cprj

AUTHOR

版本

2019年 六月 27日 星期四

目录

Table of contents

类索引

类列表

这里列出了所有类、结构、联合以及接口定义等，并附带简要说明:

文件索引

文件列表

这里列出了所有文档化的文件，并附带简要说明:

类说明

mux_t结构体 参考

互斥锁类型定义
`#include <mux.h>`

Public 属性

- pthread_mutex_t mux
  linux互斥锁

- pthread_mutexattr_t attr
  linux互斥锁属性

详细描述

互斥锁类型定义

在文件 mux.h 第 19 行定义.

该结构体的文档由以下文件生成:

- mux.h

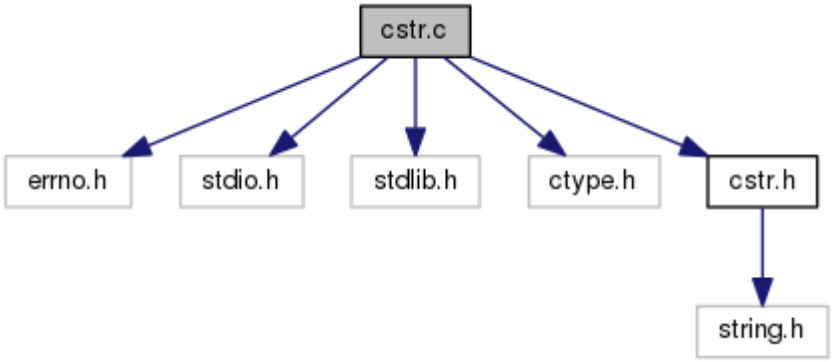
文件说明

cstr.c 文件参考

提供方便的字符串操作

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "cstr.h"
```

cstr.c 的引用(Include)关系图:



函数

- char * strlwr (char *s)
  字符串转为小写
- char * strupr (char *s)
  字符串转为大写
- int strstrip (char *s)
  删除给定字符串开始和结尾两处的所有空格
- int bin2hex (char *hex, const void *bin, size_t len)
  二进制转换为hex
- char * abin2hex (const void *bin, size_t len)
  二进制转换为hex，输出缓存由函数负责malloc
- int hex2bin (void *bin, const char *hex, size_t len)
  hex转换为二进制
- void * ahex2bin (const char *hex, size_t *bin_len)
  hex转换为二进制，输出缓存由函数负责malloc
- int memswap (void *out, const void *in, size_t len, size_t section_size)
  以字节为单位，反转指定长度的内存块

详细描述

提供方便的字符串操作

作者:

ln

在文件 cstr.c 中定义.

函数说明

char* abin2hex (const void * *bin*, size_t *len*)

二进制转换为hex，输出缓存由函数负责malloc

参数:

bin	输入的二进制数据
len	输入二进制数据的长度

返回:

成功返回hex字符串指针，失败返回NULL并设置errno

注意:

返回的hex缓存指针，需要free

在文件 cstr.c 第 127 行定义.

```
128 {
129     if (bin == NULL || len == 0) {
130         errno = EINVAL;
131         return NULL;
132     }
133 
134     char *s = (char*)malloc((len * 2) + 1);
135     if (s != NULL) {
136         bin2hex(s, bin, len);
137     }
138     return s;
139 }
```

函数调用图:

void* ahex2bin (const char * *hex*, size_t * *bin_len*)

hex转换为二进制，输出缓存由函数负责malloc

参数:

hex	输入的hex字符串
bin_len	输出缓存里已转换的字节数，输出缓存最大长度是strlen(hex); bin_len可以为NULL

返回:

成功返回二进制数据的指针，失败返回NULL并设置errno

注意:

hex串必须以0结束；返回的二进制数据指针需要free

在文件 cstr.c 第 186 行定义.

