CS 35L- Software Construction Laboratory

Fall 18

TA: Guangyu Zhou

Course Information

- Presentation

10/31	Zheng Wang		Turn 2-D video to 3-D motion sculpture	Henry Ou		Fake online personas	Jeffrey Liu		How Robots and Drones Will Change Retail Forever
11/5	Justin Jeon	Andrew Yong	Audio to Visual instruments	Miranda Tang		filling in gaps in videos	Kelly Cheng	Shawn Ma	Modifying a virtual environment

- Uploading your slides (pdf, ppt) to CCLE week 10 **before** presentation

Lab 4: Clarification

- Make sure you apply the patch to fix the “make file” bug first
  - Then run **configure, make, make install**
- When you generate the bug, make sure you run *“ls” from the directory you installed (XXX/bin/ls)*
- When using GDB, run from the directory where the compiled ls lives *(XXX/bin/ls)*
- You can run *ls* with **options** in the GDB terminal:
  - *(gdb) r -lt --full-time wwi-armistice now now1*
- Construct a new patch file:
  - *diff -u old_buggy_file new_fixed_file> patch.diff*

Lab 4: Clarification

Buggy version:

```
$ tmp=$(mktemp -d)
$ cd $tmp
$ touch -d '1918-11-11 11:00 GMT' wwi-armistice
$ touch now
$ sleep 1
$ touch now1
$ TZ=UTC0 ls -lt --full-time wwi-armistice now now1
-rw-r--r-- 1 eggert csfac 0 1918-11-11 11:00:00.000000000 +0000 wwi-armistice
-rw-r--r-- 1 eggert csfac 0 2018-10-29 16:43:16.805404419 +0000 now1
-rw-r--r-- 1 eggert csfac 0 2018-10-29 16:43:15.801376773 +0000 now
$ cd
$ rm -fr $tmp
```

Correct output should be:

-rw-r--r-- 1 eggert csfac 0 2018-10-29 16:43:16.805404419 +0000 now1

-rw-r--r-- 1 eggert csfac 0 2018-10-29 16:43:15.801376773 +0000 now

-rw-r--r-- 1 eggert csfac 0 1918-11-11 11:00:00.000000000 +0000 wwi-armistice

More hints on Lab 4

- Use “info functions” to look for relevant starting point
- Use “info locals” to check values of local variables
- Compiler optimizations: -O2 -> -O0
  - If you want to see more details for optimized variables
  - https://gcc.gnu.org/onlinedocs/gnat_ugn/Optimization-Levels.html
- ./configure … CFLAGS= "-g -O0"

Review: Pointers to Functions

- **qsort (values, 6, sizeof(int), compare);**
- User can pass in a function to the sort function
- Declaration
  - double (*func_ptr) (double, double);
  - func_ptr = &pow; // func_ptr points to pow()
- Usage
  - // Call the function referenced by func_ptr

    double result = (*func_ptr)(1.5, 2.0);
  - // The same function call

    result = func_ptr(1.5, 2.0);

Some facts about C function pointers

- Unlike normal pointers, a function pointer points to code, not data. Typically a function pointer stores the start of executable code.
- Unlike normal pointers, we do not allocate de-allocate memory using function pointers
- Many object oriented features in C++ are implemented using function pointers in C. For example virtual functions. Class methods are another example implemented using function pointers
- A function's name can also be used to get functions' address

```c
#include <stdio.h>
// A normal function with an int parameter
// and void return type
void fun(int a)
{
    printf("Value of a is %d\n", a);
}

int main()
{
    void (*fun_ptr)(int) = fun;   // & removed

    fun_ptr(10);   // * removed

    return 0;
}
```

Some facts about C function pointers

- Like normal pointers, we can have an array of function pointers.

```c
#include <stdio.h>
void add(int a, int b)
{
    printf("Addition is %d\n", a+b);
}
void subtract(int a, int b)
{
    printf("Subtraction is %d\n", a-b);
}
void multiply(int a, int b)
{
    printf("Multiplication is %d\n", a*b);
}

int main()
{
    // fun_ptr_arr is an array of function pointers
    void (*fun_ptr_arr[])(int, int) = {add, subtract, multiply};
    unsigned int ch, a = 15, b = 10;

    printf("Enter Choice: 0 for add, 1 for subtract and 2 "
           "for multiply\n");
    scanf("%d", &ch);

    if (ch > 2) return 0;

    (*fun_ptr_arr[ch])(a, b);

    return 0;
}
```

