CS444/544
Operating Systems II

Lecture 3
Virtual Memory
Protected mode
4/8/2024

Acknowledgement: Slides drawn heavily from Yeongjin Jiang
Odds and Ends

• Office Hours starts this week, where?
  • Recitations: in person
  • All the rest: remote/online via discord
  • Check Canvas → Office Hours page

• Lab setup and Lab1 have been posted
  • Read document
  • Watch tutorial video + read lab slides
Public Key Error

- It means that you did not setup your ssh keys correctly

- To solve it:
  - Generate ssh key pair using Lab1_slides (slide 11 & 12)
  - Add your public key to your GitHub account
Failed to bind socket: Address already in use

***
*** Use Ctrl-a x to exit qemu
***
qemu-system-i386 -nographic -drive file=obj/kern/kernel.img,index=0,media=disk,format=raw -serial mon:stdio -gdb tcp::29007 -D qemu.log
qemu-system-i386: -gdb tcp::29007: Failed to bind socket: Address already in use
make: *** [qemu-nox] Error 1

Run $ kill-qemu
kill-qemu

• This command will kill all running qemu instances that is owned by your account
• Please ignore the error message
  • It tries to kill qemu that is not owned by you, and has no effect to them
Add ~/bin to PATH in your .*/shrc

• For students who typed ‘n’ on .bashrc installation,
• Please add ~/bin to your PATH environmental variable. E.g.,
  • export PATH=$PATH:~/bin

• Alternatively, you can modify the conf/env.mk file, and set
  QEMU=~/.cs444/bin/qemu-system-i386

• This will remove the errors like

```plaintext
*** Error: Couldn't find a working QEMU executable.
*** Is the directory containing the qemu binary in your PATH
*** or have you tried setting the QEMU variable in conf/env.mk?
***
```
Device or Resource Busy...

This occurs when your tmux/vim/other apps working on some of the files that is required to be deleted by our ‘make’ script

Kill all tmux/vim sessions would remove the problem
  • Make sure that you saved all your work!
Killing tmux

• RUN
  • $ kill-all-tmux

• Killing vims
  • $ ps aux | grep vim | grep your_username_here
  • The command above will show your instance of vim
  • You can kill it selectively by running
    • $ kill -9 [pid of vim]
  • Or,
    • $ pkill vim
  • to kill all vim instances...
Some other error messages

X11 forwarding request failed on channel 0

• Please ignore this error
  • It’s about forwarding GUI applications from the server to the client
  • We don’t use GUI applications on the server

• To enable: in ~/.ssh/config, do the following:

  Host os2
    HostName os2.engr.oregonstate.edu
    User <USERNAME_HERE>
    ProxyJump access
    IdentityFile <Path_to_your_id_file>
    ForwardX11 yes
    ForwardX11Trusted yes
In Lab Tutorial...

• Following the boot sequence with ‘gdb’ in assembly and C code
  • Up to Exercise 6

• Learning how Intel x86 uses STACK to store a function’s local context
  • Exercise 10!
Recap – Real Mode

- Real mode segmentation, how?
  - seg * 16 + offset
  - [b000:b7ff] => 0xb000 * 16 + 0xb7ff = 0xbb7ff

- What is A20?
  - [f800:8001] => 0x100001?
  - [f800:8001] => 0x1?

- FYI, segment registers are:
  - %cs – code segment
  - %ds – data segment
  - %es – extra segment
  - %fs
  - %gs
  - %ss – stack segment
CPU / Registers / Memory

CPU

Registers, 1clk

- eax
- ebx
- ecx
- edx
- esi
- edi
- esp
- ebp
- eip
- cs
- ds
- es
- fs
- gs
- ss

Cache
- L1 (3clk)
- L2 (7clk)
- L3 (30clk)

MMU

200 ~ 300 clk

eax General-purpose registers

eip Hidden register. You cannot access it

Segment registers, stores CPL/RPL

M
M
U
A Simple CISC Computer (not to scale)

Main Memory Unit
- Operating System
- Device Drivers
- System Stack
- System Heap
- User Programs
- User Data
- etc.

