Arquitetura de Computadores 2019/20

TPC 4

Deadline: 23:59, May 20, 2020

This homework consists of two individual programming exercises. You can discuss general doubts with colleagues, but the solution and the code writing should be strictly individual. All solutions will be automatically compared, and plagiarism cases will be punished in accordance with the regulations.

Your solutions are to be submitted to DI's Mooshak (`http://mooshak.di.fct.unl.pt/~mooshak/`). You are limited to 10 submissions for each exercise (more will be ignored!). The OS is a Linux and your program is compiled with a command similar to the following one (in case of errors or warnings your program will fail!):

```
cc -m32 -Wall -std=c11 -o main main.c
```

Note that your program's output must be exactly the same as in the examples. The points obtained by Mooshak will be used as guide to your grade (100 points = 20 mark).

Problem 1 (50%)

The PNG image file format was proposed by PNG Development Group from W3 Consortium and is now standardized as RFC 2083 and ISO/IEC 15948:2004. This format starts with a header of a fixed sequence of 8 bytes, that allow any program to recognize it as a PNG file. Then is followed by several "chunks" of data, with several information, including the compressed image itself. See an example of the first 23 bytes of one PNG file, each byte represented as hexadecimal number and as char when relevant:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
89	**50**	**4e**	**47**	**0d**	**0a**	**1a**	**0a**	**00**	**00**	**00**	**0d**	**49**	**48**	**44**	**52**	**00**	**00**	**01**	**b4**	**00**	**00**	**00**	**7e**
	P	N	G	\r	\n		\n					I	H	D	R								

The first chunck, starts at byte 8, with 4 bytes of a big-endian unsigned int (32bits) representing chunk's size, then this chunck's name, "IHDR", and the image resolution as two big-endian unsigned ints. For the previous example, the IHDR chunk has 13 bytes (0x0000000d = 13), and the image resolution is 436 width (0x000001b4) and 126 height (0x0000007e).

Implement a C program that, given a filename at the command line, will read that file and detect if it's a PNG file. If it is, your program should print its width and height. As an example, if your program is "`prog`" it will be run from the command line with:

```
prog img.png
```

and for the previous example it should write:

```
PNG file: 436 x 126
```

For any file that is not a PNG it should write:

```
Not a PNG file.
```

Remember that Intel is a little-endian architecture.

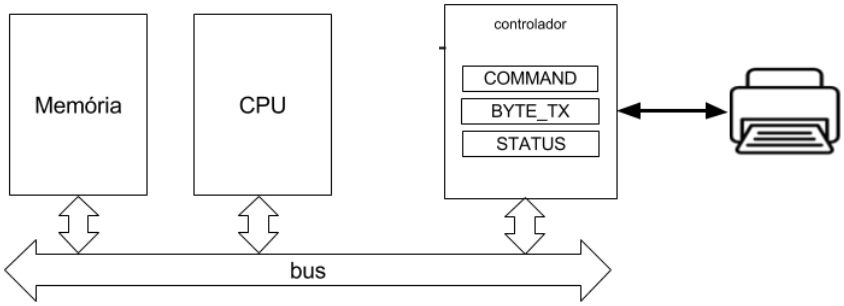
See manual pages for fopen, fread, fgetc, memcmp.

Use the following code to print the messages:

```
printf("PNG file: %d x %d\n", width, height);
printf("Not a PNG file.\n");
```

Problem 2 (50%)

Imagine a computer architecture where a printer is connected using a controller that has a programming interface accessed by IN and OUT instructions (see below). A printer is a very slow device and can process just a byte at a time. A printer must be connected and turned on to print. When the printer is busy or if the printer has no paper, it cannot print new bytes. All this is also reported by the controller.



The controller has registers in the following registers/ports:

- **`BYTE_TX` (0x100)** — can be written with a data byte that may be sent to the printer using a command (see below).
- **`COMMAND` (0x101)** — can be written with commands. Just one command is relevant for this work. When value 1 is written to this port the BYTE_TX value is sent to the printer. Note that if the printer is offline, busy, or out of paper, that data byte will be lost (not printed).
- **`STATUS` (0x102)** – can be read;
  - bit 0 (least significant) – has value 1 if printer is busy;
  - bit 1 – has value 1 if printer is out of paper. In this case the printer is not busy but also cannot print. It will go to 0 when paper is fed.
  - bit 2 – has value 1 if printer is online (connected). If printer is offline, bit 0 and bit 1 can become any value (and must be ignored).

No other programs are using this controller.

Implement in C, `sendByte` from `driver.c` to send a byte to the printer. This function must ensure that the byte is correctly printed and returns as soon as the printer finishes printing that byte. It returns 1 if byte was printed and returns 0 if printer is offline and can't confirm the that the byte was printed.

This work must be implemented using a simulator of this controller, similar to the one from labs #8, provided in `prtsim.zip`. Use the `in` and `out` functions as replacements of the CPU's IN/OUT instructions. The "printer output" will be simulated using the standard output (your terminal). There is a `main` program, that simulates printing a file to this printer, and counts the number of bytes printed. If all goes well, an exact copy of that file will appear at your terminal. Also, this program prints a message when the printer goes offline and exits. For testing, the simulator makes the printer go offline after printing 700 bytes.