EECS 2031 3.0 A Software Tools

Week 7: October 24, 2018

Input & output

- Before Unix…
- Generally a separate interface for every possible input and output
  - A read character for the terminal operation was different than the read character for the disk operation which was different than the read character for something else

DOS2.0

AH	Description	AH	Description
01	Read character from STDIN	02	Write character to STDOUT
05	Write character to printer	06	Console Input/Output
07	Direct char read (STDIN), no echo	08	Char read from STDIN, no echo
09	Write string to STDOUT	0A	Buffered input
0B	Get STDIN status	0C	Flush buffer for STDIN
0D	Disk reset	0E	Select default drive
19	Get current default drive	25	Set interrupt vector
2A	Get system date	2B	Set system date
2C	Get system time	2D	Set system time
2E	Set verify flag	30	Get DOS version
35	Get Interrupt vector		
36	Get free disk space	39	Create subdirectory
3A	Remove subdirectory	3B	Set working directory
3C	Create file	3D	Open file
3E	Close file	3F	Read file
40	Write file	41	Delete file
42	Seek file	43	Get/Set file attributes
47	Get current directory	4C	Exit program
4D	Get return code	54	Get verify flag
56	Rename file	57	Get/Set file date

So in Unix (and many OS's afterwards)

- Different devices were all mapped to 'files' in the file system
  - Look in /dev/

```
crw-rw-rw-  1 root    wheel          33,   0  1 Sep 09:15 autofs_homedirmounter
crw-rw-rw-  1 root    wheel          31,   0  1 Sep 09:15 autofs_notrigger
crw-rw-rw-  1 root    wheel          22,  59  1 Sep 09:15 autofs_nowait
crw--------  1 root    wheel          23,   0  1 Sep 09:15 bpf0
crw--------  1 root    wheel          23,   1  1 Sep 09:15 bpf1
crw--------  1 root    wheel          23,   2 23 Oct 18:40 bpf2
crw--------  1 root    wheel          23,   3 20 Oct 16:27 bpf3
crw--------  1 root    wheel          23,   4  1 Sep 09:25 bpf4
crw--------  1 jenkin  staff           0,   0 16 Oct 22:51 console
crw-rw-rw-  1 root    wheel          19,   5  1 Sep 09:15 cu.Bluetooth-Incoming-Port
crw-rw-rw-  1 root    wheel          19,   3  1 Sep 09:15 cu.lpss-serial1
crw-rw-rw-  1 root    wheel          19,   1  1 Sep 09:15 cu.lpss-serial2
brw-r-----  1 root    operator        1,   0  1 Sep 09:15 disk0
brw-r-----  1 root    operator        1,   1  1 Sep 09:15 disk0s1
brw-r-----  1 root    operator        1,   3  1 Sep 09:15 disk0s2
brw-r-----  1 root    operator        1,   2  1 Sep 09:15 disk0s3
brw-r-----  1 root    operator        1,   4  1 Sep 09:15 disk1
br--r-----  1 root    operator        1,   5 12 Sep 15:11 disk2
br--r-----  1 root    operator        1,   6 12 Sep 15:11 disk2s1
br--r-----  1 root    operator        1,   7 12 Sep 15:11 disk2s2
br--r-----  1 root    operator        1,   8 10 Oct 21:17 disk3
br--r-----  1 root    operator        1,  10 10 Oct 21:17 disk3s1
br--r-----  1 root    operator        1,   9 10 Oct 21:17 disk3s2
br--r-----  1 root    operator        1,  11 16 Oct 12:54 disk4
br--r-----  1 root    operator        1,  12 16 Oct 12:54 disk4s1
br--r-----  1 root    operator        1,  13 16 Oct 12:54 disk4s2
crw-rw-rw-  1 root    wheel          24,   2  1 Sep 09:15 dtrace
crw-rw-rw-  1 root    wheel          25,   0  1 Sep 09:15 dtracehelper
crw-rw-rw-  1 root    wheel           0,   0  1 Sep 09:15 fbt
dr-xr-xr-x  1 root    wheel                0  1 Sep 09:15 fd
crw-r--r--  1 root    wheel          13,   0  1 Sep 09:15 fsevents
```

This meant that a common API would work across (character — c) devices

- This resulted in what is (now) known as POSIX IO
  - Defined in <fcntl.h>
- open - opens a file and returns a file descriptor
- creat - create a file (create spelled poorly)
- close - closes an open file descriptor
- read - reads bytes from a file descriptor
- write -writes bytes from a file descriptor
- lseek - moves the read/write point on a file descriptor
- ioctl - operations not defined above
- And many others

Include files

- #include <sys/types.h>
- #include <unistd.h>
- #include <sys/ioctl.h>
- #include <fcntl.h>

open

- open(fname, mode, ….)
  - Mode O_RDONLY, O_WRONLY, O_RDWR, …
  - Many other options
  - Returns -1 on failure, otherwise a low positive integer
  - 0, 1, 2 are bound to stdin, stdout, stderr when you start.

