Assignment 3 Intro
Pointer Recap

Memory
• 4-bit addresses, i.e. address range 0 – 15

Examples
\[ c = 5; \quad *c = 'x' \]
\[ i = 4; \quad *i = 42 \]
Converting Page/Frame numbers from/to addresses

Examples
```
page_num = vaddr >> 12;
vaddr = page_num << 12;
```
Indexing off Pointers

Memory
• 4-bit addresses, i.e. address range 0 – 15

char *c;
int *i;

Examples
  c = 5; c[0] = ‘h’; c[1] = ‘i’;
  i = 4; i[0] = 42; i[2] = 7;
Assignment 3

– Page table and ‘region’ support
  • Virtual memory for applications
Theoretical Typical Address Space Layout

- Stack region is at top, and can grow down
- Heap has free space to grow up
- Text is typically read-only
- Kernel is in a reserved, protected, shared region
Real R3000 Address Space Layout

- **kuseg:**
  - 2 gigabytes
  - TLB translated (mapping loaded from page table)
  - Cacheable (depending on ‘N’ bit)
  - user-mode and kernel mode accessible
  - Page size is 4K

![Address Space Layout Diagram]
Set of translations per-user address space

- Switching processes switches the translations for kuseg

![Diagram showing address space with kseg0, kseg1, kseg2, Proc 1, Proc 2, Proc 3 with their respective addresses]
Kernel Address Space Layout

- **kseg0:**
  - 512 megabytes
  - Fixed translation window to physical memory
    - 0x80000000 - 0x9fffffff virtual = 0x00000000 - 0x1fffffff physical
    - TLB not used
  - Cacheable
  - Only kernel-mode accessible
  - Usually where the kernel code is placed
OS/161 Kernel

- Placed in Kseg0
  - lower part of physical memory
  - 16 meg of physical RAM
    - 31 busctl ramsize=16777216, in sys161.conf
    - #define PADDR_TO_KVADDR(paddr) ((paddr)+MIPS_KSEG0)
alloc_kpage()/free_kpage()

- The low-level functions that kmalloc()/kfree() use to allocate/free memory in its memory pool.
- Results are page aligned.
- Addresses are in the address range of kseg0
  - Need to convert to physical address to use as frame.
KUseg layout

- Stack region is at top, and can grow down
- Other regions determined by ELF file
  - see load_elf()
  - number can vary
  - permissions specified also
  - cs161-objdump -p testbin/huge

<table>
<thead>
<tr>
<th>Virtual Address Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
thresher% cs161-objdump -h ../bin/true

../bin/true:  file format elf32-tradbigmips

Sections:
<table>
<thead>
<tr>
<th>Idx</th>
<th>Name</th>
<th>Size</th>
<th>VMA</th>
<th>LMA</th>
<th>File off</th>
<th>Algn</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.reginfo</td>
<td>00000018</td>
<td>00400094</td>
<td>00400094</td>
<td>00000094</td>
<td>2**2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONTENTS, ALLOC, LOAD, READONLY, DATA, LINK_ONCE_SAME_SIZE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>.text</td>
<td>000001d0</td>
<td>004000b0</td>
<td>004000b0</td>
<td>000000b0</td>
<td>2**4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONTENTS, ALLOC, LOAD, READONLY, CODE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>.data</td>
<td>00000000</td>
<td>10000000</td>
<td>10000000</td>
<td>00001000</td>
<td>2**4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONTENTS, ALLOC, LOAD, DATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>.sbss</td>
<td>00000008</td>
<td>10000000</td>
<td>10000000</td>
<td>00001000</td>
<td>2**2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ALLOC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.bss</td>
<td>00000000</td>
<td>10000010</td>
<td>10000010</td>
<td>00001008</td>
<td>2**4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ALLOC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>.comment</td>
<td>00000036</td>
<td>00000000</td>
<td>00000000</td>
<td>00001008</td>
<td>2**0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONTENTS, READONLY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>.pdr</td>
<td>000004a0</td>
<td>00000000</td>
<td>00000000</td>
<td>00001040</td>
<td>2**2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONTENTS, READONLY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>.mdebug.abi32</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>000014e0</td>
<td>2**0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONTENTS, READONLY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

thresher%
thresher% cs161-objdump -p ../bin/true

../bin/true: file format elf32-tradbigmips

Program Header:
0x70000000 off 0x00000094 vaddr 0x00400094 paddr 0x00400094 align 2**2
  filesz 0x00000018 memsz 0x00000018 flags r--
LOAD off 0x00000000 vaddr 0x00400000 paddr 0x00400000 align 2**12
  filesz 0x00000280 memsz 0x00000280 flags r-x
LOAD off 0x00001000 vaddr 0x10000000 paddr 0x10000000 align 2**12
  filesz 0x00000000 memsz 0x00000010 flags rw-
private flags = 1001: [abi=O32] [mips1] [not 32bitmode]

thresher%

Zero fill fresh pages prior to mapping
Walk through load elf
Process Layout

- Process layout in KUseg
  - regions specified by calls to
    - `as_define_stack()`
    - `as_define_region()`
      - usually implemented as a linked list of region specifications
    - `as_prepare_load()`
      - make READONLY regions READWRITE for loading purposes
    - `as_complete_load()`
      - enforce READONLY again
Process Layout