Some facts about C function pointers

- Like normal data pointers, a function pointer can be passed as an argument and can also be returned from a function.

```c
// An example for qsort and comparator
#include <stdio.h>
#include <stdlib.h>

// A sample comparator function that is used
// for sorting an integer array in ascending order.
// To sort any array for any other data type and/or
// criteria, all we need to do is write more compare
// functions.  And we can use the same qsort()
int compare (const void * a, const void * b)
{
  return ( *(int*)a - *(int*)b );
}

int main ()
{
  int arr[] = {10, 5, 15, 12, 90, 80};
  int n = sizeof(arr)/sizeof(arr[0]), i;

  qsort (arr, n, sizeof(int), compare);

  for (i=0; i<n; i++)
     printf ("%d ", arr[i]);
  return 0;
}
```

Review: Process Memory Layout

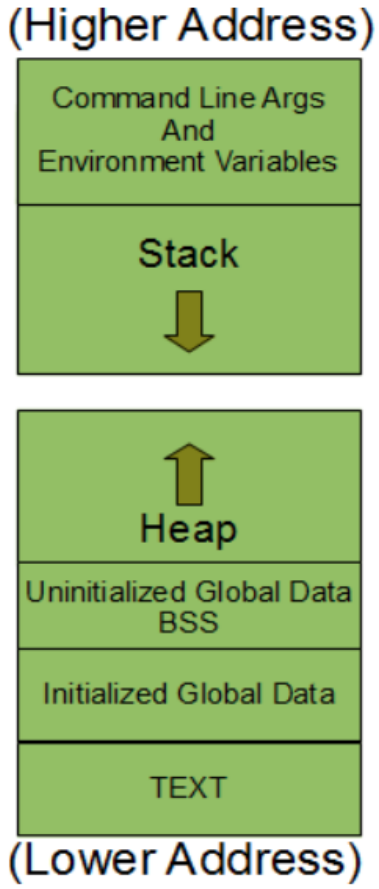


Image source : thegeekstuff.com

- TEXT segment
  - Contains machine instructions to be executed
- Global Variables
  - Initialized
  - Uninitialized
- Heap segment
  - Dynamic memory allocation
  - malloc, free
- Stack segment
  - Push frame: Function invoked
  - Pop frame: Function returned
  - Stores
    - Local variables
    - Return address, registers, etc
- Command Line arguments and Environment Variables

Review: Dynamic Memory Allocation

- malloc(size_t size): allocates a block of memory whose size is at least size
- free(void *ptr): frees the block of memory pointed to by ptr
- realloc(void *ptr, size_t newSize) : Resizes allocated memory
- Allocate memory **on heap**

```
Rectangle_t *ptr = (Rectangle_t*) malloc(sizeof
(Rectangle_t));
if(ptr == NULL)
{
  printf("Something went wrong in malloc");
  exit(-1);
}
else
{
//Perform tasks with the memory
//
//
free(ptr);
ptr = NULL;
}


ptr = (Rectangle_t*) realloc(ptr, 3*sizeof
(Rectangle_t))
```

Homework 4

- Implement a C function frobcmp as the template of qsort
  - takes two arguments a and b as input
  - returns an int result that is negative, zero, or positive depending on whether a is less than, equal to, or greater than b. Each argument is of type char const *.
  - a, b point to array of non-space bytes
- Returns an int result that is:
    - Negative if: **a** < **b**
    - Zero if: **a** == **b**
    - Positive if: **a** > **b**
    - Where each comparison is a lexicographic comparison of the unforbnicated bytes

Homework 4: hints

- Then, write a C program called *sfrob*
  - Reads stdin **byte-by-byte** (getchar)
  - Each byte is frobnicated
    - frobnicated - encoded with memfrob (XOR decimal 64)
    - Sort records without decoding (**qsort, frobcmp**)
    - Output result in frobnicated encoding to stdout (putchar)
  - Consider error checking (**fprintf**)
  - Dynamic memory allocation (**malloc, realloc, free**)
  - Program should work on empty and large files too

Note: if you don't allocate memory from heap using malloc/realloc, there will be a memory issue that makes your program fail

