CS246
Unix Directories, sort
2d Arrays, working with pointers

March 1

Typical Unix directories

In Unix everything looks like a file and is treated like a file!!
So everything is findable in the directory structure

- / the beginning - the root
- /bin — executables
- /dev — “devices” like disk drives
- /etc — configuration files
- /home — user directories
- /lib — libraries
    - parts of executables
    - usually a .so extension eg libc.so
        - this is the library that from “gcc -lc -xc xxx.c”
- /usr —
    - things that may also be in /
        - /usr/bin, /usr/include, …
    - /usr/local — stuff NOT in standard UNIX …
    - /usr/include — where all #include <xxx.h> live in Unix.
- /proc
    - NOT actual files but lots of great info
        - /proc/cpuinfo, /proc/stat, /proc/uptime
        - /proc/# one for every running process
    - frequently there are utilities that show information from proc in human readable forms

UNIX sort command (also tr, uniq)

- recall “ls -lart” or “ls -lat”
  - sort entries by time
  - suppose want to sort by size?
- The Unix sort utility! (man page)
  - ls -la | sort
  - ls -la | sort -k 5 -n -r
    - things — directories are small but NOT all the same size
    - soft links show as 17 bytes
  - ls -laS does this, probably by piping through sort

UNIX tr,uniq

- tr – “translate”
  - replace a character with another
    - ls -la | tr “d” “q”
    - cat file | tr [:punct:] “ “
      - replaces all punctuation in a file with a space
- Uniq – “unique”
  - compare consecutive lines and eliminate duplicates

list all unique words in a text

- cat text | tr [:punct:] " " | tr A-Z a-z | tr " " "\n" | sort | uniq
  - first tr removes punctuation
  - second downcase
  - third puts each word on a line (there will be lots of blank lines)
  - sort puts same words next to each other
  - uniq eliminates duplicates
  - DONE …
- How to get count?
- Efficiency?
  - ~/Public/206/a4/janeausten.txt, ~/Public/206/a4/dickens.txt, ~/Public/206/a4/ham.txt

Thursday Lab

- Infinite loop?
- Improve?
  - Add some defines for 97, 122 and 32
  - move “if.. break” inside while
- “fgets”
  - recall array are just pointers to the start of a reserved block of memory
  - so fgets is “write up to LINE_LEN-1 chars you get from stdin to memory starting at the memory loc given by line”
    - why the -1?

```
#define LINE_LEN 256
void shout()
{
    char line[LINE_LEN];
    while (1) {
        if (NULL == fgets(line, LINE_LEN, stdin))
            break;
        for (int i = 0; line[i] != '\0'; i++) {
            if (line[i] >= 97 && line[i] <= 122) {
                line[i] = line[i] - 32;
            }
        }
        printf("%s\n", line);
    }
}

int main(void) {
    shout();
}
```

Java and C and Arrays

- Java
  - Java arrays are a pointer to a block of memory + size of the memory block + type of thing in the block
  - “new” operation in java **dynamically** allocates from “heap”
    - heap is global memory space
      - size of heap is bounded by machine memory
    - Because array allocation is always in the heap space it can be used outside that function
- C
  - arrays are pointer to a block of memory
  - global arrays are allocated from heap
  - arrays in functions are allocated in “stack” space
    - stack space clears when function completes for arrays cannot be passed back from functions
    - size of array inside function is bounded by size of “stack space”

Arrays — behind the scenes

- A contiguous block of memory
  - int arr[10]
    - space of 10 ints — sizeof(int) = 4
      - 40 bytes
- So what happens when you say
  - arr[5] = 42
    - Calculate: loc = array_start + 5 * sizeof(int)
    - write the number 42 into the 4 bytes starting at loc

2D Arrays

- `int array2d[3][5];`
- Row major vs Column Major
  - C uses row major (as does Java)
  - Fortran used Column Major

Address	Row Major	Column Major
0	a[0][0]	a[0][0]
1	a[0][1]	a[1][0]
2	a[0][2]	a[2][0]
3	a[0][3]	a[0][1]
4	a[0][4]	a[1][1]
5	a[1][0]	a[2][1]
6	a[1][1]	a[0][2]
7	…	…