```
187 {
188     if (hex == NULL) {
189         errno = EINVAL;
190         return NULL;
191     }
192 
193     size_t len = strlen(hex);
194     unsigned char *b = (unsigned char*)malloc(len/2 + 1);
195     if (b != NULL) {
196         size_t ret = hex2bin(b, hex, len/2 + 1);
197         if (bin_len)
198             *bin_len = ret;
199     }
200     return (void *)b;
201 }
```

函数调用图:

int bin2hex (char * *hex*, const void * *bin*, size_t *len*)

二进制转换为hex

参数:

hex	输出的hex字符串
bin	输入的二进制数据
len	输入二进制数据的长度

返回:

成功返回hex字符串的长度(len*2)，失败返回-1并设置errno

注意:

hex缓存的大小至少是(len*2 + 1)，否则发生溢出

在文件 cstr.c 第 102 行定义.

```
103 {
104     if (bin == NULL || hex == NULL || len == 0) {
105         errno = EINVAL;
106         return -1;
107     }
108     size_t i = 0;
109     for (i = 0; i < len; i++) {
110         sprintf(&hex[i*2], "%02x", ((unsigned char *)bin)[i]);
111     }
112     hex[i*2] = '\0';
113     return (int)(i*2);
114 }
```

这是这个函数的调用关系图:

int hex2bin (void * *bin*, const char * *hex*, size_t *len*)

hex转换为二进制

参数:

bin	输出的二进制数据
hex	输入的hex字符串
len	输出二进制数据的缓存长度

返回:

成功返回输出的二进制数据的长度，失败返回-1并设置errno

注意:

hex不能以0x开头；hex的有效转换长度必须是偶数；参数len指的是输出缓存的长度而不是输入的长度

注解:

因为输入的hex字符串往往只是另一个更长串的某个部分，

例如'1f2f3f4f, 1a2a3a4a,...'，此时函数遇到非法字符则自动停止(不会导致返回-1)，

或者字符串并非以0结束，例如通信数据包中的字符串['a'，'b'，'c', 'd']，此时需要只要控制len(=2)便可避免越界

在文件 cstr.c 第 157 行定义.

```
158 {
159     if (bin == NULL || hex == NULL || len == 0) {
160         errno = EINVAL;
161         return -1;
162     }
163 
164     size_t len2 = len;
165     char buf[4] = {0};
166     while (isxdigit(*hex) && len) {
167         buf[0] = *hex++;
168         buf[1] = *hex++;
169         *((char *)bin) = (char)strtol(buf, NULL, 16);
170         len--;
171         bin = (char *)bin + 1;
172     }
173     return (int)(len2 - len);
174 }
```

这是这个函数的调用关系图:

int memswap (void * *out*, const void * *in*, size_t *len*, size_t *section_size*)

以字节为单位，反转指定长度的内存块

参数:

out	输出缓存
in	输入缓存，in和out可以相同
len	输入数据的大小
section_size	需要被反转的块的大小，len必须是块大小的整数倍，否则那些多余的字节将得不到转换， 如果section_size为0,则section_size将被视为与len相等

返回:

成功返回0，失败返回-1并设置errno

举例

按一个字节反转:

```
memswap(out,int,4,2);                    // [0][1][2][3] ===> [1][0][3][2]
memswap(out,int,4,4):                    // [0][1][2][3] ===> [3][2][1][0]
```

通过组合调用，也可以做到按多个字节作为整体进行反转:

```
char buf[] = {1,2,3,4};
memswap(buf, buf, 4, 0);
memswap(buf, buf, 4, 2);    // result is {3,4,1,2}
```

在文件 cstr.c 第 229 行定义.

```
230 {
231     if (section_size == 0) {
232         section_size = len;
233     }
234 
```

```
235     if (section_size == 0 || out == NULL || in == NULL || len % section_size) {
236         errno = EINVAL;
237         return -1;
238     }
239 
240     if (len <= 1 || section_size <= 1)  // no need to swap
241         return 0;
242 
243     size_t num = len/section_size;
244     for (size_t i=0; i<num; i++) {
245         char *pin = (char *)in + i*section_size;
246         char *pout = (char *)out + i*section_size;
247         size_t ss = 0;
248         while (ss < section_size/2) {
249             char c = pin[ss];
250             pout[ss] = pin[section_size - ss - 1];
251             pout[section_size - ss - 1] = c;
252             ss++;
253         }
254     }
255 
256     return 0;
257 }
```

char* strlwr (char * s)

字符串转为小写

参数:

s	被转换的字符串

返回值:

!NULL	成功返回字符串s
NULL	失败并设置errno

在文件 cstr.c 第 25 行定义.

