System Calls, Errors, and the /proc Filesystem

UNX511 Week 2 Class 2

John Sellens

May 15, 2025

Seneca Polytechnic

Outline

System Calls and Errors

Functions and Return Values

- Many standard functions return something useful on success
    - e.g. `fopen()` will return a file pointer for reading or writing
    - Many system functions (e.g. `chmod()` return 0 or a positive number on success
- And a simple failure indicator
    - e.g. `fopen()` will return NULL on failure
    - Many system functions will return -1 to indicate failure
- See the man pages for the functions for details

System Calls vs Library Calls

- A system call is a function that calls the kernel to do something (typically) privileged
    - e.g. `open()` is a system call, but `fopen()` is a library call function – which calls `open()`
    - Man page section 2 is for system calls, section 3 is for library functions

A Function Failed, But Why?

- If `fopen()` fails, it's tempting to just print out `could not open file` and exit from the program
- It's more useful for humans (and computers) if we say why it failed
    - e.g. file does not exist, not permitted, disk failure, ...
- Most system calls (and many library calls) set the `errno` global failure to indicate the reason for the failure
    - See `errno(3)` and `/usr/include/asm-generic/errno-base.h`
- There is a standard error message for each `errno` value
- `errno` might get reset on any library function call – either use it immediately, or save the value of `errno` in another variable

perror() and strerror()

- `perror()` prints a prefix string, and error to stderr

  ```
  fp = fopen( ”myfile”, ”r” );
  if ( fp == NULL ) {
      perror( ”could not open myfile” );
  }
  ```

  yields

  ```
  could not open myfile: No such file or directory
  ```

- `strerror( errno )` returns a pointer to a string containing the message for `errno`

Sample Error Messages: Bad

```
fp = fopen( "myfile", "r" );
if ( fp == NULL ) {
    printf( "oops\n" );
    exit( 1 );
}
```

Provides no way for the user to know what the problem is, or how to fix it.

Sample Error Messages: Less Bad

```
fp = fopen( "myfile", "r" );
if ( fp == NULL ) {
    perror( "couldn't open file" );
    exit( 1 );
}
```

Gives a little hint, but not much.

Sample Error Messages: Better

```
fname = "myfile";
fp = fopen( fname, "r" );
if ( fp == NULL ) {
    fprintf( stderr, "%s: could not open file '%s': %s\n",
        argv[0], fname, strerror( errno ) );
    exit( 1 );
}
```

Identifies the program with the error, what action was attempted (open), on what object (myfile), and why it failed

Error Messages Matter

- Always check for error returns from functions
- Take the time to provide useful and complete error messages

The /proc Filesystem

The /proc Filesystem

- Most Linux machines have the “proc” filesystem mounted at `/proc`
- Provides a vast amount of system status information
- Some system tuning can be done by writing into “files” in `/proc`
    - Remember: in UNIX, “everything is a file”
- There is a directory for every process on the system, named for the (numeric) process ID (pid)
    - Remember that the shell variable `$$` is the shell’s pid
    - `ls /proc/$$`
- See the man page `proc(5)`
- Let’s have a look ...

Summary

- Catch your errors and do something useful
- “Everything is a file” is powerful