I/O Unit
- Virtual Memory Interface
- Virtual File System Interface
- I/O Buffers
- Network interface
- etc.

CPU
- Arithmetic/Logic Unit (ALU)
- Internal Bus
  - General registers
  - Status Register
  - Control Register
  - μMemory (μPrograms)
  - μIP
  - Starting Address Generator (SAG)
  - Instruction Decoder
  - Instruction Register (IR)
  - Instruction Pointer (IP)
  - Adder
  - System Clock
- Addressing Unit
  - Memory Address Register (MAR)
  - Memory Data Register (MDR)
- Data Bus
- Address Bus
- Control Bus
- Peripheral Devices

I/O Unit
- Virtual Memory Interface
- Virtual File System Interface
- I/O Buffers
- Network interface
- etc.
Recap - JOS Boot Sequence

• 0xf000:0xffff0 – BIOS

• Loads boot sector – runs 0x7c00

• Enable A20

• Enable protected mode (enabling 4GB memory access)

• Read kernel ELF (Executable Linkable Format)

• ...


JOS Bootloader (boot/main.c)

• After enabling protected mode, boot.S will run ‘ljmpl’ (long jump, far jump) to apply the new segment assigned by the GDT.

• Then, it will call bootmain in boot/main.c

• Read kernel ELF (Executable Linkable Format)
  • https://en.wikipedia.org/wiki/Executable_and_Linkable_Format
  • Load binary program into memory
  • Read header, map memory, copy data...

• Then, run Kernel!
Need for Protected Mode: No Memory Privilege in Real Mode

- Suppose two programs run at the same time
  - Program A attempts to modify memory used by program B
  
  - **No SECURITY!**
i386 Protected Mode

• Look at GDT (Global Descriptor Table)
  • Indexed by a segment register
  • (selector)

i386 Protected Mode

- **Base**
  - Any 32-bit address

- **Limit**
  - 20-bit, but could be multiplied by 4096 bytes
  - E.g., 1 means 4096, 2 means 8192, etc.

https://wiki.osdev.org/Global_Descriptor_Table
i386 Protected Mode

• Look at GDT (Global Descriptor Table)
  • Indexed by a segment register
  • (selector)

• Retrieve base address
  • \textbf{Address} = \textbf{base} + \textbf{offset}

• Can access \textbf{if} \ (\textbf{offset} < \textbf{limit}) \ \textbf{or}
• Can access \textbf{if} \ (\textbf{offset} < \textbf{limit} \times 4096)
• Depending on the values in flags!

i386 Protected Mode

- Address 0x0008:0x00003400

- In the real mode
  - \(0x0008 \times 16 + 0x3400 = 0x3480\)

- In the i386 protected mode
  - \(GDT[1].base + 0x3400\)
    - Access ok if \(0x3400\) is less than \(GDT[1].limit\)
    - Otherwise, raise an exception!

i386 Protected Mode

- **G** - Granularity (0 = byte, 1 = page)
  - 0: Limit will be byte granularity (i.e., limit, only access $2^{20}$, 1MB)
  - 1: Limit will be page granularity (i.e., limit * $4096$, $2^{20} * 2^{12} = 2^{32}$)

- **D** – Default operand size (0 = 16-bit, 1 = 32-bit)
  - Set the values of IP/SP with respect to this bit

- **R,X** – Readable/Executable

- **DPL** – **Descriptor Privilege Level (a.k.a. Ring Level)**
  - 0 (highest priv), 1, 2, 3 (lowest priv)

For more information: [https://en.wikipedia.org/wiki/Protected_mode](https://en.wikipedia.org/wiki/Protected_mode)
A Segment

