GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	storage/filesystem/littlefs/source/LittleFileSystem.cpp	Lines:	200	323	61.9 %
Date:	2021-05-06 12:39:05	Branches:	52	111	46.8 %

Line	Branch	Exec	Source
1			/* mbed Microcontroller Library
2			* Copyright (c) 2017 ARM Limited
3			* SPDX-License-Identifier: Apache-2.0
4			*
5			* Licensed under the Apache License, Version 2.0 (the "License");
6			* you may not use this file except in compliance with the License.
7			* You may obtain a copy of the License at
8			*
9			* http://www.apache.org/licenses/LICENSE-2.0
10			*
11			* Unless required by applicable law or agreed to in writing, software
12			* distributed under the License is distributed on an "AS IS" BASIS,
13			* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14			* See the License for the specific language governing permissions and
15			* limitations under the License.
16			*/
17			#include "filesystem/mbed_filesystem.h"
18			#include "littlefs/LittleFileSystem.h"
19			#include "errno.h"
20			#include "littlefs/lfs.h"
21			#include "littlefs/lfs_util.h"
22			#include "MbedCRC.h"
23			
24			namespace mbed {
25			
26		56598	extern "C" void lfs_crc(uint32_t *crc, const void *buffer, size_t size)
27			{
28		56598	uint32_t initial_xor = lfs_rbit(*crc);
29			// lfs_cache_crc calls lfs_crc for every byte individually, so can't afford
30			// start-up overhead for hardware acceleration. Limit to table-based.

```
31  ✓✗  56598      MbedCRC<POLY_32BIT_ANSI, 32, CrcMode::TABLE> ct(initial_xor, 0x0, true, true);
32  ✓✗  56598      ct.compute(buffer, size, crc);
33      56598  }
34
35             ////// Conversion functions //////
36      1748   static int lfs_toerror(int err)
37             {
    ✓✗✓✗
38  ✗✗✗✗  1748     switch (err) {
      ✗✓✓
39       919          case LFS_ERR_OK:
40       919              return 0;
41                    case LFS_ERR_IO:
42                        return -EIO;
43       107          case LFS_ERR_NOENT:
44       107              return -ENOENT;
45                    case LFS_ERR_EXIST:
46                        return -EEXIST;
47                    case LFS_ERR_NOTDIR:
48                        return -ENOTDIR;
49                    case LFS_ERR_ISDIR:
50                        return -EISDIR;
51                    case LFS_ERR_INVAL:
52                        return -EINVAL;
53                    case LFS_ERR_NOSPC:
54                        return -ENOSPC;
55                    case LFS_ERR_NOMEM:
56                        return -ENOMEM;
57         8          case LFS_ERR_CORRUPT:
58         8              return -EILSEQ;
59       714          default:
60       714              return err;
61             }
62             }
63
64       407   static int lfs_fromflags(int flags)
65             {
66             return (
67  ✓✓   814                (((flags & 3) == O_RDONLY) ? LFS_O_RDONLY : 0) |
68  ✓✓   814                (((flags & 3) == O_WRONLY) ? LFS_O_WRONLY : 0) |
```

```
69  ✗✓  814                (((flags & 3) == O_RDWR)   ? LFS_O_RDWR   : 0) |
70      814                ((flags & O_CREAT)  ? LFS_O_CREAT  : 0) |
71      814                ((flags & O_EXCL)   ? LFS_O_EXCL   : 0) |
72      407                ((flags & O_TRUNC)  ? LFS_O_TRUNC  : 0) |
73      407                ((flags & O_APPEND) ? LFS_O_APPEND : 0));
74                         }
75
76      101  static int lfs_fromwhence(int whence)
77                         {
78  ✓✗✗✗  101      switch (whence) {
79      101          case SEEK_SET:
80      101              return LFS_SEEK_SET;
81                       case SEEK_CUR:
82                           return LFS_SEEK_CUR;
83                       case SEEK_END:
84                           return LFS_SEEK_END;
85                       default:
86                           return whence;
87                   }
88                   }
89
90                 static int lfs_tomode(int type)
91                 {
92                     int mode = S_IRWXU | S_IRWXG | S_IRWXO;
93                     switch (type) {
94                         case LFS_TYPE_DIR:
95                             return mode | S_IFDIR;
96                         case LFS_TYPE_REG:
97                             return mode | S_IFREG;
98                         default:
99                             return 0;
100                    }
101                }
102
103     4  static int lfs_totype(int type)
104                {
105 ✓✓✗     4      switch (type) {
106     2          case LFS_TYPE_DIR:
107     2              return DT_DIR;
108     2          case LFS_TYPE_REG:
```

```
109        2             return DT_REG;
110                    default:
111                        return DT_UNKNOWN;
112                }
113            }
114
115
116            ////// Block device operations //////
117         4489       static int lfs_bd_read(const struct lfs_config *c, lfs_block_t block,
118                                   lfs_off_t off, void *buffer, lfs_size_t size)
119            {
120         4489           BlockDevice *bd = (BlockDevice *)c->context;
121         4489           return bd->read(buffer, (bd_addr_t)block * c->block_size + off, size);
122            }
123
124          429       static int lfs_bd_prog(const struct lfs_config *c, lfs_block_t block,
125                                   lfs_off_t off, const void *buffer, lfs_size_t size)
126            {
127          429           BlockDevice *bd = (BlockDevice *)c->context;
128          429           return bd->program(buffer, (bd_addr_t)block * c->block_size + off, size);
129            }
130
131          429       static int lfs_bd_erase(const struct lfs_config *c, lfs_block_t block)
132            {
133          429           BlockDevice *bd = (BlockDevice *)c->context;
134          429           return bd->erase((bd_addr_t)block * c->block_size, c->block_size);
135            }
136
137          326       static int lfs_bd_sync(const struct lfs_config *c)
138            {
139          326           BlockDevice *bd = (BlockDevice *)c->context;
140          326           return bd->sync();
141            }
142
143
144            ////// Generic filesystem operations //////
145
146            // Filesystem implementation (See LittleFileSystem.h)
147            4  LittleFileSystem::LittleFileSystem(const char *name, BlockDevice *bd,
148                                                 lfs_size_t read_size, lfs_size_t prog_size,
```

```
149        4                                       lfs_size_t block_size, lfs_size_t lookahead)
150                : FileSystem(name)
151                , _lfs()
152                , _config()
153                , _bd(NULL)
154                , _read_size(read_size)
155                , _prog_size(prog_size)
156                , _block_size(block_size)
157        4       , _lookahead(lookahead)
158                {
159    ✓✗  4           if (bd) {
160    ✓✗  4               mount(bd);
161                    }
162        4       }
163
164       12   LittleFileSystem::~LittleFileSystem()
165            {
166                // nop if unmounted
167        4       unmount();
168        8   }
169
170       12   int LittleFileSystem::mount(BlockDevice *bd)
171            {
172       12       _mutex.lock();
173                LFS_INFO("mount(%p)", bd);
174       12       _bd = bd;
175       12       int err = _bd->init();
176    ✗✓ 12       if (err) {
177                    _bd = NULL;
178                    LFS_INFO("mount -> %d", err);
179                    _mutex.unlock();
180                    return err;
181                }
182
183       12       memset(&_config, 0, sizeof(_config));
184       12       _config.context = bd;
185       12       _config.read  = lfs_bd_read;
186       12       _config.prog  = lfs_bd_prog;
187       12       _config.erase = lfs_bd_erase;
188       12       _config.sync  = lfs_bd_sync;
```

```
189          12      _config.read_size   = bd->get_read_size();
190   ✗✓     12      if (_config.read_size < _read_size) {
191                      _config.read_size = _read_size;
192                  }
193          12      _config.prog_size   = bd->get_program_size();
194   ✗✓     12      if (_config.prog_size < _prog_size) {
