bzip2 and libbzip2, version 1.0.6

A program and library for data compression

bzip2 and libbzip2, version 1.0.6: A program and library for data compression

by Julian Seward

Version 1.0.6 of 6 September 2010

Table of Contents

2. How to use bzip2

Table of Contents

--verbouse

than a block. For example, compressing a file 20,000 bytes long with the flag `-9` will cause the compressor to

How to use bzip2

3. Programming with

Yoshioka also contributed modifications to allow the library to be built as a Windows DLL.

3.2. Error handling

The library is designed to recover cleanly in all situations, including the worst-case situation of decompressing random data. I'm not 100% sure that it can always do this, so you might want to add a signal handler to catch segmentation violations during decompression if you are feeling especially paranoid. I would be interested in hearing more about the robustness of the library to corrupted compressed data.

Version 1.0.3 more robust in this respect than any previous version. Investigations with Valgrind (a tool for detecting problems with m 0 0 rfs.6egmen(h)-421rini(cae rf)-2(thas,)4764hat forte rfworhte

the standard sorting algorithm to a fallback algorithm. The fallback is slower than the standard algorithm by perhaps a factor of three, but always behaves reasonably, no matter how bad the input.

Lower values of `workFactor` reduce the amount of effort theworkFactor

A second purpose of BZ2_bzCompress is to request a change of mode of the compressed stream.

Conceptually, a compressed stream can be in one of four states: IDLE, RUNNING, FLUSHING and FINISHING. Before initialisation (BZ2_bzCompressInit) and after termination (BZ2_bzCompressEnd), a stream is regarded as IDLE.

Upon initialisation (BZ2_bzCompressInit), the stream is placed in the RUNNING state. Subsequent calls to

2. Shovel data in and shlurp out its compressed form using zero or more calls of `BZ2_bzCompress` with action = `BZ_RUN`.

3. Finish [illegible]

```
BZ_CONFIG_ERROR
  if the library has been mis-compiled
BZ_PARAM_ERROR
  if ( small != 0 && small != 1 )
  or (verbosity <; 0 || verbosity > 4)
BZ_MEM_ERROR
  if insufficient memory is available
```

Allowable next actions:

```
BZ2_bzDecompress
  if BZ_OK was returned
  no specific action required in case of error
```

3.3.5. BZ2_bzDecompress

```
int BZ2_bzDecompress ( bz_stream *strm );
```

Provides more input and/out output buffer space for the library. The caller maintains input and output buffers, and uses `BZ2_bzDecompress` to transfer data between them.

Before each call to `BZ2_bzDecompress`, `next_in` should point at the compressed data, and `avail_in`

```
BZ_PARAM_ERROR
  if strm is NULL or strm->s is NULL
  or strm->avail_out < 1
BZ_DATA_ERROR
  if a data integrity error is detected in the compressed stream
BZ_DATA_ERROR_NvICR
  if the compressed stream oesn’utin with the
BZMEAM_ERROR
  if thrhe
BZSTRERAM_NDR
  if thf thf data stream detected
   in cnsumed,t -m->avail_out
BZOKR
  thrwise2
```

velmintrface2

- If

```
Pointer to an abstract BZFILE
  if bzerror is BZ_OK
NULL
  otherwise
```

Allowable next actions:

```
BZ2_bzRead
  if bzerror is BZ_OK
BZ2_bzClose
  otherwise
```

3.4.2. BZ2_bzRead

```
int BZ2_bzRead ( int *bzerror, BZFILE *b, void *buf, int len );
```

```
BZ_PARAM_ERROR
  if b is NULL
  or unused is NULL or nUnused is NULL
BZ_SEQUENCE_ERROR
  if BZ_STREAM_END has not been signalled
  or if b was opened with BZ2_bzWriteOpen
BZ_OK
  otherwise
```

Allowable next actions:

```
BZ2_bzReadClose
```

3.4.4BZ2_bzReadClose

```
void BZ2_bzWriteClose( int *bzerror, BZFILE* f,
                       int abandon,
                       unsigned int* nbytes_in,
                       unsigned int* nbytes_out );

void BZ2_bzWriteClose64( int *bzerror, BZFILE* f,
                         int abandon,
                         unsigned int* nbytes_in_lo32,
                         unsigned int* nbytes_in_hi32,
                         unsigned int* nbytes_out_lo32,
                         unsigned int* nbytes_out_hi32 );
```

```
FILE*   f;
BZFILE* b;
int     nBuf;
char    buf[ /* whatever size you like */ ];
int     bzerror;
int     nWritten;

f = fopen ( "myfile.bz2", "r" );
if ( !f ) {
  /* handle error */
}
b = BZ2_bzReadOpen ( &bzerror, f, 0, NULL, 0 );
if ( bzerror != BZ_OK ) {
  BZ2_bzReadClose ( &bzerror, b );
  /* handle error */
}

bzerror = BZ_OK;
while ( bzerror == BZ_OK && /* arbitrary other conditions */) {
  nBuf = BZ2_bzRead ( &bzerror, b, buf, /* size of buf */ );
  if ( bzerror == BZ_OK ) {
    /* do something with buf[0 .. nBuf-1] */
  }
}
if ( bzerror != BZ_STREAM_END ) {
   BZ2_bzReadClose ( &bzerror, b );
   /* handle error */
} {
   BZ2_bzReadClose ( &bzerror, b );-
```

For the meaning of parameters `blockSize100k`, `verbosity` and `workFactor`, see `BZ2_bzCompressInit`.

To guarantee that the compressed data will fit in its buffer, allocate an output buffer of size 1% larger than the uncompressed data, plus six hundred extra bytes.

`BZ2_bzBuffToBuffDecompress` will not write data at or beyond `dest[*destLen]`, even in case of buffer overflow.

Possible return values:

```
BZ_CONFIG_ERROR
  if the library has been mis-compiled
BZ_PARAM_ERROR
  if dest is NULL or destLen is NULL
  or blockSize100k < 1 or blockSize100k > 9
  or verbosity < 0 or verbosity > 4
  or workFactor < 0 or workFactor > 250
BZ_MEM_ERROR
  if insufficient memory is available
BZ_OUTBUFF_FULL
  if the size of the compressed data exceeds *destLen
BZ_OK
  otherwise
```

3.5.2. BZ2_bzBuffToBuffDecompress

```
desG_F25(v]TJ000rg000.5Td[8 F210ise)]TJ000output>
int BZ2_bzBuffToBuffDecompress(mis-cdr)-315(be)ty
```

```
BZ_CONFIG_ERROR
  if the library has been mis-compiled
BZ_PARAM_ERROR
  if dest is NULL or destLen is NULL
  or small != 0 && small != 1
  or verbosity < 0 or verbosity > 4
BZ_MEM_ERROR
  if insufficient memory is available
BZ_OUTBUFF_FULL
  if the size of the compressed data exceeds *destLen
BZ_DATA_ERROR
  if a data integrity error was detected in the compressed data
BZ_DATA_ERROR_MAGIC
  if the compressed data doesn’t begin with the right magic bytes
BZ_UNEXPECTED_EOF
  if the compressed data ends unexpectedly
BZ_OK
  otherwise
```

3.6. zlib compatibility functions

My vague understanding of what to do is: using Visual C++ 5.0, open the project file `libbz2.dsp`, and build. That's all.

If you can't open the project file for some reason, make a new one, naming these files: `blocksort.c`, `bzlib.c`, `compress.c`, `crctable.c`, `decompress.c`

•

•

The Manber-Myers suffix array construction algorithm is described in a paper available from:

```
   http://www.cs.arizona.edu/people/gene/PAPERS/suffix.ps
```