IPC and Sockets:

* Key Idea: Communication between processes and across the world looks like File I/O
* Introduce Pipes and Sockets
* Introduce TCP/IP Connection set up for Web Server

pid\_t fork() 🡪 Copy current process

* State of original process duplicated in parent and child
* Address Space (Memory), FD, etc…
* Return value from fork():
  + When > 0: Running in original Parent process
  + When = 0: Running in new Child process
  + When < 0: Error!

Uniformity – Everything is a File!

* File operations, device I/O, and IPC through open, read/write, close
* Allows for simple composition of programs (find | grep | wc… )
* Open before use:
  + Provides opportunity for access control and arbitration
  + Sets up underlying machinery (ie, data structures)
* Byte-oriented
  + Even if blocks are transferred
  + (agnostic, doesn’t matter what data looks like)
* Kernel buffered reads
  + To give byte-oriented access (streaming and block devices looks the same)
* Kernel buffered writes
* A picture containing text, screenshot, diagram, font

  Description automatically generatedExplicit close

Putting it together: Web server (single process)

* Steps 3, 8 are kernel buffered reads
* Step 11 is kernel buffered write

High Level File API – Streams

* Operates on “streams”, unformatted sequence of bytes
* fopen returns FILE\*, data structure:
  + Error reported by returning nullptr
  + Pointer used in subsequent operations on stream
  + Data buffered in user space

Low-Level File I/O (RAW system-call interface):

* A screenshot of a computer code

  Description automatically generated with low confidenceInteger return from open() is a file descriptor
  + File descriptor used in subsequent operations on the file
* Streams (opened with fopen()) have file descriptor inside of them!
  + Retrievable with fileno(FILE\* stream) 🡪 internal file descriptor

Representation of Process (inside kernel):

A diagram of a file

Description automatically generated with low confidence

Can change position with seek operations

Communication between processes:

* What if processes wish to communicate with one another?
  + Why? Shared Task, Cooperative Venture w/ Security Implications
* Process Abstraction designed to Discourage IPC
* So, must do something special (and agreed upon by both processes)
  + Must “Punch Hole” in security
* Called “Interprocess Communication” 🡺 “IPC”

Recall: Processes protected from each other (translation mapping)

* Producer (writer) and consumer (reader) may be distinct processes
  + Potentially separated in time
  + How to allow selective communication?
* A picture containing text, screenshot, font, logo

  Description automatically generatedSimple option: Use a file!
  + Parents and children share file descriptions
  + Very slow, requires persisting information to disk just to allow processes to communicate
  + On the right track… processes are communicating with file descriptors
* Can use shared memory through translation mapping (both processes have shared piece of virtual memory that maps onto same physical address)

Suppose we ask Kernel to help?

* Consider an in-memory queue?
* Accessed via system calls (for security reasons)
* Data written by A is held in memory until B reads it
  + Same interface as we use for files
  + Internally more efficient, since nothing goes to disk
  + How to set it up?
    - What if A generates data faster than B can consume it?
    - What if B generates data faster than A can produce it?
    - A green and pink rectangle with black text

      Description automatically generated with low confidenceUse WAIT

Example: POSIX/Unix PIPE

* Memory buffer is finite
  + If producer(A) tries to write when buffer is full, it blocks
  + If consumer(B) tries to read when buffer empty, it blocks
* int pipe(int fileds[2]);
  + Allocates two new file descriptors in the process
  + Writes to fileds[1] read from fileds[0]
  + A green and black logo

    Description automatically generated with low confidenceImplemented as fixed-size queue
* Kernel knows when A writes and B reads (will wake up waiting process)