Main Memory

Program A

0x81000000

Program B

0x40200000

0x80000000
Size 1MB

0x40000000
Size 2MB

0x10:0 ~ 0x10:0x100000 are valid address for Program A
0x80000000 ~ 0x80100000

0x08:0 ~ 0x08:0x200000 are valid address for Program B
0x40000000 ~ 0x40200000

GDT index | 32-bit Base | 20-bit Limit | 12-bit Flags
---|---|---|---
16 | 0x80000000 | 0xffffffff | G=0
8 | 0x40000000 | 0x00200 | G=1
0 | 0x0 | 0x0 | G=0
Protected Mode - Examples

• 0x8:0x8080
  • Base: 0x40000000
  • Limit (addr): 0x80000000
  • Offset: 0x8080

• 0x8080 < 0x80000000

• Address: 0x40008080

<table>
<thead>
<tr>
<th>GDT index</th>
<th>32-bit Base</th>
<th>20-bit Limit</th>
<th>12-bit Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0x31310000</td>
<td>0x1000</td>
<td>G=0</td>
</tr>
<tr>
<td>8</td>
<td>0x40000000</td>
<td>0x8000</td>
<td>G=1</td>
</tr>
<tr>
<td>0</td>
<td>0x00000000</td>
<td>0x0000</td>
<td>G=0</td>
</tr>
</tbody>
</table>
Protected Mode - Examples

16

- 0x10:0x333
  - Base: 0x31310000
  - Limit (addr): 0x1000
  - Offset: 0x333
  - Address: 0x31310333

- Offset < limit

- 0x10:0x8080
  - Base: 0x31310000
  - Limit (addr): 0x1000
  - Offset: 0x8080
  - Offset > limit
  - Access denied!

<table>
<thead>
<tr>
<th>GDT index</th>
<th>32-bit Base</th>
<th>20-bit Limit</th>
<th>12-bit Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0x31310000</td>
<td>0x1000</td>
<td>G=0</td>
</tr>
<tr>
<td>8</td>
<td>0x40000000</td>
<td>0x800000</td>
<td>G=1</td>
</tr>
<tr>
<td>0</td>
<td>0x0</td>
<td>0x0</td>
<td>G=0</td>
</tr>
</tbody>
</table>
Protected Mode – Memory Privilege

• DPL (Descriptor Privilege Level)

• Protected mode – four levels of memory privilege
  • 0 (00) – highest, OS kernel
  • 1 (01) – OS kernel
  • 2 (10) – highest user-level privilege
  • 3 (11) – user-level privilege
Protected Mode – Memory Privilege

• No memory privilege in real mode

• Protected mode – four levels of memory privilege
  • 0 – highest, OS kernel
  • 1 – OS kernel
  • 2 – highest user-level privilege
  • 3 – user-level privilege

• Typically, 0 is for kernel, 3 is for user...

Descriptor Privilege Level Defines Ring Level

- CPL = Current Privilege Level
  - Defined in the last 2 bits of the %cs register
  - You can change %cs only via lcall/ljmp/trap/int

<table>
<thead>
<tr>
<th>GDT index</th>
<th>32-bit Base</th>
<th>20-bit Limit</th>
<th>12-bit Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 USER</td>
<td>0x31310000</td>
<td>0x1000</td>
<td>G=0, DPL=3</td>
</tr>
<tr>
<td>8 KERNEL</td>
<td>0x40000000</td>
<td>0x80000</td>
<td>G=1, DPL=0</td>
</tr>
<tr>
<td>0 KERNEL</td>
<td>0x0</td>
<td>0xffffffff</td>
<td>G=1, DPL=0</td>
</tr>
</tbody>
</table>
Descriptor Privilege Level Defines Ring Level

• CPL = Current Privilege Level
  • Defined in the last 2 bits of the %cs register
  • You can change %cs only via lcall/ljmp/trap/int

• Examples
  • %cs == 0x8  == 1000 in binary, last 2 bits are ZERO -> KERNEL!
  • %cs == 0x13 == 10011 in binary, last 2 bits are 3 -> USER!
  • %cs == 0x10 == 10000 in binary, last 2 bits are 0 -> KERNEL!
  • %cs == 0xb  == 1011....