close

- close(fd)
  - Closes the file, releases the file descriptor to be re-used
  - There is a maximum number of these, so if you open many (many) files, you can run out
  - Returns -1 on error

read

- read(fd, void *ptr, int count)
  - Reads from the fd, into ptr at most count bytes
  - Returns the number of bytes read (blocks)
  - -1 on EOF, 0 if none available

write

- write(fd, ptr, count)
  - Writes bytes on fd, from ptr, up to count bytes
  - returns number of bytes written, -1 on error

lseek

- lseek(fd, offset, whence) - sets read/write location
  - Offset is set to this number of bytes
  - whence is SEEK_SET (from start), SEEK_CUR (from current), SEEK_END (from end)
  - Allows (among other things) to append to a file

ioctl

- ioctl(fd, request, …) - do some special thing on the device
- There are many other functions, but these cover the basics.

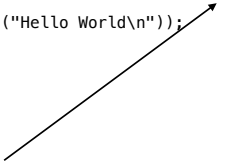
```c
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>


int main(int argc, char *argv[])
{
  int fd = open("test.text", O_CREAT | O_TRUNC | O_WRONLY, 0600);
  if(fd >= 0) {
    write(fd, "Hello World\n", strlen("Hello World\n"));
    close(fd);
  } else
    perror("open fails");
  return 0;
}
```

Create the file, truncate it and we want to write

```c
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>


int main(int argc, char *argv[])
{
  int fd = open("test.text", O_CREAT | O_TRUNC | O_WRONLY, 0600);
  if(fd >= 0) {
    write(fd, "Hello World\n", strlen("Hello World\n"));
    close(fd);
  } else
    perror("open fails");
  return 0;
}
```



File permission (rw- - - - - - -)

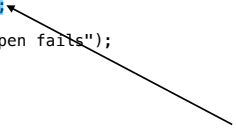
```c
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>


int main(int argc, char *argv[])
{
  int fd = open("test.text", O_CREAT | O_TRUNC | O_WRONLY, 0600);
  if(fd >= 0) {
    write(fd, "Hello World\n", strlen("Hello World\n"));
    close(fd);
  } else
    perror("open fails");
  return 0;
}
```

Write “Hello World\n”

```c
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>


int main(int argc, char *argv[])
{
  int fd = open("test.text", O_CREAT | O_TRUNC | O_WRONLY, 0600);
  if(fd >= 0) {
    write(fd, "Hello World\n", strlen("Hello World\n"));
    close(fd);
  } else
    perror("open fails");
  return 0;
}
```



Close the file descriptor

```c
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>


int main(int argc, char *argv[])
{
  int fd = open("test.text", O_CREAT | O_TRUNC | O_WRONLY, 0600);
  if(fd >= 0) {
    write(fd, "Hello World\n", strlen("Hello World\n"));
    close(fd);
  } else
    perror("open fails");
  return 0;
}
```

Do ‘man perror’

On top of this API

- The stdio library (#include <stdio.h>)
  - fopen - open
  - fclose - close
  - fread - read
  - fwrite - write
  - fseek - seek
  - flush - flush buffers
  - And others….

fopen

- fopen(file,mode)
  - Returns a FILE* ptr (stream)
  - Mode “r”, “w”, “a” (others too)

fclose

- fclose(fd) - close the stream
  - Returns 0 on success
  - Flushes all text being written

fread

- fread(ptr, size, n, fd) - read into ptr, n items of size n from fd
- Returns number of items read
- Use feof(fd) to test for end of file

fwrite

- fwrite(ptr, size, n, fd) - write from ptr, n items of size n to fd
- Returns number of items written (if less than n, an error)

fflush

- fflush(fd) - flush the stream
- Streams do not necessarily write when you say to (they optimize things)
- If you need the write to have taken place, fflush makes this happen
- fflush happens automatically on fclose

```
#include <stdio.h>
#include <stdlib.h>


int main(int argc, char *argv[])
{
  FILE *fd;
  int n;
  char buf[101];

  if((fd = fopen("test.text", "r")) == (FILE *)NULL) {
    perror("open of test.text failed");
    (void) exit(1);
  }
  n = fread(buf, 1, 100, fd);
  if(n == 0) {
    perror("read fails!");
    exit(1);
  }
  buf[n] = '\0';
  fprintf(stdout, "Read '%s'\n", buf);
  (void) fclose(fd);
  return 0;
}
```

Open file test.text for reading

```c
#include <stdio.h>
#include <stdlib.h>


int main(int argc, char *argv[])
{
  FILE *fd;
  int n;
  char buf[101];

  if((fd = fopen("test.text", "r")) == (FILE *)NULL) {
    perror("open of test.text failed");
    (void) exit(1);
  }
  n = fread(buf, 1, 100, fd);
  if(n == 0) {
    perror("read fails!");
    exit(1);
  }
  buf[n] = '\0';
  fprintf(stdout, "Read '%s'\n", buf);
  (void) fclose(fd);
  return 0;
}
```