195                      _config.prog_size = _prog_size;
196                  }
197          12      _config.block_size  = bd->get_erase_size();
198   ✗✓     12      if (_config.block_size < _block_size) {
199                      _config.block_size = _block_size;
200                  }
201          12      _config.block_count = bd->size() / _config.block_size;
202          12      _config.lookahead = 32 * ((_config.block_count + 31) / 32);
203   ✗✓     12      if (_config.lookahead > _lookahead) {
204                      _config.lookahead = _lookahead;
205                  }
206
207          12      err = lfs_mount(&_lfs, &_config);
208   ✓✓     12      if (err) {
209           8          _bd = NULL;
210                      LFS_INFO("mount -> %d", lfs_toerror(err));
211           8          _mutex.unlock();
212           8          return lfs_toerror(err);
213                  }
214
215           4      _mutex.unlock();
216                  LFS_INFO("mount -> %d", 0);
217           4      return 0;
218              }
219
220           8  int LittleFileSystem::unmount()
221              {
222           8      _mutex.lock();
223                  LFS_INFO("unmount(%s)", "");
224           8      int res = 0;
225   ✓✓      8      if (_bd) {
226           4          int err = lfs_unmount(&_lfs);
227 ✗✓✗✗      4          if (err && !res) {
228                          res = lfs_toerror(err);
```

```
229                 }
230
231         4       err = _bd->deinit();
232 ✗✓✗     4       if (err && !res) {
233                     res = err;
234                 }
235
236         4       _bd = NULL;
237             }
238
239             LFS_INFO("unmount -> %d", res);
240         8   _mutex.unlock();
241         8   return res;
242         }
243
244         4   int LittleFileSystem::format(BlockDevice *bd,
245                                          lfs_size_t read_size, lfs_size_t prog_size,
246                                          lfs_size_t block_size, lfs_size_t lookahead)
247         {
248             LFS_INFO("format(%p, %ld, %ld, %ld, %ld)",
249                      bd, read_size, prog_size, block_size, lookahead);
250   ✓✗    4       int err = bd->init();
251   ✗✓    4       if (err) {
252                     LFS_INFO("format -> %d", err);
253                     return err;
254                 }
255
256                 lfs_t _lfs;
257                 struct lfs_config _config;
258
259         4       memset(&_config, 0, sizeof(_config));
260         4       _config.context = bd;
261         4       _config.read  = lfs_bd_read;
262         4       _config.prog  = lfs_bd_prog;
263         4       _config.erase = lfs_bd_erase;
264         4       _config.sync  = lfs_bd_sync;
265   ✓✗    4       _config.read_size   = bd->get_read_size();
266   ✗✓    4       if (_config.read_size < read_size) {
267                     _config.read_size = read_size;
268                 }
```

```
269  ✓✗  4      _config.prog_size   = bd->get_program_size();
270  ✗✓  4      if (_config.prog_size < prog_size) {
271                  _config.prog_size = prog_size;
272                }
273  ✓✗  4      _config.block_size  = bd->get_erase_size();
274  ✗✓  4      if (_config.block_size < block_size) {
275                  _config.block_size = block_size;
276                }
277  ✓✗  4      _config.block_count = bd->size() / _config.block_size;
278      4      _config.lookahead = 32 * ((_config.block_count + 31) / 32);
279  ✗✓  4      if (_config.lookahead > lookahead) {
280                  _config.lookahead = lookahead;
281                }
282
283  ✓✗  4      err = lfs_format(&_lfs, &_config);
284  ✗✓  4      if (err) {
285                  LFS_INFO("format -> %d", lfs_toerror(err));
286                  return lfs_toerror(err);
287                }
288
289  ✓✗  4      err = bd->deinit();
290  ✗✓  4      if (err) {
291                  LFS_INFO("format -> %d", err);
292                  return err;
293                }
294
295                LFS_INFO("format -> %d", 0);
296      4         return 0;
297              }
298