- Need to keep translation table for KUSEG
2-level page table in ‘C’

paddr_t **pagetable;
• as_create()
  – allocate a data structure used to keep track of an address space
    • i.e. regions
    • proc_getas() used to get access to current address space struct
      – struct addrspace *as;

• as_destroy() (crossed out)
  – deallocate book keeping and page tables.
    • deallocate frames used
• as_copy()
  – allocates a new (destination) address space
  – adds all the same regions as source
  – roughly, for each mapped page in source
    • allocate a frame in dest
    • copy contents from source frame to dest frame
    • add PT entry for dest

• as_activate()
  – flush TLB
  – (or set the hardware asid)

• as_deactivate()
  – flush TLB
  – (or flush an asid)
VM Fault Approximate Flow Chart

- **vm_fault**
- **VM_FAULT.READONLY**
- **lookup PT**
- **Valid Translation**
- **Allocate Frame, Zero-fill, Insert PTE**
- **Load TLB**
- **Valid Region**
- **Newly allocated user-level pages are expected to be zero-filled**

- **Yes**
- **No**
kprintf()

• Do not use it in vm_fault()

  • kprintf() blocks current process while printing
    – Switches to another process
    – Context switch flushes TLB
      • Flushes what you just inserted
      • Endless loop
trace161 can help with debugging

http://cgi.cse.unsw.edu.au/~cs3231/06s1/os161/man/sys161/index.html

• The following additional options control trace161's tracing and are ignored by sys161:

• -f tracefile
  – Set the file trace information is logged to. By default, stderr is used. Specifying -f- sends output to stdout instead of stderr.

• -t traceflags
  – Tell System/161 what to trace. The following flags are available:
    • d Trace disk I/O
    • e Trace emufs I/O
    • j Trace jumps and branches
    • k Trace instructions in kernel mode
    • n Trace network I/O
    • t Trace TLB/MMU activity
    • u Trace instructions in user mode
    • x Trace exceptions

• Caution: tracing instructions generates huge amounts of output that may overwhelm smaller host systems.
wagner% trace161 -tt kernel
sys161: System/161 release 2.0.8, compiled Feb 19 2017 14:31:56
sys161: Tracing enabled: tlb
trace: 00 tlbp:  81000/000 -> 00000 ----: [0]
trace: 00 tlbp:  81001/000 -> 00000 ----: [1]
trace: 00 tlbp:  81002/000 -> 00000 ----: [2]
trace: 00 tlbp:  81003/000 -> 00000 ----: [3]
trace: 00 tlbp:  81004/000 -> 00000 ----: [4]
trace: 00 tlbp:  81005/000 -> 00000 ----: [5]
trace: 00 tlbp:  81006/000 -> 00000 ----: [6]
trace: 00 tlbp:  81007/000 -> 00000 ----: [7]
trace: 00 tlbp:  81008/000 -> 00000 ----: [8]
trace: 00 tlbp:  81009/000 -> 00000 ----: [9]
trace: 00 tlbp:  8100a/000 -> 00000 ----: [10]
trace: 00 tlbp:  8100b/000 -> 00000 ----: [11]
trace: 00 tlbp:  8100c/000 -> 00000 ----: [12]
trace: 00 tlbp:  8100d/000 -> 00000 ----: [13]
trace: 00 tlbp:  8100e/000 -> 00000 ----: [14]
trace: 00 tlbp:  8100f/000 -> 00000 ----: [15]
trace: 00 tlbp:  81010/000 -> 00000 ----: [16]
trace: 00 tlbp:  81011/000 -> 00000 ----: [17]
trace: 00 tlbp:  81012/000 -> 00000 ----: [18]
trace: 00 tlbp:  81013/000 -> 00000 ----: [19]
trace: 00 tlbp:  81014/000 -> 00000 ----: [20]
..........
trace: 00 tlbp: 8103f/000 -> 00000 ----: [63]
trace: 00 tlbp: 81040/000 -> NOT FOUND
trace: 00 tlbwi: [ 0] 81000/000 -> 00000 ---- ==> 81040/000 -> 00000 ----
trace: 00 tlbp: 81001/000 -> NOT FOUND
trace: 00 tlbwi: [ 1] 81001/000 -> 00000 ---- ==> 81041/000 -> 00000 ----
trace: 00 tlbp: 81002/000 -> NOT FOUND
trace: 00 tlbwi: [ 2] 81002/000 -> 00000 ---- ==> 81042/000 -> 00000 ----
trace: 00 tlbp: 81003/000 -> NOT FOUND
trace: 00 tlbwi: [ 3] 81003/000 -> 00000 ---- ==> 81043/000 -> 00000 ----
trace: 00 tlbp: 81004/000 -> NOT FOUND
trace: 00 tlbwi: [ 4] 81004/000 -> 00000 ---- ==> 81044/000 -> 00000 ----
trace: 00 tlbp: 81005/000 -> NOT FOUND
trace: 00 tlbwi: [ 5] 81005/000 -> 00000 ---- ==> 81045/000 -> 00000 ----
trace: 00 tlbp: 81006/000 -> NOT FOUND
trace: 00 tlbwi: [ 6] 81006/000 -> 00000 ---- ==> 81046/000 -> 00000 ----
trace: 00 tlbp: 81007/000 -> NOT FOUND
trace: 00 tlbwi: [ 7] 81007/000 -> 00000 ---- ==> 81047/000 -> 00000 ----
trace: 00 tlbp: 81008/000 -> NOT FOUND
trace: 00 tlbwi: [ 8] 81008/000 -> 00000 ---- ==> 81048/000 -> 00000 ----
trace: 00 tlbwi: [60] 8103c/000 -> 00000 ---- ==> 8107c/000 -> 00000 ----
trace: 00 tlbp: 8107d/000 -> NOT FOUND
trace: 00 tlbwi: [61] 8103d/000 -> 00000 ---- ==> 8107d/000 -> 00000 ----
trace: 00 tlbp: 8107e/000 -> NOT FOUND
trace: 00 tlbwi: [62] 8103e/000 -> 00000 ---- ==> 8107e/000 -> 00000 ----
trace: 00 tlbp: 8107f/000 -> NOT FOUND
trace: 00 tlbwi: [63] 8103f/000 -> 00000 ---- ==> 8107f/000 -> 00000 ----

