

OpenSolaris

Discussions Communities Projects Download Source Browser

clofi.general 1.2

07/09/06 SMI

Design Specification for adding compression support to lofi(7d)

1. Requirements

This project aims to extend lofi(7D) to support reading from a compressed file via on-the-fly decompression.

This project will also add extensions to the lofiadm command to optionally compress files at the time of mapping them to a block device as well decompressing the compressed files.

So, the requirements for adding this support to lofi are -

A user should be able to map a compressed file. Reads from the mapped file should decompress the data on-the-fly and return uncompressed data.

A user must be able to compress a given file with a supported compression algorithm.

A user must be able to decompress a compressed file.

A user must be able to ascertain from the lofiadm output whether a file is compressed or not and if it is, which compression algorithm was used to compress it.

2. Constraints

The design requires the availability of compression algorithms in the Solaris kernel. Currently lzjb and gzip are the two algorithms available in the kernel. However, gzip will be the only algorithm supported at this time (See the section on Compression Algorithm Choices).

Any changes to the public gzip interfaces would need to be propagated to lofi(7d) as well.

3. Basic Design

3.1 Compressed File Format

The basic design is to split a given file into fixed size segments (typically 128K but configurable) and compress each segment individually. The compressed segments are then stored sequentially. The segment size is constrained to be a power of 2 for ease of calculation.

A header that contains the following information is then prepended to the file containing the compressed segments -

Signature - A string containing the compression algorithm used

Segment Size - The size of each uncompressed segment

Number of Segments - The number of compressed segments

Size of the Last Segment - The uncompressed size of the last segment. This is needed since the file size may not be a multiple of the segment size.

Array of Index Entries - Each of the array entries contains an offset of the compressed segments.

It must be noted that the header information is written to the file in network byte order.

A graphical representation of the compressed file format can be found here -

http://www.genunix.org/distributions/belenix_site/?q=compression

3.2 Changes to map a compressed file to a block device

When lofiadm(1M) is used to map a file to a block device, it ultimately results in a call to lofi_map_file(). This function will be changed to additionally check for the existence of a signature.

If a valid signature is found, it will call a new function lofi_map_compressed_file(). This function will read in the compression header information into memory and update the appropriate fields in the lofi state structure. Since the header information is always in the network byte order, lofi_map_compressed_file will byte swap at the time of reading this information in.

3.3 Changes to the lofi state structure

The lofi state structure will be expanded to include the following fields -

```
/*
 * index into lofi_decompress_table for the compression algorithm used
 */
uint32_t      ls_comp_algorithm_index;
uint32_t      ls_comp_algorithm_len;
```

The lofi_decompress_table contains information about the compression algorithms supported by lofi(7D) (See below for more information). Just like the comment points out, ls_comp_algorithm_index contains an index into the lofi_decompress_table for the compression algorithm in use for the mapped file.

```
/* size of an uncompressed segment */
uint32_t      ls_uncomp_seg_sz;
/*
 * number of bytes to allocate for an uncompressed segment
 * (dependant upon the type of compression algorithm being used)
 */
uint32_t      ls_uncomp_seg_alloc;
/* number of index entries */
uint32_t      ls_comp_index_sz;
/* exponent for byte shift, power of 2 */
uint32_t      ls_comp_seg_shift;
```

ls_comp_seg_shift is the exponent of 2 that indicates the size of an uncompressed segment. It is used to compute the correct offsets at the time of doing reads from the file.

```
/*
 * size of the last uncompressed segment (the file may
 * not be an exact multiple of the segment size ls_comp_seg_sz)
 */
uint32_t      ls_uncomp_last_seg_sz;
/*
 * the offset in the file where the header ends and the
 * actual compressed data begins. The segment offsets in
 * the index do not account for the header so this value must
 * be added
 */
uint64_t      ls_comp_offbase;
/*
 * the array holding the segment index. This is a pointer
 * into ls_comp_index_data
 */
uint64_t      *ls_comp_seg_index;
/*
 * holds the index pages loaded from the file. This is aligned
 * on a disk block boundary so comp_seg_index above is used to point
 * to the actual array
 */
caddr_t      ls_comp_index_data;
/* size of ls_comp_index_data */
uint32_t      ls_comp_index_data_sz;
```

```

/*
 * incase of compressed files, ls_vp_size above is
 * tweaked to represent the actual uncompressed file
 * size so that fake_disk_geometry gives the correct
 * values. However we need the actual compressed file
 * size while faulting in pages from the compressed
 * file in lofi_mapped_rdwr
 */
u_offset_t      ls_vp_comp_size;

```

3.4 Changes to do on the fly decompression

A read request to lofi results in a call to lofi_strategy_task(). lofi_strategy_task will be changed to first compute the starting and ending compressed segment numbers that will contain the requested data.