```
26 {
27     if (s == NULL) {
28         errno = EINVAL;
29         return NULL;
30     }
31     char *p = s;
32     while (*p) {
33         *p = (char)tolower((int)(*p));
34         p++;
35     }
36     return s;
37 }
```

int strstrip (char * s)

删除给定字符串开始和结尾两处的所有空格

参数:

s	被操作的字符串

返回:

成功返回过滤后的字符串长度，失败返回-1并设置errno

在文件 cstr.c 第 66 行定义.

```
67 {
68     char *last = NULL ;
69     char *dest = s;
70 
71     if (s == NULL) {
72         errno = EINVAL;
73         return -1;
74     }
75 
76     last = s + strlen(s);
```

```
77     while (isspace((int)*s) && *s)
78         s++;
79     while (last > s) {
80         if (!isspace((int)*(last-1)))
81             break ;
82         last-- ;
83     }
84     *last = (char)0;
85 
86     memmove(dest, s, last-s+1);
87     return last - s;
88 }
```

char* strupr (char * s)

字符串转为大写

参数:

s	被转换的字符串

返回值:

!NULL	成功返回字符串s
NULL	失败并设置errno

在文件 cstr.c 第 47 行定义.

```
48 {
49     if (s == NULL) {
50         errno = EINVAL;
51         return NULL;
52     }
53     char *p = s;
54     while (*p) {
55         *p = (char)toupper((int)(*p));
56         p++;
57     }
58     return s;
59 }
```

cstr.c

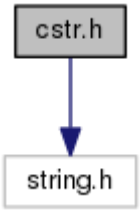
```
 1 
 7 #include <errno.h>
 8 #include <stdio.h>
 9 #include <stdlib.h>
10 #include <ctype.h>
11 #include "cstr.h"
12 
13 #ifdef __cplusplus
14 extern "C" {
15 #endif
16 
25 char* strlwr(char *s)
26 {
27     if (s == NULL) {
28         errno = EINVAL;
29         return NULL;
30     }
31     char *p = s;
32     while (*p) {
33         *p = (char)tolower((int)(*p));
34         p++;
35     }
36     return s;
37 }
38 
47 char* strupr(char *s)
48 {
49     if (s == NULL) {
50         errno = EINVAL;
51         return NULL;
52     }
53     char *p = s;
54     while (*p) {
```

