Lab 1

Introduction to Systems Programming in Linux

1.1 Introduction

1.1.1 Objectives

This lab is to introduce system programming in a general Linux Development Environment at ECE Department. In this lab, students will:

- apply essential Linux commands to interact with the Linux system through shell
- apply standard Linux C programming tools for system programming, including gcc, make, and ddd
- utilize Linux manual pages
- create a program to interact with Linux file systems by applying the relevant system and library calls; the program will perform file I/O, directory traversal, and read/write operations on selected files
- practice memory allocation, pointer manipulation, and working with buffers

1.1.2 Topics

Concretely, the lab will cover the following topics:

- Basic Linux commands
- C programming toolchain including `gcc`, `make`, and `ddd`
- Linux manual pages

- Linux system calls and file I/O library calls to traverse a directory and perform read/write operations on selected files.

1.2 Starter Files

The starter files are on GitHub at url: https://git.uwaterloo.ca/ece252-s25/ece252_starter/-/tree/main/lab1/starter. It contains the following sub-directories where we have example code and image files to help you get started:

- the cmd_arg demonstrates how to capture command line input arguments;
- the images contains some image files;
- the ls demonstrates how to list all files under a directory and obtain file types;
- the png_util provides a set of utility functions to process a PNG image file;
- the pointer demonstrates how to use pointers to access a C structure; and
- the segfault contains a broken program that has a segmentation fault bug, which you will debug in the last Pre-lab exercise.

Using the code in the starter files is permitted and will not be considered as plagiarism.

1.3 Pre-lab Preparation

Do the pre-lab exercises in Section 1.3.2. Do the pre-lab programming exercise (see 1.3.3).

1.3.1 Working Environment Setup

1. Use the MobaXterm to login onto `eceubuntu1.uwaterloo.ca`, using your UWID as username.

   `ssh username@eceubuntu1.uwaterloo.ca`

   You are now inside the Linux shell and in your home directory. The home directory usually has a path name in the format of `/home/username`, where username normally is your UWID. For example, a user with UWID of `jsmith` has a home directory of `/home/jsmith`.

2. Create a directory as the work space of labs under your home directory. Name the newly created directory as `labs`. Read the man page of the command `mkdir` to see how to do it.

3. Change directory to the newly create directory of `labs`. Read the man page of command `cd` to find out how to change directory.

4. Clone the ece252_starter repository by using the command:

   `git clone https://git.uwaterloo.ca/ece252-s25/ece252_starter`

5. Clone your ECE252 repo:

A new directory named `ece252_starter` will be created. It has starter code of ECE252 labs.

1.3.2 Basic Linux Commands Exercises

These pre-lab exercises are to practice some basic commands on Linux. These are optional if you are already familiar with Linux and programming on Linux.

1. Use the MobaXterm to
   codessh and login onto `eceubuntu.uwaterloo.ca` .

2. Use the `pwd` command to print the full filename of the current working directory. You should see your home directory name printed on the screen. For example: `/home/jsmith`.

3. Use the `echo $HOME` command to print your home directory path name. You will notice that the output matches the `pwd` output of exercise 2.

4. Use the `env` command to list all the environment variables and their values. Note that HOME is one of the many environment variables

5. One important environment variable is `PATH`. It specifies a set of directories the system searches for executible programs. Use `echo $PATH` to see your `PATH` environment variable setting.

6. Execute command `which ls` to locate the directory the ls command is in. You will notice the directory is listed in `PATH` environment variable. When you issue a command and get an error message of "command not found", it means the command cannot be found after searching all the directories listed in `PATH` environment variable. A commonly seen error is that a command in your current working directory gives you `command not found` error. This is normally due to the fact that the current working directory `.` or `./` is not in the PATH. Consequently, you need to add the path to the command name for the system to know where the command is. For example `./a.out` tells the system to run the command `a.out` located in the current working directory.

