CS444/544
Operating Systems II

Lecture 16

Quiz 3 sol. And Final Course Review

6/5/2024

Acknowledgement: Slides drawn heavily from Yeongjin Jiang

Oregon State University

Due Reminders

- 6/10 11:59 pm: 100% for Lab 4
- 6/12 11:59 pm: 75% for Lab 4 and 50% for Lab 1-3
- After 6/12 11:59 pm: 0%

- Questions?
  - I will do another round of regrade for lab 1-3 this weekend

Today's topic

- Quiz 3 Report
- Final course review

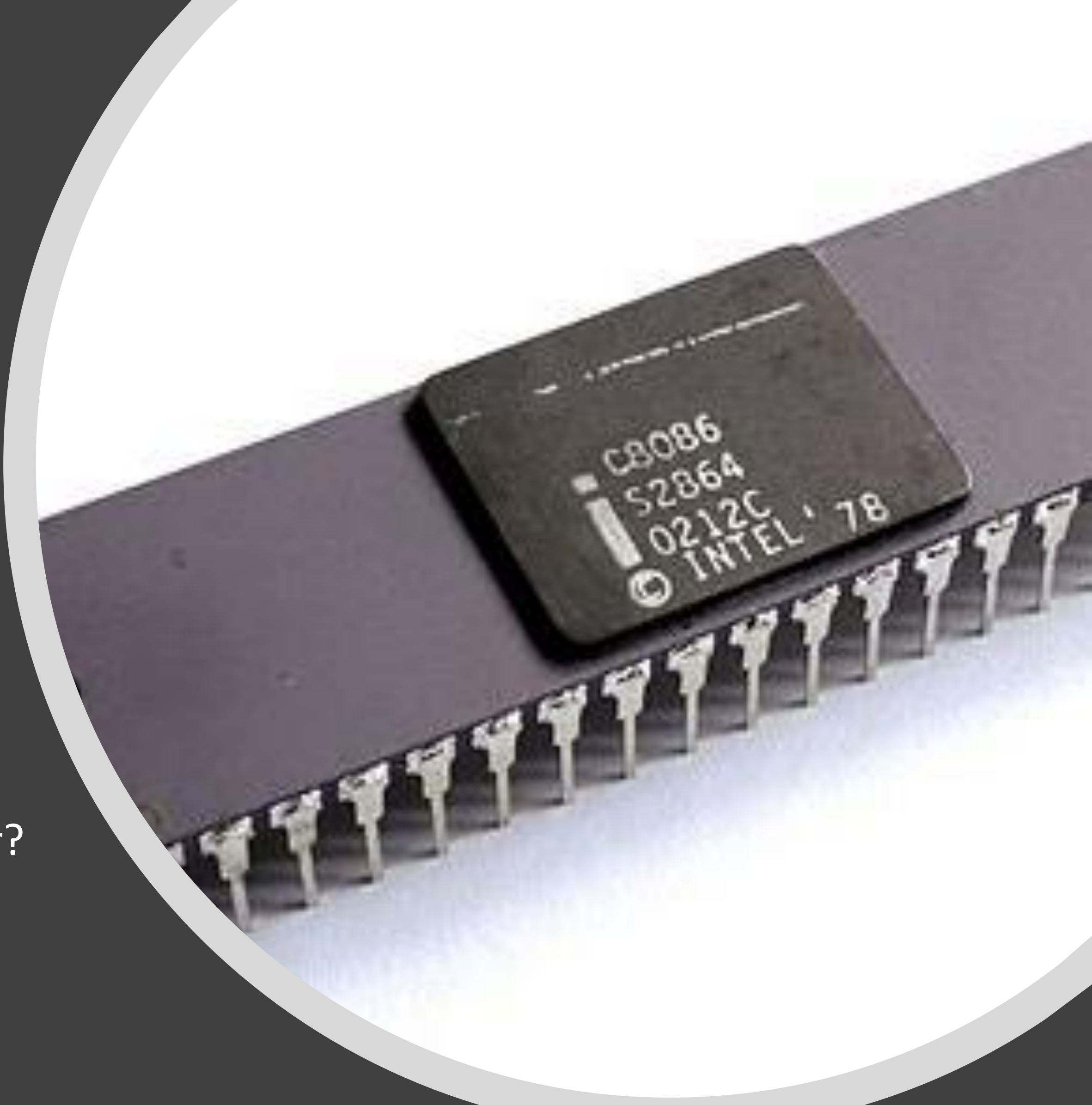
Topics Covered

- Week 1: Booting
- Week 2: Address translation
- Week 3: Virtual Memory Management
- Week 4: Quiz on Virtual Memory
- Week 5: User/Kernel Context Switch
- Week 6: System Calls and Page Fault
- Week 7: Quiz on Syscalls, Faults, and Exceptions
- Week 8: Lock and Thread Synchronization
- Week 9: Concurrency Bugs and Deadlock
- Week 10: Quiz 3 & Review

Booting
(Week 1, JOS Lab 1)

- How does x86 Processors boot with BIOS?
  - Which mode does the processor start with?
    - Real mode!
  - Addressing model in early stage of the booting
    - Seg * 16 + offset

- BIOS / Boot sector
  - Where (which address) does BIOS load the boot sector?
    - 0x7c00
  - How does boot sector load the kernel?
    - ELF header
- Processor modes: Real / Protected
  - How does CPU use memory segmentation in those modes?
    - Global descriptor Table (GDT)

