LZO stream format as understood by Linux's LZO decompressor

Introduction

This is not a specification. No specification seems to be publicly available for the LZO stream format. This document describes what input format the LZO decompressor as implemented in the Linux kernel understands. The file subject of this analysis is lib/lzo/lzo1x_decompress_safe.c. No analysis was made on the compressor nor on any other implementations though it seems likely that the format matches the standard one. The purpose of this document is to better understand what the code does in order to propose more efficient fixes for future bug reports.

Description

The stream is composed of a series of instructions, operands, and data. The instructions consist in a few bits representing an opcode, and bits forming the operands for the instruction, whose size and position depend on the opcode and on the number of literals copied by previous instruction. The operands are used to indicate:

- a distance when copying data from the dictionary (past output buffer)
- a length (number of bytes to copy from dictionary)
- the number of literals to copy, which is retained in variable "state" as a piece of information for next instructions.

Optionally depending on the opcode and operands, extra data may follow. These extra data can be a complement for the operand (eg: a length or a distance encoded on larger values), or a literal to be copied to the output buffer.

The first byte of the block follows a different encoding from other bytes, it seems to be optimized for literal use only, since there is no dictionary yet prior to that byte.

Lengths are always encoded on a variable size starting with a small number of bits in the operand. If the number of bits isn't enough to represent the length, up to 255 may be added in increments by consuming more bytes with a rate of at most 255 per extra byte (thus the compression ratio cannot exceed around 255:1). The variable length encoding using #bits is always the same:

```
length = byte & ((1 << #bits) - 1)
if (!length) {
        length = ((1 << #bits) - 1)
        length += 255*(number of zero bytes)
        length += first-non-zero-byte
}
length += constant (generally 2 or 3)
```

For references to the dictionary, distances are relative to the output pointer. Distances are encoded using very few bits belonging to certain ranges, resulting in multiple copy instructions using different encodings. Certain encodings involve one extra byte, others involve two extra bytes forming a little-endian 16-bit quantity (marked LE16 below).

After any instruction except the large literal copy, 0, 1, 2 or 3 literals are copied before starting the next instruction. The number of literals that were copied may change the meaning and behaviour of the next instruction. In practice, only one instruction needs to know whether 0, less than 4, or more literals were copied. This is the information stored in the <state> variable in this implementation. This number of immediate literals to be copied is generally encoded in the last two bits of the instruction but may also be taken from the last two bits of an extra operand (eg: distance).

End of stream is declared when a block copy of distance 0 is seen. Only one instruction may encode this distance (0001HLLL), it takes one LE16 operand for the distance, thus requiring 3 bytes.

> **Important**
>
> In the code some length checks are missing because certain instructions are called under the assumption that a certain number of bytes follow because it has already been guaranteed before parsing the instructions. They just have to "refill" this credit if they consume extra bytes. This is an implementation design choice independent on the algorithm or encoding.

Versions

0: Original version 1: LZO-RLE

Version 1 of LZO implements an extension to encode runs of zeros using run length encoding. This improves speed for data with many zeros, which is a common case for zram. This modifies the bitstream in a backwards compatible way (v1 can correctly

decompress v0 compressed data, but v0 cannot read v1 data).

For maximum compatibility, both versions are available under different names (lzo and lzo-rle). Differences in the encoding are noted in this document with e.g.: version 1 only.