<table>
<thead>
<tr>
<th>GDT index</th>
<th>32-bit Base</th>
<th>20-bit Limit</th>
<th>12-bit Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>USER</td>
<td>0x31310000</td>
<td>0x1000</td>
</tr>
<tr>
<td>8</td>
<td>KERNEL</td>
<td>0x40000000</td>
<td>0x80000</td>
</tr>
<tr>
<td>0</td>
<td>KERNEL</td>
<td>0x0</td>
<td>0xffffffff</td>
</tr>
</tbody>
</table>
Descriptor Privilege Level Defines Ring Level

- CPL = Current Privilege Level
  - Defined in the last 2 bits of the %cs register
  - You can change %cs only via lcall/ljmp/trap/int

- mov %ax, %cs \text{ impossible!}

- Can only move down...
  - CPL==0, then \text{ jmp 0x3:0x1234 is OK to execute}
  - CPL==3, then \text{ jmp 0x0:0x1234 is not allowed}

<table>
<thead>
<tr>
<th>GDT index</th>
<th>32-bit Base</th>
<th>20-bit Limit</th>
<th>12-bit Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>16  USER</td>
<td>0x31310000</td>
<td>0x1000</td>
<td>G=0, DPL=3</td>
</tr>
<tr>
<td>8   KERNEL</td>
<td>0x40000000</td>
<td>0x80000</td>
<td>G=1, DPL=0</td>
</tr>
<tr>
<td>0   KERNEL</td>
<td>0x0</td>
<td>0xffffffff</td>
<td>G=1, DPL=0</td>
</tr>
</tbody>
</table>
OK, Kernel (Ring 0) can execute code in (Ring 3) via ljmp 0x3:0x1234

• Then, how can we go back to kernel?

• We can switch from ring 0 to ring 3 via ljmp
  • ljmp 0x3:0x1234

• We cannot switch from ring 3 to ring 0 via ljmp
  • ljmp 0x0:0x1234 ← illegal instruction

• We use iret / sysexit / sysret to switch from ring 3 to ring 0
  • We will learn this in week 4
Enabling Protected Mode (part 1): Create Global Descriptor Table (GDT)

- In boot/boot.S
  - `%cs` to point `0 ~ 0xffffffff` in DPL 0
  - `%ds` to point `0 ~ 0xffffffff` in DPL 0

- Only kernel can access those two segment

```assembly
# Bootstrap GDT
.p2align 2

gdt:
    SEG_NULL       # null seg
    SEG(STA_X|STA_R, 0x0, 0xffffffff) # code seg
    SEG(STA_W, 0x0, 0xffffffff)        # data seg

.set PROTO_MODE_CSEG, 0x8        # kernel code segment selector
.set PROTO_MODE_DSEG, 0x10       # kernel data segment selector
```

<table>
<thead>
<tr>
<th>GDT index</th>
<th>32-bit Base</th>
<th>20-bit Limit</th>
<th>12-bit Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0x0</td>
<td>0xffffffff</td>
<td>G=1,W DPL=0</td>
</tr>
<tr>
<td>8</td>
<td>0x0</td>
<td>0xffffffff</td>
<td>G=1, XR DPL=0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Enabling Protected Mode (part 2): Change CR0 (Control Register 0)

Set **PE** (Protected enabled) to **1** will enable Protected Mode

In JOS:

```
lgdt gdt descr
movl %cr0, %eax
orl $CR0_PE_ON, %eax
movl %eax, %cr0
```

1. Load GDT
2. Read CR0, store it to eax
3. Set PE_ON (1) on eax
4. Put eax back to CR0 (PE_ON to CR0!!)
How to Change CPL?

