CIS 3800 C Recitation

Tuesday, September 5th 2023

Contents

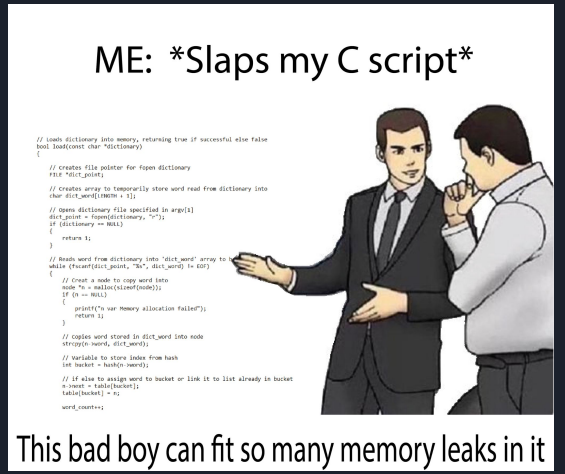
Quick C History

- C came out of Bell Laboratories along with Unix around the same time (1972ish)
- Created by Dennis Ritchie and Ken Thompson who also created Unix
- Linux and MacOS are both based upon Unix
- Linux (used in this course) is written completely in C (but possibly soon Rust as well)
- Despite its old age and simplicity, C is still an extremely popular language
- https://github.com/torvalds/linux

Tux the Penguin!

Why C?

- C is fast and very portable
- It does not include quality of life aspects other programming languages have
- No garbage collection, bounds checks and objects
- Can create powerful programs but also monstrous errors (memory safety)
- Since Unix/Linux are written in C, C's use of system calls matches what the OS uses
- Soft bound paper to learn more about bound checking
  https://acg.cis.upenn.edu/papers/pldi09_softbound.pdf



Pointers

- Literally point to a place in memory
- A variable which contain a memory location
- Declared with a type and an * ex (int *ptr;)
- * is used to dereference a pointer or operand
- Dereference is to “go to” the memory location in the pointer
- & is used to get the memory location of the passed operand
- & is used in conjunction with pointers ex (int *ptr = &integer)
- Can dereference multiple times
- pointers.c

Arrays

- Contiguous memory of the same type
- Arrays are referenced by using a pointer to the first element
- Very easy to go past the bounds of an array and cause memory errors
- arrays.c

Strings

- No “String” type in C
- Strings are just arrays of characters
- All strings in C must end in ‘\0’, the null character
- Functions often continue reading until they find a null character
- This is referred to as a null terminated string
- String literals are super funky
- Very important for project 0!
- strings.c

Structs

- Objects do not exist in C, but structs do!
- Custom data types which contain inner custom fields
- Structs are allocated as contiguous memory
- Very similar to arrays, but filled with possibly different data types
- structs.c and structs.h

Memory Management

- Stack
  - Static storage / local scope
  - Automatically allocated/deallocated
  - Small upper bound in size (stack overflow)
  - char str[6] = “hello”;
- Heap
  - Dynamic storage / program scope
  - Allocated with system calls and freed with free(3)
  - Large size
  - char* str = malloc(6 * sizeof(char));

Memory Management

- Proj 0 can be done solely on the stack, but proj1 and proj2 will be tough to only allocate stack memory
- System calls to allocate memory
  - **malloc(3),** calloc(3), etc.
- System call to free allocated memory
  - free(3)
- What to allocate?
  - ANY pointers / arrays unless we tell you otherwise
  - Strings (char*)
- Make sure to FREE all memory before exit(2)!
- Example code memory.c

C coding style

See c_style.c, c_style.h

Valgrind

- Memory error checking program
- Very useful for finding memory leaks and memory errors
- Valgrind runs around the program running
- Common valgrind errors are memory leaks, invalid reads/writes, and uninitialised bytes
- Simply run `valgrind ./program <program arguments>`
- Useful valgrind arguments:
  - `--trace-children=<yes|no> [default: no]`
  - `--track-origins=<yes|no> [default: no]`
  - `--leak-check=<no|summary|yes|full> [default: summary]`

GDB

- GDB is a useful debugger that allows you to inspect a program
- Compile a c program with flag -g allows you to use gdb on it
  - Your provided makefiles will have this option already
- gdb [program name] to run gdb on the program
- List of gdb commands next slide
- Example debug code gdb.c

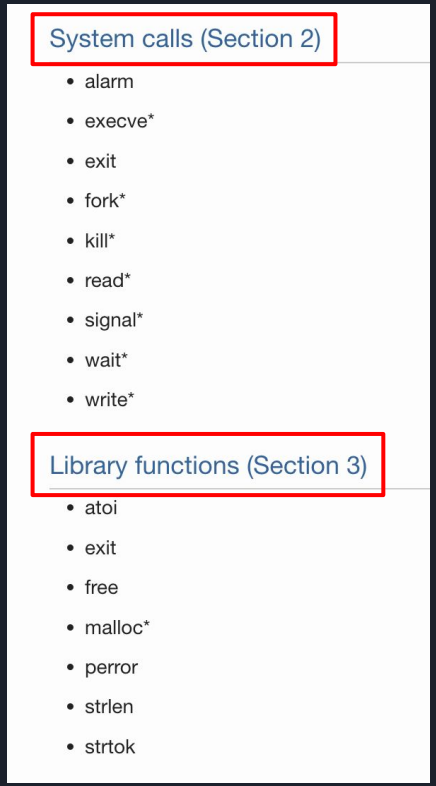
GDB Commands

- **file [program]** - mounts the program onto gdb
- **run** - runs the program
- **backtrace/bt** - show the stack of this program
- **print [variable]** - shows variable's value at this instant of program execution
- **break/br [line]** - set a breakpoint at line number
- **continue/c** - continues execution
- **list** - prints the source code around the current line.
- **step** - runs the next line of code, and stops again. If the current line is a function call, it steps into the function call.
- **next** - is similar to step, but steps *over` any function calls.
- **watch [variable]**- stops each time the variable var changes.

7. Helpful sites for C reference

Linux man pages: https://man7.org/linux/man-pages/

- Either access online (link above) or in terminal:
  - $ man [section number] [func name]
    - i.e.: man 2 alarm, man 3 malloc
  - Press q to exit
  - “Section number”:
    - Man pages are broken into sections, including commands (sec 1), system calls (sec 2), C library functions (sec 3)



Some C++ references contain sections on C

https://cplusplus.com/reference/

- Contains really nice documentation for C string functions: https://cplusplus.com/reference/cstring/
- Be careful not to stray out of the C library

https://en.cppreference.com/w/c

- C language basics