7. Use the `ls` command to list all files in your current working directory.

8. Read the online manual of the `ls` command by issuing `man ls` command to the shell. Find out from the manual what options `-l`, `-a` and `-la` do. Execute the `ls` command with these three options and see the execution results.

9. Change the directory to the `ece252_starter` .

10. Read the man page of the `find` command by issuing `man find` command to the shell. Read what the `-name` option does. Use `find` with the `-name` option to find all the files with `.png` file extension in the `$HOME/labs/ece252` directory.

11. Change directory to where the `WEEF_1.png` is. Use `file WEEF_1.png` command to obtain the file type and image properties such as dimensions and bit depth.

12. Use `file` command to obtain the file type information of `Disguise.png`. You should see that this is not an image file though the file exension is .png. Use a text editor to open the file and see the contents. This exercise is to show you that the `file` command does not obtain the file type information based on the file extension. It looks into the contents of file to extract the file type information [1].

13. You can use the command `display` to display an image. For example, to view the `WEEF_1.png` file, use the command `display WEEF_1.png`.

14. Execute `cat red-green-16x16.png` command and you will notice the output are gibberish funny characters. This is because cat displays plain text file in a human readable form, not a binary file. The PNG image file is a binary file. Both vi and emacs have hexadecimal mode which displays the bytes in binary files in a hexadecimal format. They are pretty good for small size binary files. There are also few linux commands that perform hex dump of a file. The `xxd` is one of them. Try `xxd red-green-16x16.png` and see the output. Other similar commands include `od` and `hexdump`. Refer to the man pages of these commands for detailed usage instructions.

15. The `pngcheck` command test PNG image files for corruption. Some image viewers are able to display the image even part of the data are corrupted. An image can be opened by an image viewer does not guarantee it is not corrupted. You will notice that the `display` command will not display a corrupted PNG image file on Linux. But the same corrupted image file most likely can be displayed by Windows Paint program. Execute the following two commands and see what pngcheck output tells you.

    - `pngcheck red-green-16x16.png`

[1] A file has a magic number to indicate its type. The magic number is a sequence of bytes usually appearing near the beginning of the file. The file command checks the magic number. The PNG file's magic number is 89 50 4E 47 in hexadecimal, which is .PNG in ASCII.

- `pngcheck red-green-16x16-corrupted.png`

16. Images are binary files. To compare two binary files, we can use the `cmp` command. Read the man page of the `cmp` command and especially pay attention to the `-l` option. Use the `cmp` command with the `-l` option to find out which bytes in `red-green-16x16.png` and `red-green-16x16-corrupted.png` are different. Another tool is `vbindiff`, which displays and compares hexadecimal file(s).

17. Use gdb or ddd to run the program under segfault subdirectory of the `lab1` directory of the starter code. When the code generates a segmentation fault inside the debugger, run the gdb command `where` to see the stack trace and fix the segmentation fault problem of the code.

1.3.3 Pre-lab Programming Exercise

We will write some small pieces of code that can be used in the final lab project assignment code. We will create a command line program named `pnginfo` that prints the dimensions of a valid PNG image file and an error message to the standard output if the input file is not a PNG file or is a corrupted PNG file. The command takes one input argument, which is the path name of a file. Both absolute path name and relative path name are accepted. For example, command `./pnginfo WEEF_1.png` will output the following line:

```
WEEF_1.png: 450 x 229
```

If the input file is not a PNG file. Then output an error message. For example, command `./pnginfo Disguise.png` will output the following line:

```
Disguise.png: Not a PNG file
```

If the input file is a PNG file, but certain chunks has CRC error. That is the CRC value in the chunk does not match the CRC value computed by your program, then the program output something similar as what pngcheck does. For example, command `./pnginfo red-green-16x16-corrupted.png` will output the following line:

```
red-green-16x16-corrupted.png: 16 x 16
IDAT chunk CRC error: computed 34324f1e, expected dc5f7b84
```

