CS 61
Unit V: Process Management
Lecture 19: Application Process Management
November 11, 2014
Scribe Notes By: Avery Dao, Samuel Green, Mateusz Kulesza

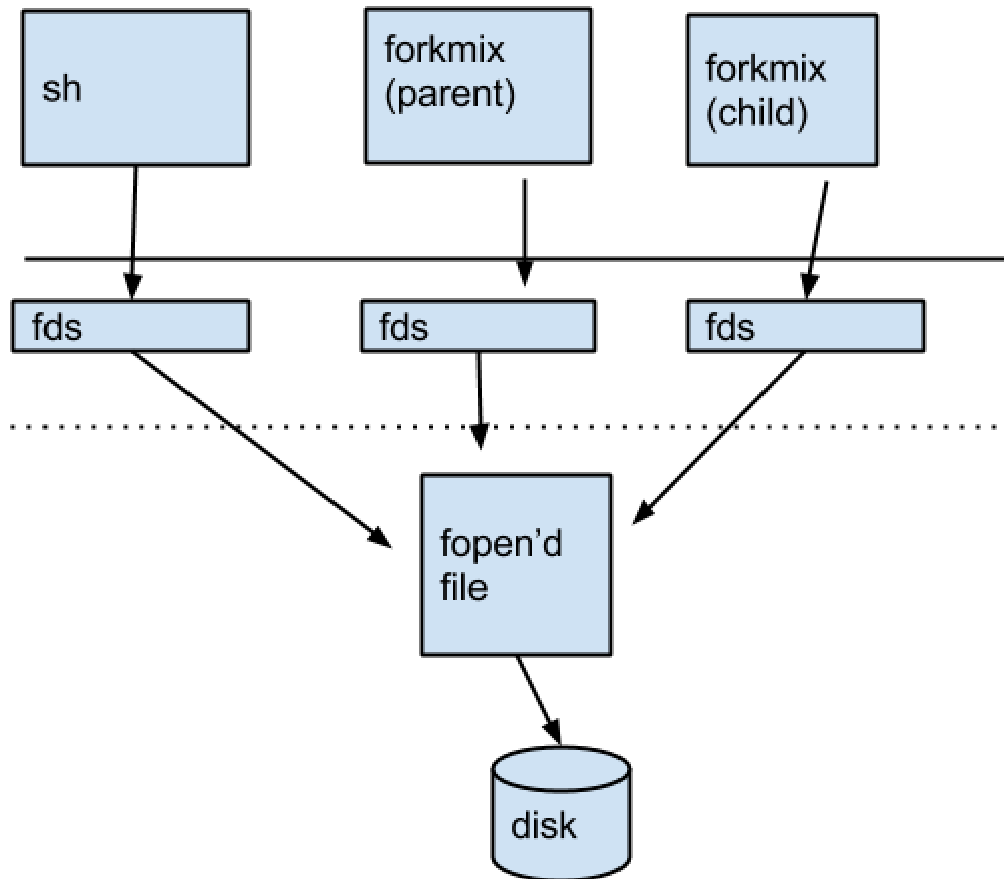
Fork:
- Parent does not necessarily have priority over the child: parent might not run first.
    - Both parent and child are isolated processes; each has equal access to the resources of the machine, but neither has priority.
- Use waitpid(**pid_t** pid_of_child, **int** *status**, int** options) to ensure parent waits for child to terminate before continuing.

Race condition:
- Process behavior depends on the order in which the process is run relative to other processes - depends on the process scheduler.
- Cause of very rare, hard-to-locate bugs.
- Ex. Race condition in fork:
    - Normally, parent runs first because fork() returns immediately to the process that called it
    - However, sometimes, child might run first in the rare case that there is a timer interrupt between fork() and the print.
- Good practice to program in a way that makes race conditions more common and therefore ensures your code handles them adequately.
- Linux has an (intentional) race conditional bug when redirecting output into a file.