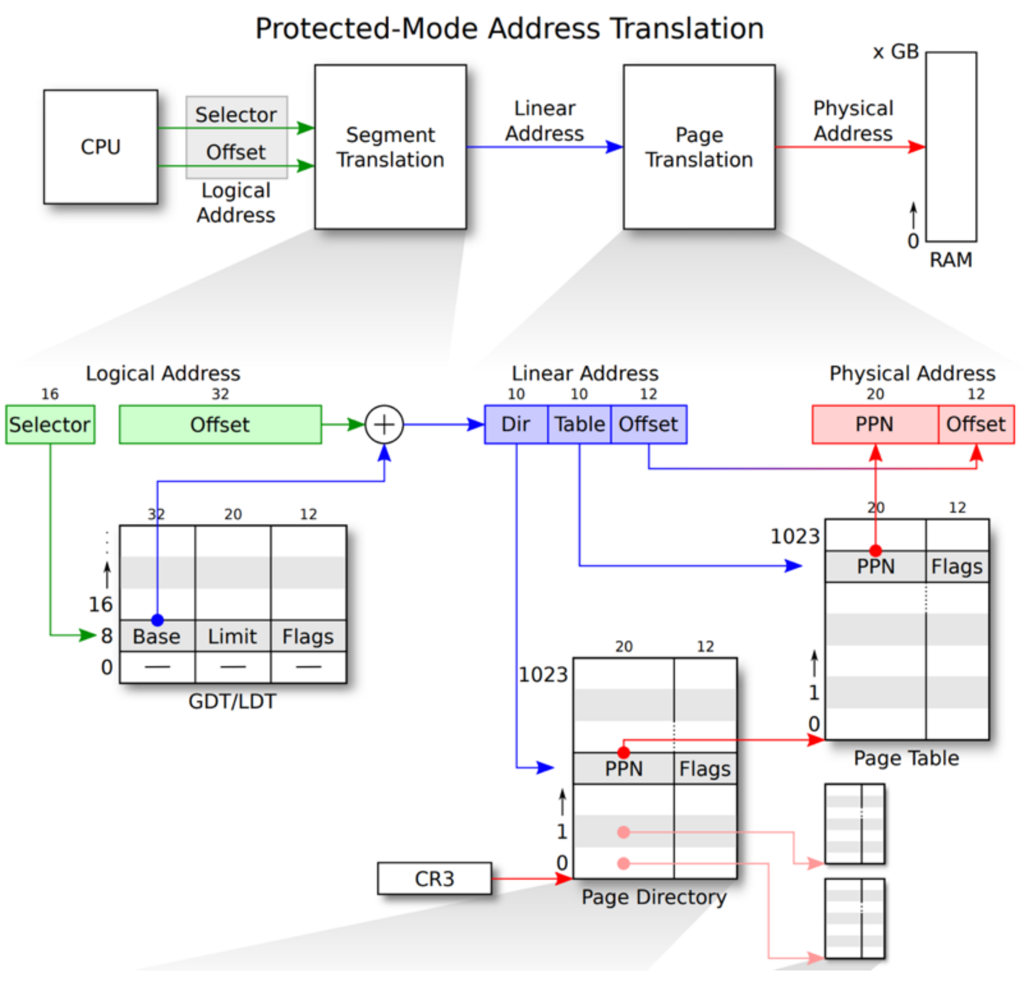


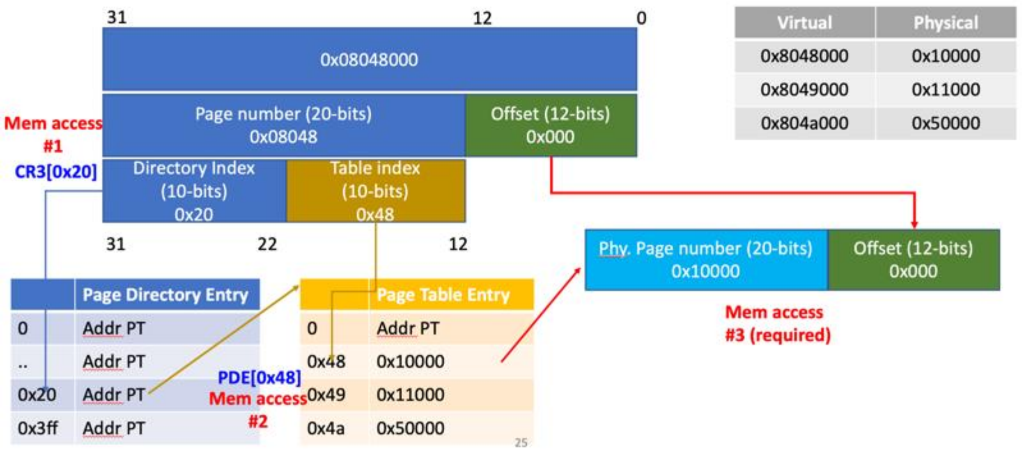
Oregon State University

Address Translation (Week 2)

- Segmentation
  - Seg * 16 + offset
  - GDT – base + offset, offset < limit

- Paging
  - Page table / page directory
  - ☆ Translation Lookaside Buffer (TLB)
  - When to invalidate TLB?
    - When updates CR3 (invalidate all entries)
    - When updates PTE (invalidate 1 entry)





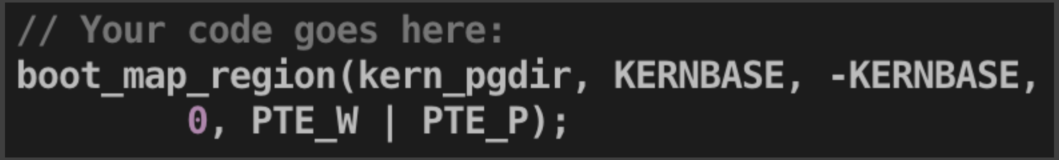
Virtual	Physical
0x8048000	0x10000
0x8049000	0x11000
0x804a000	0x50000

	Page Directory Entry
0	Addr PT
..	Addr PT
0x20	Addr PT
0x3ff	Addr PT

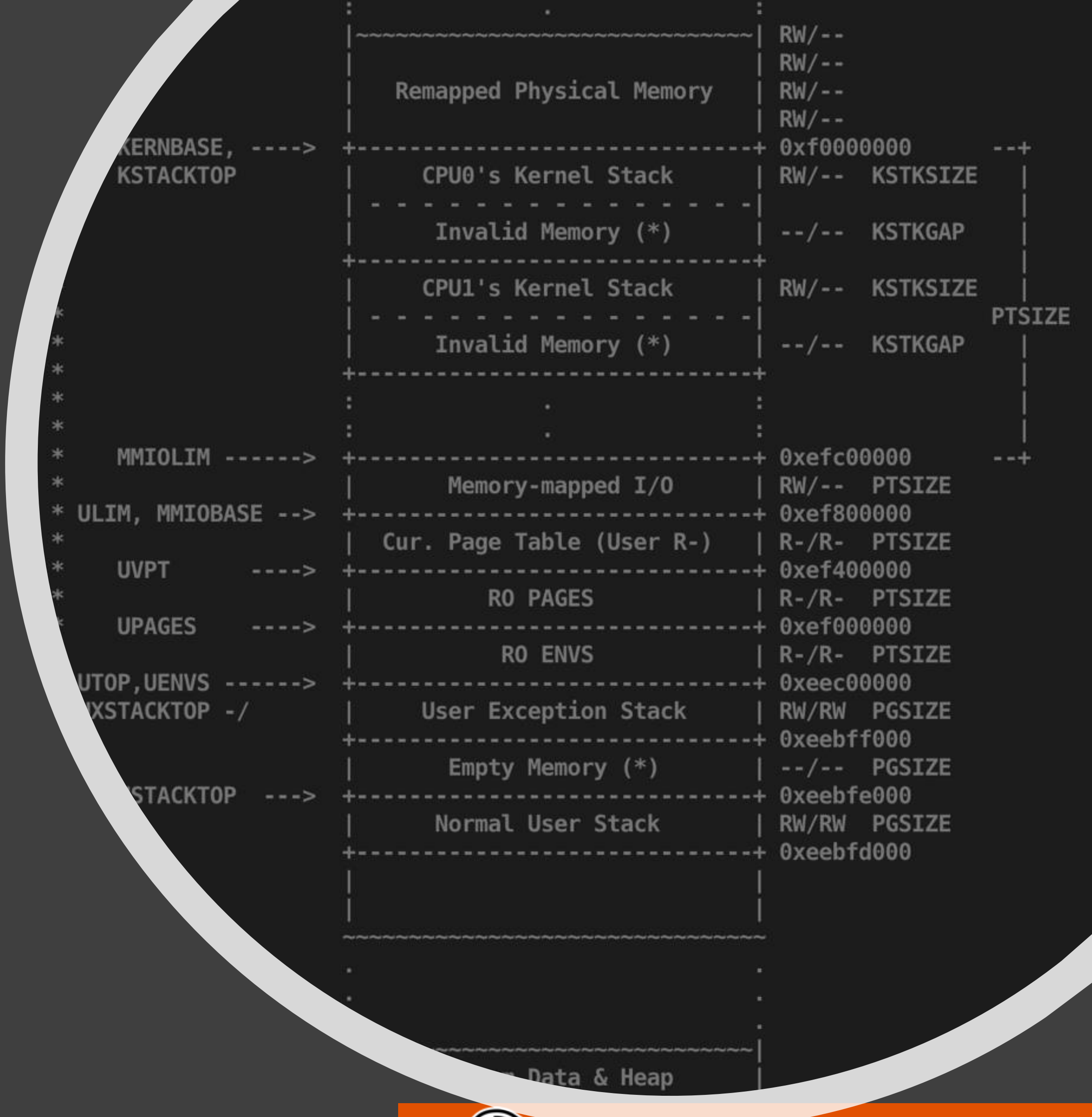
	Page Table Entry
0	Addr PT
0x48	0x10000
0x49	0x11000
0x4a	0x50000