The actual file offsets and ranges of the compressed segments will be gotten from the segment index array and the start offset will be aligned to a disk block boundary. The data will then be read into memory via lofi_mapped_rdwr().

The data thus read contains all the compressed segments required. Each of the compressed segments will be subsequently uncompressed on the fly and the uncompressed data will be returned to the caller. If an error is encountered while uncompressing the data an EIO will be returned to the caller.

3.5 Compression Algorithm Choices

Currently, lzjb and gzip are the two compression algorithms available in the Solaris kernel.

The lzjb algorithm provides very fast compression as compared to gzip. gzip, even though it is slower than lzjb, is fast enough for the typical use case of a LiveCD. Additionally, gzip provides a much better compression ratio as compared to lzjb (on the order of ~25% in some cases) and it allows for more data to be packed on to the media. These characteristics make gzip well suited for a LiveCD. Thus, gzip will be the only supported algorithm at this time.

3.6 Changes to allow for pluggable compression algorithms

As more compression algorithms become available in the Solaris kernel, it will be relatively easy to plug them into lofi. Here's how it will be accomplished -

Each compression algorithm will be described by the lofi_decompress_info structure -

```

typedef struct lofi_decompress_info {
    lofi_decompress_func_t  *l_decompress;
    int                      l_level;
    char                     *l_name;      /* algorithm name */
} lofi_decompress_info_t;

```

where lofi_decompress_func_t is used to convey a common signature for all lofi decompress functions -

```

typedef size_t lofi_decompress_func_t(void *src, size_t srclen,
    void *dst, size_t destlen, int level);

```

The list of compression algorithms will be stored in lofi_decompress -

```

enum lofi_decompress {
    LOFI_DECOMPRESS_GZIP = 0,
    LOFI_DECOMPRESS_FUNCTIONS
};

```

and a global table of the supported algorithms in the form of

```

lofi_decompress_func_t's will be stored in lofi_decompress_table -
lofi_decompress_info_t lofi_decompress_table[LOFI_DECOMPRESS_FUNCTIONS] = {
    {gzip_decompress,          9,      "gzip"}
};

```

To add a new compression algorithm would mean adding the appropriate entries to lofi_decompress, lofi_decompress_table and providing a lofi_decompress_func_t wrapper function that wraps a call to

the actual decompression routine.

4.0 Command Changes

4.1 Changes to lofiadm(1M)

lofiadm(1M) will take the following additional arguments -

- c Compress the file with a specified compression algorithm or with the default compression algorithm if one isn't specified. The only supported compression algorithm at this time is gzip which is also the default.
- s The segment size to use to divide the file being compressed. If no segment size is specified, a default segment size of 128k is used.
- u Uncompress a compressed file

Examples:

1. Compress a file with gzip

```
# lofiadm -c gzip /export/home/solaris.iso
```

2. Compress a file with gzip and compress it in 64k chunks

```
# lofiadm -c gzip -s 64k /export/home/solaris.iso
```

3. Check if there are any compressed files that are mapped

```
# lofiadm
```

Block Device	File	Type
/dev/lofi/1	/export/home/solaris.iso	Compressed (gzip)
/dev/lofi/2	/export/home/regular.iso	Regular

4. Uncompress a compressed file

```
# lofiadm -u /export/home/solaris.iso
```

4.1 New ioctl to support extended lofiadm(1M) output

As seen in the examples above, the output of lofiadm will change to add another field to reflect if the mapped file(s) is a compressed file or a regular file.

The lofi_ioctl structure will have an additional member -
char li_compress_info[MAXPATHLEN 1];

This field will be populated with the algorithm used to compress the file (or set to NULL if it isn't compressed) in response to the LOFI_GET_FILENAME ioctl.

5. References

On the Fly Decompression in BeleniX

http://www.genunix.org/distributions/belenix_site/?q=compression
gzip for ZFS update

http://blogs.sun.com/ahl/entry/gzip_for_zfs_update