```
 55        *p = (char)toupper((int)(*p));
 56        p++;
 57     }
 58     return s;
 59  }
 60
 66  int strstrip(char *s)
 67  {
 68     char *last = NULL ;
 69     char *dest = s;
 70
 71     if (s == NULL) {
 72        errno = EINVAL;
 73        return -1;
 74     }
 75
 76     last = s + strlen(s);
 77     while (isspace((int)*s) && *s)
 78        s++;
 79     while (last > s) {
 80        if (!isspace((int)*(last-1)))
 81           break ;
 82        last-- ;
 83     }
 84     *last = (char)0;
 85
 86     memmove(dest, s, last-s+1);
 87     return last - s;
 88  }
 89
 90
102  int bin2hex(char *hex, const void *bin, size_t len)
103  {
104     if (bin == NULL || hex == NULL || len == 0) {
105        errno = EINVAL;
106        return -1;
107     }
108     size_t i = 0;
109     for (i = 0; i < len; i++) {
110        sprintf(&hex[i*2], "%02x", ((unsigned char *)bin)[i]);
111     }
112     hex[i*2] = '\0';
113     return (int)(i*2);
114  }
115
116
127  char* abin2hex(const void *bin, size_t len)
128  {
129     if (bin == NULL || len == 0) {
130        errno = EINVAL;
131        return NULL;
132     }
133
134     char *s = (char*)malloc((len * 2) + 1);
135     if (s != NULL) {
136        bin2hex(s, bin, len);
137     }
138     return s;
139  }
140
141
157  int hex2bin(void *bin, const char *hex, size_t len)
158  {
159     if (bin == NULL || hex == NULL || len == 0) {
160        errno = EINVAL;
161        return -1;
162     }
163
164     size_t len2 = len;
165     char buf[4] = {0};
166     while (isxdigit(*hex) && len) {
167        buf[0] = *hex++;
168        buf[1] = *hex++;
169        *((char *)bin) = (char)strtol(buf, NULL, 16);
170        len--;
171        bin = (char *)bin + 1;
172     }
173     return (int)(len2 - len);
174  }
175
186  void* ahex2bin(const char *hex, size_t *bin_len)
187  {
188     if (hex == NULL) {
```

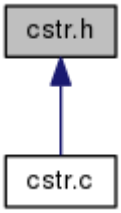
```
189         errno = EINVAL;
190         return NULL;
191     }
192 
193     size_t len = strlen(hex);
194     unsigned char *b = (unsigned char*)malloc(len/2 + 1);
195     if (b != NULL) {
196         size_t ret = hex2bin(b, hex, len/2 + 1);
197         if (bin_len)
198             *bin_len = ret;
199     }
200     return (void *)b;
201 }
202 
229 int memswap(void *out, const void *in, size_t len, size_t section_size)
230 {
231     if (section_size == 0) {
232         section_size = len;
233     }
234 
235     if (section_size == 0 || out == NULL || in == NULL || len % section_size) {
236         errno = EINVAL;
237         return -1;
238     }
239 
240     if (len <= 1 || section_size <= 1)  // no need to swap
241         return 0;
242 
243     size_t num = len/section_size;
244     for (size_t i=0; i<num; i++) {
245         char *pin = (char *)in + i*section_size;
246         char *pout = (char *)out + i*section_size;
247         size_t ss = 0;
248         while (ss < section_size/2) {
249             char c = pin[ss];
250             pout[ss] = pin[section_size - ss - 1];
251             pout[section_size - ss - 1] = c;
252             ss++;
253         }
254     }
255 
256     return 0;
257 }
258 
259 #ifdef __cplusplus
260 }
261 #endif
262 
```

cstr.h 文件参考

提供方便的字符串操作
`#include <string.h>`
cstr.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



宏定义

- #define _MAKE_CSTR(s) #s
  构造字符串，如果s是宏，则结果为宏名本身

- #define MAKE_CSTR(s) _MAKE_CSTR(s)
  构造字符串，如果s是宏，则结果为宏所表示的内容

- #define _CONCAT_STRING(l, r) l##r
  连接字符串，如果l,r是宏，则结果为宏名本身

- #define CONCAT_STRING(l, r) _CONCAT_STRING(l, r)
  连接字符串，如果l,r是宏，则结果为宏所表示的内容

函数

- char * strlwr (char *s)
  字符串转为小写

- char * strupr (char *s)
  字符串转为大写

- int strstrip (char *s)
  删除给定字符串开始和结尾两处的所有空格

- int bin2hex (char *hex, const void *bin, size_t len)
  二进制转换为hex

- char * abin2hex (const void *bin, size_t len)
  二进制转换为hex，输出缓存由函数负责malloc

- int hex2bin (void *bin, const char *hex, size_t len)
  hex转换为二进制

- void * ahex2bin (const char *hex, size_t *bin_len)
  hex转换为二进制，输出缓存由函数负责malloc

- int memswap (void *out, const void *in, size_t len, size_t section_size)
  以字节为单位，反转指定长度的内存块

详细描述

提供方便的字符串操作

作者:

ln

在文件 cstr.h 中定义.

函数说明

char* abin2hex (const void * *bin*, size_t *len*)

二进制转换为hex，输出缓存由函数负责malloc

参数:

bin	输入的二进制数据
len	输入二进制数据的长度

返回:

成功返回hex字符串指针，失败返回NULL并设置errno

注意:

返回的hex缓存指针，需要free

在文件 cstr.c 第 127 行定义.

