

CS 3204 Operating Systems

Lecture 11
Godmar Back



Announcements

- Thursday, Oct 2:
 - Midterm
 - No office hours
 - Project 2 help session 7-9pm, Room McB 209
- Group switch deadline is Sunday, Oct 5
 - No need to notify us if group doesn't change
 - Must notify if group did change for P2
- See forum for CVS instructions
- Reading:
 - Read carefully 2.1-2.4



CS 3204 Fall 2008

9/30/2008

2

Linking and Loading



Today: what you need to know for Project 2 (and life)

- Compiling, Linking, Loading
 - Where are my variables?
 - How are programs loaded into memory?
- Virtual Memory Basics
 - How are virtual addresses checked and how are they mapped?
 - What happens on a context-switch with respect to the MMU?



CS 3204 Fall 2008

9/30/2008

4

Accessing Information

- All information a program reads/writes is stored somewhere in memory
- Programmer uses **symbolic names**:
 - local variables, global variables, assembly constants
- CPU instructions use **virtual addresses**:
 - absolute addresses (at 0xC0000024)
 - relative addresses (at \$esp - 16)
- Actual memory uses **physical addresses**:
- Big Question: who does the translation & when?

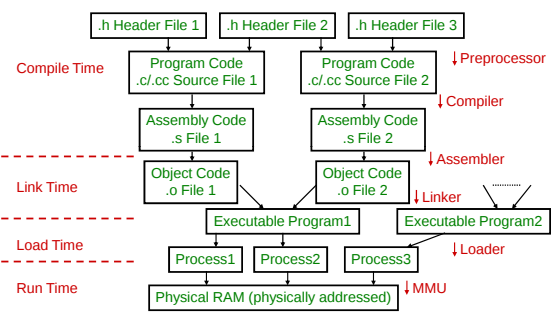


CS 3204 Fall 2008

9/30/2008

5

The Big Picture



CS 3204 Fall 2008

9/30/2008

6

Step 1: Compilation

```

int initialized_variable = 1;
int zero_initialized_variable;

int global_function(int argument)
{
    volatile int local_variable = 1;
    return argument
    + local_variable
    + initialized_variable
    + zero_initialized_variable;
}

```

whereismystuff.c

```

.data
initialized_variable:
.long 1

.comm zero_initialized_variable,4,4

global_function:
pushl %ebp
subl $16,%esp
movl %esp,%ebp
subl $1,-4(%ebp) /* local_variable */
movl -4(%ebp),%eax
addl 8(%ebp),%eax /* argument */
addl initialized_variable,%eax
addl zero_initialized_variable,%eax
leave
ret

```

whereismystuff.s

- Compiler resolves local variable names (and struct field offsets!)

CS 3204 Fall 2008 9/30/2008 7

Step 2: Assembly

```

global_function:
pushl %ebp
movl %esp,%ebp
subl $16,%esp
movl $1,-4(%ebp) /*
00000000 <global_function+0>:
0: 55      pushl %ebp
1: 89 e5    movl %esp,%ebp
3: 83 ec 10 subl $0x10,%esp
6: c7 45 fc 01 00 00 00 movl $0x1,0xffffffff(%ebp)
d: 8b 45 fc movl 0xffffffff(%ebp),%eax
10: 03 45 08 addl 0x8(%ebp),%eax
13: 03 05 00 00 00 00 addl 0x0,%eax // initialized_variable
19: 03 05 00 00 00 00 addl 0x0,%eax // zero_initialized_variable
1f: c9      leave
20: c3      ret

```

whereismystuff.o

OFFSET	TYPE	VALUE
00000015	R_386_32	initialized_variable
0000001b	R_386_32	zero_initialized_variable

Contents of section .data:

```

0000 01000000 ....

```

CS 3204 Fall 2008 9/30/2008 8

Step 3: Linking

```

0804837c <global_function>:
804837c: 55      push %ebp
804837d: 89 e5    mov %esp,%ebp
804837f: 83 ec 10 subl $0x10,%esp
8048382: c7 45 fc 01 00 00 00 movl $0x1,0xffffffff(%ebp)
8048389: 8b 45 fc movl 0xffffffff(%ebp),%eax
804838c: 03 45 08 addl 0x8(%ebp),%eax
804838f: 03 05 00 00 00 00 addl 0x0,%eax
8048395: 03 05 00 00 00 00 addl 0x0,%eax
804839b: c9      leave
804839c: c3      ret

```

whereismystuff(.exe)

- Linker resolves global addresses, stores instructions for loader in executable
 - Key: linker links multiple, independently assembled .o files into executable
 - Must decide on layout & then assign addresses & relocate

CS 3204 Fall 2008 9/30/2008 9

Step 4: Loading (Conceptual)