- `ljmp` (instruction)
  - Long jump

```
# Jump to next instruction, but in 32-bit code segment.
# Switches processor into 32-bit mode.
ljmp  $PROT_MODE_CSEG, $protcseg

0x8 == 1000, Last 2 bits are zero..
```

```
.set PROT_MODE_CSEG, 0x8        # kernel code segment selector
.set PROT_MODE_DSEG, 0x10       # kernel data segment selector
# Bootstrap GDT
.p2align 2                       # force 4

```

```
gdt:
    SEG_NULL                      # null seg
    SEG(STA_X|STA_R, 0x0, 0xfffffffff) # code seg
    SEG(STA_W, 0x0, 0xfffffffff)    # data seg
```
Protected Mode Summary

• Segment access via GDT
  • Base + Offset < Limit * 4096 (if G == 1)
  • Base + Offset < Limit (if G == 0)

• Last two bits in %cs - CPL
  • Memory Privilege - Ring level
  • 0 for OS kernel
  • 3 for user application

• Changing CR0 to enable protected mode
  • CR0_PE_ON == 1, set via eax

• Changing CPL?
  • ljmp %cs:xxxxx, set the last 2 bits of %cs as 0 for kernel, 3 for user
Virtual Memory

• Three goals

  • Transparency

  • Efficiency

  • Protection
Uniprogramming Environment

- Run one program
- The program can use memory space freely...

```
Stack - 1
Program Data - 1
Program Code - 1
```

- OS:
  - 0x00000 ~ 0x10000
  - (0 ~ 64KB)

- Free (576 KB):
  - 0x10000 ~ 0xa0000
  - (64KB ~ 640KB)
Uniprogramming Environment

• Run one program

• The program can use memory space freely...
Uniprogramming Environment

- Run one program

- The program can use memory space freely...

<table>
<thead>
<tr>
<th></th>
<th>Free (64 KB)</th>
<th>Program Code - 1 (128KB)</th>
<th>Program Data - 1 (64 KB)</th>
<th>Stack - 1 (64KB)</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000 ~ 0x10000</td>
<td>0x10000 ~ 0x30000</td>
<td>0x40000 ~ 0x50000</td>
<td>0x80000 ~ 0x90000</td>
<td>0x50000 ~ 0x60000</td>
<td>0x00000 ~ 0x10000</td>
</tr>
<tr>
<td>0x10000 ~ 0x20000</td>
<td>0x30000 ~ 0x40000</td>
<td>0x60000 ~ 0x70000</td>
<td>0x90000 ~ 0xa0000</td>
<td>0x60000 ~ 0x70000</td>
<td>(576KB ~ 640KB)</td>
</tr>
<tr>
<td>0x20000 ~ 0x30000</td>
<td>0x50000 ~ 0x60000</td>
<td>0x70000 ~ 0x80000</td>
<td>0xa0000 ~ 0xb0000</td>
<td>0x70000 ~ 0x80000</td>
<td>(0 ~ 64KB)</td>
</tr>
<tr>
<td>0x30000 ~ 0x40000</td>
<td>0x60000 ~ 0x70000</td>
<td>0x80000 ~ 0x90000</td>
<td>0xb0000 ~ 0xc0000</td>
<td>0x80000 ~ 0x90000</td>
<td>(64KB ~ 192KB)</td>
</tr>
<tr>
<td>0x40000 ~ 0x50000</td>
<td>0x90000 ~ 0xa0000</td>
<td>0x10000 ~ 0x11000</td>
<td>0xc0000 ~ 0xd0000</td>
<td>0x90000 ~ 0xa0000</td>
<td>(320KB ~ 512KB)</td>
</tr>
<tr>
<td>0x50000 ~ 0x60000</td>
<td>0xa0000 ~ 0xb0000</td>
<td>0x11000 ~ 0x12000</td>
<td>0xd0000 ~ 0xe0000</td>
<td>0xa0000 ~ 0xb0000</td>
<td>(192KB ~ 256KB)</td>
</tr>
<tr>
<td>0x60000 ~ 0x70000</td>
<td>0xb0000 ~ 0xc0000</td>
<td>0x12000 ~ 0x13000</td>
<td>0xe0000 ~ 0xf0000</td>
<td>0xb0000 ~ 0xc0000</td>
<td>(256KB ~ 320KB)</td>
</tr>
<tr>
<td>0x70000 ~ 0x80000</td>
<td>0xc0000 ~ 0xd0000</td>
<td>0x13000 ~ 0x14000</td>
<td>0xf0000 ~ 0x10000</td>
<td>0xc0000 ~ 0xd0000</td>
<td>(512KB ~ 576KB)</td>
</tr>
<tr>
<td>0x80000 ~ 0x90000</td>
<td>0xd0000 ~ 0xe0000</td>
<td>0x14000 ~ 0x15000</td>
<td>0x10000 ~ 0x11000</td>
<td>0xd0000 ~ 0xe0000</td>
<td>(128KB)</td>
</tr>
<tr>
<td>0x90000 ~ 0xa0000</td>
<td>0xe0000 ~ 0xf0000</td>
<td>0x15000 ~ 0x16000</td>
<td>0x11000 ~ 0x12000</td>
<td>0xe0000 ~ 0xf0000</td>
<td>(64 KB)</td>
</tr>
</tbody>
</table>
## Multi-programming Environment