```
128 {
129     if (bin == NULL || len == 0) {
130         errno = EINVAL;
131         return NULL;
132     }
133 
134     char *s = (char*)malloc((len * 2) + 1);
135     if (s != NULL) {
136         bin2hex(s, bin, len);
137     }
138     return s;
139 }
```

函数调用图:

void* ahex2bin (const char * *hex*, size_t * *bin_len*)

hex转换为二进制，输出缓存由函数负责malloc

参数:

hex	输入的hex字符串
bin_len	输出缓存里已转换的字节数，输出缓存最大长度是strlen(hex); bin_len可以为NULL

返回:

成功返回二进制数据的指针，失败返回NULL并设置errno

注意:

hex串必须以0结束；返回的二进制数据指针需要free

在文件 cstr.c 第 186 行定义.

```
187 {
188     if (hex == NULL) {
189         errno = EINVAL;
190         return NULL;
191     }
192 
193     size_t len = strlen(hex);
194     unsigned char *b = (unsigned char*)malloc(len/2 + 1);
195     if (b != NULL) {
196         size_t ret = hex2bin(b, hex, len/2 + 1);
197         if (bin_len)
198             *bin_len = ret;
199     }
200     return (void *)b;
201 }
```

函数调用图:

int bin2hex (char * *hex*, const void * *bin*, size_t *len*)

二进制转换为hex

参数:

hex	输出的hex字符串
bin	输入的二进制数据
len	输入二进制数据的长度

返回:

成功返回hex字符串的长度(len*2)，失败返回-1并设置errno

注意:

hex缓存的大小至少是(len*2 + 1)，否则发生溢出

在文件 cstr.c 第 102 行定义.

```
103 {
104     if (bin == NULL || hex == NULL || len == 0) {
105         errno = EINVAL;
106         return -1;
107     }
108     size_t i = 0;
109     for (i = 0; i < len; i++) {
110         sprintf(&hex[i*2], "%02x", ((unsigned char *)bin)[i]);
111     }
112     hex[i*2] = '\0';
113     return (int)(i*2);
114 }
```

这是这个函数的调用关系图:

int hex2bin (void * *bin*, const char * *hex*, size_t *len*)

hex转换为二进制

参数:

bin	输出的二进制数据
hex	输入的hex字符串
len	输出二进制数据的缓存长度

返回:

成功返回输出的二进制数据的长度，失败返回-1并设置errno

注意:

hex不能以0x开头；hex的有效转换长度必须是偶数；参数len指的是输出缓存的长度而不是输入的长度

注解:

因为输入的hex字符串往往只是另一个更长串的某个部分，

例如'1f2f3f4f, 1a2a3a4a,...'，此时函数遇到非法字符则自动停止(不会导致返回-1)，

或者字符串并非以0结束，例如通信数据包中的字符串['a'，'b'，'c', 'd']，此时需要只要控制len(=2)便可避免越界

在文件 cstr.c 第 157 行定义.

```
158 {
159     if (bin == NULL || hex == NULL || len == 0) {
160         errno = EINVAL;
161         return -1;
162     }
163 
164     size_t len2 = len;
165     char buf[4] = {0};
166     while (isxdigit(*hex) && len) {
167         buf[0] = *hex++;
168         buf[1] = *hex++;
169         *((char *)bin) = (char)strtol(buf, NULL, 16);
170         len--;
171         bin = (char *)bin + 1;
172     }
173     return (int)(len2 - len);
174 }
```

这是这个函数的调用关系图:

int memswap (void * *out*, const void * *in*, size_t *len*, size_t *section_size*)

以字节为单位，反转指定长度的内存块

参数:

out	输出缓存
in	输入缓存，in和out可以相同
len	输入数据的大小
section_size	需要被反转的块的大小，len必须是块大小的整数倍，否则那些多余的字节将得不到转换， 如果section_size为0,则section_size将被视为与len相等

返回:

成功返回0，失败返回-1并设置errno

举例

按一个字节反转:

```
memswap(out,int,4,2);                    // [0][1][2][3] ===> [1][0][3][2]
memswap(out,int,4,4):                    // [0][1][2][3] ===> [3][2][1][0]
```

通过组合调用，也可以做到按多个字节作为整体进行反转：

```
char buf[] = {1,2,3,4};
memswap(buf, buf, 4, 0);
memswap(buf, buf, 4, 2);    // result is {3,4,1,2}
```

在文件 cstr.c 第 229 行定义.