You will find starter files under png_util directory are helpful. To make our pre-lab code reusable in the final lab, we will create two functions. One is `is_png()` which takes eight bytes and check whether they match the PNG image file signature. Another function is `get_data_IHDR()` which extracts the image meta information including height and width from a PNG file IHDR chunk. You are free to design

the signatures of these two functions so that they will be re-used in your lab1 final solution and future labs 2 and 3 solutions[2]. However in `lab_png.h` you will find some existing function prototypes that we put there to show one possible function prototype design. Feel free to modify these function prototypes to fit your own design. For computing the CRC of a sequence of bytes, the starter file already provides the crc.c and crc.h and the main.c that demos how to call the crc function to do the computation.

1.4 LAB Project Assignment

1.4.1 Problem statement

You are given a directory under which some files are PNG images and some files are not. The directory may contain nested sub-directories[3]. All valid PNG images under the given directory are horizontal strips of a bigger whole image. They all have the same width. The height of each image might be different. The PNG images have the naming convention of `*_N.png`, where `N` is the image strip sequence number and `N=0, 1, 2, ...`. However a file with .png or .PNG extension may not be a real PNG image file. You need to located all the real PNG image files under the given directory first. Then you will concatenate these horizontal strip images sequentially based on the sequence number in the file name to restore the original whole image. The sequence number indicates the order the image should be concatenated from top to bottom. For example, file `img_1.png` is the first horizontal strip and `img_2.png` is the second horizontal strip. To concatenate these two strips, the pixel data in `img_1.png` should be followed immediately by the pixel data in `img_2.png` file. Figure 1.1 illustrates the concatenation order.

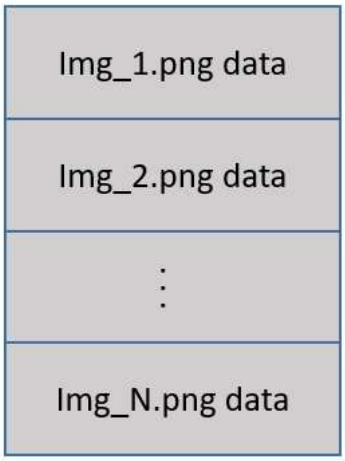


Figure 1.1: Image Concatenation Illustration

To solve the problem, first you will create a tool named `findpng` to search the given directory hierarchy to find all the real PNG files under it. Secondly you will create an image data concatenation tool named `catpng` to concatenate pixel data of a set of PNG files to form a single PNG image file. The `catpng` only processes PNG images with the same width in dimension.

[2] lab2 and lab3 are based on the code of lab1

[3] A nested sub-directory is a sub-directory that may contain many layers of sub-directories.

1.4.2 The findpng command

The expected behaviour of the `findpng` is given in the following manual page of the command.

Man page of findpng

NAME

findpng - search for PNG files in a directory hierarchy

SYNOPSIS

findpng DIRECTORY

DESCRIPTION

Search for PNG files under the directory tree rooted at DIRECTORY and return the search results to the standard output. The command does not follow symbolic links.

OUTPUT FORMAT

The output of search results is a list of PNG file relative path names[4], one file pathname per line. The order of listing the search results is not specified. If the search result is empty, then output "findpng: No PNG file found".

EXAMPLES

findpng .

Find PNG of the current working directory. A non-empty search results might look like the following:

```
lab1/sandbox/new_bak.png
lab1/sandbox/t1.png
png_img/rgba_scanline.png
png_img/v1.png
```

It might also look like the following:

[4]It is relative to the command input directory path name

```
./lab1/sandbox/new_bak.png
./lab1/sandbox/t1.png
./png_img/rgba_scanline.png
./png_img/v1.png
```

An empty search result will look like the following:

```
findpng: No PNG file found
```

Searching PNG files under a given directory

UNIX file system is organized as a tree. A file has a type. Three file types that this assignment will deal with are regular, directory and symbolic link. A PNG file is a regular file. A directory is a directory file. A link created by `ls -n` is a symblic link. Read the section 2 of `stat` family system calls man page for information about other file types. The `ls/ls_ftype.c` in the starter code gives a sample program to determine the file type of a given file. Note that the `struct dirent` returned by the `readdir()` has a field `d_type` that also gives the file type information. However it is not supported by all file system types. We will be using eceubuntu machines to test your submission. If you want to use the `d_type` in your code, make sure you test its behaviour on eceubuntu machines.