- Array Initialization
  - `int array[6] = {1,1,2,3,5,8,13,21,34};`
  -
```
int array2d[3][5]={{1,2,3,4,5},
                   {6,7,8,9,10},
                   {11,12,13,14,15}};
```

More 2d Array

- `int array2d[3][5]={{1,2,3,4,5},`
  `{6,7,8,9,10},`
  `{11,12,13,14,15}};`
- Row-major or col-major, does not matter
- `int array2d[3][5]`
  `={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};`
  - LEGAL — identical to previous initializer
    - RM vs CM matters
  - LOC = start + row_length*row_number*sizeof(int) + col_num*sizeof(int)

Passing 2d arrays

- recall
  - `void f2(int arrSize, int farr[arrSize])`
    - the arrSize in array declaration is optional
- For 2d arrays
  - `void ap(int rows, int cols, int arr[][cols])`
  - sizes of all dimensions other than first is REQUIRED
    - Another manifestation of R-M ordering
    - How do you do the LOC calculation without knowing column numbers

n dimensional arrays

- `int arr[5][12][4]….;`
- When passing n dim arrays every dimension other than first must be specified!
  - e.g.`void printArray(int rows, int cols, int d3, int arr[][cols][d3])`

Order [in]dependence

- Recall that C compile is single pass
  - so function/global vars must be “known” before they can be used
    - PUT all global vars at top of file
      - cannot do that for functions
    - put signature of functions at top of file!
      - shoutc2.c

- Alternate solution .h files

string.h

- Has other includes
- Defines a bunch of “string” functions
  - __functions
    - DO NOT OVERWRITE
- “extern” will be implemented elsewhere
  - assumed in .h files
- .h files may also have variables

```
#include <bits/libc-header-start.h>
#include <stddef.h>

/* Copy N bytes of SRC to DEST.  */
extern void *memcpy (void *__restrict __dest, const void *__restrict __src,
                     size_t __n) __THROW __nonnull ((1, 2));

/* Copy SRC to DEST, returning the address of the terminating '\0' in DEST.
*/
extern char *__stpcpy (char *__restrict __dest, const char *__restrict __src
    __THROW __nonnull ((1, 2));
extern char *stpcpy (char *__restrict __dest, const char *__restrict __src)
    __THROW __nonnull ((1, 2));

/* Copy no more than N characters of SRC to DEST, returning the address of
  the last character written into DEST.  */
extern char *__stpncpy (char *__restrict __dest,
                        const char *__restrict __src, size_t __n)
    __THROW __nonnull ((1, 2));
extern char *stpncpy (char *__restrict __dest,
                      const char *__restrict __src, size_t __n)
    __THROW __nonnull ((1, 2));
```

writing .h files

- generally include anything you want to share
  - think the “public” variables and functions of java
  - may include
    - signatures of functions
    - includes
    - global variables
    - defines
  - Order still matters
    - OK to include .h more than once
      - loops are bad!

Using .h you write

- #include “my.h”
  - “”: look for .h starting from here
  - <>: look for .h starting from /usr/include

shout.h

```
void shout();
```

shoutc2.h

```
#include <stdio.h>
#include <shout.h>

#define LINE_LEN 256
```

Breaking things up using .h

- using .h files you can break things up something like java
- problem
  - Not every .c file contains “main”
  - How to you tell gcc what to do?

shoutc3b.c
    contains the shout function
shoutcMain.c
    contains the main function
shoutc.h
    just the prototype of shout();

shout3Main.c

```
#include <stdio.h>
#include "shoutc.h"

int main(void) {
    shout();
}
```

compile but do not create executable. Instead stop and output shoutc3b.o

compile shout3Main.c, then link it to shout3b.o to create the executable “shout”

```
[gtowell@powerpuff L05]$ gcc -c shoutc3b.c
[gtowell@powerpuff L05]$ gcc -o shout shoutc3b.o shout3Main.c
[gtowell@powerpuff L05]$ shout
```

Writing your own versions of library functions

- atoi
  - ascii to integer
    - takes a “string” as input, returns an integer
    - atoigt.c

LAB

- Write your own implementation of strcpy
  - `void strcpy(int destLen, char dest[destLen], char source[]);`
  - You version should take two char arrays
    - copy from first into second until
      - string end in first
      - out of space in second
    - You MUST be sure than the second ends with \0