Homework 4: example

- 1) Input: printf 'sybjre\nobl’
  - $ printf 'sybjre\nobl**\n**' | ./sfrob
- 2) $ cat 'sybjre\nobl’ > foo.txt
- Input: contents of foo.txt
  - $ ./sfrob < foo.txt
- Read the records: sybjre, obl
- Compare records using *frobcmp* function
- Use frobcmp as compare function in qsort
- Output:

  obl

  sybjre

Homework 4: instruction

- Use ***gdb*** for debugging
- Use *exit*, not *return* when exiting with error
- When performing malloc/realloc, do check for whether it is successful
- Array of pointers to char arrays to store strings (char ** arr)
- Use the right cast while passing frobcmp to qsort
  - cast from void * to char ** and then dereference because frobcmp takes a char *
- Use realloc to reallocate memory for every string and the array of strings itself, dynamically

- Assignment 5 **requires** having a solid handle on assignment 4, so this is important!
- Your code must do thorough error checking, and print an appropriate message on errors.
- Plug all memory leaks!

Valgrind

- Powerful dynamic analysis tool
- Useful to detect memory leaks
- Example:
  $ valgrind --leak-check-full
  ./sfrob < foo.txt

  88 (...) bytes in 1 blocks are definitely lost ...

  at 0x.....: malloc (vg_replace_malloc.c:...)

  by 0x.....: mk (**leak-tree.c:11**)

  by 0x.....: main (leak-tree.c:25)

Valgrind Example

```
1  #include <stdlib.h>
2
3  void f(void)
4  {
5     int* x = malloc(10 * sizeof(int));
6     x[10] = 0;
7  }
8
9  int main(void)
10 {
11    f();
12    return 0;
13 }
```

1st Problem: heap block overrun

```
==19182== Invalid write of size 4
==19182==    at 0x804838F: f (example.c:6)
==19182==    by 0x80483AB: main (example.c:11)
==19182==  Address 0x1BA45050 is 0 bytes after a block of size 40 alloc'd
==19182==    at 0x1B8FF5CD: malloc (vg_replace_malloc.c:130)
==19182==    by 0x8048385: f (example.c:5)
==19182==    by 0x80483AB: main (example.c:11)
```

2nd Problem: memory leak

```
==19182== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==19182==    at 0x1B8FF5CD: malloc (vg_replace_malloc.c:130)
==19182==    by 0x8048385: f (a.c:5)
==19182==    by 0x80483AB: main (a.c:11)
```

There are several kinds of leaks; the two most important categories are:

- "definitely lost": your program is leaking memory -- fix it!
- "probably lost": your program is leaking memory, unless you're doing funny things with pointers (such as moving them to point to the middle of a heap block).

Make sure you fix all of them to get full grade!

Segmentation fault

- A segmentation fault is caused when the code attempts to access memory that it **doesn't have permission to access**.
- Possible Reasons:
  - Accessing a NULL or uninitialized pointer
  - Accessing a dangling pointer
  - Stack overflow
  - Wild pointers
  - Attempting to read past the end of an array
  - Forgetting a NUL terminator on a C string
  - Attempting to modify a string literal
  - Mismatching Allocation and Deallocation methods

Segfault examples

```c
// C program to illustrate
// Core Dump/Segmentation fault
#include <stdio.h>
#include<alloc.h>
int main(void)
{
    // allocating memory to p
    int* p = malloc(8);
    *p = 100;

    // deallocated the space allocated to p
    free(p);

    // core dump/segmentation fault
    //  as now this statement is illegal
    *p = 110;

    return 0;
}
```

Some exercises

1. Write a C program using getchar() and putchar() which continuously takes user input and prints it on the screen. This should keep on happening till the user inputs a string containing ‘#’ and Enters. Hint: use while(getchar() != ‘#’)