- Run two programs

<table>
<thead>
<tr>
<th>Stack - 2 (64KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Data - 2 (64 KB)</td>
</tr>
<tr>
<td>Program Code - 2 (128KB)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000 ~ 0x10000 (0 ~ 64KB)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program Code - 1 (128KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10000 ~ 0x30000 (64KB ~ 192KB)</td>
</tr>
<tr>
<td>Program Data - 1 (64 KB)</td>
</tr>
<tr>
<td>0x40000 ~ 0x50000 (256KB ~ 320KB)</td>
</tr>
<tr>
<td>Stack - 1 (64KB)</td>
</tr>
<tr>
<td>0x80000 ~ 0x90000 (512KB ~ 576KB)</td>
</tr>
<tr>
<td>Free (192 KB)</td>
</tr>
<tr>
<td>0x50000 ~ 0x80000 (320KB ~ 512KB)</td>
</tr>
<tr>
<td>Free (64 KB)</td>
</tr>
<tr>
<td>0x30000 ~ 0x40000 (192KB ~ 256KB)</td>
</tr>
<tr>
<td>Program Code - 2 (128KB)</td>
</tr>
<tr>
<td>0x90000 ~ 0xa0000 (576KB ~ 640KB)</td>
</tr>
<tr>
<td>Free (64 KB)</td>
</tr>
<tr>
<td>0x30000 ~ 0x40000 (192KB ~ 256KB)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program Data - 2 (64 KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40000 ~ 0x50000 (256KB ~ 320KB)</td>
</tr>
<tr>
<td>Stack - 1 (64KB)</td>
</tr>
<tr>
<td>0x80000 ~ 0x90000 (512KB ~ 576KB)</td>
</tr>
<tr>
<td>Free (192 KB)</td>
</tr>
<tr>
<td>0x50000 ~ 0x80000 (320KB ~ 512KB)</td>
</tr>
<tr>
<td>Free (64 KB)</td>
</tr>
<tr>
<td>0x30000 ~ 0x40000 (192KB ~ 256KB)</td>
</tr>
<tr>
<td>Program Code - 2 (128KB)</td>
</tr>
<tr>
<td>0x90000 ~ 0xa0000 (576KB ~ 640KB)</td>
</tr>
<tr>
<td>Free (64 KB)</td>
</tr>
<tr>
<td>0x30000 ~ 0x40000 (192KB ~ 256KB)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program Code - 1 (128KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10000 ~ 0x30000 (64KB ~ 192KB)</td>
</tr>
<tr>
<td>Program Data - 1 (64 KB)</td>
</tr>
<tr>
<td>0x40000 ~ 0x50000 (256KB ~ 320KB)</td>
</tr>
<tr>
<td>Stack - 1 (64KB)</td>
</tr>
<tr>
<td>0x80000 ~ 0x90000 (512KB ~ 576KB)</td>
</tr>
<tr>
<td>Free (192 KB)</td>
</tr>
<tr>
<td>0x50000 ~ 0x80000 (320KB ~ 512KB)</td>
</tr>
<tr>
<td>Free (64 KB)</td>
</tr>
<tr>
<td>0x30000 ~ 0x40000 (192KB ~ 256KB)</td>
</tr>
<tr>
<td>Program Code - 2 (128KB)</td>
</tr>
<tr>
<td>0x90000 ~ 0xa0000 (576KB ~ 640KB)</td>
</tr>
<tr>
<td>Free (64 KB)</td>
</tr>
<tr>
<td>0x30000 ~ 0x40000 (192KB ~ 256KB)</td>
</tr>
</tbody>
</table>
### Multi-programming Environment