299      4   int LittleFileSystem::reformat(BlockDevice *bd)
300          {
301      4       _mutex.lock();
302              LFS_INFO("reformat(%p)", bd);
303  ✗✓  4       if (_bd) {
304                  if (!bd) {
305                      bd = _bd;
306                  }
307
308                  int err = unmount();
```

```
309                 if (err) {
310                     LFS_INFO("reformat -> %d", err);
311                     _mutex.unlock();
312                     return err;
313                 }
314             }
315
316 ✗✓ 4       if (!bd) {
317                 LFS_INFO("reformat -> %d", -ENODEV);
318                 _mutex.unlock();
319                 return -ENODEV;
320             }
321
322             int err = LittleFileSystem::format(bd,
323 4                                              _read_size, _prog_size, _block_size, _lookahead);
324 ✗✓ 4       if (err) {
325                 LFS_INFO("reformat -> %d", err);
326                 _mutex.unlock();
327                 return err;
328             }
329
330 4           err = mount(bd);
331 ✗✓ 4       if (err) {
332                 LFS_INFO("reformat -> %d", err);
333                 _mutex.unlock();
334                 return err;
335             }
336
337             LFS_INFO("reformat -> %d", 0);
338 4           _mutex.unlock();
339 4           return 0;
340         }
341
342 100     int LittleFileSystem::remove(const char *filename)
343         {
344 100         _mutex.lock();
345             LFS_INFO("remove(\"%s\")", filename);
346 100         int err = lfs_remove(&_lfs, filename);
347             LFS_INFO("remove -> %d", lfs_toerror(err));
348 100         _mutex.unlock();
```

```
349        100      return lfs_toerror(err);
350                 }
351
352                 int LittleFileSystem::rename(const char *oldname, const char *newname)
353                 {
354                     _mutex.lock();
355                     LFS_INFO("rename(\"%s\", \"%s\")", oldname, newname);
356                     int err = lfs_rename(&_lfs, oldname, newname);
357                     LFS_INFO("rename -> %d", lfs_toerror(err));
358                     _mutex.unlock();
359                     return lfs_toerror(err);
360                 }
361
362          4      int LittleFileSystem::mkdir(const char *name, mode_t mode)
363                 {
364          4          _mutex.lock();
365                     LFS_INFO("mkdir(\"%s\", 0x%lx)", name, mode);
366          4          int err = lfs_mkdir(&_lfs, name);
367                     LFS_INFO("mkdir -> %d", lfs_toerror(err));
368          4          _mutex.unlock();
369          4          return lfs_toerror(err);
370                 }
371
372                 int LittleFileSystem::stat(const char *name, struct stat *st)
373                 {
374                     struct lfs_info info;
375                     _mutex.lock();
376                     LFS_INFO("stat(\"%s\", %p)", name, st);
377                     int err = lfs_stat(&_lfs, name, &info);
378                     LFS_INFO("stat -> %d", lfs_toerror(err));
379                     _mutex.unlock();
380                     st->st_size = info.size;
381                     st->st_mode = lfs_tomode(info.type);
382                     return lfs_toerror(err);
383                 }
384
385                 static int lfs_statvfs_count(void *p, lfs_block_t b)
386                 {
387                     *(lfs_size_t *)p += 1;
388                     return 0;
```

```
389          }
390
391          int LittleFileSystem::statvfs(const char *name, struct statvfs *st)
392          {
393              memset(st, 0, sizeof(struct statvfs));
394
395              lfs_size_t in_use = 0;
396              _mutex.lock();
397              LFS_INFO("statvfs(\"%s\", %p)", name, st);
398              int err = lfs_traverse(&_lfs, lfs_statvfs_count, &in_use);
399              LFS_INFO("statvfs -> %d", lfs_toerror(err));
400              _mutex.unlock();
401              if (err) {
402                  return err;
403              }
404
405              st->f_bsize  = _config.block_size;
