C Primer and Virtual Memory

15-213 / 18-213: Introduction to Computer Systems
10th Recitation, Oct. 31, 2011

Slides by: Pranav Senthilnathan (psenthi1)
Today

- Shell Lab
- C Primer – Basics of C, style and conventions
- Virtual Memory
- Malloc Lab
Shell Lab

- Due Thursday, November 3!!!
- Don’t worry about trace 10
Today

- Shell Lab
- C Primer – Basics of C, style and conventions
- Virtual Memory
- Malloc Lab
C Primer – Basics of C, style and conventions

- Saving you from malloc misery
- Basics
- Useful language features
- Debugging conventions
- The C Programming Language by Kernighan and Ritchie
Basics (just review)

- **Statically allocated arrays:**
  - `int prices[100];`
  - Getting 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
Pg 101 K&R

- `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 pg. 101)
  - Always use parenthesis!!
  - Typedef
Typedef

- For convenience and readable code

- Example:
  ```c
  typedef struct {
    int x;
    int y;
  } point;
  ```

- Function Pointer example:
  ```c
  typedef int (*pt2Func)(int, int);
  ```
  `pt2Func` is a pointer to a function that takes 2 int arguments and returns an int
Macros

- C Preprocessor looks at macros in 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 fun(int a, int b)`
  - Requests compiler to insert assembly of max wherever a call to max is made
- Both useful for malloc
Debugging

- **Using the DEBUG flag:**
  - `#define DEBUG
   ...
   #ifdef DEBUG
   .. // debugging print statements, etc.
   #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`
Other stuff

- **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;`
Today

- Shell Lab
- C Primer – Basics of C, style and conventions
- Virtual Memory
- Malloc Lab
Recall: Address Translation With a Page Table

**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)

Page table base register (PTBR)

Page table address for process

Valid bit = 0: page not in memory (page fault)
Translating with a k-level Page Table
Example
- 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)
    ______________________________________________________
    ```
Example

- **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)&0xfff))
  ```

- **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))
  ```
Today

- Shell Lab
- C Primer – Basics of C, style and conventions
- Virtual Memory
- Malloc Lab
Malloc Lab

- Lab goes out Thursday November 3, 2011
- Due Thursday November 17, 2011
- As always, read the documentation
- Start early
Today

- Shell Lab
- C Primer – Basics of C, style and conventions
- Virtual Memory
- Malloc Lab
- Happy Halloween!