Virtual Memory Management (Week 3, JOS Lab 2)

- Page Permission
  - How can we set access permissions to a memory page?
    - Read/Write, Kernel/User
  - How can we set a conflicting memory permissions, e.g.,
    - Kernel RW, User R



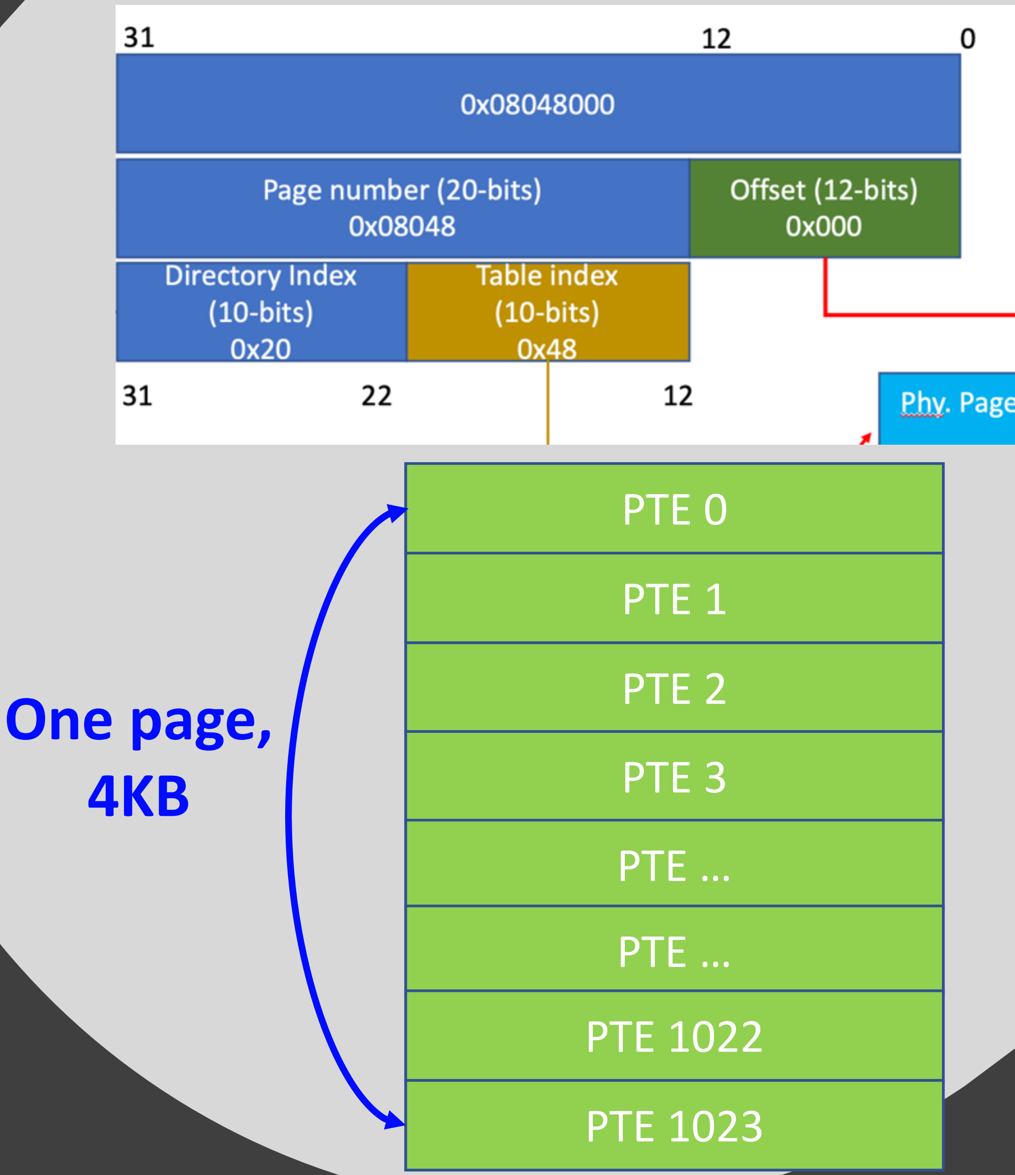
```
// Your code goes here:
boot_map_region(kern_pgdir, KERNBASE, -KERNBASE,
        0, PTE_W | PTE_P);
```



```
                          :              .               :
                          |~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~| RW/--
                          |                              | RW/--
                          |   Remapped Physical Memory   | RW/--
                          |                              | RW/--
  KERNBASE, ---->         +------------------------------+ 0xf0000000      --+
  KSTACKTOP               |     CPU0's Kernel Stack      | RW/--  KSTKSIZE   |
                          | - - - - - - - - - - - - - - -|                   |
                          |      Invalid Memory (*)      | --/--  KSTKGAP    |
                          +------------------------------+                   |
                          |     CPU1's Kernel Stack      | RW/--  KSTKSIZE   |
*                         | - - - - - - - - - - - - - - -|                 PTSIZE
*                         |      Invalid Memory (*)      | --/--  KSTKGAP    |
*                         +------------------------------+                   |
*                         :              .               :                   |
*                         :              .               :                   |
*   MMIOLIM ------>       +------------------------------+ 0xefc00000      --+
*                         |       Memory-mapped I/O      | RW/--  PTSIZE
* ULIM, MMIOBASE -->      +------------------------------+ 0xef800000
*                         |  Cur. Page Table (User R-)   | R-/R-  PTSIZE
*   UVPT      ---->       +------------------------------+ 0xef400000
*                         |          RO PAGES            | R-/R-  PTSIZE
*   UPAGES    ---->       +------------------------------+ 0xef000000
                          |           RO ENVS            | R-/R-  PTSIZE
  UTOP,UENVS ------>      +------------------------------+ 0xeec00000
  XSTACKTOP -/            |     User Exception Stack     | RW/RW  PGSIZE
                          +------------------------------+ 0xeebff000
                          |       Empty Memory (*)       | --/--  PGSIZE
   STACKTOP  --->         +------------------------------+ 0xeebfe000
                          |      Normal User Stack       | RW/RW  PGSIZE
                          +------------------------------+ 0xeebfd000
                          |                              |
                          |                              |
                          ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                          .                              :
                          .                              :
                                                         .
                             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~|
                                    Data & Heap          |
```

