Boring Stuff

- Shell Lab due tomorrow Tuesday, 29 Oct 2013
  - Your first concurrency assignment!
  - If you like process level concurrency, take OS 😊
- Malloc Lab out the same time Shell Lab is due
  - My favorite lab!
  - Design and implement a memory allocator
  - Start early
- Pressing concerns?
Menu for Today

- Advice, not Motivation
- The Complete Picture
- Virtual Memory
- Address Translation
- Extra: C Primer
“To use this process, a programmer explains code to an inanimate object, such as a rubber duck, with the expectation that upon reaching a piece of incorrect code and trying to explain it, the programmer will notice the error.”
THE COMPLETE PICTURE

#include <stdio.h>

int main(int argc, char **argv) {
    int *val = NULL;
    *val = 0;
    printf("This is %d\n", *val);
    return 0;
}

VM: Problems with Direct Mapping

Questions to ponder:
- How can we grow processes safely?
- What to do about fragmentation?
- How can we make large contiguous chunks fit easier?

Direct Mapping Fragmentation

<table>
<thead>
<tr>
<th>Process 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Process 2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Process 3</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Process 4</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Process 5</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Process 6</td>
</tr>
</tbody>
</table>
HOW DO WE SOLVE THESE PROBLEMS?

- We are scientists (and engineers)...
  - Insert a level of indirection
VIRTUAL MEMORY

..Is the Best Thing Ever™
  • Demand paging
  • Memory Management
  • Protection

Allows the illusion of infinite memory
  • Kernel manages page faults

Each process gets its own virtual address space
  • Mapping is the heart of virtual memory
VM OF A LINUX PROCESS

Different for each process

- Process-specific data structs (ptables, task and mm structs, kernel stack)

Identical for each process

- Physical memory
- Kernel code and data
- User stack
- Memory mapped region for shared libraries
- Runtime heap (malloc)
- Uninitialized data (.bss)
- Initialized data (.data)
- Program text (.text)

Kernel virtual memory

Process virtual memory

0x08048000 (32)
0x00400000 (64)
VM: Address Translations

Virtual address

Virtual page number (VPN)  Virtual page offset (VPO)

Physical address

Physical page number (PPN)  Physical page offset (PPO)

Page table

Valid  Physical page number (PPN)

Valid bit = 0: page not in memory (page fault)

Page table address for process

Page table base register (PTBR)
Overview of a Hit

CPU Chip

1. VA

CPU

MMU

2. PTEA
3. PTE
4. PA

Cache/Memory

Data

5.
**Two-Level Page Table**

**Level 1 page table**
- PTE 0
- PTE 1
- PTE 2 (null)
- PTE 3 (null)
- PTE 4 (null)
- PTE 5 (null)
- PTE 6 (null)
- PTE 7 (null)
- PTE 8
- (1K - 9) null PTEs

**Level 2 page tables**
- PTE 0
- ... (null)
- PTE 1023
- PTE 0
- ... (null)
- PTE 1023
- 1023 null PTEs
- PTE 1023

**Virtual memory**
- VP 0
- ... (null)
- VP 1023
- VP 1024
- ... (null)
- VP 2047
- Gap
- 1023 unallocated pages
- VP 9215
- ...

32 bit addresses, 4KB pages, 4-byte PTEs
TRANSLATING W/ A K-LEVEL PAGE TABLE

VIRTUAL ADDRESS

PHYSICAL ADDRESS
But Memory Accesses are Slow

- At least 2 memory accesses
  - Fetch page-table entry (PTE) from memory
  - Then fetch data from memory
- In x86, 3 memory accesses
  - Page directory, page table, physical memory
- In x86_64, 4 level page-mapping system
- What should we do?
  - Please don’t say insert a level of “indirection”
**Translation Lookaside Buffer (TLB)**

- Super fast hardware cache of PTEs
- Idea: Locality exists between memory accesses
  - Typically access nearby memory
  - Usually on the same page as current data
    - Arrays with loops
    - Program instructions
**VM: TRANSLATIONS w/ TLB AND TABLES**