Byte sequences

First byte encoding:

```
0..16   : follow regular instruction encoding, see below. It is worth
          noting that code 16 will represent a block copy from the
          dictionary which is empty, and that it will always be
          invalid at this place.

17      : bitstream version. If the first byte is 17, and compressed
          stream length is at least 5 bytes (length of shortest possible
          versioned bitstream), the next byte gives the bitstream version
          (version 1 only).
          Otherwise, the bitstream version is 0.

18..21  : copy 0..3 literals
          state = (byte - 17) = 0..3  [ copy <state> literals ]
          skip byte

22..255 : copy literal string
          length = (byte - 17) = 4..238
          state = 4 [ don't copy extra literals ]
          skip byte
```

Instruction encoding:

```
0 0 0 0 X X X X  (0..15)
  Depends on the number of literals copied by the last instruction.
  If last instruction did not copy any literal (state == 0), this
  encoding will be a copy of 4 or more literal, and must be interpreted
  like this :

     0 0 0 0 L L L L  (0..15)  : copy long literal string
     length = 3 + (L ?: 15 + (zero_bytes * 255) + non_zero_byte)
     state = 4  (no extra literals are copied)

  If last instruction used to copy between 1 to 3 literals (encoded in
  the instruction's opcode or distance), the instruction is a copy of a
  2-byte block from the dictionary within a 1kB distance. It is worth
  noting that this instruction provides little savings since it uses 2
  bytes to encode a copy of 2 other bytes but it encodes the number of
  following literals for free. It must be interpreted like this :

     0 0 0 0 D D S S  (0..15)  : copy 2 bytes from <= 1kB distance
     length = 2
     state = S (copy S literals after this block)
   Always followed by exactly one byte : H H H H H H H H
     distance = (H << 2) + D + 1

  If last instruction used to copy 4 or more literals (as detected by
  state == 4), the instruction becomes a copy of a 3-byte block from the
  dictionary from a 2..3kB distance, and must be interpreted like this :

     0 0 0 0 D D S S  (0..15)  : copy 3 bytes from 2..3 kB distance
     length = 3
     state = S (copy S literals after this block)
   Always followed by exactly one byte : H H H H H H H H
     distance = (H << 2) + D + 2049

0 0 0 1 H L L L  (16..31)
     Copy of a block within 16..48kB distance (preferably less than 10B)
     length = 2 + (L ?: 7 + (zero_bytes * 255) + non_zero_byte)
  Always followed by exactly one LE16 :  D D D D D D D D : D D D D D D S S
     distance = 16384 + (H << 14) + D
     state = S (copy S literals after this block)
     End of stream is reached if distance == 16384
     In version 1 only, to prevent ambiguity with the RLE case when
     ((distance & 0x803f) == 0x803f) && (261 <= length <= 264), the
     compressor must not emit block copies where distance and length
     meet these conditions.

  In version 1 only, this instruction is also used to encode a run of
     zeros if distance = 0xbfff, i.e. H = 1 and the D bits are all 1.
     In this case, it is followed by a fourth byte, X.
     run length = ((X << 3) | (0 0 0 0 0 L L L)) + 4
```

```
0 0 1 L L L L L  (32..63)
     Copy of small block within 16kB distance (preferably less than 34B)
     length = 2 + (L ?: 31 + (zero_bytes * 255) + non_zero_byte)
  Always followed by exactly one LE16 :  D D D D D D D D : D D D D D D S S
     distance = D + 1
     state = S (copy S literals after this block)

0 1 L D D D S S  (64..127)
     Copy 3-4 bytes from block within 2kB distance
     state = S (copy S literals after this block)
     length = 3 + L
   Always followed by exactly one byte : H H H H H H H H
     distance = (H << 3) + D + 1

1 L L D D D S S  (128..255)
     Copy 5-8 bytes from block within 2kB distance
     state = S (copy S literals after this block)
     length = 5 + L
   Always followed by exactly one byte : H H H H H H H H
     distance = (H << 3) + D + 1
```

Authors

This document was written by Willy Tarreau <w@1wt.eu> on 2014/07/19 during an analysis of the decompression code available in Linux 3.16-rc5, and updated by Dave Rodgman <dave.rodgman@arm.com> on 2018/10/30 to introduce run-length encoding. The code is tricky, it is possible that this document contains mistakes or that a few corner cases were overlooked. In any case, please report any doubt, fix, or proposed updates to the author(s) so that the document can be updated.