MULTILEVEL

<table>
<thead>
<tr>
<th>page directory</th>
<th>page of PT (@PPN:0x3)</th>
<th>page of PT (@PPN:0x92)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPN</td>
<td>valid</td>
<td>PPN</td>
</tr>
<tr>
<td>0x3</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>0x92</td>
<td>1</td>
<td>0x25</td>
</tr>
</tbody>
</table>

20-bit address:

outer page (4 bits)  inner page (4 bits)  page offset (12 bits)
**Problem with 2 Levels?**

Problem: page directories (outer level) may not fit in a page

64-bit address:

<table>
<thead>
<tr>
<th>outer page?</th>
<th>inner page (10 bits)</th>
<th>page offset (12 bits)</th>
</tr>
</thead>
</table>

Solution:

- Split page directories into pieces
- Use another page dir to refer to the page dir pieces.

VPN

<table>
<thead>
<tr>
<th>PD idx 0</th>
<th>PD idx 1</th>
<th>PT idx</th>
<th>OFFSET</th>
</tr>
</thead>
</table>

How large is virtual address space with 4 KB pages, 4 byte PTEs, each page table fits in page given 1, 2, 3 levels?

4KB / 4 bytes → 1K entries per level

---

**TLB Question**

Why are fully associative TLBs less collision prone than the non-fully associative TLB?

What does collision actually mean over here?
**Translation Lookaside Buffer (TLB)**

TLB: Translation Lookaside Buffer (this is special hardware!)

---

**TLB Example**

Various ways to organize a 16-entry TLB (artificially small)

- Direct mapped
  - 30 % 16 = ?
- Two-way set associative
  - 30 % 8 = ?
- Four-way set associative
  - 30 % 4 = ?
- Fully associative

**Lookup**
- Calculate set (tag % num_sets)
- Search for tag within resulting set
TLB ASSOCIATIVITY TRADE-OFFS

Higher associativity
+ Better utilization, fewer collisions (or conflicts)
  – Slower
  – More hardware

Lower associativity
+ Fast
+ Simple, less hardware
  – Greater chance of collisions (or conflicts)

TLBs usually fully associative

PRESENT VS VALID BIT

• Virtual memory when page is not allocated in physical memory (RAM); instead on disk
• Why is a present bit needed? Why not just use valid bit?
VIRTUAL ADDRESS SPACE
MECHANISMS

Each page in virtual address space maps to one of three:
• Nothing (error): Free
• Physical main memory: Small, fast, expensive
• Disk (persistent storage): Large, slow, cheap

Extend page tables with an extra bit: present
• permissions (r/w), valid, present
• Page is not allocated or mapped (not valid)
  • Segmentation fault
• Page in memory: present bit set in PTE, hold PPN
• Page on disk: present bit cleared
  • PTE points to block address on disk
  • Causes trap into OS when page is referenced
  • Trap: page fault

PRESENT BIT

What if access vpn 0xb?
SWAPPING

Assume: when process starts, all the code that runs has to be loaded in from the disk due to page faults occurring, is this correct?

- Yes, with pure demand paging

Why in diagram 21.1 is proc0’s VPN 0 page in memory but not on the disk? Wouldn’t proc0’s VPN 0 page still be on the disk except it was also copied into main memory?

GOOD LUCK!

- TAs may review for exam more in discussion section or might go over Project material
- Use form if you care!