**Virtual address**

- Virtual page number (VPN)
- Virtual page offset (VPO)

**Page table**

- Valid
- Physical page number (PPN)

**Physical address**

- Physical page number (PPN)
- Physical page offset (PPO)

**Translation Lookaside Buffer (TLB)**

- TLB Hit: Fetch straight from TLB
- TLB Miss: Do a page walk to fetch the entry

**Page table address for process**

- Page table base register (PTBR)

- Virtual address
  - $n-1$
  - $p$
  - $p-1$
  - $0$

- Physical address
  - $m-1$
  - $p$
  - $p-1$
  - $0$
OVERVIEW OF A TLB HIT

CPU Chip

CPU

MMU

TLB

VPN

PTE

VA

PA

Cache/Memory

Data

1

2

3

4

5
OVERVIEW OF A TLB MISS

CPU Chip

CPU

MMU

TLB

VPN

PTE

PTEA

PA

Cache/ Memory

VA

Data
WORDS AND SYMBOLS

- **Basic Parameters**
  - $N = 2^n$: Number of addresses in virtual address space
  - $M = 2^m$: Number of addresses in physical address space
  - $P = 2^p$: Page size (bytes)

- **Components of the virtual address (VA)**
  - TLBI: TLB index
  - TLBT: TLB tag
  - VPO: Virtual page offset
  - VPN: Virtual page number

- **Components of a Physical Address**
  - PPO: Physical page offset (same as VPO)
  - PPN: Physical page number
  - CO: Byte offset within cache line
  - CI: Cache index
  - CT: Cache tag
EXTRA WORDS

- **PDE: Page Directory Entry**
  - Entry in first level page table in a 2-level structure

- **PTE: Page Table Entry**
  - Entry in a page table

- **PTI/PDI: Page table/ Page Directory index**
  - The index into the page table for an entry

- **Page Aligned**
  - Address is a multiple of the page size
    - Page directory/ Page Table Base addresses are page aligned
**EQUATIONS**

- # virtual address bits = \( \log_2(\text{virtual address size}) \)
- # physical address bits = \( \log_2(\text{physical address size}) \)
- # bits in VPO = \( \log_2(\text{page size}) \)
- # bits in VPN = (\# VA bits) - \( \log_2(\text{page size}) \)

Depending on the (problem) these two need to be aligned:
- PDE Address = Base Address + sizeof(PDE)*\( \text{(PDI)} \)
- PTE Address = PDE[PDI] + sizeof(PTE)*\( \text{(PTI)} \)
TUTORIAL: VIRTUAL ADDRESS TRANSLATION

- **Addressing**
  - 32 bit virtual address
  - 32 bit physical address
  - Page size = 4 kb

- **Paging**
  - 10 bit page directory index
  - 10 bit page table index
  - 12 bit offset

- **TLB**
  - Direct Mapped
  - 4 entries

Diagram:

- TLB Tag
- TLBT
- TLBI
- VPO/PPO
- PDI
- PTI
- PDI
- VPO/PPO
Always access TLB first
**TUTORIAL: ADDRESS TRANSLATION HIT**

VPO and PPO are always the same!

- **TLBT**: 3A7AE
- **TLBI**: 0
- **VPO**: F00

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3A7AE</td>
<td>5F7F7</td>
<td>1</td>
</tr>
</tbody>
</table>

**Physical Page Number**

**Physical Page Offset**
TUTORIAL: ADDRESS TRANSLATION MISS

TLBT  TLBI  VPO

3B8AC  3  BEE

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3B8AC</td>
<td>DEAD</td>
<td>0</td>
</tr>
</tbody>
</table>

- TLB Miss! Do page walk
# Tutorial: Address Translation Miss

<table>
<thead>
<tr>
<th>Page Directory Index</th>
<th>Page Table Base Address</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3B8</td>
<td>0xFAFF8034</td>
<td>1</td>
</tr>
<tr>
<td>0x3B9</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>...</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>
# Tutorial: Address Translation Miss

