BUCKNELL UNIVERSITY
Computer Science

CSCI 315 Operating Systems Design

I/O Systems
Part 3

Notice: The slides for this lecture have been largely based on those accompanying the textbook *Operating Systems Concepts*, 9th edition by Silberschatz, Galvin, and Gagne (2013). Many, if not all, of the illustrations contained in this presentation come from this source. Revised by Xiannong Meng based on Professor Perrone's notes.

Block and Character Devices

- Block devices include disk drives.
  - Commands include read, write, seek.
  - Raw I/O or file-system access.
  - Memory-mapped file access possible.
- Character devices include keyboards, mice, serial ports.
  - Single character (byte) input and output
  - Libraries layered on top allow line editing.

Network Devices

- Different enough from block and character to have their own interface.
- Unix/Linux and Windows NT/9x/2000 include socket interface:
  - Separates network protocol from network operation.
  - Includes `select` functionality.
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes).

Clocks and Timers

- Provide current time, elapsed time, timer.
- If programmable interval time used for timings, periodic interrupts.
- `ioctl` (on UNIX/Linux) covers odd aspects of I/O such as clocks and timers.

Blocking and Nonblocking I/O

- **Blocking** - process suspended until I/O completed.
  - Easy to use and understand.
  - Insufficient for some needs.
- **Nonblocking** - I/O call returns as much as available.
  - User interface, data copy (buffered I/O).
  - Implemented via multi-threading.
  - Returns quickly with count of bytes read or written.
- **Asynchronous** - process runs while I/O executes.
  - Difficult to use.
  - I/O subsystem signals process when I/O completed.

Code example: select

```
fd_set readmask;   /* a bit-mask to represent active devices */
int maxfdp;        /* maximum number of devices to check */
int nfound;        /* number of devices found to have data */

/* initialize the read mask */
FD_ZERO(&readmask);

/* we will be checking standard input (file 0) and the network (file sock) */
/* both the keyboard and the network socket is assumed ready */
FD_SET(0, &readmask);
FD_SET(sock, &readmask);

nfound = select(maxfdp, &readmask, (fd_set *)0, (fd_set*)0, (struct timeval *)0);

if (FD_ISSET(0, &readmask)) { /* do keyboard reading */ }
if (FD_ISSET(sock, &readmask)) { /* do network reading */}
```

http://www.eg.bucknell.edu/~cs315/2013-fall/code-examples/io/select.c

Code example: ioctl

```
struct termio savetty;
struct termio * newtty;
…
/* retrieve tty attributes */
if (ioctl(0, TCGETA, &savetty) != -1) { /* get tty attributes succeeds */}
…
/* save savetty to newtty, change newtty features */
…
/* set tty with new attributes */
if (ioctl(0, TCSETAF, newtty) != -1) { /* set attributes succeeds */ }
```

http://www.eg.bucknell.edu/~cs315/2013-fall/code-examples/io/rawread.c

Some of the termio attributes

- c_iflag: input attributes
- c_oflag: output attributes
- c_lflag: controls various terminal functions
- c_cc: specifies an array that defines the special control character

Code example: tty

- How to use tty?
- In Linux/Unix, everything (device) is a file, so are the terminals.
- We write to/ read from a terminal as if they were regular files.

http://www.eg.bucknell.edu/~cs315/2013-fall/code-examples/io/tty.c