Lecture 3, Jan 12, 2024

Libraries

- The definition of an operating system depends on the application; the OS allows you to run the application you want
  - An OS consists of a kernel and *libraries* required for your application
- During normal compilation, code is complied into object files, and then linked against libraries to produce an executable
- *Static libraries* (.a files) are included in the binary at link time
  - A bunch of object files are linked together to form an archive (a static library), and then when an executable is linked, this archive is copied into the executable
  - This means the static libraries have to be copied for each executable that needs to use them, so it is inefficient
  - Additionally, updates to the static library requires the executable to be recompiled
- *Dynamic libraries* (.so files) are included at runtime and not embedded in the executable binary
  - e.g. the C standard library (libc)
  - Multiple applications can use the same library; the OS only loads the library once and shares it with all applications that need it
    * This is more efficient and applications don't need to be recompiled if the library is updated
    * But if the dynamic library has a bug, all applications that use it will be affected
  - The `ldd` utility shows the dynamic libraries used by an executable
- Updating dynamic libraries may subtly change the ABI and cause incompatibilities and bugs
  - e.g. if the order of fields in a struct declaration changes, the ABI would change due to different memory layout, leading to subtle bugs
  - If a dynamic library exposes a struct, it should never be changed again!
- To avoid compatibility issues, use *semantic versioning*
- The `LD_LIBRARY_PATH` environment variable changes the path that dynamic libraries are searched for
- The `LD_PRELOAD` environment variable causes dynamic libraries in the specified path to be loaded first and overrides the usual dynamic libraries
  - This is how `valgrind` works; it overrides the standard `malloc()`/`free()` so that memory usage can be analyzed
- Another tool is clang's AddressSanitizer (ASan)
  - Add the `-Db_sanitize=address` flag to `meson setup build`
- Most system calls have corresponding function calls in C, but the wrappers may do additional tasks:
  - Set `errno` (error detection)
  - Buffer reads and writes to reduce the number of syscalls (since syscalls are slow)
  - Simplify interfaces (e.g. combining two syscalls into the same function)
  - Adding new features
  - e.g. the C `exit()` also runs functions registered with `atexit()` before an `exit_group` syscall; returning from `main()` is the same as calling `exit()` with the return value