- Run two programs
- System’s memory usage determines allocation
- Program need to be aware of the environment
  - Where does system loads my code?
  - You can’t determine... system does...

**No Transparency...**
Multi-programming Environment

- Run two programs

<table>
<thead>
<tr>
<th>Stack - 2 (64KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Data - 2 (64 KB)</td>
</tr>
<tr>
<td>Program Code - 2 (160KB)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program Code - 1 (160KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10000 ~ 0x38000</td>
</tr>
<tr>
<td>64KB ~ 224KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program Data - 1 (64 KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x58000 ~ 0x68000</td>
</tr>
<tr>
<td>352KB ~ 416KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack - 1 (64KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000 ~ 0x90000</td>
</tr>
<tr>
<td>64KB ~ 224KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Free (64 KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x38000 ~ 0x58000</td>
</tr>
<tr>
<td>224KB ~ 352KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Free (96 KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x68000 ~ 0x80000</td>
</tr>
<tr>
<td>416KB ~ 512KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Free (128 KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x90000 ~ 0xa0000</td>
</tr>
<tr>
<td>576KB ~ 640KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000 ~ 0x10000</td>
</tr>
<tr>
<td>0 ~ 64KB</td>
</tr>
</tbody>
</table>
Multi-programming Environment

- Run two programs
  - Program size: 64KB + 64KB + 160K = 288KB

- Free mem
  - 64 + 96 + 128 = 288KB

- Cannot run Program – 2
  - Can’t fit...

Not efficient.. Suffers memory fragmentation problem..
Multi-programming Environment

• Run two programs

• What if Program-2’s stack underflows?

• What if Program-2’s data overflows?

• Without virtual memory
  • Programs can affect to the other’s execution

Virtual Memory

• Three goals
  • Transparency: does not need to know system’s internal state
    • Program A is loaded at 0x8048000. Can Program B be loaded at 0x8048000?
  • Efficiency: do not waste memory; manage memory fragmentation
    • Can Program B (288KB) be loaded if 288 KB of memory is free, regardless of its allocation?
  • Protection: isolate program’s execution environment
    • Can we prevent an overflow from Program A from overwriting Program B’s data?
Paging

• A method of implementing virtual memory

• Split memory into multiple 4,096 byte blocks (12-bit)
  • Last 3 digits of page address are ZERO (in hexadecimal)
  • E.g., 0x0, 0x1000, 0x2000, ..., 0x8048000, 0x804a000, ..., 0x7fffe000, etc.

• Having an indirect map between virtual page and physical page
  • Set an arbitrary virtual address for a page, e.g., 0x81815000
  • Set a physical address to that page as a map, e.g., 0x32000
  • 0x81815000 ~ 0x81815fff will be translated into
  • 0x32000 ~ 0x32fff
Virtual Memory - Paging

• Having an indirect table that maps virt-addr to phys-addr

<table>
<thead>
<tr>
<th>Virtual</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048000</td>
<td>0x10000</td>
</tr>
<tr>
<td>0x8049000</td>
<td>0x11000</td>
</tr>
<tr>
<td>0x804a000</td>
<td>0x14000</td>
</tr>
<tr>
<td>0xbffdf000</td>
<td>0x12000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Paging: Virtual Memory