<table>
<thead>
<tr>
<th>PDI</th>
<th>PTA</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>0x3B8</td>
<td>0xFAFF8035</td>
<td>1</td>
</tr>
<tr>
<td>0x3B9</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

| ..    | ..        | ..    |

<table>
<thead>
<tr>
<th>PDI</th>
<th>PTA</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFAFF9000</td>
<td>PTI</td>
<td>PPN</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>...</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

| 0xFAFF8000 | PTI | PPN | Valid |
| ...        | ... | ... |     |
| ...        | ... | ... |     |
| ...        | ... | ... |     |

| 0xFAFF8000 | PTI | PPN | Valid |
| ...        | ... | ... |     |
| ...        | ... | ... |     |
| ...        | ... | ... |     |

| 0xFAFF8000 | PTI | PPN | Valid |
| ...        | ... | ... |     |
| ...        | ... | ... |     |
| ...        | ... | ... |     |

| 0xFAFF8000 | PTI | PPN | Valid |
| ...        | ... | ... |     |
| ...        | ... | ... |     |
| ...        | ... | ... |     |

The diagram illustrates the address translation process, showing how the starting address (0x3B8) leads to the PDI, PTI, and VPO in the translation table. The table entries show the mapping between PDI and PTA, with 0x3B8 mapping to 0xFAFF8035. The translation miss is indicated by the absence of a valid entry for the requested address.
**TUTORIAL: ADDRESS TRANSLATION MISS**

<table>
<thead>
<tr>
<th>Page Table Index</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2B2</td>
<td>0x2D00D</td>
<td>1</td>
</tr>
<tr>
<td>0x2B3</td>
<td>0x2D00D</td>
<td>1</td>
</tr>
</tbody>
</table>

- **PDI**: 3B8
- **PTI**: 2B3
- **VPO**: BEE

Page Table Index: 0x2B2
PPN: 0x2D00D
Valid: 1

Page Table Index: 0x2B3
PPN: 0x2D00D
Valid: 1
TUTORIAL: ADDRESS TRANSLATION MISS

Physical Page Number (PPN) = 0x2D00D
Physical Page Offset (PPO) = 0x2B3
VPO = 0x2B8
PDI = 0x2B3
PTI = 0x2B3
Valid = 1

PTI | PPN | Valid
---|-----|-----
0x2B3 | 0x2D00D | 1
EVERY BIT COUNTS

![Diagram of memory management with page directory and page table entries. The diagram illustrates various flags and bits for present, writable, user, write-through, cache disable, accessed, dirty, available for systems programmer use, and the page frame address 31..12. Additionally, there are reserved fields by Intel Corporation (should be zero).]
TRANSLATION MACRO EXERCISE

- 32 bit address: 10 bit VPN1, 10 bit VPN2, 12 bit VPO
- 4KB pages
- Define the following function like macros:
  - Page align
    #define PAGE_ALIGN(v_addr) _________________________________
  - Gets VPN1/VPN2 as unsigned int from virtual address
    #define VPN1(v_addr) _______________________________________
    #define VPN2(v_addr) _______________________________________
  - Gets VPO as unsigned int from virtual address
    #define VPO(v_addr) ________________________________________
  - Calculates the address of the page directory index
    #define PDEA(pd_addr, v_addr) _________________________________
  - Calculate address of page table entry
    #define PTEA(pd_addr, v_addr)_______________________________
  - Calculate physical address
    #define PA(pd_addr, v_addr) _________________________________
TRANSLATION MACRO SOLUTION

