Contents

Notes on ish

`ish.c`

Preamble and `static const`

- can work either on library hearders or on our own files
- max input length is 255
- max path length is 1024
- env variables `HOME` and `PATH`
- builtin commands `cd` and `exit`
- standard input file flag is `O_RDONLY`
- standard output file mode is `00644` (user r/w, other r)

Main Method

- three arguments in `main()`, `int argc`, `char **argv`, `char **envp`)
  - just in UNIX, ISO C does not have this
  - `argc` and `argv` are the same
  - `envp` is the programs environment
  - information like home directory, terminal type, current locale
- get `*home`, home directory of current user
- get `*paths` dirs where executables are
- create string for all input
- loop of `read` syscalls – main program loop
  - replace newline with `\0`
  - create an array of arguments as long as the input array
  - check if the first argument is not empty
  - check if command is `cd`
    - if no arguments, `dir` is `home`, else it's argv[1]
    - `chdir` syscall
  - is command `exit`
    - if no exit code is given, use 0
    - `exit` syscall
  - if command is not built-in, start a separate program
  - use the syscall `stat` to see if the executable exists, if not:

        * search for executable in all the paths
        * if you don't find it, simply continue
    - see if stdin redirection lexeme (basic lexical unit) was used (if the is stdin from a file)
        * if yes get file descriptor with syscall `read`
    - see if stdout was redirected
        * if yes, get that file descriptor with syscall `creat`
    - create a child duplicate of the current process and get its `pid` using syscall `fork` – spawn a child process
        * if we're in child process (`pid == 0`)
            · if stdin is redirected, redirect stdin to file using syscall `dup2`
            · then close the originally open file using syscall `close`
            · now only stdin points to the file
            · if there is an ouput file, redirect stdout to that file
            · close the file, now only stdout points to the file
            · run the provided command using syscall `execve`, supplying path of the program, arguments given for it, and environment variables
            · in case `execve` fails, error out with -1
        * if the process `id > 0` (we're in the parent process)
            · wait for the child process to stop (exit, terminate, fail) using syscall `waitpid`
            · get status of child process
        * if (`pid < 0`) means there was some error in `fork`, write error message to stderr (number 2) using syscall `write`

`ish_common.c`

- defines whether to use `stdlib` or `ish_syscalls.h`
- defines infinite loop that only stops on 0 values

`ish_cstring_utilities.h`

- defines string functions independent of the standard library
- defines path separator `:` and directory separator `/`

`ish_cstring_utilities.c`

- implements `ish_cstring_utilities.h`
- contains functions to check for string attributes, convert to lower, upper
- `char[] ish_carve_token_in_cstring` returns a pointer to the start of the string delimited by char, a pointer to first char after the end of the delimited string
- then the same method but does not properly terminate the string

`ish_shell_utilities.h`

- defines the input and output redirection lexemes
- processes arguments

`ish_syscalls.h`

- contains the preprocessor macro that decides between library syscalls and our own syscalls

`ish_syscalls.c`

- contains preprocessor macroes that decide which syscall file to use based on the target architecture