- Having an indirect table that maps virt-addr to phys-addr

<table>
<thead>
<tr>
<th>Virtual</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048000</td>
<td>0x10000</td>
</tr>
<tr>
<td>0x8049000</td>
<td>0x11000</td>
</tr>
<tr>
<td>0x804a000</td>
<td>0x14000</td>
</tr>
<tr>
<td>0xbffdf000</td>
<td>0x12000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Virtual-2</th>
<th>Physical-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048000</td>
<td>0x13000</td>
</tr>
<tr>
<td>0x8049000</td>
<td>0x15000</td>
</tr>
<tr>
<td>0x804a000</td>
<td>0x16000</td>
</tr>
<tr>
<td>0xbffdf000</td>
<td>0x17000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Physical Memory

- Stack-2 0x17000
- Program code-2 0x16000
- Program code-2 0x15000
- Program code-2 0x14000
- Program code-2 0x13000
- Stack 0x12000
- Program code 0x11000
- Program code 0x10000
Transparency: does not need to know system’s internal state
Program A is loaded at 0x8048000.
Can Program B be loaded at 0x8048000?

• Having an indirect table that maps virt-addr to phys-addr

<table>
<thead>
<tr>
<th>Stack-2 0xbffdf000</th>
<th>Stack 0xbffdf000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program code-2 0x804a000</td>
<td></td>
</tr>
<tr>
<td>Program code-2 0x8049000</td>
<td></td>
</tr>
<tr>
<td>Program code-2 0x8048000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Virtual</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048000</td>
<td>0x10000</td>
</tr>
<tr>
<td>0x8049000</td>
<td>0x11000</td>
</tr>
<tr>
<td>0x804a000</td>
<td>0x14000</td>
</tr>
<tr>
<td>0xbffdf000</td>
<td>0x12000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Virtual-2</th>
<th>Physical-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048000</td>
<td>0x13000</td>
</tr>
<tr>
<td>0x8049000</td>
<td>0x15000</td>
</tr>
<tr>
<td>0x804a000</td>
<td>0x16000</td>
</tr>
<tr>
<td>0xbffdf000</td>
<td>0x17000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Physical Memory

- Stack 0x17000
- Program code-2 0x16000
- Program code-2 0x15000
- Program code 0x14000
- Program code-2 0x13000
- Stack 0x12000
- Program code 0x11000
- Program code 0x10000
Efficiency: do not waste memory
Can Program B (288KB) be loaded if only 288 KB of memory is free, regardless of its allocation?

- Having an indirect table that maps virt-addr to phys-addr

<table>
<thead>
<tr>
<th>Stack-2</th>
<th>Program code-2</th>
<th>Program code-2</th>
<th>Program code-2</th>
<th>Program code-2</th>
<th>Program code-2</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xbffdf000</td>
<td>0x804a000</td>
<td>0x8049000</td>
<td>0x8048000</td>
<td>0xbffdf000</td>
<td>0x10000</td>
<td>0x17000</td>
</tr>
<tr>
<td>Program code-2</td>
<td>Program code-2</td>
<td>Program code-2</td>
<td>Program code-2</td>
<td>Program code-2</td>
<td>Program code-2</td>
<td>Stack</td>
</tr>
<tr>
<td>0x8049000</td>
<td>0x804a000</td>
<td>0xbffdf000</td>
<td>0x10000</td>
<td>0x11000</td>
<td>0x12000</td>
<td>0x16000</td>
</tr>
<tr>
<td>Virtual</td>
<td>Physical</td>
<td>Virtual-2</td>
<td>Physical-2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x8048000</td>
<td>0x10000</td>
<td>0x8048000</td>
<td>0x13000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x8049000</td>
<td>0x11000</td>
<td>0x8049000</td>
<td>0x15000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x804a000</td>
<td>0x14000</td>
<td>0x804a000</td>
<td>0x16000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xbffdf000</td>
<td>0x12000</td>
<td>0xbffdf000</td>
<td>0x17000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Protection: isolate program’s execution environment
Can we prevent an overflow from Program A from overwriting Program B’s data?

No mappings, fault!

No mappings, fault!