ELF Header:
- start address 0x080482d8

BSS:
- size & start of zero initialized data

Data:
Contents of section .data:
0804827c 00000000 00000000 00000000 00000000

Code:
Contents of section .text:
0804827c: 55 push %ebp
0804827f: 89 e5 mov %esp,%ebp
08048282: c7 45 fc 01 00 00 00 movl \$0x1,0xffffffff(%ebp)
08048289: 8b 45 fc movl 0xffffffff(%ebp),%eax
0804828c: 03 45 08 addl 0x8(%ebp),%eax
0804828f: 03 05 00 00 00 00 addl 0x0,%eax
08048295: 03 05 00 00 00 00 addl 0x0,%eax
0804829b: c9 leave
0804829c: c3 ret

Picture compiler & linker had in mind when building executable

- sections become segments

MAX_VIRTUAL

Stack

↑

Heap

↑

BSS

↑

Data

↑

Code

↑

0

start program here

Executable = set of instructions stored on disk for loader

- consists of sections

CS 3204 Fall 2008 9/30/2008 10

Virtual Memory & Paging (Simplified)

MAX_VIRTUAL

0..2³²-1
range depends on processor architecture

IA32: sizeof(void*)=4

pages

Page Table

MMU

MAX_PHYSICAL

range depends on how much RAM you bought

frames

- Maps virtual addresses to physical addresses (indirection)
 - x86: page table is called *page directory* (one per process)
 - mapping at page granularity (x86: 1 page = 4 KB = 4,096 bytes)

CS 3204 Fall 2008 9/30/2008 11

Step 5: Loading (For Real)

FFFFF000

1 GB

C0400000

C0000000

3 GB

0

P1

kheap

kbss

kdata

kcode

ustack (1)

udata (1)

ucode (1)

kernel heap

kernel bss

kernel data

kernel code

user (1)

user (1)

user (1)

kernel

kernel

kernel

free

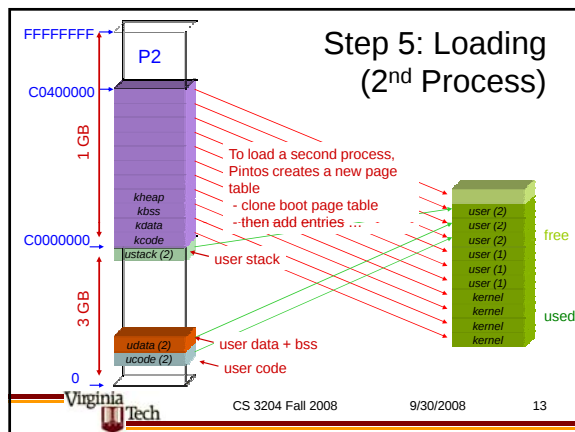
used

after Pintos boots, all physical memory is mapped at C0000000 and up

part is already used by kernel

part is free (but mapped)

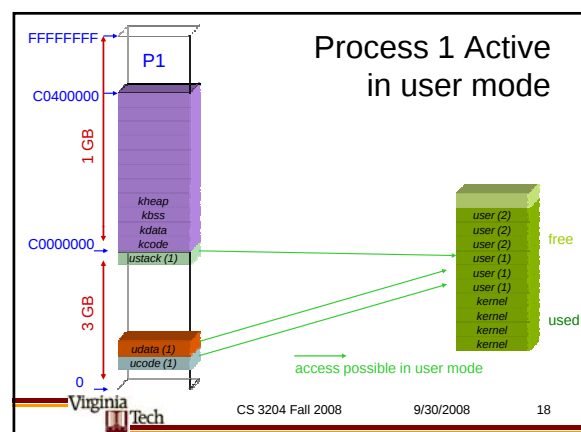
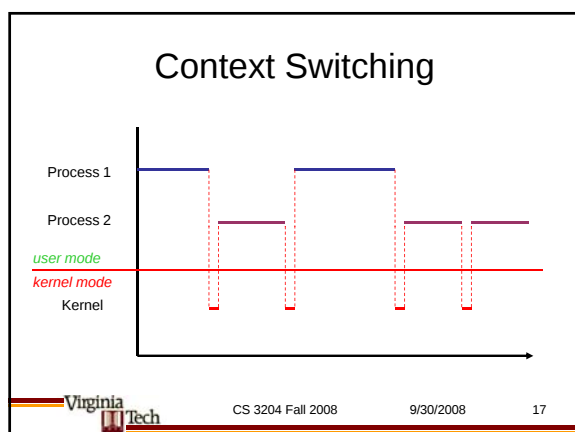
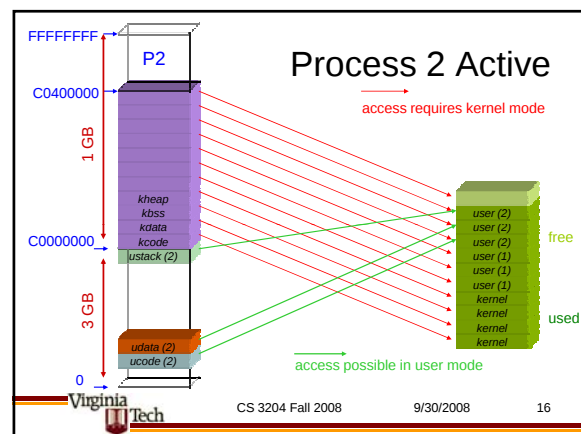
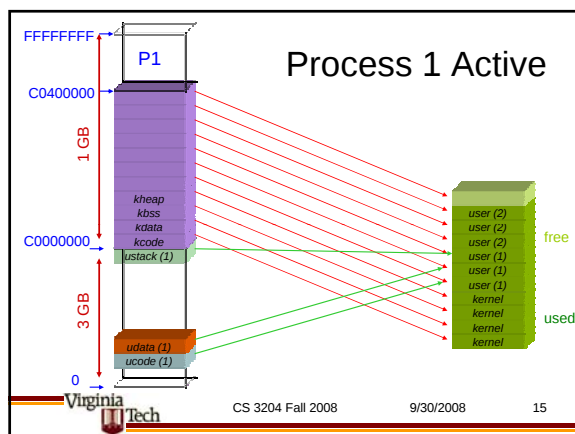
CS 3204 Fall 2008 9/30/2008 12