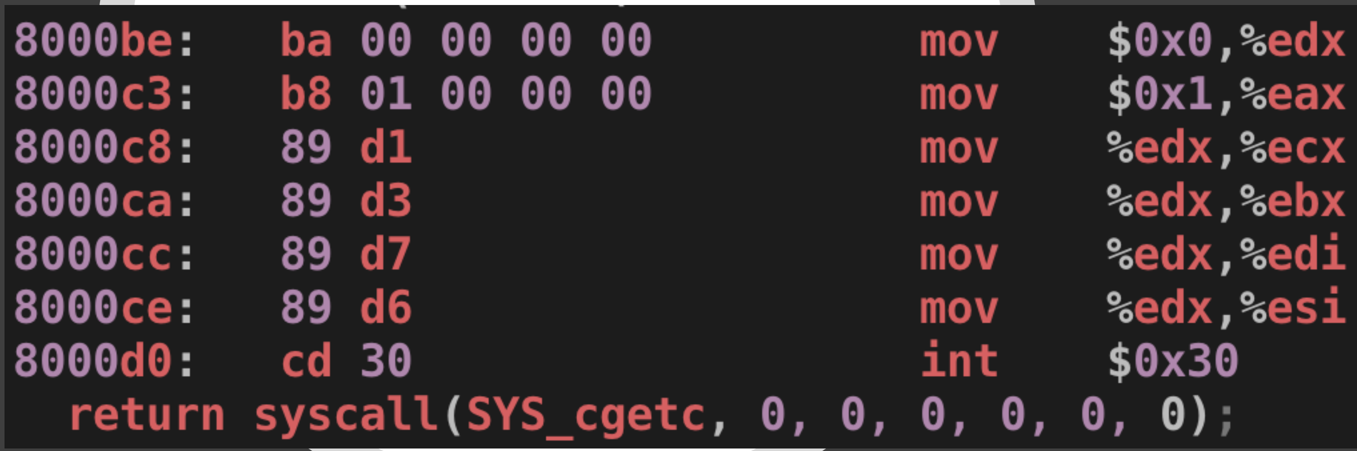
Virtual Memory Management (Week 3, JOS Lab 2)

- Indexing Page Directory / Tables
  - [10 bit] [10 bit] [12 bit]
  - Why 10 bits?
    - 12 bits for page offset: 4096 bytes
    - 4 byte per each page directory/table entries
    - 1024 entries, indexed by 10 bits
  - [6 bit] [6 bit] [6 bit] [6 bit] [8 bit]
    - Page size: 256 bytes, entries: 64, 4 levels → slow
  - [6 bit] [12 bit] [14 bit]
    - Page size: 16384 bytes, entries: 4096, wasting memory



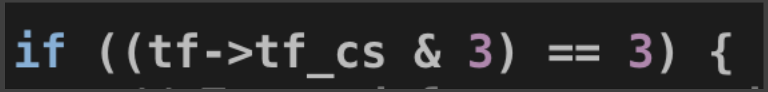
Oregon State University

User/Kernel Switch (Week 5, JOS Lab 3)



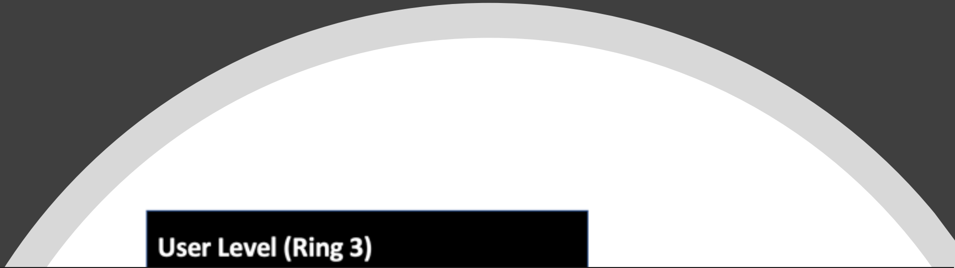
```
8000be:    ba 00 00 00 00          mov    $0x0,%edx
8000c3:    b8 01 00 00 00          mov    $0x1,%eax
8000c8:    89 d1                   mov    %edx,%ecx
8000ca:    89 d3                   mov    %edx,%ebx
8000cc:    89 d7                   mov    %edx,%edi
8000ce:    89 d6                   mov    %edx,%esi
8000d0:    cd 30                   int    $0x30
  return syscall(SYS_cgetc, 0, 0, 0, 0, 0, 0);
```

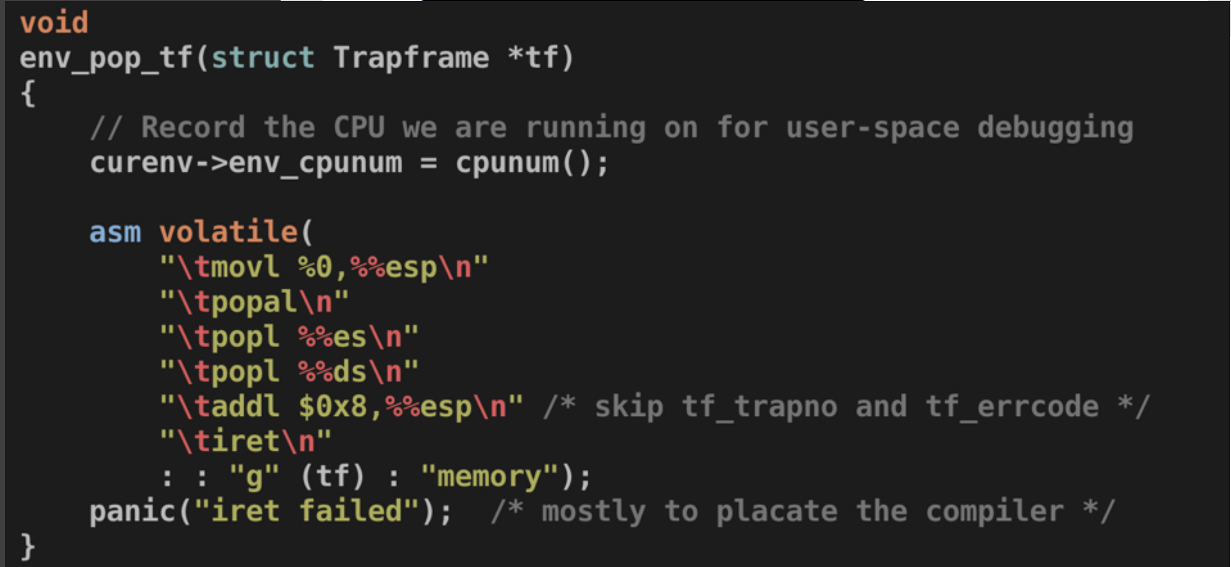
- Ring
  - How many rings are available in x86 processor?
    - 4 levels
  - Which ring level do we use for kernel? For user?
    - Ring 0 for kernel, Ring 3 for user
  - Where does CPU store the current ring level?
    - **The last 2 bits of the CS register**



```
if ((tf->tf_cs & 3) == 3) {
```

- User/Kernel Switch
  - Difference between library call and system call
  - How can we switch an execution from
    - User -> kernel?
      - syscalls (software interrupt)
    - Kernel -> User?
      - iret