```
230 {
231     if (section_size == 0) {
232         section_size = len;
233     }
234
235     if (section_size == 0 || out == NULL || in == NULL || len % section_size) {
236         errno = EINVAL;
237         return -1;
238     }
239
240     if (len <= 1 || section_size <= 1)  // no need to swap
241         return 0;
242
243     size_t num = len/section_size;
244     for (size_t i=0; i<num; i++) {
245         char *pin = (char *)in + i*section_size;
246         char *pout = (char *)out + i*section_size;
247         size_t ss = 0;
248         while (ss < section_size/2) {
249             char c = pin[ss];
250             pout[ss] = pin[section_size - ss - 1];
251             pout[section_size - ss - 1] = c;
252             ss++;
253         }
254     }
255
256     return 0;
257 }
```

char* strlwr (char * s)

字符串转为小写

参数:

s	被转换的字符串

返回值:

!NULL	成功返回字符串s
NULL	失败并设置errno

在文件 cstr.c 第 25 行定义.

```
26 {
27     if (s == NULL) {
28         errno = EINVAL;
29         return NULL;
30     }
31     char *p = s;
32     while (*p) {
33         *p = (char)tolower((int)(*p));
34         p++;
35     }
36     return s;
37 }
```

int strstrip (char * s)

删除给定字符串开始和结尾两处的所有空格

参数:

s	被操作的字符串

返回:

成功返回过滤后的字符串长度，失败返回-1并设置errno

在文件 cstr.c 第 66 行定义.

```
67 {
68     char *last = NULL ;
69     char *dest = s;
70 
71     if (s == NULL) {
72         errno = EINVAL;
73         return -1;
74     }
75 
76     last = s + strlen(s);
77     while (isspace((int)*s) && *s)
78         s++;
79     while (last > s) {
80         if (!isspace((int)*(last-1)))
81             break ;
82         last-- ;
83     }
84     *last = (char)0;
85 
86     memmove(dest, s, last-s+1);
87     return last - s;
88 }
```

char* strupr (char * s)

字符串转为大写

参数:

s	被转换的字符串

返回值:

!NULL	成功返回字符串s
NULL	失败并设置errno

在文件 cstr.c 第 47 行定义.

```
48 {
49     if (s == NULL) {
50         errno = EINVAL;
51         return NULL;
52     }
53     char *p = s;
54     while (*p) {
55         *p = (char)toupper((int)(*p));
56         p++;
57     }
58     return s;
59 }
```

cstr.h

```
 1 
 7 #ifndef __C_STRING_H__
 8 #define __C_STRING_H__
 9 
10 #include <string.h>
11 
12 #ifdef __cplusplus
13 extern "C" {
14 #endif
15 
16 #define _MAKE_CSTR(s)          #s
17 #define MAKE_CSTR(s)           _MAKE_CSTR(s)
18 
19 #define _CONCAT_STRING(l, r)    l##r
```

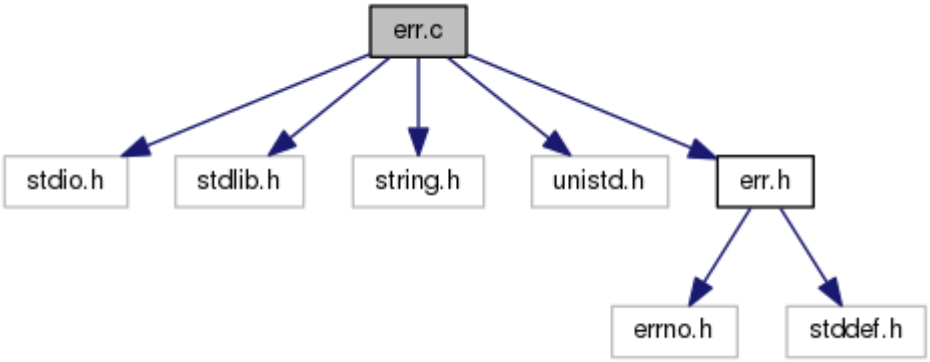
```
20 #define CONCAT_STRING(l, r)     _CONCAT_STRING(l, r)
21
22 extern char*    strlwr          (char *s);
23 extern char*    strupr          (char *s);
24
25 extern int      strstrip        (char *s);
26
27 extern int      bin2hex         (char *hex, const void *bin, size_t len);
28 extern char*    abin2hex        (const void *bin, size_t len);
29
30 extern int      hex2bin         (void *bin, const char *hex, size_t len);
31 extern void*    ahex2bin        (const char *hex, size_t *bin_len);
32
33 extern int      memswap         (void *out, const void *in, size_t len, size_t section_size);
34
35 #ifdef __cplusplus
36 }
37 #endif
38
39 #endif  /* __C_STRING_H__ */
40
```

err.c 文件参考

将系统的、库的、app 的错误码统一起来

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "err.h"
```