To search all the files under a given directory and its subdirectories, one need to traverse the given directory tree to its leaf nodes. The library call of `opendir` returns a directory stream for `readdir` to read each entry in a directory. One need to call `closedir` to close the directory stream once operations on it is completed. The control flow is to go through each entry in a directory and check the file type. If it is a regular file, then further check whether it is a PNG file by comparing the first 8 bytes with the PNG file header bytes (see Section 1.4.3). If it is a directory file, then you need to check files under the sub-directory and repeat what you did in the parent directory. The `ls/ls_fname.c` in the starter code gives a sample program that lists all file entries of a given directory.

Always check the man page of the systems calls and library calls for detailed information.

1.4.3 The catpng command

The expected behaviour of the `catpng` is given in the following manual page of the command.

Man page of the catpng

NAME

catpng - concatenate PNG images vertically to a new PNG named all.png

SYNOPSIS

catpng [PNG_FILE]...

DESCRIPTION

Concatenate PNG_FILE(s) vertically to all.png, a new PNG file.

OUTPUT FORMAT

The concatenated image is output to a new PNG file with the name of all.png.

EXAMPLES

```
catpng ./img1.png ./png/img2.png
```

Concatenate the listed PNG images vertically to all.png.

File I/O

There are two sets of functions for file I/O operations under Linux. At system call level, we have the *unbufferred I/O* functions: `open` , `read` , `write` , `lseek` and `close` . At library call level, we have standard I/O functions: `fopen` , `fread` , `fwrite` , `fseek` and `fclose` . The library is built on top of unbufferred I/O functions. It handles details such as buffer allocation and performing I/O in optimal sized chunks to minimize the number of `read` and `write` usage, hence is recommended to be used for this lab.

The `fopen` returns a FILE pointer given a file name and the mode. A PNG image file is a binary file, hence when you call `fopen` , use mode " `rb` " for reading and " `wb+` " for reading and writing, where the "b" indicates it is a binary file that we are opening. Read the man page of `fopen` for more mode options.

After the file is opened, use `fread` to read the number of bytes from the stream pointed by the FILE pointer returned by `fopen` . Each opened file has an internal state of file position indicator. The file position indicator sets to the beginning of the file when it is just opened. The `fread` operation will advance the file position

indicator by the number of bytes that has been read from the file. The `fseek` sets the file position indicator to the user specified location. The `fwrite` writes user specified number of bytes to the stream pointed by the FILE pointer. The file position indicator also advances by the number of bytes that has been written. It is important to call `fclose` to close the file stream when I/O operations are finished. Failure to do so may result in incomplete files.

The man pages of the standard I/O library is the main reference for details including function prototypes and how to use them.

PNG File Format

In order to finish this assignment, one need to have some understanding of the png file format and how an image is represented in the file. One way to store an image is to use an array of coloured dots referred to as *pixels*. A row of pixels within an image is called a *scanline*. Pixels are ordered from left-to-right within each scanline. Scanlines appear top-to-bottom in the pixel array. In this assignment, each pixel is represented as four 8-bit [5] unsigned integers (ranging from 0 to 255) that specify the red, green, blue and alpha intensity values. This encoding is often referred to as the RGBA encoding. RGB values specify the colour and the alpha value specifies the opacity of the pixel. The size of each pixel is determined by the number of bits per pixel. The dimensions of an image is described in terms of horizontal and vertical pixels.

