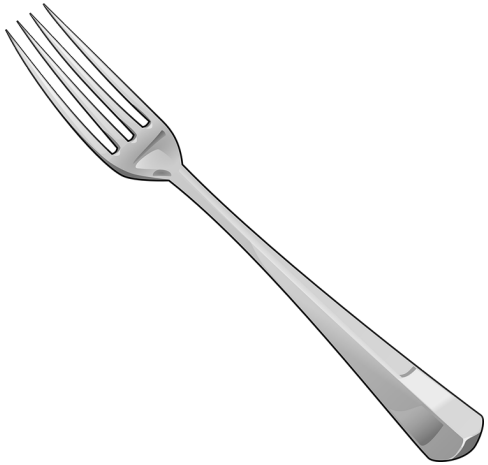
Using Fork and Pipe

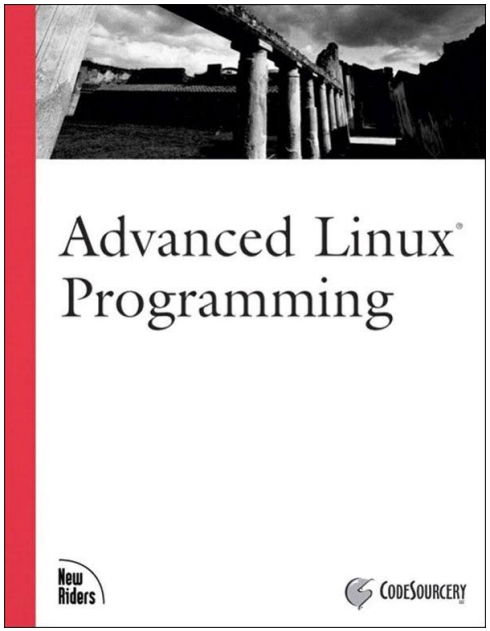
Methods & Tools for Software Engineering (MTSE)
Fall 2017

Prof. Arie Gurfinkel



UNIVERSITY OF WATERLOO

Additional Information



Advanced Linux Programming

- Chapter 4 (Processes)
- Chapter 5.4 (Pipes)

Old link appears to be broken

The book is still available from the links below

https://github.com/MentorEmbedded/advancedlinuxprogramming/blob/gh-pages/alp-folder/advanced-linux-programming.pdf

https://github.com/MentorEmbedded/advancedlinuxprogramming/tree/gh-pages

fork() system call

Forks an execution of the process

- after a call to fork(), a new process is created (called child)
- the original process (called parent) continues to execute concurrently
- in the parent, fork() returns the process id of the child that was created
- in the child, fork() return 0 to indicate that this is a child process

Man(ual) Page

- man 2 fork

exec() – executing a program in a process

exec() series of functions are used to start another program in the current process

- after a call to exec() the current process is replaced with the image of the specified program
- different versions allow for different ways to pass command line arguments and environment settings
- `int execv(const char *file, char *const argv[ ])`
  - file is a path to an executable
  - argv is an array of arguments. By convention, argv[0] is the name of the program being executed

Man page

- man 3 exec

kill() – sending a signal

A process can send a signal to any other process

- usually the parent process sends signals to its children
- `int kill(pid_t pid, int sig)`
  - send a signal `sig` to a process `pid`
- useful signal: SIGTERM
  - asks a process to terminate

When a parent process exits, the children processes are terminated

It's a good practice to kill and wait for children to terminate before exiting

Man page

- man 2 kill

waitpid() – Waiting for a child

A parent process can wait for a child process to terminate

- `pid_t waitpid(pid_t pid, int *stat_loc, int options)`
  - block until the process with the specified `pid` terminates
  - the return code from the terminating process is placed in `stat_loc`
  - options control whether the function blocks or not
    - 0 is a good choice for options

Man page

- man 2 wait

pipe() and dup2() – Inter-Process Communication

pipe() creates a ONE directional pipe

- two file descriptors: one to write to and one to read from the pipe
- a process can use the pipe by itself, but this is unusual
- typically, a parent process creates a pipe and shares it with a child, or between multiple children
- some processes read from it, and some write to it
  - there can be multiple writers and multiple readers
    - although multiple writers is more common

