Lecture 18, Feb 27, 2024

Sockets

- Sockets are another form of IPC that allows communication over a network (in addition to on the same machine)
  - All network connections have to go through sockets
- After sockets are set up, a file descriptor is returned that we can `read()` and `write()` to as usual and `close()` when done
- The server follows these steps:
  1. `int socket(int domain, int type, int protocol);`: Creates the socket
     - `domain` specifies the general protocol
       - `AF_UNIX`: local communication of the same machine
       - `AF_INET`: IPv4
       - `AF_INET6`: IPv6
     - `type` is either `SOCK_STREAM` or `SOCK_DGRAM` (TCP or UDP)
       - For stream connections, the data sent arrives in the same order; we have a persistent connection (we'll know when we lose it), which is reliable but slow
       - For datagram connections, there is no guarantee of arrival order and connection persistence, but is faster
     - `protocol` further specifies the protocol and is mostly unused
     - Returns a file descriptor (but for a server we shouldn't read/write to this)
  2. `int bind(int socket, const struct sockaddr *addr, socklen_t addr_len);`: Attach the socket to some location (a file, IP and port, etc)
     - 3 different types of `sockaddr` structures: `sockaddr_un` (UNIX socket, i.e. a path), `sockaddr_in` (IPv4 address), `sockaddr_in6` (IPv6 address)
     - `addr_len` is `sizeof(sockaddr)`
     - Set `sun_type` of the `sockaddr` struct to the same as the domain of the socket and `sun_path` to the path (note this is a `char[]`, not `char*`, so size is limited)
     - For UNIX sockets, we should use `int unlink(const char *pathname);` to clean up the socket path (after closing the socket); otherwise the file corresponding to the socket will remain
  3. `int listen(int socket, int backlog);`: Listen for connections on the socket and sets the queue limit
     - `backlog` is the limit of outstanding connections queue, managed by the kernel; passing 0 uses the default kernel queue size
       - If the queue is full, new connections will not be allowed
  4. `int accept(int socket, struct sockaddr *address, socklen_t *address_len);`: Accept an incoming connection
     - `address`, `address_len` is an optional return of the connecting address (`NULL` to ignore)
     - This returns a file descriptor we can read and write to, corresponding to the new client connection
     - Will block until a client connects
- The client follows these steps:
  1. `int socket(int domain, int type, int protocol);`: Creates the socket
  2. `int connect(int sockfd, const struct sockaddr *addr, socklen_t addr_len);`: Connects to some location, giving a file descriptor
     - This will use the same name as the `bind()` call of the server
     - On success, `sockfd` may be used as a normal file descriptor (the function returns 0 on success)
- Instead of read and write, we can use `send()` and `recv()` syscalls, which are similar but take additional flags
  - e.g. `MSG_OOB` (send/receive out-of-band data), `MSG_PEEK` (look at data without reading), `MSG_DONTROUTE` (send without routing packets)
  - `sendto()` and `recvfrom()` take an additional address
    - Ignored for stream sockets since there's a persistent connection