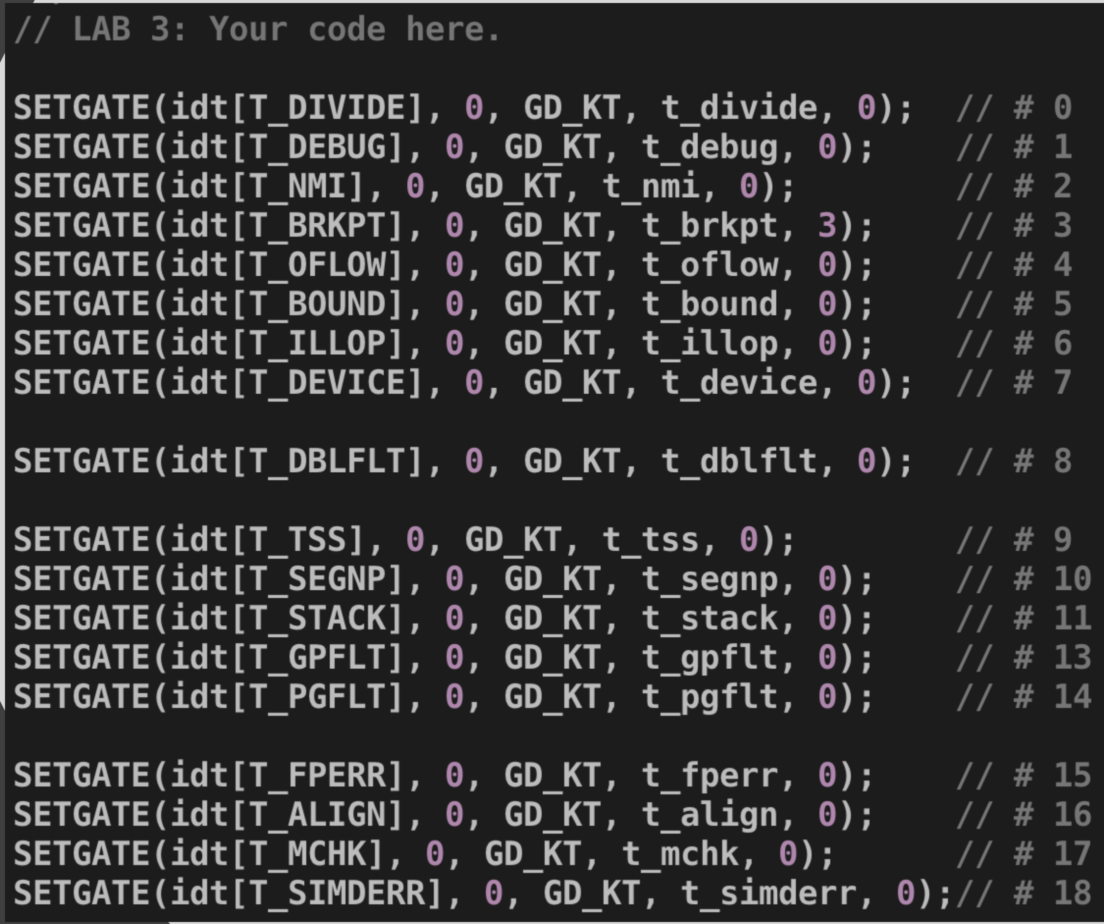


```
void
env_pop_tf(struct Trapframe *tf)
{
    // Record the CPU we are running on for user-space debugging
    curenv->env_cpunum = cpunum();

    asm volatile(
        "\tmovl %0,%%esp\n"
        "\tpopal\n"
        "\tpopl %%es\n"
        "\tpopl %%ds\n"
        "\taddl $0x8,%%esp\n" /* skip tf_trapno and tf_errcode */
        "\tiret\n"
        : : "g" (tf) : "memory");
    panic("iret failed");  /* mostly to placate the compiler */
}
```

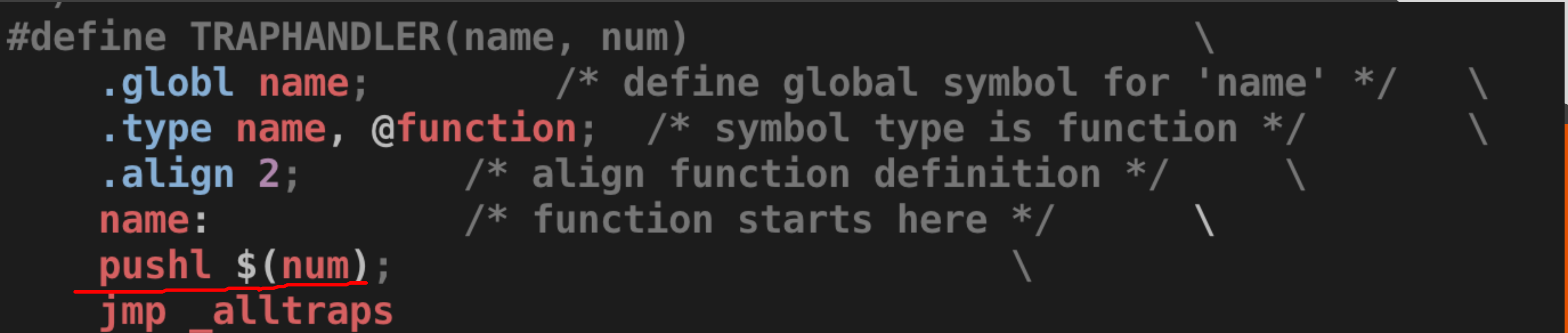
Interrupt, Syscall, Exception (Week 5, JOS Lab 3)

- Interrupt & Exceptions
  - Interrupt
  - Exceptions and Fault

- Interrupt Descriptor Table (IDT) and Interrupt handlers
  - How can we set interrupt handlers?
  - How can we determine which interrupt the current one is?
    - E.g., how can we get the interrupt number?
    - Pushed by CPU? Pushed by JOS?



