Operating Systems
COMS W4118
Reading Notes 6

Alexander Roth

2015 – 02 – 23

Advanced Programming in the Unix Environment Chapter 14: Advanced I/O

14.2 Nonblocking I/O

- "Slow" system calls are those that can block forever.
- Slow systems calls are
  - Reads that can block the caller forever if data isn't present with certain file types (pipes, terminal devices, and network devices).
  - Writes that can block the caller forever if the data can't be accepted immediately by these same file types (e.g. no room in the pipe, network flow control)
  - Opens that block until some condition occurs on certain file types (such as an open of a terminal device that waits until an attached modem answers the phone, or an open of a FIFO for writing only, when no other process has the FIFO open for reading)
  - Reads and writes of files that have mandatory record locking enabled.
  - Certain `ioctl` operations
  - Some of the interprocess communication functions.
- System calls related to disk I/O are not considered slow, even though they can block the caller temporarily.
- Nonblocking I/O lets us issue an I/O operation and not have it block forever.
- If the operation cannot be completed, teh call returns immediately with an error noting that the operation would have blocked.

- There are two ways to specify nonblocking I/O for a given descriptor.
  1. If we call `open` to get the descriptor, we can specify the `O_NONBLOCK` flag
  2. For a descriptor that is already open, we call `fcntl` to turn the `O_NONBLCOK` file status flag.

14.4 I/O Multiplexing

- *Asynchronous I/O* is the technique by which we tell the kernel to notify us with a signal when a descriptor is ready for I/O.
- It is not portable and we cannot tell which descriptor is ready.
- *I/O multiplexing* is when we build a list of descriptors that we are interested in and call a function that doesn't return until one of the descriptors is ready for I/O.
- `poll`, `pselect`, and `select` allow us to perform I/O multiplexing.
- On return from these functions, we are told which descriptors are ready for I/O.

14.4.1 `select` and `pselect` Functions

- The `select` function is portable to all POSIX-compatible platforms.
- The arguments we pass to `select` tell the kernel
  - Which descriptors we're interested in.
  - Which conditions we're interested in for each descriptor
  - How long we want to wait
- On the return from select, the kernel tell us
  - The total count of the number of descriptors that are ready
  - Which descriptors are ready for each of the three conditions (read, write, or exception condition)

```
#include <sys/select.h>
int select(int maxfdp1, fd_set *restrict readfds, fd_set *restrict
writefds, fd_set *restrict exceptfd, struct timeval *restrict tvptr);
```
Returns: count of ready descriptors, 0 on timeout, -1 on error

- The last argument specifies how long we want to wait in terms of seconds and microseconds.

`tvptr == NULL` Wait forever. Can be interrupted if we catch a signal. Return is made when one of the specified descriptors is ready or when a signal is caught. If a signal is caught, returns -1 with `errno` set to `EINTR`.

`tvptr->tv_sec == 0 && tvptr->tv_usec == 0` Don't wait at all. All specified descriptors are tested, and return is made immediately. Polls the system to find out the status of descriptors without blocking in the function.

`tvptr->tv_sec  = 0 || tvptr->tv_usec != 0` Wait the specified number of seconds and microseconds. Return is made when one of the specified descriptors is ready or when the timeout value expires.

- The middle three arguments are pointers to *descriptor sets*.
- A descriptor set is stored in an `fd_set` data type.
- The only thing we can do with the `fd_set` data type is allocate a variable of this type, assign a variable of this type to another variable of the same type, or use one of the following four functions on a variable of this type.

```
#include <sys/select.h>
int FD_ISSET(int fd, fd_set *fdset);
```
Returns: nonzero if *fd* is in set, 0 otherwise

```
void FD_CLR(int fd, fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_ZERO(fd_set *fdset);
```

- These interface can be implemented as either macros or functions.
- The first argument of `select` is really a count of the number of descriptor to check.
- There are three possible return values from `select`.
    1. A return value of -1 means that an error occurred.
    2. A return value of 0 means that no descriptors are ready.
    3. A positive return value specifies the number of descriptors that are ready.
- A descriptor is said to be ready when
    - if it is in the read set and if a `read` from that descriptor won't block.
    - if it is in the write set and if a `write` from that descriptor won't block.
    - If it is in the exception set if an exception condition is pending on that descriptor.

  - File descriptors for regular files always return ready for reading, writing, and exception conditions.

- A descriptor blocking does not affect if `select` blocks.