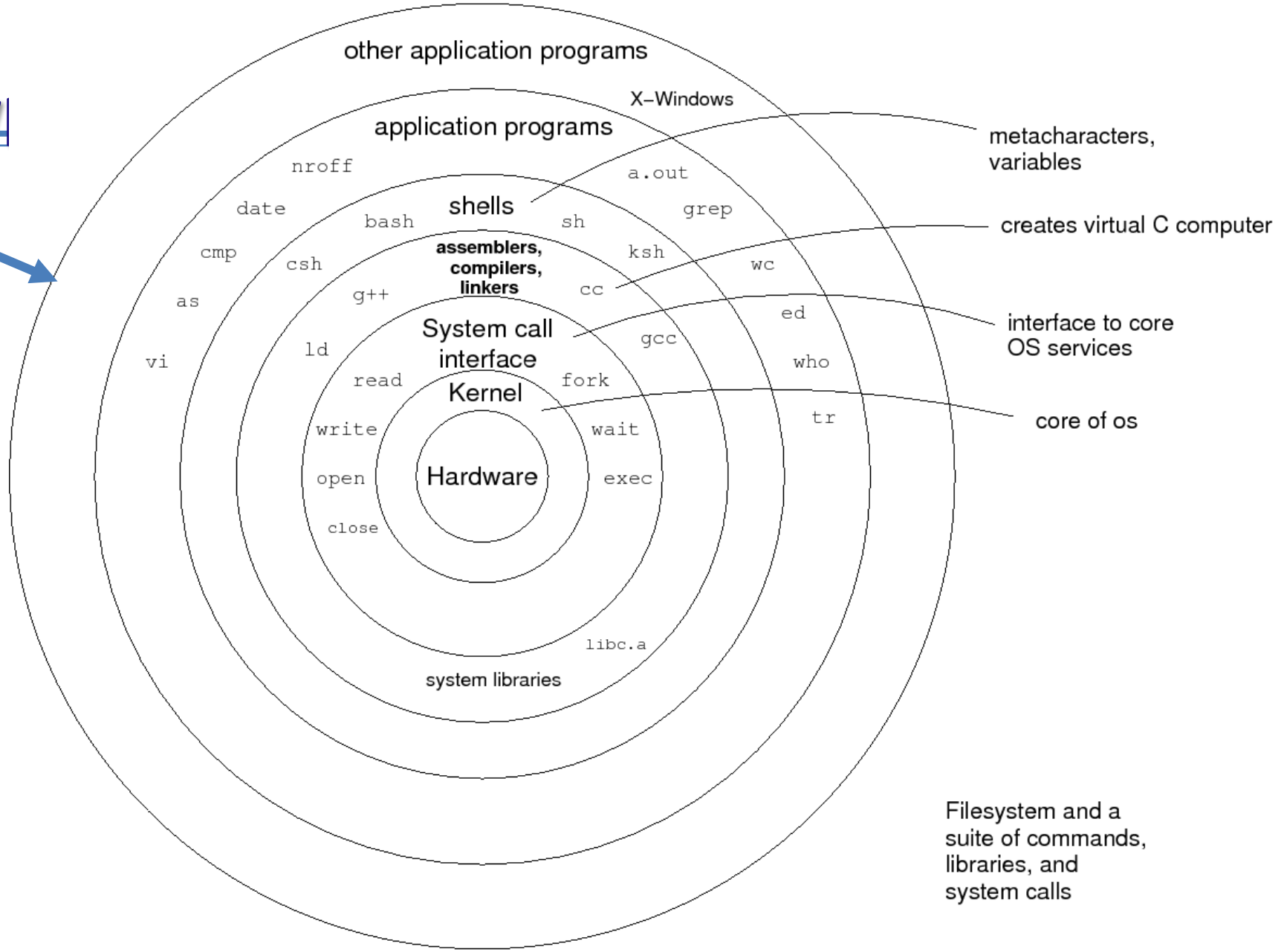
```
#include <stdio.h>
/* --  Copy input to output -- */
int main(void)
{
        char c;
        c = getchar();
        while ( c != "#" ) {
           putchar(c);
           c = getchar();
        }
        return 0;
}
```

Some exercises

2. Write the following line in a file called file.txt. The value stored is 100. Use fscanf to read the value 100 from file.txt and store it in a variable <var>. Then write this value to another file file1.txt “Value read is <var>” using fprintf

Hint: fscanf(fp, "This is the value %d", &a); fprintf(fp1, "Value read is %d",a);

```
#include <stdio.h>
#include <stdlib.h>
int main() {
int a;
FILE * fp;
FILE * fp1;
fp = fopen("file.txt","r+");
fp1 = fopen("file1.txt", "w+");
fscanf(fp, "This is the value %d", &a);
fprintf(fp1, "Value read is %d",a);
fclose(fp);
return 0;}
```



Conceptual Architecture of UNIX SYSTEMS

From: The Design of the UNIX Operating System