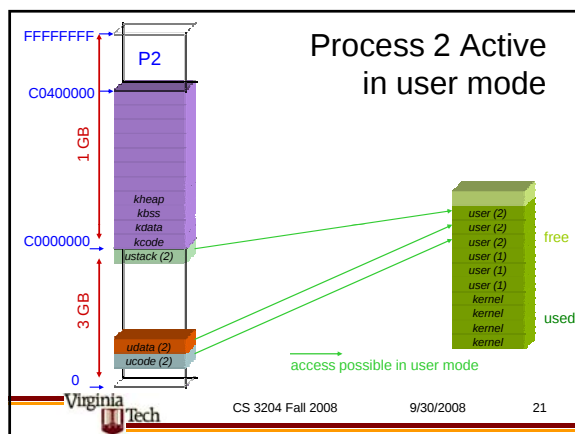
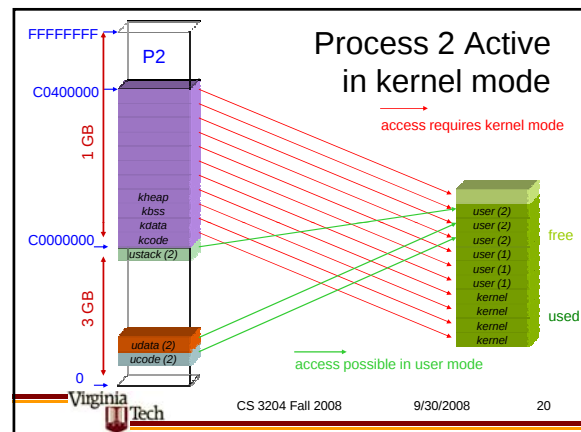
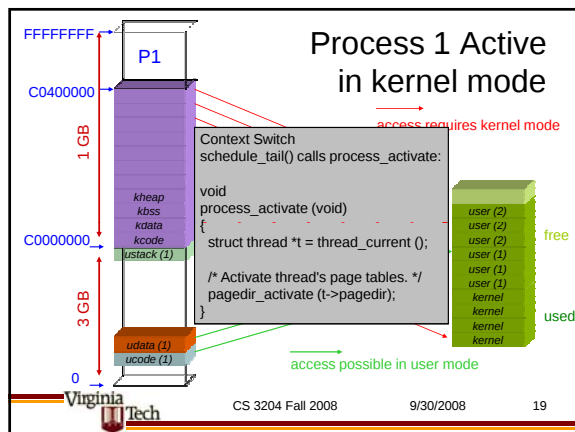


Context Switching

- Each process has its own *address space*
 - This means that the meaning of addresses (say 0x08c4000) may be different depending on which process is active
 - Maps to different page in physical memory
- When processes switch, address spaces must switch
 - MMU must be reprogrammed

Virginia Tech CS 3204 Fall 2008 9/30/2008 14





Summary

- All you have to do in Project 2/3/4 is:
 - Be aware of what memory is currently accessible
 - Your kernel executes in kernel mode, so you can access all memory (if you use >C0000000 addresses)
 - But you must set it up such that your programs can run with the memory you allow them to access (at <C0000000)
- Don't let user programs fool you into accessing other processes' (or arbitrary kernel) memory
 - Kill them if they try
- Keep track of where *stuff* is
 - Virtually (toolchain's view):
 - below C0000000 (PHYS_BASE) user space
 - above C0000000 (PHYS_BASE) kernel space
 - Physically (note: "physical" addresses are represented as 0xC0000000 + phys_addr in Pintos b/c of permanent mapping)

Virginia Tech CS 3204 Fall 2008 9/30/2008 22