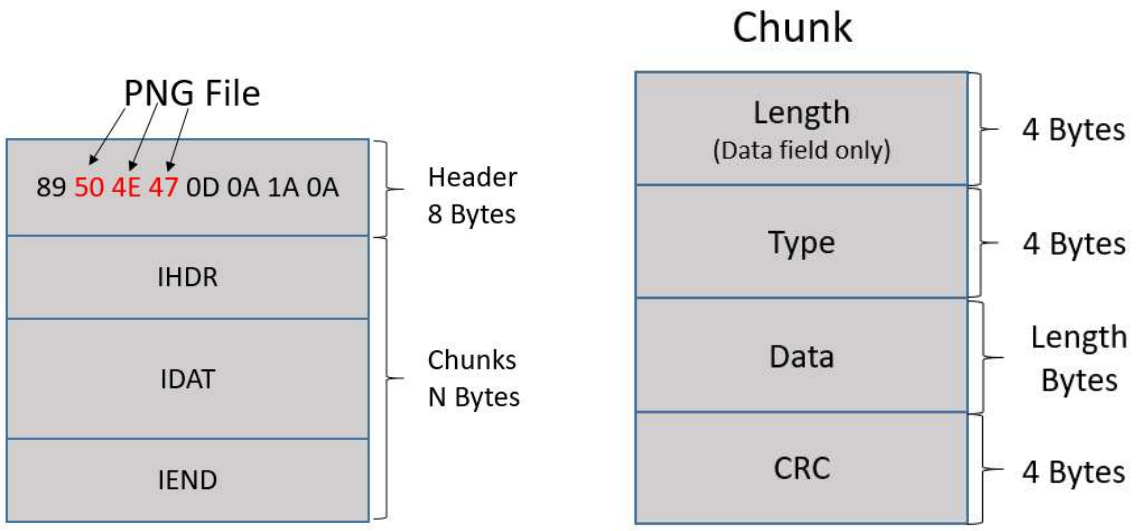
PNG stands for "Portable Network Graphics". It is a computer file format for storing, transmitting and displaying images[1]. A PNG file is a binary file. It starts with an 8-byte header followed by a series of chunks. You will notice the second, third and fourth bytes are the ASCII code of 'P', 'N' and 'G' respectively (see Figure 1.2a).

The first chunk is the IHDR chunk, which contains meta information of the image such as the dimensions of the pixels. The last chunk is always the IEND chunk, which marks the end of the image datastream. In between there is at least one IDAT chunk which contains the compressed filtered pixel array of the image. There are other types chunks that may appear between IHDR chunk and IEND chunk. For all the PNG files we are dealing with in this assignment, we use the format that only has one IHDR chunk, one IDAT chunk and one IEND chunk (see Figure 1.2a).

Each chunk consists of four parts. A four byte length field, a four byte chunk type code field, the chunk data field whose length is specified in the chunk length field, and a four byte CRC (Cyclic Redundancy Check) field (see Figure 1.2b).

The length field stores the length of the data field in bytes. PNG file uses *big endian* byte order, which is the network byte order. When we process any PNG data that is more than one byte such as the length field, we need to convert the network byte order to host order before doing arithmetic. The `ntohl` and `htonl` library

[5] Formally, we say the image has a bit depth of 8 bits per sample.



(a) PNG File Format (b) PNG Chunk format

Figure 1.2: Structure of a PNG file.

calls convert a 32 bit unsigned integer from network order to host order and vice versa respectively.

The chunk type code consists four ASCII character. IHDR, IDAT and IEND are the three chunk type code that this assignment involves with.

The data field contains the data bytes appropriate to the chunk type. This field can be of zero length.

The CRC field calculates on the proceeding bytes in the type and data fields of the chunk. Note that the length field is not included in the CRC calculation. The `crc` function under the `png_util` starter code can be used to calculate the CRC value.

