INP111 Lab02 Week #3

Date: 2022-09-29

#1 PAKO File Unpacker

The purpose of this lab is to practice binary file reading and handle binary data in data structure.

Structure of PAKO file

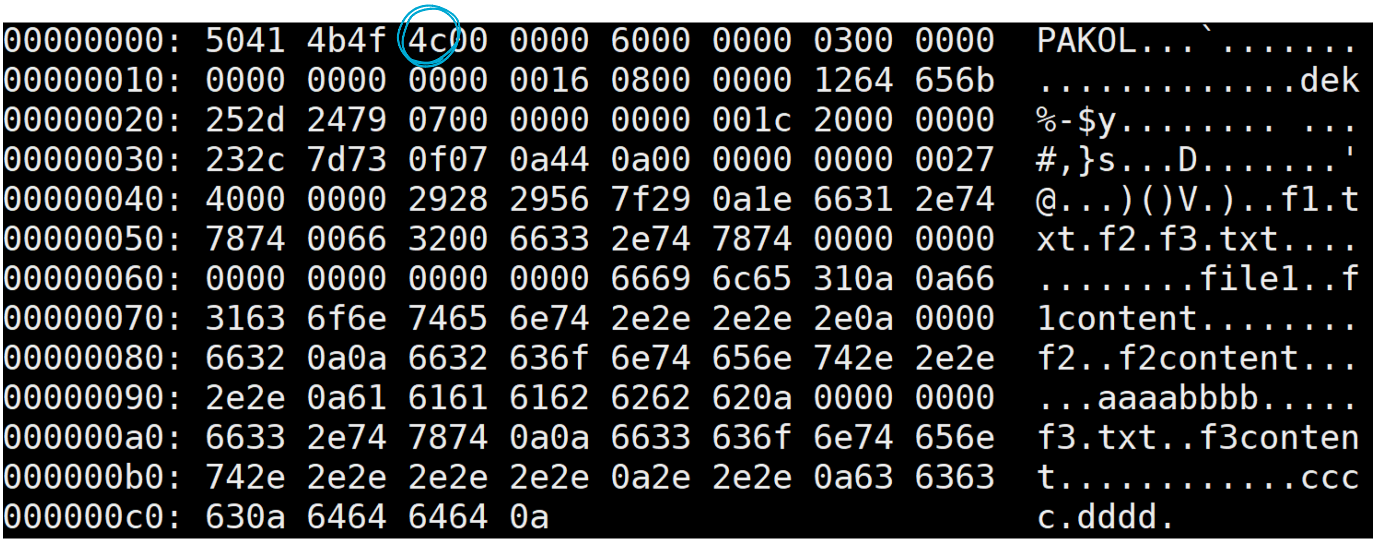
- A PAKO file contains four sections: header section, file entries, string section, and content section.
- header section
  - 4 bytes - magic number: `P`, `A`, `K`, `O`
  - 4 bytes - file offset of the string section
  - 4 bytes - file offset of the content section
  - 4 bytes - number of files packed in this file
- file entries
  - array of `FILE_E` (see the descriptions below) contains the information about each file.
- string section
  - The string section stores strings used in this file, each string is ended with a **null byte**.
- content section
  - The content section stores the content of all packed files.

`FILE_E` data structure

- 4 bytes - offset of the filename string in the string section
- 4 bytes - file size (**big-endian**)
- 4 bytes - offset of the file content in the content section
- 8 bytes - checksum (**big endian**)
  - Split the content of a file into segments of 8 bytes. The total number of segments should be $\lceil \text{(size-of-PAKO-file)}/8 \rceil$. 上高斯
  - Consider each segment as an `uint64_t` . The checksum is the **XOR** value of all segments.

Sample PAKO File

- There are three files packed in the sample PAKO file. The filenames are:
  - `f1.txt`
  - `f2`
  - `f3.txt`
- The sample PAKO file's hexdump is shown below.



```
00000000: 5041 4b4f 4c00 0000 6000 0000 0300 0000  PAKOL...`.......
00000010: 0000 0000 0000 0016 0800 0000 1264 656b  .............dek
00000020: 252d 2479 0700 0000 0000 001c 2000 0000  %-$y........ ...
00000030: 232c 7d73 0f07 0a44 0a00 0000 0000 0027  #,}s...D.......'
00000040: 4000 0000 2928 2956 7f29 0a1e 6631 2e74  @...)()V.)..f1.t
00000050: 7874 0066 3200 6633 2e74 7874 0000 0000  xt.f2.f3.txt....
00000060: 0000 0000 0000 0000 6669 6c65 310a 0a66  ........file1..f
00000070: 3163 6f6e 7465 6e74 2e2e 2e2e 2e0a 0000  1content........
00000080: 6632 0a0a 6632 636f 6e74 656e 742e 2e2e  f2..f2content...
00000090: 2e2e 0a61 6161 6162 6262 620a 0000 0000  ...aaaabbbb.....
000000a0: 6633 2e74 7874 0a0a 6633 636f 6e74 656e  f3.txt..f3conten
000000b0: 742e 2e2e 2e2e 2e2e 0a2e 2e2e 0a63 6363  t............ccc
000000c0: 630a 6464 6464 0a                        c.dddd.
```