err.c 的引用(Include)关系图:



函数

- int err_init (void)
  err模块初始化

- int err_add (int errnum, const char *str)
  注册一个错误码和它所对应的字符串

- char * err_string (int errnum, char *buf, size_t size)
  输出给定错误码所对应的字符串，线程安全，用法类似标准库函数strerror_r()

变量

- const char unknown_str [] = "Unknown error"
  如果错误码未定义，则err_string()输出该字符串

详细描述

将系统的、库的、app 的错误码统一起来

作者:

ln

模块初始化(err_init())完成后，需要先调用err_add()增加自定义的错误码和对应字符串，
之后就可以通过err_string()输出错误码所对应的字符串。

函数err_string()是线程安全的，和它用法类似的标准库函数是strerror_r()

在文件 err.c 中定义.

函数说明

int err_add (int *errnum*, const char * *str*)

注册一个错误码和它所对应的字符串

参数:

errnum	错误码
str	错误码所对应的字符串，函数内部会使用strdup()对该字符串进行拷贝

返回值:

0	成功
-1	失败并设置errno

在文件 err.c 第 59 行定义.

```
60 {
61     char *p;
62     if (errnum < LIB_ERRNO_BASE ||
63         errnum > (LIB_ERRNO_MAX_NUM + LIB_ERRNO_BASE - 1) || str == NULL) {
64         errno = EINVAL;
65         return -1;
66     }
67     if ((p = strdup(str)) == NULL)
68         return -1;
69     int ix = errnum - LIB_ERRNO_BASE;
70     if (errtbl[ix] != NULL)
71         free(errtbl[ix]);
72     errtbl[ix] = p;
73     return 0;
74 }
```

这是这个函数的调用关系图:

int err_init (void)

err模块初始化

返回值:

0	成功
-1	失败并设置errno

在文件 err.c 第 33 行定义.

```
34 {
35     int ret = 0;
36     ret += err_add(LIB_ERRNO_QUE_EMPTY, "Queue empty");
37     ret += err_add(LIB_ERRNO_QUE_FULL, "Queue full");
38     ret += err_add(LIB_ERRNO_MEM_ALLOC, "Malloc fail");
39     ret += err_add(LIB_ERRNO_MBLK_SHORT, "Block size of memory-pool not enough");
40     ret += err_add(LIB_ERRNO_BUF_SHORT, "Local buffer not enough");
41     ret += err_add(LIB_ERRNO_RES_LIMIT, "Resources exceed limit");
42     ret += err_add(LIB_ERRNO_SHORT_MPOOL, "Short of memory-pool");
43     ret += err_add(LIB_ERRNO_NOT_EXIST, "Objectives not exist");
44     if (ret != 0)
45         return -1;
46     else
47         return 0;
48 }
```

函数调用图:

char* err_string (int *errnum*, char * *buf*, size_t *size*)

输出给定错误码所对应的字符串，线程安全，用法类似标准库函数strerror_r()

参数:

errnum	错误码
buf	字符串的输出buf，函数会确保输出的字符串含有结束符('\0')； 所以如果buf的size不足，输出的字符串会被截断。
size	输出buf的大小

返回值:

