CSF 2019 Programming Assignment #1

Overview

For this assignment, you will be writing a program in C that converts the date fed to in on the standard input file descriptor into a hexadecimal format on the standard output file descriptor. For instance, if you program is fed a text file containing the sixteen hexadecimal digits, your program should output:

```
0000  30 31 32 33 34 35 36 37  01234567
0008  38 39 41 42 43 44 45 46  89ABCDEF
```

Each line is composed of:

1. A 16 bit address, displayed as a 4 digit hexadecimal number followed by two spaces. If the address reaches `FFF8` the next address should be `0000`.
2. Eight bytes of data, displayed as 2 digit hexadecimal numbers separated by single spaces and followed by two spaces.
3. Eight bytes of data, displayed as characters, followed by a newline. Any unprintable character (ASCII less than 32 or greater than 126) is displayed as a period.

This format is inspired by the early home computers of the 1975-1985 era which had 64kbytes (65536 bytes) of memory and a limited number of columns for displaying text (using fixed-width fonts).

Requirements

1. Your code must compile with the command `cc -o hex hex.c`. Among other implications, this means that your source code file must be called `hex.c` and it must be written in C.
2. Your code may not generate errors or warnings when compiled.
3. You are only permitted one include statement: `#include <unistd.h>`. You are not permitted to copy the contents of a system include file to get around this restriction. Among other implications, this means that you must use `read()` and `write()` for input and output.
4. You may assume that file descriptors 0, 1 and 2 are available: 0 is the input (for use with `read()`), 1 is for output (for use with `write()`) and 2 is for errors (also for use with `write()`).
5. If you want partial credit or CA assistance, your code must be readable. **CA are allowed to refuse to review messy and/or unreadable code!**

Examples

The supplied example input files are:

- `ex1_in`: a 16 byte file containing the sixteen hexadecimal digits.
- `ex2_in`: a 17 byte file containing the sixteen hexadecimal digits and a newline.
- `ex3_in`: a 65544 byte file containing random bytes.

Each example input file has a corresponding output file. To test your code output, use `diff`. For instance:

```
cc -o hex hex.c
./hex < ex1_in > my1_out
diff my1_out ex1_out
```

Documentation

The Linux system comes with extensive documentation on commands and library functions. These are accessed with the `man` command. For instance:

- `man diff`
- `man od` (a utility for displaying data in octal, decimal, hex, *etc.*)
- `man xxd` (a utility for displaying data in hex which is part of some Linux distributions).

For C library functions, you sometimes need to specify the manual *section*. For instance, `man read` usually returns information from section 1 on the read command built into the shell (bash). Library functions are usually either in section 2 or section 3:

- `man 2 read`
- `man 2 write`
- `man 3 printf` (but don't use `printf` in your code.

Error handling

The `man` page on `read()` notes that the function may return an error (*i.e.* return a value less than zero) with an error of type `EINTR` (interrupted system call). Normally you should just retry the read if this happens, but it is difficult to portably check this condition without additional `#includes`, so for this assignment, just ignore it and perform a `return(1)` on **any** error.