The IHDR chunk data field has a fixed length of 13 bytes and they appear in the order as shown in Table 1.1. Width and height are four-byte unsigned integers giving the image dimensions in pixels. You will need these two values to complete this assignment. Bit depth gives the number of bits per sample. In this assignment, all images have a bit depth of 8. Colour type defines the PNG image type. All png images in this assignment have a colour type of 6, which is truecolor with alpha (i.e. RGBA image). The image pixel array data are filtered to prepare for the next step of compression. The Compression method and Filter method

Name	Length	Value
Width	4 bytes	N/A
Height	4 bytes	N/A
Bit depth	1 byte	8
Colour type	1 byte	6
Compression method	1 byte	0
Filter method	1 byte	0
Interlace method	1 byte	0

Table 1.1: IHDR data field and value

bytes encode the methods used. Both
only have 0 values defined in the current standard. The Interlace method indicates the transmission order of the image data. 0 (no interlace) and 1 (Adam7 interlace) are the only two defined. In this assignment, all PNG images are non-interlaced. Table 1.1 Value column gives the typical IHDR values the PNG images you will be processing.

The IDAT chunk data field contains compressed filtered pixel data. For each scanline, first an extra byte is added at the very beginning of the pixel array to indicate the filter method used. Filtering is for preparing the next step of compression. For example, if the raw pixel scanline is 4 bytes long, then the scanline after applying filter will be 5 bytes long. This added one byte per scanline will help to achieve better compression result. After all scanlines have be filtered, then the data are compressed according to the compression method encoded in IHDR chunk. The compressed data stream conforms to the zlib 1.0 format.

The IEND chunk marks the end of the PNG datastream. It has an empty data field.

Concatenate the pixel data

To concatenate two horizontal image strips, the natural way of thinking is to start with the pixel array of each image and then concatenate the two pixel arrays vertically. Then we apply the filter to each scanline. Lastly we compress the filtered pixel array to fill the data field of the new IDAT chunk of the concatenated image. However a simpler way exists. We can start with the filtered pixel data of each image and then concatenate the two chunks of filtered pixel data arrays vertically, then apply the compression method to generate the data field of the new IDAT chunk.

How do we get filtered pixel data from a PNG IDAT chunk? Recall that the data field in IDAT chunk is compressed data that conforms to zlib format 1.0. We can use zlib functions to uncompress(i.e. inflate) the data. The `mem_inf` in the starter code takes in memory compressed(i.e. deflated) data as input and returns the uncompressed data to a another memory location. For each IDAT chunk you want to concatenate, call this function and stack the returned data in the order you wish and then you have the concatenated filtered pixel array. To create an IDAT chunk, we need to compress the filtered pixel data. The `mem_def` function in the starter code uses the zlib to compress (i.e. deflate) the input in memory data and returns the deflated data. The `png_util` directory in the starter code demos how to use the aforementioned two functions.

To create a new PNG for the concatenated images, IHDR chunk also needs to have the new dimension information of the new PNG file. The rest of the fields of IHDR chunk can be kept the same as one of the PNG files to be concatenated. In this assignment, we assume that `catpng` can only process PNG files whose IHDR chunks only differ in the height field. So the new image will have a different height field, the rest of fields are the same as the input images.

1.5 Deliverables

1.5.1 Pre-lab deliverables

None.

1.5.2 Post-lab Deliverables

Develop findpng and catpng. Submit your lab project 1 on ECE Marmoset Submission and Testing Server.

1.6 Marking Rubric

Points	Sub-points	Description
100		Post-lab
	10	clean project files organization
	5	Makefile correctly builds and cleans
	35	Implementation of `findpng`
	50	Implementation of `catpng`

Table 1.2: Lab1 Marking Rubric

Table 1.2 shows the rubric for marking the lab. Note that if your code generates a segmentation fault, the maximum lab grade you can achieve is 40/100.