406              st->f_frsize = _config.block_size;
407              st->f_blocks = _config.block_count;
408              st->f_bfree  = _config.block_count - in_use;
409              st->f_bavail = _config.block_count - in_use;
410              st->f_namemax = LFS_NAME_MAX;
411              return 0;
412          }
413
414          ////// File operations //////
415      407 int LittleFileSystem::file_open(fs_file_t *file, const char *path, int flags)
416          {
417      407     lfs_file_t *f = new lfs_file_t;
418      407     _mutex.lock();
419              LFS_INFO("file_open(%p, \"%s\", 0x%x)", *file, path, flags);
420      407     int err = lfs_file_open(&_lfs, f, path, lfs_fromflags(flags));
421              LFS_INFO("file_open -> %d", lfs_toerror(err));
422      407     _mutex.unlock();
423   ✓✓ 407     if (!err) {
424      304         *file = f;
425              } else {
426      103         delete f;
427              }
428      407     return lfs_toerror(err);
```

```
429                 }
430
431     304     int LittleFileSystem::file_close(fs_file_t file)
432                 {
433     304         lfs_file_t *f = (lfs_file_t *)file;
434     304         _mutex.lock();
435                 LFS_INFO("file_close(%p)", file);
436     304         int err = lfs_file_close(&_lfs, f);
437                 LFS_INFO("file_close -> %d", lfs_toerror(err));
438     304         _mutex.unlock();
439     304         delete f;
440     304         return lfs_toerror(err);
441                 }
442
443     302     ssize_t LittleFileSystem::file_read(fs_file_t file, void *buffer, size_t len)
444                 {
445     302         lfs_file_t *f = (lfs_file_t *)file;
446     302         _mutex.lock();
447                 LFS_INFO("file_read(%p, %p, %d)", file, buffer, len);
448     302         lfs_ssize_t res = lfs_file_read(&_lfs, f, buffer, len);
449                 LFS_INFO("file_read -> %d", lfs_toerror(res));
450     302         _mutex.unlock();
451     302         return lfs_toerror(res);
452                 }
453
454     206     ssize_t LittleFileSystem::file_write(fs_file_t file, const void *buffer, size_t len)
455                 {
456     206         lfs_file_t *f = (lfs_file_t *)file;
457     206         _mutex.lock();
458                 LFS_INFO("file_write(%p, %p, %d)", file, buffer, len);
459     206         lfs_ssize_t res = lfs_file_write(&_lfs, f, buffer, len);
460                 LFS_INFO("file_write -> %d", lfs_toerror(res));
461     206         _mutex.unlock();
462     206         return lfs_toerror(res);
463                 }
464
465             int LittleFileSystem::file_sync(fs_file_t file)
466             {
467                 lfs_file_t *f = (lfs_file_t *)file;
468                 _mutex.lock();
```

```
469                     LFS_INFO("file_sync(%p)", file);
470                     int err = lfs_file_sync(&_lfs, f);
471                     LFS_INFO("file_sync -> %d", lfs_toerror(err));
472                     _mutex.unlock();
473                     return lfs_toerror(err);
474                 }
475
476         101     off_t LittleFileSystem::file_seek(fs_file_t file, off_t offset, int whence)
477                 {
478         101         lfs_file_t *f = (lfs_file_t *)file;
479         101         _mutex.lock();
480                     LFS_INFO("file_seek(%p, %ld, %d)", file, offset, whence);
481         101         off_t res = lfs_file_seek(&_lfs, f, offset, lfs_fromwhence(whence));
482                     LFS_INFO("file_seek -> %d", lfs_toerror(res));
483         101         _mutex.unlock();
484         101         return lfs_toerror(res);
485                 }
486
487                 off_t LittleFileSystem::file_tell(fs_file_t file)
488                 {
489                     lfs_file_t *f = (lfs_file_t *)file;
490                     _mutex.lock();
491                     LFS_INFO("file_tell(%p)", file);
492                     off_t res = lfs_file_tell(&_lfs, f);
493                     LFS_INFO("file_tell -> %d", lfs_toerror(res));
494                     _mutex.unlock();
495                     return lfs_toerror(res);
