MYY502 – SYSTEMS PROGRAMMING

Lab #7
1/12/2015

I. BINARY FILE COPY (MINI CP)

You have to process binary files using `open()`, `read()`, `write()`, `close()`, and implement a simple version of the "`cp`" command.

You have to write a program where `main()` accepts, as arguments, the name of two (binary) files and copies the first onto the second. In particular, it should open the first file for reading, and the second one for writing (with truncation)—give initial permissions of the `0755` type, using the appropriate flags. In a loop, read 1 byte at a time from the first file and write it to the second file.

You can try your program e.g. to copy `a.out` to `new.out`, which clearly, should be made executable. If all goes OK, `new.out` should run exactly as `a.out` did.

II. AVI VIDEO FILES

In this problem, you have to create a program which finds and displays basic information about AVI video files. Among the first bytes of all these files there is information ("meta-data") related to the video they contain. These bytes constitute the *header*. The AVI header **starts on the 32nd byte of the file** and occupies exactly 56 bytes; the information these bytes store is displayed in Table 1.

You are asked to write a program which will open an AVI file, read some bytes from its header and display information about the video. In particular, `main()` should accept the name of the video file as an argument, and open it as a binary file using `open()`. Then, using `read()` and `lseek()` calls, it should read the appropriate integers from the header, in order to caclulate and display the following info:

1. Frame size in pixels (width, height)
2. Frame rate (frames-per-second / fps)
3. Total duration (in sec)

For 1, everything is included in the header (width/height of video image in pixels). For 2, you have to calculate:

$$\text{fps} = \text{data\_rate}/\text{time\_scale},$$

which are given towards the end of the header. Finally, for 3, the total duration follows from the frame rate and the

Table 1: AVI header (offsets/sizes in bytes). Remember that the header starts on the 32nd byte of the file.

offset	size	description
0	4	time delay between frames in microseconds
4	4	data rate of AVI data
8	4	padding multiple size, typically 2048
12	4	parameter flags
16	4	number of video frames
20	4	number of preview frames
24	4	number of data streams (1 or 2)
28	4	suggested playback buffer size in bytes
32	4	width of video image in pixels
36	4	height of video image in pixels
40	4	time scale, typically 30
44	4	data rate (frame rate = data rate / time scale)
48	4	starting time, typically 0

total number of frames:

$$\text{duration (sec)} = \text{number\_of\_video\_frames/fps}.$$

III. VARIADIC FUNCTIONS

In this problem, you will create a function with a variable number of arguments. In particular, you have to create function `printnums()`, which accepts, as its first argument, the number of values to print. However, the values can be either integers or floats. Hence, before the actual value there is an extra parameter that marks the type of the value that follows: 'd' means an integer follows, 'f' means a float follows. For example, the call:

```
printnums(3, 'd', 12, 'f', 23.4, 'd', 14)
```

has 7 arguments in total and it should print 3 values. The first and third values are integers, while the second is a float.

IV. BITWISE OPERATORS

Write a function `void printbin(int n)` which prints number `n` in binary form (i.e. it prints its bits). Use *bitwise operators* (&, |, <<, >>, ~) so as to discover the 32 bits one-by-one, using the algorithm that follows. Try `printbin()`, by calling it from `main()` using various numbers; you should try it at least for 0, 1 and also for -1.

Print Bits Algorithm

```
Input: Number to print ( n )

1. Create a mask for extracting the most significant bit (MSB):
     mask = 1, shifted 31 positions to the left

2. Print the bits:
     Repeat 32 times:
        If (using the mask) the MSB is 0:
           print 0
        else
           print 1
        End-If
        Shift n by 1 position to the left
     End-Repeat
```

V. MORE PRACTICE WITH BINARY FILES

Search the Internet for information about the structure of files that interest you, e.g.:

- music files (mp3, wav, κλπ)
- video files (avi, 3gp, κλπ)
- system files (e.g. the ELF structure of "a.out")

Based on the information about their structure, try to build programs that open such files and display useful information about them.