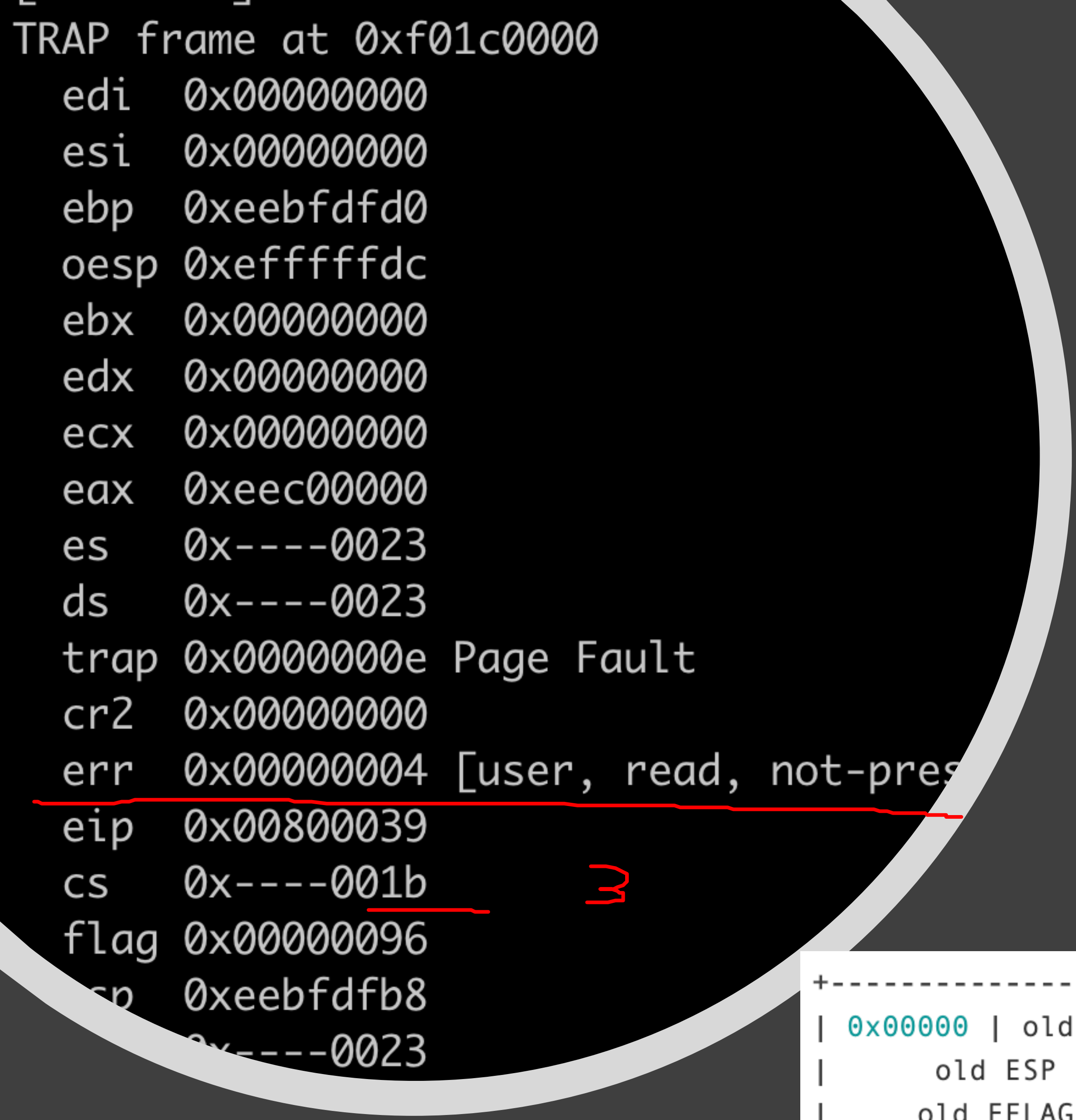
```
// LAB 3: Your code here.

SETGATE(idt[T_DIVIDE], 0, GD_KT, t_divide, 0);   // # 0
SETGATE(idt[T_DEBUG], 0, GD_KT, t_debug, 0);     // # 1
SETGATE(idt[T_NMI], 0, GD_KT, t_nmi, 0);         // # 2
SETGATE(idt[T_BRKPT], 0, GD_KT, t_brkpt, 3);     // # 3
SETGATE(idt[T_OFLOW], 0, GD_KT, t_oflow, 0);     // # 4
SETGATE(idt[T_BOUND], 0, GD_KT, t_bound, 0);     // # 5
SETGATE(idt[T_ILLOP], 0, GD_KT, t_illop, 0);     // # 6
SETGATE(idt[T_DEVICE], 0, GD_KT, t_device, 0);   // # 7

SETGATE(idt[T_DBLFLT], 0, GD_KT, t_dblflt, 0);   // # 8

SETGATE(idt[T_TSS], 0, GD_KT, t_tss, 0);         // # 9
SETGATE(idt[T_SEGNP], 0, GD_KT, t_segnp, 0);     // # 10
SETGATE(idt[T_STACK], 0, GD_KT, t_stack, 0);     // # 11
SETGATE(idt[T_GPFLT], 0, GD_KT, t_gpflt, 0);     // # 13
SETGATE(idt[T_PGFLT], 0, GD_KT, t_pgflt, 0);     // # 14

SETGATE(idt[T_FPERR], 0, GD_KT, t_fperr, 0);     // # 15
SETGATE(idt[T_ALIGN], 0, GD_KT, t_align, 0);     // # 16
SETGATE(idt[T_MCHK], 0, GD_KT, t_mchk, 0);       // # 17
SETGATE(idt[T_SIMDERR], 0, GD_KT, t_simderr, 0);// # 18
```



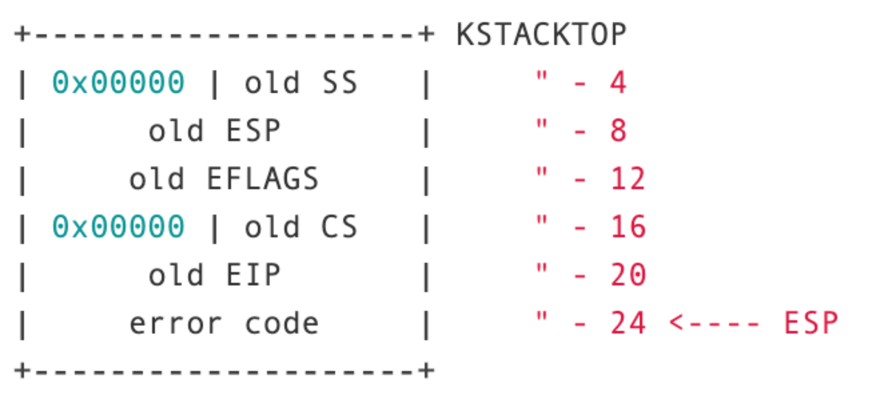
```
#define TRAPHANDLER(name, num)                                  \
    .globl name;        /* define global symbol for 'name' */  \
    .type name, @function;  /* symbol type is function */      \
    .align 2;       /* align function definition */     \
    name:           /* function starts here */      \
    pushl $(num);                       \
    jmp _alltraps
```

Interrupt, Syscall, Exceptions (Week 6, JOS Lab 3)

- Understanding the Trapframe
  - What kind of fault it is? PF
  - What is the faulting address? cr2
  - What is the reason for the fault? err
  - What is the address of instruction that causes the fault? eip
  - Which values were generated by CPU?
  - Which values were generated by JOS?
  - Which ring level it is? 3



```
TRAP frame at 0xf01c0000
  edi   0x00000000
  esi   0x00000000
  ebp   0xeebfdfd0
  oesp  0xefffffdc
  ebx   0x00000000
  edx   0x00000000
  ecx   0x00000000
  eax   0xeec00000
  es    0x----0023
  ds    0x----0023
  trap  0x0000000e Page Fault
  cr2   0x00000000
  err   0x00000004 [user, read, not-pres
  eip   0x00800039
  cs    0x----001b        3
  flag  0x00000096
    sp  0xeebfdfb8
        0x----0023
```



```
+--------------------+ KSTACKTOP
| 0x00000 | old SS   |     " - 4
|      old ESP       |     " - 8
|     old EFLAGS     |     " - 12
| 0x00000 | old CS   |     " - 16
|      old EIP       |     " - 20
|     error code     |     " - 24 <---- ESP
+--------------------+
```