496                 }
497
498         101     off_t LittleFileSystem::file_size(fs_file_t file)
499                 {
500         101         lfs_file_t *f = (lfs_file_t *)file;
501         101         _mutex.lock();
502                     LFS_INFO("file_size(%p)", file);
503         101         off_t res = lfs_file_size(&_lfs, f);
504                     LFS_INFO("file_size -> %d", lfs_toerror(res));
505         101         _mutex.unlock();
506         101         return lfs_toerror(res);
507                 }
508
```

```
509                 int LittleFileSystem::file_truncate(fs_file_t file, off_t length)
510                 {
511                     lfs_file_t *f = (lfs_file_t *)file;
512                     _mutex.lock();
513                     LFS_INFO("file_truncate(%p)", file);
514                     int err = lfs_file_truncate(&_lfs, f, length);
515                     LFS_INFO("file_truncate -> %d", lfs_toerror(err));
516                     _mutex.unlock();
517                     return lfs_toerror(err);
518                 }
519
520
521                 ////// Dir operations //////
522           107   int LittleFileSystem::dir_open(fs_dir_t *dir, const char *path)
523                 {
524           107       lfs_dir_t *d = new lfs_dir_t;
525           107       _mutex.lock();
526                     LFS_INFO("dir_open(%p, \"%s\")", *dir, path);
527           107       int err = lfs_dir_open(&_lfs, d, path);
528                     LFS_INFO("dir_open -> %d", lfs_toerror(err));
529           107       _mutex.unlock();
530      ✓✓   107       if (!err) {
531           103           *dir = d;
532                     } else {
533             4           delete d;
534                     }
535           107       return lfs_toerror(err);
536                 }
537
538           103   int LittleFileSystem::dir_close(fs_dir_t dir)
539                 {
540           103       lfs_dir_t *d = (lfs_dir_t *)dir;
541           103       _mutex.lock();
542                     LFS_INFO("dir_close(%p)", dir);
543           103       int err = lfs_dir_close(&_lfs, d);
544                     LFS_INFO("dir_close -> %d", lfs_toerror(err));
545           103       _mutex.unlock();
546           103       delete d;
547           103       return lfs_toerror(err);
548                 }
```

```
549
550  5  ssize_t LittleFileSystem::dir_read(fs_dir_t dir, struct dirent *ent)
551     {
552  5      lfs_dir_t *d = (lfs_dir_t *)dir;
553         struct lfs_info info;
554  5      _mutex.lock();
555         LFS_INFO("dir_read(%p, %p)", dir, ent);
556 ✓✗ 5      int res = lfs_dir_read(&_lfs, d, &info);
557         LFS_INFO("dir_read -> %d", lfs_toerror(res));
558  5      _mutex.unlock();
559 ✓✓ 5      if (res == 1) {
560  4          ent->d_type = lfs_totype(info.type);
561  4          strcpy(ent->d_name, info.name);
562         }
563  5      return lfs_toerror(res);
564     }
565
566     void LittleFileSystem::dir_seek(fs_dir_t dir, off_t offset)
567     {
568         lfs_dir_t *d = (lfs_dir_t *)dir;
569         _mutex.lock();
570         LFS_INFO("dir_seek(%p, %ld)", dir, offset);
571         lfs_dir_seek(&_lfs, d, offset);
572         LFS_INFO("dir_seek -> %s", "void");
573         _mutex.unlock();
574     }
575
576     off_t LittleFileSystem::dir_tell(fs_dir_t dir)
577     {
578         lfs_dir_t *d = (lfs_dir_t *)dir;
579         _mutex.lock();
580         LFS_INFO("dir_tell(%p)", dir);
581         lfs_soff_t res = lfs_dir_tell(&_lfs, d);
582         LFS_INFO("dir_tell -> %d", lfs_toerror(res));
583         _mutex.unlock();
584         return lfs_toerror(res);
585     }
586
587     void LittleFileSystem::dir_rewind(fs_dir_t dir)
588     {
```

```
589                 lfs_dir_t *d = (lfs_dir_t *)dir;
590                 _mutex.lock();
591                 LFS_INFO("dir_rewind(%p)", dir);
592                 lfs_dir_rewind(&_lfs, d);
593                 LFS_INFO("dir_rewind -> %s", "void");
594                 _mutex.unlock();
595             }
596
597             } // namespace mbed
```

Generated by: GCOVR (Version 4.2)