OS/161 base system version 2.0.3
(with locks/CVs, system calls solutions)

President and Fellows of Harvard College. All rights reserved.

Put-your-group-name-here's system version 0 (ASST3 #29)

16208k physical memory available
Device probe...
lamebus0 (system main bus)
emu0 at lamebus0
End of trace from bin/true

trace: 00 tlblookup: 00400/000 -> no match
trace: 00 tlbwr: [58] 8003a/000 -> 00000 ---- ==> 00400/000 -> 00034 -V--
trace: 00 tlblookup: 00400/000 -> 00034 -V--: [58] - OK
trace: 00 tlblookup: 00410/000 -> no match
trace: 00 tlbwr: [34] 80022/000 -> 00000 ---- ==> 00410/000 -> 00036 -VD-
trace: 00 tlblookup: 00410/000 -> 00036 -VD-: [34] - OK
trace: 00 tlblookup: 00400/000 -> 00034 -V--: [58] - OK
trace: 00 tlblookup: 00410/000 -> 00036 -VD-: [34] - OK
trace: 00 tlblookup: 00400/000 -> 00034 -V--: [58] - OK
trace: 00 tlblookup: 7ffff/000 -> 00035 -VD-: [25] - OK
trace: 00 tlblookup: 00400/000 -> 00034 -V--: [58] - OK
trace: 00 tlblookup: 00400/000 -> 00034 -V--: [58] - OK
TLB refill

- Use tlb_random()
- Cost of book keeping to do something smarter costs more than potential benefit
Disable interrupts when writing to the TLB in vm_fault!

`spl = splhigh();
tlb_random(entry_hi, entry_lo);
splx(spl);`
Advance Assignment

- Shared pages and copy-on-write
- Sbrk()
- Demand loading and mmap
- Paging
Two (or more) processes running the same program and sharing a section

Proc 1 Address Space

Proc 2 Address Space

Page Table

Page Table

Physical Address Space
**COW**

- `fork()` can be more efficient
- `as_copy` is underlying routine
- `set pages read_only`
  - Keep reference count in frame table
  - On write-fault, `vm_fault` copies, decrement count.
sbrk

- The "break" is the end address of a process's heap region.
- The sbrk call adjusts the "break" by the amount.
- It returns the old "break". Thus, to determine the current "break", call sbrk(0).

- The heap region is initially empty, so at process startup, the beginning of the heap region is the same as the end and may thus be retrieved using sbrk(0).
Memory-mapped files and paging

Memory mapped file

Disk

Physical Address Space
mmap semantics

```c
void *mmap(size_t length, int prot, int fd, off_t offset);
int munmap(void *addr);
```