CS5460/6460: Operating Systems

Lecture 16: Midterm recap, sample questions

Anton Burtsev
February, 2014
Describe the x86 address translation pipeline (draw figure), explain stages.
Logical Address (or Far Pointer) → Segment Selector → Offset → Linear Address Space

Global Descriptor Table (GDT) → Segment Descriptor → Segment Base Address

Segment → Lin. Addr. → Linear Address Space

Dir → Table → Offset → Physical Address Space

Segmentation → Paging
Logical Address (or Far Pointer) 

Segment Selector 
Offset 

Global Descriptor Table (GDT) 
Segment Descriptor 
Segment Base Address 

Segment 
Lin. Addr. 

Linear Address Space 
Dir 
Table 
Offset 

Page Table 
Entry 
Page Directory 
Entry 

Page 

Physical Address Space 
Phy. Addr. 

Segmentation 
Paging
What is the linear address? What address is in the registers, e.g., in %eax?
Logical and linear addresses

- Segment selector (16 bit) + offset (32 bit)
What segments do the following instructions use? push, jump, mov
Describe the linear to physical address translation with the paging mechanism (use provided diagram, mark and explain the steps).
Page translation

Linear Address
31  22  21  12  11  0
- Directory  - Table  - Offset

Page Directory
- PDE with PS=0

Page Table
- PTE

Physical Address
- 4-KByte Page

CR3
Page directory entry (PDE)

| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | Address of page table | Ignored | 0 |Ign | A | P | C | D | P W | U | S | R | W | 1 | PDE: page table |

- 20 bit address of the page table
  - Pages 4KB each, we need 1M to cover 4GB
- R/W – writes allowed?
  - To a 4MB region controlled by this entry
- U/S – user/supervisor
  - If 0 – user-mode access is not allowed
- A – accessed
Page translation

Linear Address

Directory  Table  Offset

4-KByte Page

Page Table

Physical Address

Page Directory

PDE with PS=0

CR3

PTE

32

10

12

10

20

20
### Page table entry (PTE)

<table>
<thead>
<tr>
<th>Address of 4KB page frame</th>
<th>Ignored</th>
<th>G</th>
<th>P</th>
<th>T</th>
<th>D</th>
<th>A</th>
<th>P</th>
<th>C</th>
<th>D</th>
<th>P</th>
<th>T</th>
<th>U</th>
<th>S</th>
<th>R</th>
<th>W</th>
<th>1</th>
<th>PTE: 4KB page</th>
</tr>
</thead>
</table>

- 20 bit address of the 4KB page
  - Pages 4KB each, we need 1M to cover 4GB
- R/W – writes allowed?
  - To a 4KB page
- U/S – user/supervisor
  - If 0 user-mode access is not allowed
- A – accessed
- D – dirty – software has written to this page
Page translation

Linear Address
31  22  21  12  11  0
Directory  Table  Offset

Page Directory

PDE with PS=0

CR3

Page Table

PTE

Physical Address

4-KByte Page

12
Describe the steps and data structures involved into a user to kernel transition (draw diagrams)
Interrupt path

Process

User stack of a process (can grow up to 2GBs)

Code, data, heap

Timer: IRQ0 -> vector 32

Interrupt Vector #

Kernel Stack of a process (4K)

User state (saved by hardware)

User state

Argument 1
Argument 2
Calling EIP ++
Old EBP
Local variables
Saved local values, e.g. push EAX, etc

EBP

Kernel Stack of a process (4K)

EIP: <kernel>
EIP: <kernel>
GDT: gdt
TSS: tss
IDT: idt
CR3: pt

CS : #1
SS : #2

GDT: gdt
TSS: tss
CS
SS
ESP
EFLAGS
Code, data, heap

Page table

Level 1

Level 2

Kernel code

vector 32

0 - 4KB
4 - 8KB
2GB - 2GB + 4MB
0 - 4K
4K - 8K
...
(4MB-4K) - 4MB

0 - 4KB
4 - 8KB
2GB - 2GB + 4MB
0 - 4K
4K - 8K
...
(4MB-4K) - 4KB

0 - 4KB
4 - 8KB
2GB - 2GB + 4MB
0 - 4K
4K - 8K
...
(4MB-4K) - 4KB

CS : HANDLER ADDR
TSS
SS:
ESP:
...
What segment is specified in the interrupt descriptor? Why?
Interrupt descriptor