- 32 bit address: 10 bit VPN1, 10 bit VPN2, 12 bit VPO
- 4KB pages
- Define the following function like macros:
  - Page align
    ```c
    #define PAGE_ALIGN(v_addr) ((unsigned int) v_addr & ~0xfff)
    ```
  - Gets VPN1/VPN2 as unsigned int from virtual address
    ```c
    #define VPN1(v_addr) ((unsigned int) (((v_addr)>>22)&0x3ff))
    #define VPN2(v_addr) ((unsigned int) (((v_addr)>>12)&0x3ff))
    ```
  - Gets VPO as unsigned int from virtual address
    ```c
    #define VPO(v_addr) ((unsigned int) ((v_addr)&0x3fff))
    ```
  - Calculates the address of the page directory index
    ```c
    #define PDEA(pd_addr, v_addr) (((void **)pd_addr)+VPN1(v_addr))
    ```
  - Calculate address of page table entry
    ```c
    #define PTEA(pd_addr, v_addr)
    (((void **)PAGE_ALIGN(*PDEA(pd_addr, v_addr)))+VPN2(v_addr))
    ```
  - Calculate physical address
    ```c
    #define PA(pd_addr, v_addr)
    (((PAGE_ALIGN(*PTEA(pd_addr, v_addr)))) | VPO(v_addr))
    ```
EXTRA STUFF

- For next week, or for your enjoyment
ALL THE C!

“Saving you from malloc misery…”
Basics
Useful C Stuff
Debugging

C AND POINTER BASICS

- Statically allocated arrays:
  - `int prices[100];`
  - Get rid of magic numbers:
    - `int prices[NUMITEMS];`

- Dynamically allocated arrays:
  - `int *prices2 = (int *) malloc(sizeof(int) * var);`

- Which is valid:
  - `prices2 = prices;`
  - `prices = prices2;`

- The & operator:
  - `&prices[1]` is the same as `prices+1`

- Function Pointer:
  - `int (*fun)();`
  - Pointer to function returning int
Peeling the Onion (K&R p.101)

- **char **argv**
  - argv: pointer to a pointer to a char

- **int (*daytab)[13]**
  - daytab: pointer to array[13] of int

- **int *daytab[13]**
  - daytab: array[13] of pointer to int

- **char (*(*x())[])()**
  - x: function returning pointer to array[] of pointer to function returning char

- **char (*(*x[3]()[])[5])**

Takeaway
- There is an algorithm to decode this (see K&R p. 101)
- Always use parenthesis!!
Why Typedefs?

- For convenience and readable code
- Example:
  - `typedef struct {
    int x;
    int y;
  } point;`
- Function Pointer example:
  - `typedef int(*pt2Func)(int, int);`
  - `pt2Func` is a pointer to a function that takes 2 int arguments and returns an int
MACROS ARE COOL

- C Preprocessor looks at macros during the preprocessing step of compilation
- Use `#define` to avoid magic numbers:
  - `#define TRIALS 100`
- Function like macros – short and heavily used code snippets
  - `#define GET_BYTE_ONE(x) ((x) & 0xff)`
  - `#define GET_BYTE_TWO(x) ((x) >> 8) & 0xff)`
- Also look at inline functions (example prototype):
  - `inline int max(int a, int b)`
  - Requests compiler to insert assembly of max wherever a call to max is made
- Both useful for malloc lab
DEBUGGING – FAVORITE METHODS

- Using the DEBUG flag:
  - `#define DEBUG`

    ```
    #ifdef DEBUG
    #define dbg_printf(...) printf(__VA_ARGS__)
    #else
    #define dbg_printf(...) 
    #endif
    ```

- Compiling (if you want to debug):
  - `gcc -DDEBUG foo.c -o foo`

- Using assert
  - `assert(posvar > 0);`
  - `man 3 assert`

- Compiling (if you want to turn off asserts):
  - `gcc -DNDEBUG foo.c -o foo`
Little Things

- Usage messages
  - Putting this in is a good habit – allows you to add features while keeping the user up to date
  - `man -h`

- `fopen/fclose`
  - Always error check!

- `malloc()`
  - Error check
  - Free everything you allocate

- Global variables
  - Namespace pollution
  - If you must, make them private:
    - `static int foo;`
Questions and References Slide

- Rubber Duck 1
- Rubber Duck Debugging on Wiki
- Florentijn Hofman’s Duck
- Indirection on Wiki
- Pictures stolen from lecture slides
- Stole from 15-410 Virtual Memory Slides
  - Lectures reside here
  - BTW, Prof. Eckhardt is super cool
- IA32 Page Directory/Table Bits