ISO C/POSIX/Unix/Linux I/O & Fork Concepts

Prof. James L. Frankel
Harvard University

Version of 7:16 PM 25-Apr-2017

ISO C/POSIX Streams vs. File Descriptors

- ISO C: streams may be buffered
- POSIX: file descriptor I/O is not buffered

- stream I/O is built on top of file descriptor I/O

- file descriptors are non-negative ints
- streams are FILE *

Standard streams

- stdin
- stdout
- stderr

Standard file descriptors

- 0 is standard input (but use STDIN_FILENO)
- 1 is standard output (but use STDOUT_FILENO)
- 2 is standard error (but use STDERR_FILENO)

<stdio.h> (1 of 3)

- int fileno(FILE *stream); /* Returns the associated file descriptor */
- FILE *fdopen(int fildes, const char *mode);

- FILE *fopen(const char *restrict path, const char *restrict mode);
- int fclose(FILE *stream);

- int fgetc(FILE *stream);
- int getc(FILE *stream); /* Implemented as a macro; may evaluate stream more than once */
- int getchar(void);
- char *fgets(char *restrict, int, FILE *restrict);
- char *gets(char *s);

<stdio.h> (2 of 3)

- int fputc(int c, FILE *stream);
- int putc(int c, FILE *stream);/* Implemented as a macro; may evaluate stream more than once */
- int putchar(int);
- int fputs(const char *restrict, FILE *restrict);
- int puts(const char *);

- int fgetpos(FILE *restrict stream, fpos_t *restrict pos);
- int fsetpos(FILE *stream, const fpos_t *pos);

- long ftell(FILE *stream);
- int fseek(FILE *stream, long offset, int whence);

- int feof(FILE *stream);

<stdio.h> (3 of 3)

- int fflush(FILE *stream);

- int ferror(FILE *stream);
- void clearerr(FILE *stream);

- int setvbuf(FILE *stream, char *buf, int mode, size_t size);
  /* IOFBF (for full buffering),
  _IOLBF (for line buffering), or
  _IONBF (for unbuffered input/output) */

<fcntl.h>

- int open(const char *path, int oflag, ...);
- int fcntl(int fildes, int cmd, ...);

<unistd.h>

- STDIN_FILENO
- STDOUT_FILENO
- STDERR_FILENO
- int close(int fildes);
- ssize_t read(int fildes, void *buf, size_t nbyte);
- ssize_t write(int, const void *, size_t);
- int fstat(int fd, struct stat *buf);

<sys/stat.h>

- int fchmod(int fd, mode_t mode);
- int fchown(int fd, uid_t owner, gid_t group);

Implementation of file descriptors

- A file descriptor indexes into a per-process "descriptor table" to locate a file entry
  - In our system, this would be either in the PCB or in a separate data structure pointed to by the PCB
- The file entry is an entry in the system-wide file table
  - That entry provides a file mode (read, write, etc.) and a pointer to an underlying object (the vnode, or virtual-node) that enables object-oriented I/O

Spawn vs. Fork

- How Fork is implemented in Unix
  - #include <unistd.h>
  - pid_t fork(void);
  - The fork() function shall create a new process. The new process (child process) shall be an exact copy of the calling process (parent process) except for many details.
  - Upon successful completion, fork() shall return 0 to the child process and shall return the process ID of the child process to the parent process. Both processes shall continue to execute from the fork() function. Otherwise, -1 shall be returned to the parent process, no child process shall be created, and errno shall be set to indicate the error.
- How Unix Fork can be efficient