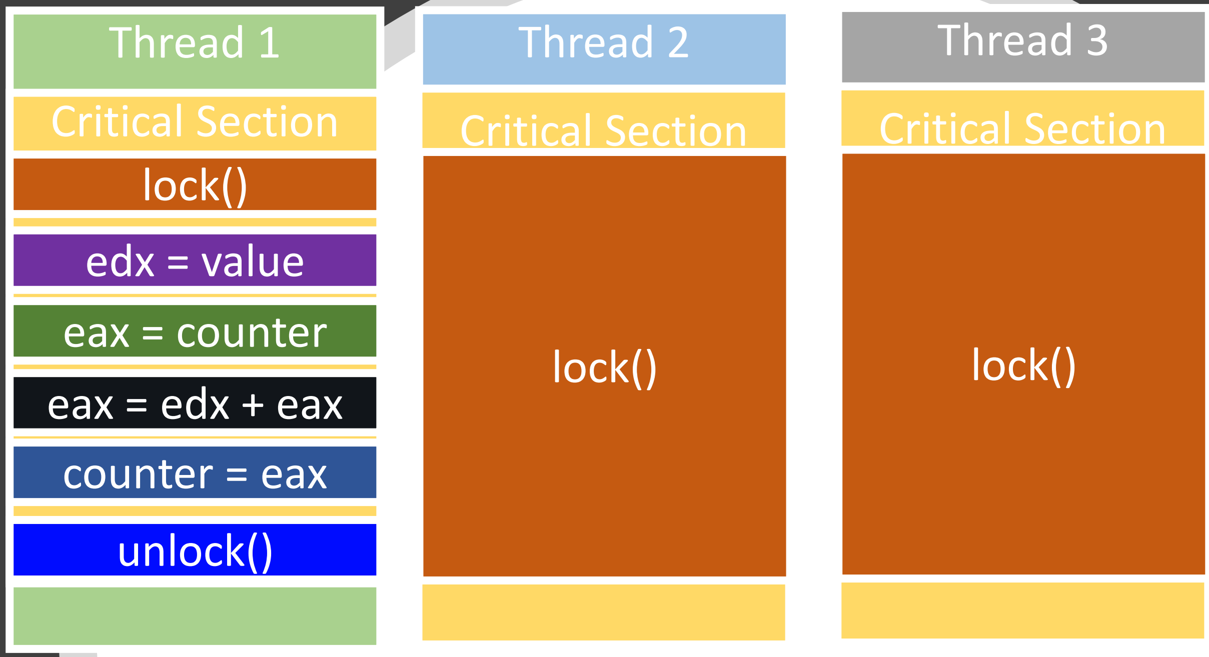
Page Fault & Copy-on-Write (Week 6, JOS Lab3&4)

- Page fault workflow
  - When does it happen?
  - How can we know the faulting address and the cause of the fault?
  - How can we resolve the fault and get back to the normal execution?

- Page fault use cases (refer to the slide of Lecture 10)
  - Automatic stack allocation
  - Copy-on-write
  - Memory Swapping

Synchronization and Locks (Week 8)

- Data racing
  - What is this?
  - Why is this bad?
    - Inconsistent/incorrect result
  - How can we resolve this?
    - Mutual exclusion
- Lock
  - How can we implement locks?
  - What’s the difference between
    - Test-and-set (atomic)
    - Test and test-and-set (atomic)
    - *Backoff



Thread 1

Critical Section

lock()

edx = value

eax = counter

eax = edx + eax

counter = eax

unlock()

Thread 2

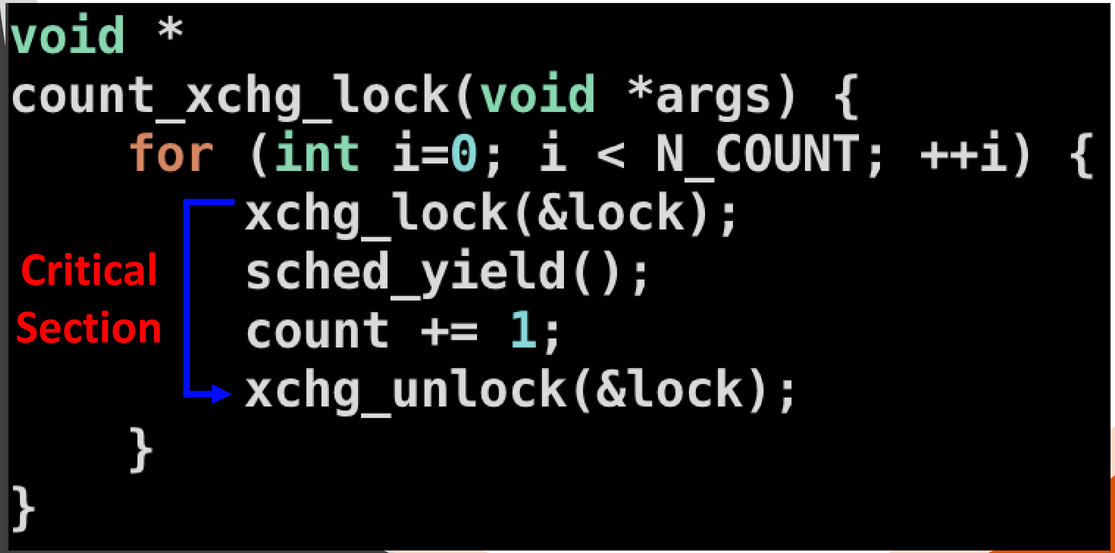
Critical Section

lock()

Thread 3

Critical Section

lock()