Interrupt Gate

Offset 31..16

Segment Selector

Offset 15..0
- Interrupt gate disables interrupts
  - Clears the IF flag in EFLAGS register
- Trap gate doesn't
  - IF flag is unchanged
Which stack is used for execution of an interrupt handler? How does hardware find it?
Why does xv6 uses 4MB pages for the first page table during boot?
Describe organization of the memory allocator in xv6?
Describe how a per-CPU variables can be stored?
swtch in xv6 doesn’t explicitly save and restore all fields of struct context. Why is it okay that swtch doesn’t contain any code that saves %eip?
Stack inside swtch()

User state (saved by hardware)
- SS
- ESP
- EFLAGS
- CS
- CS
- 0
- 32
- DS
- ES
- FS
- GS
- All registers
- ESP
- EIP (alltraps)
- ...
- EIP (trap)
- ...
- EIP (yield)
- ...
- &proc->context
- cpu->scheduler
- EIP (sched)

Kernel Stack of a process (4K)

Trap frame

Call stack:
- vector32()
- alltraps()
- trap()
- yield()
- sched()
- switch(&proc->context, cpu->scheduler)

Context
- EIP (line: 2479)
- EBP
- EBX
- ESI
- EDI
Describe how does RCU work?
Read copy update

- Goal: remove “cat” from the list
  - There might be some readers of “cat”
- Idea: control the pointer dereference
  - Make it atomic
Read copy update (2)

- Remove “cat”
  - Update the “boa” pointer
  - All subsequent reader will get “gnu” as boa->next
Read copy update (2)

- Wait for all readers to finish
  - synchronize_rcu()
Read copy update (3)

- Readers finished
  - Safe to deallocate “cat”
Read copy update (4)

- New state of the list
Under what conditions RCU is a good idea?
In the following piece of code explain the use of memory barriers?
Reference counting is a potential scalability bottleneck, what can be done to improve it?
Reference counting is a potential scalability bottleneck, what can be done to improve it?

- Sloppy counters
Why $O(1)$ is really $O(1)$?
Why $O(1)$ is really $O(1)$?

- Hint: analyze all operations and explain why they are constant.
Alyssa runs xv6 on a machine with 8 processors and 8 processes. Each process calls sbrk (3451) continuously, growing and shrinking its address space. Alyssa measures the number of sbrks per second and notices that 8 processes achieve the same total throughput as 1 process, even though each process runs on a different processor. She profiles the xv6 kernel while running her processes and notices that most execution time is spent in kalloc (2838) and kfree (2815), though little is spent in memset. Why is the throughput of 8 processes the same as that of 1 process?
kalloc(void)
{
    struct run *r;

    if(kmem.use_lock)
        acquire(&kmem.lock);
    r = kmem.freelist;
    if(r)
        kmem.freelist = r->next;
    if(kmem.use_lock)
        release(&kmem.lock);
    return (char*)r;
}

kfree(char *v) {
    struct run *r;

    memset(v, 1, PGSIZE);
    if(kmem.use_lock)
        acquire(&kmem.lock);
    r = (struct run*)v;
    r->next = kmem.freelist;
    kmem.freelist = r;
    if(kmem.use_lock)
        release(&kmem.lock);
}
What can be done to improve performance?
Suppose you wanted to change the system call interface in xv6 so that, instead of returning the system call result in EAX, the kernel pushed the result on to the user space stack. Fill in the code below to implement this. For the purposes of this question, you can assume that the user stack pointer points to valid memory.
void syscall(void)
{
  int num;

  num = proc->tf->eax;
  if (num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    proc->tf->eax = syscalls[num]();
  } else {
    cprintf("%d %s: unknown sys call %d\n", proc->pid, proc->name, num);
    proc->tf->eax = -1;
  }
}
void syscall(void) {
    int num;
    num = proc->tf->eax;
    if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
        // proc->tf->eax = syscalls[num]();
        proc->tf->esp -= 4;
        *(int*)ptoc->tf->esp = syscalls[num]();
    } else {
        cprintf("%d %s: unknown sys call %d
", proc->pid, proc->name, num);
        // proc->tf->eax = -1;
        proc->tf->esp -= 4;
        *(int*)ptoc->tf->esp = -1;
    }
}
 acquire(struct spinlock *lk)
{
    pushcli();
    if(holding(lk))
      panic("acquire");

    while(xchg(&lk->locked, 1) != 0)
      ;

    Why does acquire disable interrupts?
acquire(struct spinlock *lk)
{
    pushcli();
    if(holding(lk))
        panic("acquire");
    ... 
    while(xchg(&lk->locked, 1) != 0)
        ;
    ... 
    What would go wrong if you
    replaced pushcli() with just cli(),
    and popcli() with just sti()?
}
Explain why it would be awkward for xv6 to give a process different data and stack segments (i.e. have DS and SS refer to descriptors with different BASE fields).
Thank you!