Read 1 byte, up to 100 of them

```c
#include <stdio.h>
#include <stdlib.h>


int main(int argc, char *argv[])
{
  FILE *fd;
  int n;
  char buf[101];

  if((fd = fopen("test.text", "r")) == (FILE *)NULL) {
    perror("open of test.text failed");
    (void) exit(1);
  }
  n = fread(buf, 1, 100, fd);
  if(n == 0) {
    perror("read fails!");
    exit(1);
  }
  buf[n] = '\0';
  fprintf(stdout, "Read '%s'\n", buf);
  (void) fclose(fd);
  return 0;
}
```

Null terminate the buffer

```c
#include <stdio.h>
#include <stdlib.h>


int main(int argc, char *argv[])
{
  FILE *fd;
  int n;
  char buf[101];

  if((fd = fopen("test.text", "r")) == (FILE *)NULL) {
    perror("open of test.text failed");
    (void) exit(1);
  }
  n = fread(buf, 1, 100, fd);
  if(n == 0) {
    perror("read fails!");
    exit(1);
  }
  buf[n] = '\0';
  fprintf(stdout, "Read '%s'\n", buf);
  (void) fclose(fd);
  return 0;
}
```

Close the file stream

Lab 06

- Linux maps devices to files, which makes it easier to interface with them.
- In your kit you have a temperature sensor that can be configured to do exactly that.

1-wire

- 1-wire is a (one wire) protocol for serial devices
- The temperature sensor supports this.
- By default 1-wire is not enabled, so you have to enable it
  - You end up loosing one GPIO pin to the 1-wire protocol

1-wire

Serial number of your device

```
pi@mypi:/sys/bus/w1/devices/28-0118423e95ff$ ls -l
total 0
lrwxrwxrwx 1 root root    0 Oct 24 03:44 driver -> ../../../bus/w1/drivers/w1_slave_driver
drwxr-xr-x 3 root root    0 Oct 24 03:44 hwmon
-r--r--r-- 1 root root 4096 Oct 24 03:44 id
-r--r--r-- 1 root root 4096 Oct 24 03:44 name
drwxr-xr-x 2 root root    0 Oct 24 03:44 power
lrwxrwxrwx 1 root root    0 Oct 24 03:44 subsystem -> ../../../bus/w1
-rw-r--r-- 1 root root 4096 Oct 24 03:44 uevent
-rw-r--r-- 1 root root 4096 Oct 24 03:44 w1_slave
```

1-wire

```
pi@mypi:/sys/bus/w1/devices/28-0118423e95ff$ ls -l
total 0
lrwxrwxrwx 1 root root    0 Oct 24 03:44 driver -> ../../../bus/w1/drivers/w1_slave_driver
drwxr-xr-x 3 root root    0 Oct 24 03:44 hwmon
-r--r--r-- 1 root root 4096 Oct 24 03:44 id
-r--r--r-- 1 root root 4096 Oct 24 03:44 name
drwxr-xr-x 2 root root    0 Oct 24 03:44 power
lrwxrwxrwx 1 root root    0 Oct 24 03:44 subsystem -> ../../../bus/w1
-rw-r--r-- 1 root root 4096 Oct 24 03:44 uevent
-rw-r--r-- 1 root root 4096 Oct 24 03:44 w1_slave
```

Structure of the directory

1-wire

```
pi@mypi:/sys/bus/w1/devices/28-0118423e95ff$ cat w1_slave
46 01 4b 46 7f ff 0c 10 2f : crc=2f YES
46 01 4b 46 7f ff 0c 10 2f t=20375
```

File of two lines of text

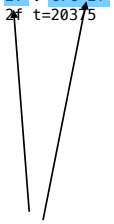
1-wire

```
pi@mypi:/sys/bus/w1/devices/28-0118423e95ff$ cat w1_slave
46 01 4b 46 7f ff 0c 10 2f : crc=2f YES
46 01 4b 46 7f ff 0c 10 2f t=20375
```

Raw message format (duplicated)

1-wire

```
pi@mypi:/sys/bus/w1/devices/28-0118423e95ff$ cat w1_slave
46 01 4b 46 7f ff 0c 10 2f : crc=2f YES
46 01 4b 46 7f ff 0c 10 2f t=20375
```



Message has a CRC which passed

1-wire

```
pi@mypi:/sys/bus/w1/devices/28-0118423e95ff$ cat w1_slave
46 01 4b 46 7f ff 0c 10 2f : crc=2f YES
46 01 4b 46 7f ff 0c 10 2f t=20375
```

Temperature in C * 1000 = 20.375C

Lab06

- Monitor the temperature
- Connect to IFTTT server whenever the temperature changes by more than 1C from the last measurement

Summary

- Unix has a file-based device model (very general)
- C supports POSIX (open/close) and stdio (fopen/fclose) API's
- stdio is a wrapper to posix, but exists on non-unix machines to often more portable.
- stdio can be much more efficient but you have to be aware of the nature of the efficiency.
- Character devices map to a file in the file system.