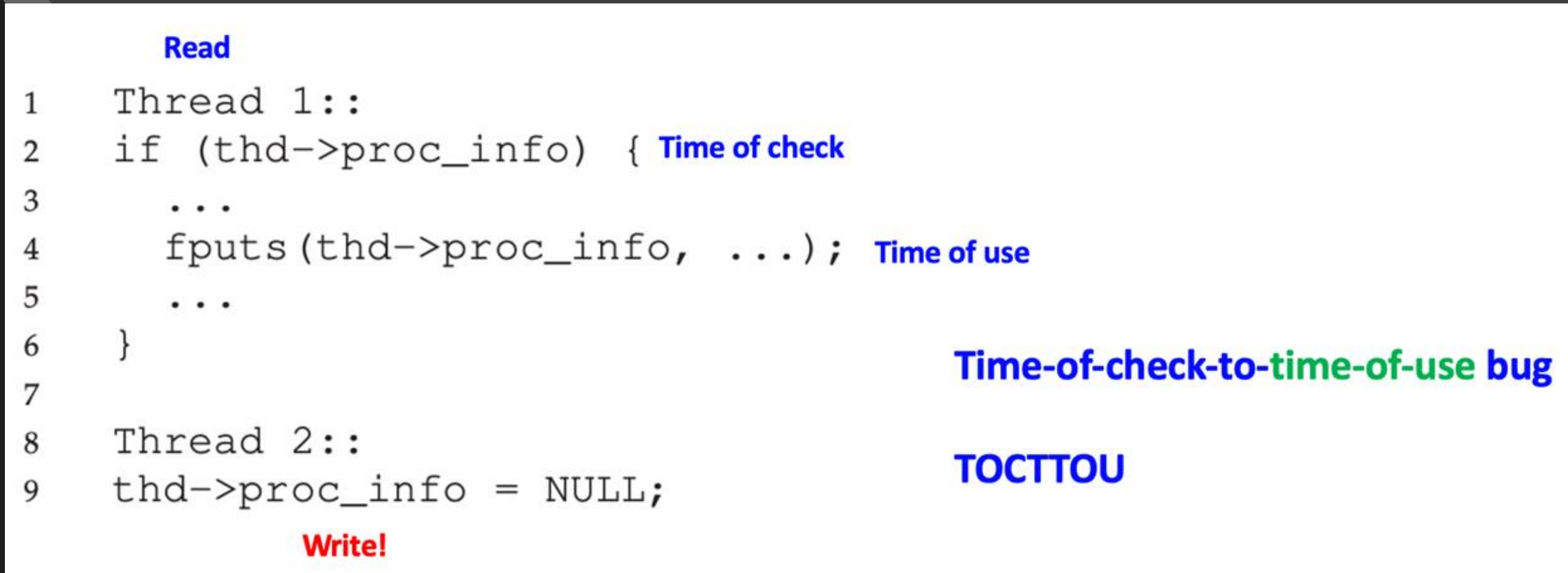


Critical Section

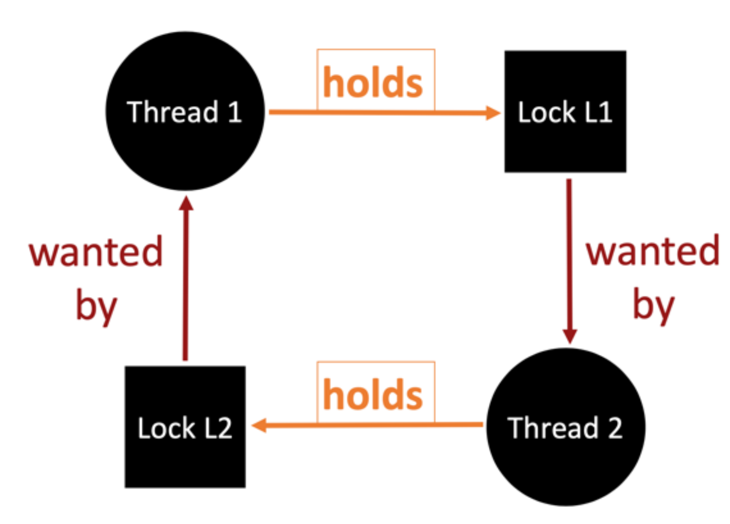
```
void *
count_xchg_lock(void *args) {
    for (int i=0; i < N_COUNT; ++i) {
        xchg_lock(&lock);
        sched_yield();
        count += 1;
        xchg_unlock(&lock);
    }
}
```

Concurrency Bugs and Deadlock (Week 9)

- TOCTTOU (Time of check to time of use) bug
    - What is this and when does it happen?
        - Another thread executes between time of check and time of use
    - How can we prevent this?
        - Use lock/unlock
- Deadlock
    - Four necessary conditions of deadlock
        - Mutual Exclusion
            - Critical section
        - Hold-and-wait
            - Meta lock
        - No Preemption
            - Unlock if fail to acquire a lock
        - Circular wait

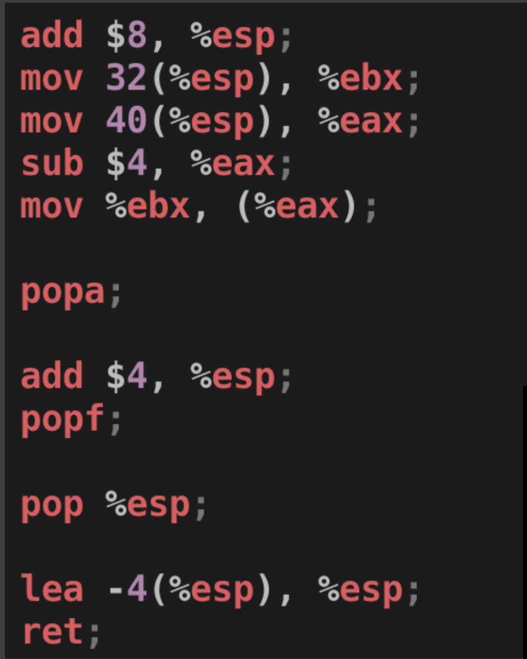


```
Thread 1::
if (thd->proc_info) {
  ...
  fputs(thd->proc_info, ...);
  ...
}

Thread 2::
thd->proc_info = NULL;
```

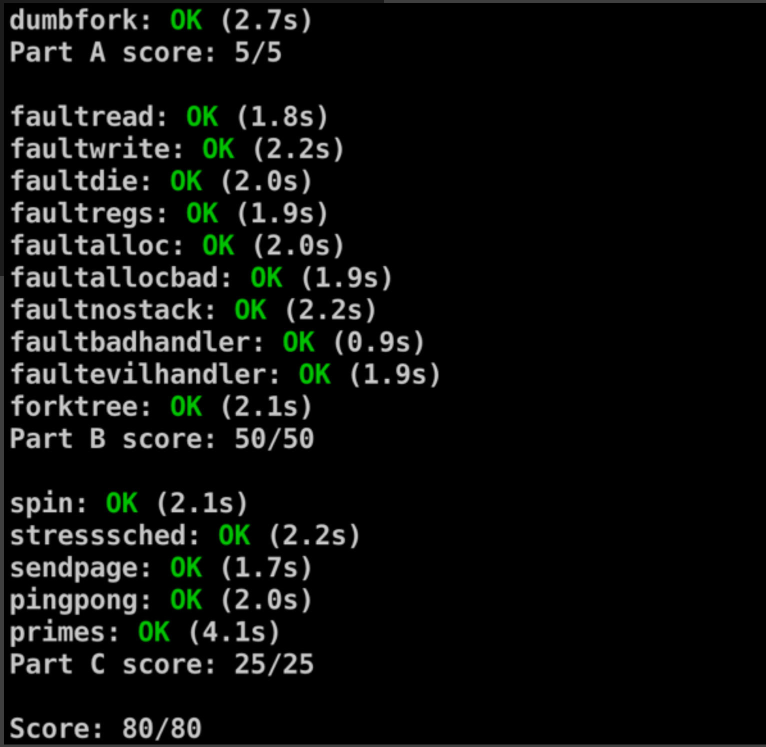


You Have Learned Many Things from CS 444/544

- How to build OS internals in a nutshell
  - Bootloader (JOS Lab 1)
  - Setting up virtual memory (JOS Lab 2)
  - Setting up interrupt handlers (JOS Lab 3a)
  - Implementing system calls (JOS Lab 3b)
  - Implementing locks (lock-example repository)
  - Implementing page fault handler and copy-on-write fork (JOS Lab 4)



```
add $8, %esp;
mov 32(%esp), %ebx;        // ebx = eip
mov 40(%esp), %eax;        // eax = esp
sub $4, %eax;              // eax = esp-4
mov %ebx, (%eax);          // *(esp-4) = eip

popa;

add $4, %esp;
popf;

pop %esp;

lea -4(%esp), %esp;
ret;
```



```
dumbfork: OK (2.7s)
Part A score: 5/5

faultread: OK (1.8s)
faultwrite: OK (2.2s)
faultdie: OK (2.0s)
faultregs: OK (1.9s)
faultalloc: OK (2.0s)
faultallocbad: OK (1.9s)
faultnostack: OK (2.2s)
faultbadhandler: OK (0.9s)
faultevilhandler: OK (1.9s)
forktree: OK (2.1s)
Part B score: 50/50

spin: OK (2.1s)
stresssched: OK (2.2s)
sendpage: OK (1.7s)
pingpong: OK (2.0s)
primes: OK (4.1s)
Part C score: 25/25

Score: 80/80
```

Oregon State University

Be Confident in Computer Systems

- Now you have experience in
  - Terminal IDE tools (tmux, git, vim, ctags, make)
  - ☆ Kernel-level (Ring 0) Debugging (via remote gdb)
  - x86 Assembly
  - Paging and address translation
  - Software/hardware interrupt and exception handling
  - Enabling preemptive multitasking
- And you wrote code for multi-core OS (pedagogical)

Final Remarks

- Thank you so much for your commitment to this course
- Submit all your work by 6/10 11:59pm
  - 100% for lab4
  - 75% for lab4 and 50% for lab1-3 if submitted by 6/12 11:59 pm
- Future improvements?