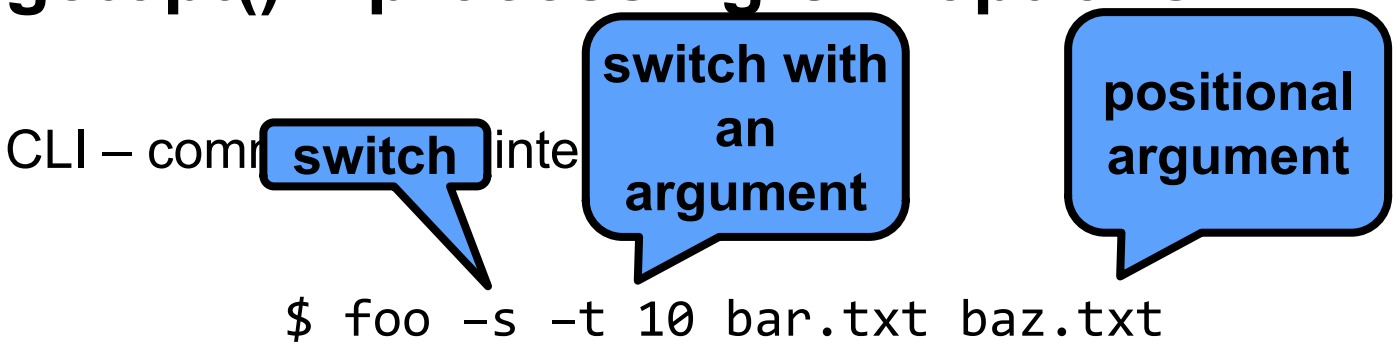
dup2() duplicates a file descriptor

- used to redirect standard input, standard output, and standard error to a pipe (or another file)
- STDOUT_FILENO is the number of the standard output

Man pages

- man 2 pipe
- man 2 dup2

getopt() – processing CLI options



At a start of the program, main(argc, argv) is called, where

- argc is the number of CLI arguments
- argv is an array of 0 terminated strings for arguments
  - e.g., argv[0] is “foo”, argv[1] is “-s”, argv[2] is “-t”, argv[2] is “10”, …

getopt() is a library function to parse CLI arguments

- `getopt(argc, argv, “st:”)`
- input: arguments and a string describing desired format
- output: returns the next argument and an option value
- see example in `using_getopt.cpp`

/dev/urandom – Really Random Numbers

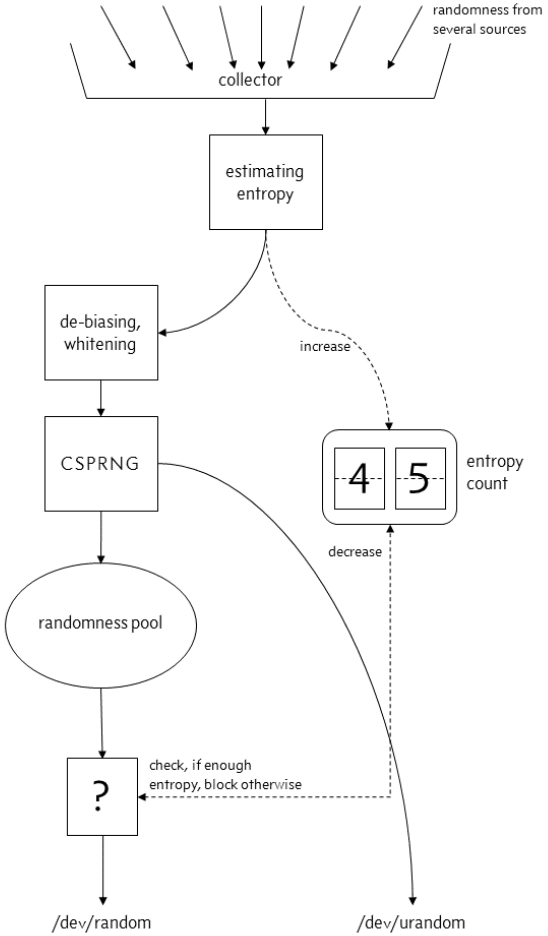
/dev/urandom is a special file (device) that provides supply of “truly” random numbers

”infinite size file” – every read returns a new random value

To get a random value, read a byte/word from the file

see `using_rand.cpp` for an example

Have to use it for Assignment 3!



https://www.2uo.de/myths-about-urandom/