Overview and Tools

Table of Contents

Prepare Your Unix Environment

A Brief Introduction to the Unix Environment

- Unix Architecture
- Booting Process
  - os loader
  - `/sbin/init`, `/etc/init`, `/bin/init`, and `/bin/sh`
- File System Architecture
  - filesystem hierarchy standard (FHS)
- Working with Unix Commands
  - Linux standard base (LSB)
- Common Notations in Unix Documents
  - `%` or `$`: run command as a regular user
  - `#`: run command as a privileged user
  - `[]`: optional part
  - `...`: multiple arguments
  - `-` or `--`: options
- Redirection and Pipe
- Read the Manual Pages
  - `man -k regexp`: search matched man page
  - **1: User commands**
  - **2: System calls**
  - **3: C library functions**
  - **4: Devices and special files**
  - **5: File formats and conventions**
  - **6: Games et al.**
  - **7: Miscellaneous**
  - **8: System administration tools and daemons**

Fundamental Unix Programming Practices

- Return Value of Previous Execution: `echo $?`

- Arguments in Command Line
  - `"$HOME"` will resolve `$HOME`
  - `'$HOME'` will not resolve `$HOME`
- Handle Program Options: `getopt(3)`, `getopt_long(3)`
  - `int getopt(int argc, char * const argv[], const char *optstring);`
    - return option character, `:` or `?` invalid option, -1 no more option
    - `:` in `optstring`: require an additional argument after previous option
    - global variables
      - `optind`: number of arguments consumed
      - `optarg`: additional argument of current option
- Unix Time Representations
  - `time_t`: seconds
    - epoch: `00:00:00, January 1st, 1970 UTC`
    - usually `int_32`
      - overflow after `03:14:07, January 19th, 2038`
  - `struct timeval`: microsecond
    - `time_t` plus microsecond precision timestamp
  - `clock_t`: CPU time
    - `CLOCKS_PER_SEC` constant
    - POSIX requires CLOCKS_PER_SEC to be 1,000,000 independent of the actual clock resolution
  - `time(3)`: `time_t time(time_t *t);`
  - `gettimeofday(2)`: `int gettimeofday(struct timeval *tv, struct timezone *tz);`
  - `clock(3)`: `clock_t clock(void);`
- Measure Program Performance
  - `time(1)`: real, user, and sys
    - `gettimeofday(3)`: get wall clock time (timeval format)
    - `clock(3)`: CPU ticks, i.e. user + sys
    - `getrusage(3)`: get CPU time (timeval format)
  - see `testtime.c` and `rusage.c`
- Error Handling
  - `errno` global variable
    - not thread-safe
    - not a problem if system supports thread local storage (TLS)
    - check right after receiving an error return value
  - `<stdio.h>` provide `perror`
  - `<string.h>` provide `strerror()`
- Error Recovery
  - fatal errors: no way to recovery
  - no fatal errors
    - delay and retry
    - e.g. `EAGAIN`, `ENFILE`, `ENOBUFS`, `EWOULDBLOCK`

Tools

- The Compiler

    - `gcc` and `g++`
        - `-S`: do not compile, generate assembly only (output to .s)
        - `-E`: do not compile, perform preprocessing only (output to stdout)
- Linking C and C++ Files
    - `nm`: list symbols
    - `c++filt`: demangled names
    - Example
        - `gcc -c add.cc`
        - `gcc -c main.c`
        - `gcc -o add.o main.o`

```
/* add.h */
#ifdef __cplusplus
extern "C" {
#endif
int add(int a, int b);
#ifdef __cplusplus
}
#endif
```

```
/* add.cc */
#include "add.h"
int add(int a, int b) {
  return a + b;
}
```

```
/* main.c */
#include <stdio.h>
#include "add.h"
int main() {
  printf("%d\n", add(1, 2));
}
```

Make and Makefile

- Command
    - `-C {dir}`: switch to dir and run make
    - `-f {makefile}`: use non-default makefile
    - `-I {dir}`: specify include directory search path
    - `-j {n}`: simultaneously jobs
- Automatic Variables
    - `$@`: target name
    - `$<`: first prerequisite
    - `$?`: all prerequisites that are newer than the target

    - `$^`: all prerequisites with duplicated entries removed
    - `$+`: all prerequisites without duplicated entries removed
- Special Rules
    - `.SUFFIXES`: old fashioned
    - `.PHONY`: don't check if a target is an existing file
- Pattern Rules: `%` symbol

Debug With GDB

- commands
    - list [line # | function | file:line # | file:function]: show source codes
    - run [arguments ...]: run a program
    - next: will not enter function
    - step: will enter a function
    - print: display
    - break [line # | function | file:line # | file:function]: set break point
    - clear [line # | function | file:line # | file:function]: delete break point
    - info breakpoints: show breakpoints
    - continue: run until a breakpoint
- debug
    - enable core dump: `ulimit -c unlimited` or `ulimit -c [limit size]`
    - `./bug2`
    - `gdb bug2 core`
    - `bt`