- `[0x0, 0x4)` : `50414b4f` is the magic number
- `[0x4, 0x8)` : `0x0000004c` is offset of the string section in this file. (The section starts with `66312e74` ... in this example.)
- `[0x8, 0xc)` : `0x00000060` is offset of the content section in this file. (The section starts with `00000000` ... in this example.)
- `[0xc, 0x10)` : `0x00000003` is the number of files packed in the sample PAKO file.
- Offsets `[0x10, 0x24)` , `[0x24, 0x38)` , `[0x38, 0x4c)` : The three `FILE_E` structures for the three files packed in this PAKO file, respectively.
- Take the second file as an example:
  - `[0x24, 0x38)` : the `FILE_E` structure of the second file (named `f2`).

if i%8=7

- `[0x24, 0x28)` : `0x00000007` indicates the filename string's offset in the string section. As a result, the filename string is located at offset `0x4c+0x7=0x53` . The string at offset `0x53` is `f2` (ended with a null byte). val=0 用0隔開
- `[0x28, 0x2c)` : `0x0000001c` (big endian) is the size of the file.
  16+12=28
- `[0x2c, 0x30)` : `0x00000020` is the offset of `f2` 's content in the content section. The content of the file is located at offset `0x60+0x20=0x80` . (start with `66320a0a6632636f` ...)
- `[0x30, 0x38)` : `0x232c7d730f070a44` is the checksum of the file. `checksum = 0x6f6332660a0a3266 ^ 0x2e2e2e746e65746e ^ 0x62616161610a2e2e ^ 0x0a626262` . If the length of the last segment is less than 8 bytes, pad zeros at the end of the segment to ensure its length is correct.

- `[0x53, 0x55]` : filename string of the second file.
- `[0x80, 0x9c]` : file content of the second file.

Steps

1. Implement an unpacker program in C/C++ to unpack the PAKO file. You should invoke your program using the following command: `./unpacker <src.pak> <dst>` , where `src.pak` is the input PAKO file and `dst` is the destination directory. You should unpack the files packed in the PAKO file to `dst` directory. When your program unpacks the files, it must calculate the checksum of each file. **Do not unpack the files having incorrect checksums.**
2. You can test your program with example.pak (http://inp111.zoolab.org/lab02.1/example.pak)
   - Note that the checksum of `f3.txt` in this example is incorrect. Therefore, your program should not unpack `f3.txt` .

int end = lseek(fd,0, seek_end);

You may need the following functions/headers to complete this lab.

- functions
  - open(2) (https://man7.org/linux/man-pages/man2/open.2.html)
  - read(2) (https://man7.org/linux/man-pages/man2/read.2.html)
  - write(2) (https://man7.org/linux/man-pages/man2/write.2.html)
  - lseek(2) (https://man7.org/linux/man-pages/man2/lseek.2.html)
- headers
  - <stdint.h>

Reference: https://man7.org/linux/man-pages/ (https://man7.org/linux/man-pages/)

Demo

1. Use the following command to test your program.

   ```
   wget http://inp111.zoolab.org/lab02.1/testcase.pak
   mkdir /tmp/inplab2test
   ./unpacker testcase.pak /tmp/inplab2test
   cd /tmp/inplab2test
   chmod +x checker
   ./checker
   ```

Checkpoints

1. [20%] Your program shows the number of files packed in `testcase.pak` .
2. [30%] Your program shows the information (filename, file length in bytes) of each file packed in `testcase0.pak` (http://inp111.zoolab.org/lab02.1/testcase.pak).
3. [20%] The extracted `checker` script can run without a crash.
4. [30%] `checker` runs without a crash and shows `Bingo` message successfully.

0~4 : PAKO.

4~8 : string section offset.

8~12 : content section offset.

12~16 : # files packed.

16~36 . 36~56 . 56~76 : 3個 FILE_E 的值.

FILE

4	offset of file name string.
4	file size
4	file content offset
8	checksum.

- scp -P zzzzz test.cpp Alison@localhost:/tmp