Throughput and Latency:
- **Latency**: the time-delay involved in performing a given operation on one chunk of data.
- **Throughput**: the amount of data that can be processed in a given period of time.
- Leaf pile analogy: Moving one leaf from the leaf pile to the dump has low latency and low throughput. Using a wheelbarrow (a cache/buffer for leaves) to move the entire pile to the dump all at once has high latency and high throughput.
- In general, there is a tradeoff between latency and throughput.
- When stdio detects that STDOUT is the console, it prints immediately (low latency); when it detects that STDOUT is a file on disk, it buffers, prioritizes throughput over latency

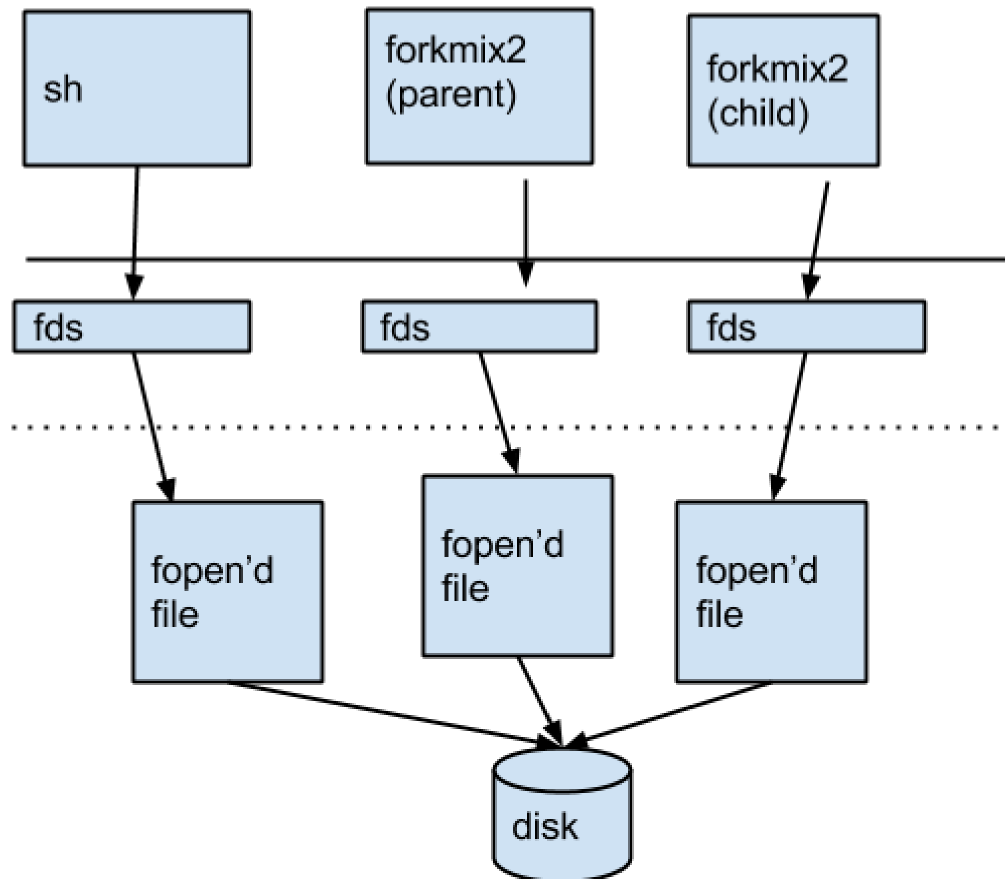
./forkmix > x

- Calls fopen() before forking.
- Parent and child processes have different file descriptor tables but share the same file pointer structure.
- Writes from either process advance the file pointer, ensuring that neither process overwrites the work of the other process.



./forkmix2 > x

- Calls fopen() after forking.
- Parent and child processes have different file descriptor tables and different file pointer structures. Both file pointer structures still point to the same disk space.
- Writes from either process can overwrite the writes of the other process because the file pointers are advanced independently - the parent does not know that the child has written to the file and vice versa.



Running a program: execvp

- Replaces the current process with a new process
- Either never returns (because process that called it disappears) or returns - 1
- Allows us to set up new environments for processes before calling them

The Sieve of Eratosthenes:

- Algorithm for finding prime numbers.
- Start with a list of all real numbers.
- Look for primes sequentially starting at 2: iterate through your real numbers list.
- If you find a prime, drop all multiples of that prime from your list of real numbers.
- Need pipes to implement this as a program.

Pipes:

- Pipes have two ends - a READ end and a WRITE end
- The READ end can only read data that has been written from WRITE end
- EOF indicated by all WRITE ends being closed - no processes point to the WRITE end.
    - All WRITE ends closed ⇒ READ end receives EOF
    - All READ ends closed ⇒ if process tries to write to WRITE end, it receives signal to die
- **Pipe hygiene** = Closing all ends of pipes that we do not need open
- Pipe buffer is 64 kB
- pipe() system call takes in an array int pipefd[2] (for example), which will contain the write (pipefd[0]) and read (pipefd[1]) file descriptors after the pipe is opened.

An Anecdote and Book Recommendation:

- Book: *Programming Pearls* by Jon Bentley
- Done Knuth and literate programming