THE C STANDARD LIBRARY & MAKING YOUR OWN LIBRARY

ISA 563: Fundamentals of Systems Programming
Overview: the Standard Library

- A language is:
  - The grammar of the language (keywords, operators, expressions, etc.)
  - The execution environment (e.g., an OS, JVM, CLR)
  - A library of supporting functions

- “Language design is library design.”
  -- Bjarne Stroustrup

- Example: Java (very large object library and API)

- Hint: read the man pages for the C library functions!
What is a Library?

- A collection of functions with a common purpose
- The collection provides a well-defined standard interface or API to the library’s core purpose:
  - I/O
  - Math
  - Graphics/GUI
  - Crypto
  - …many others
Header Files

- Header files are C source files that hold the definition of functions and data structures
  - Header files end in “.h”

- The C standard library is composed of many header files as well as their corresponding implementation (i.e., .c ) files
  - You know one already: stdio.h
Basic C data types provide storage for data when it is “in” your program’s memory space.

- Collections of data: structs, arrays, unions (last lecture)

What about feeding data into these variables and sending data to other programs or files on disk?

- Streams or collections of bytes
- Files
Basic Concepts of Unix Files

- No markup (contrast with NTFS files)
  - Every byte is addressable

- Access is byte by byte (char by char)
  - Can perform “random” access (cover this later)
  - Treat a file as a stream or sequence of bytes

- Everything in Unix is a file (in one form or another)
  - So file I/O is important in C programs
  - …and so is having a robust, standard way of manipulating data in files!
Every C program is given 3 files automatically

- Standard output (what you see on screen)
- Standard input (usually attached to keyboard device)
- Standard error (also usually on screen)

But via the “magic” of Unix, can be easily redirected to or from other sources and sinks

- Shell redirection
- See ‘dup’ system call
Naming “Standard” Files

- The header file `<stdio.h>` defines three handles to these objects (of type FILE, a struct)
  - Stdin
  - stdout
  - stderr

- These are variable names you can use in any code that “includes” `stdio.h`
Interesting I/O Functions

- Char output: putchar(), getchar(), putc(), getc()
- String input/output: fprintf(), fscanf()
- File I/O:
  - fopen() / fclose()
  - fread() / fwrite()

- These are different from the OS system calls: open, close, read, write
  - They operate on C library FILE objects rather than OS-level file descriptors
The FILE Structure Abstraction

- A data type defined in stdio.h

- A struct named FILE
  - A common data type for use with most of the C I/O library functions
  - So library design involves designing and defining appropriate data structures as well as functions

- See page 176 in TCPL for the definition
Key Idea: translate a file name to something the OS can manipulate
- The C library steps in the way

Concept stack
- A filename: a character sequence humans understand
- A FILE object: something your program (via stdio.h) understands
- A file descriptor (an integer the OS uses to keep track of unique file handles)
Opening Files via stdio.h

//consult ‘man fopen’ for details!
#include <stdio.h>

//two arguments: ‘file name’ and ‘mode’
FILE* fin = fopen("/tmp/name", "rb");

//now ‘fin’ represents a valid FILE object, right?
//wrong! … need to test the result of fopen()!
if(NULL==fin){… //an error occurred, handle it
Contract vs. Implementation

- fopen’s contract is:
  - Give me a valid file path and a mode (read, write, append, truncate, etc., see man page) AND I might return to you a valid pointer to a valid FILE object

- How does C library do all that?
  - It doesn’t do it all. It asks the OS for help.
Many standard library functions employ a system call (some don’t) to help accomplish the underlying task.

System calls define the OS’s API:
- A collection of services the OS will provide to application programs
- But can be tedious to use and set up
- So C library is a higher level of abstraction
fopen employs the ‘open()’ system call

//see ‘man 2 open’
int open(const char* pathname, int flags);
Other C Libraries
Character manipulation

- `#include <ctype.h>`
- `isascii(int)`, `islower(int)`, `isupper(int)`, `isdigit(int)`...
- `tolower(int)`, `toupper(int)`...
String Manipulation

- `#include <string.h>`
- Defines the symbol NULL
- Memory copy routines, the `strlen()` routine, string tokenization, some error output routines, … more on those when we get to memory management
Collection of many utility functions

- exit, abort, atoi, atof, system()
- malloc, calloc, realloc, free (will talk about these in a later lecture, not now...)
- getenv, putenv, setenv
- rand, srand
errno.h

- Defines a list of standard error names (rather than keeping track of error numbers…)

- Defines the ‘errno’ integer variable

- ‘perror()’ from stdio.h is related (but in a different library)

- Get in the habit of testing errno’s value!
math.h

- Defines common math symbols (pi, e, etc.)
- Defines values for representing limits of primitive types (INFINITY, NAN, etc.)
- Defines tan, cos, sin, exp, abs, floor, ceil, log, round, etc.
Create Your Own Library
Anyone Can Create a Library

- Just a collection of:
  - Contract definitions
  - Symbol and data type definitions
  - Function implementations

- Components:
  - Header files
  - Library binary (or source) files
Note: Library Interception

- Linking is not done until runtime
- Can dynamically replace function implementations
  - “DLL Injection”
  - “Library interposition”

- Unix: LD_PRELOAD environment variable
  - Affects search path for library function implementation