0	成功
-1	失败并设置errno

在文件 err.c 第 87 行定义.

```
 88 {
 89     if (errnum < 0 || buf == NULL || size < 1) {
 90         errno = EINVAL;
 91         return NULL;
 92     }
 93     if (errnum < LIB_ERRNO_BASE) {
 94         strerror_r(errnum, buf, size);
 95     } else {
 96         int ix = errnum - LIB_ERRNO_BASE;
 97         if (errtbl[ix] != NULL) {
 98             strncpy(buf, errtbl[ix], size);
 99         } else {
100             snprintf(buf, size, "%s: %d", unknown_str, errnum);
101         }
102         buf[size-1] = '\0';
103     }
104     return buf;
105 }
```

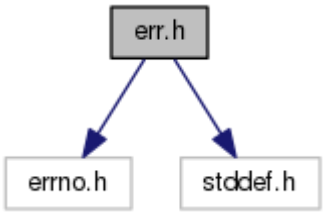
err.c

```
  1
 12 #include <stdio.h>
 13 #include <stdlib.h>
 14 #include <string.h>
 15 #include <unistd.h>
 16 #include "err.h"
 17
 18 #ifdef __cplusplus
 19 extern "C" {
 20 #endif
 21
 23 const char unknown_str[] = "Unknown error";
 24
 26 static char* errtbl[LIB_ERRNO_MAX_NUM] = {0};
 27
 33 int err_init(void)
 34 {
 35     int ret = 0;
 36     ret += err_add(LIB_ERRNO_QUE_EMPTY, "Queue empty");
 37     ret += err_add(LIB_ERRNO_QUE_FULL, "Queue full");
 38     ret += err_add(LIB_ERRNO_MEM_ALLOC, "Malloc fail");
 39     ret += err_add(LIB_ERRNO_MBLK_SHORT, "Block size of memory-pool not enough");
 40     ret += err_add(LIB_ERRNO_BUF_SHORT, "Local buffer not enough");
 41     ret += err_add(LIB_ERRNO_RES_LIMIT, "Resources exceed limit");
 42     ret += err_add(LIB_ERRNO_SHORT_MPOOL, "Short of memory-pool");
 43     ret += err_add(LIB_ERRNO_NOT_EXIST, "Objectives not exist");
 44     if (ret != 0)
 45         return -1;
 46     else
 47         return 0;
 48 }
 49
 59 int err_add(int errnum, const char *str)
 60 {
 61     char *p;
 62     if (errnum < LIB_ERRNO_BASE ||
 63         errnum > (LIB_ERRNO_MAX_NUM + LIB_ERRNO_BASE - 1) || str == NULL) {
 64         errno = EINVAL;
 65         return -1;
```

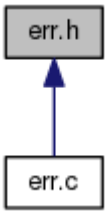
```
 66    }
 67    if ((p = strdup(str)) == NULL)
 68        return -1;
 69    int ix = errnum - LIB_ERRNO_BASE;
 70    if (errtbl[ix] != NULL)
 71        free(errtbl[ix]);
 72    errtbl[ix] = p;
 73    return 0;
 74 }
 75
 87 char* err_string(int errnum, char *buf, size_t size)
 88 {
 89    if (errnum < 0 || buf == NULL || size < 1) {
 90        errno = EINVAL;
 91        return NULL;
 92    }
 93    if (errnum < LIB_ERRNO_BASE) {
 94        strerror_r(errnum, buf, size);
 95    } else {
 96        int ix = errnum - LIB_ERRNO_BASE;
 97        if (errtbl[ix] != NULL) {
 98            strncpy(buf, errtbl[ix], size);
 99        } else {
100            snprintf(buf, size, "%s: %d", unknown_str, errnum);
101        }
102        buf[size-1] = '\0';
103    }
104    return buf;
105 }
106
107 #ifdef __cplusplus
108 }
109 #endif
110
```

err.h 文件参考

将系统的、库的、app 的错误码统一起来
```
#include <errno.h>
#include <stddef.h>
```
err.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



宏定义

- #define LIB_ERRNO_BASE 256
  系统错误码: 0 ~ 255，函数库错误码: 256 ~ 511

- #define LIB_ERRNO_END 512
- #define LIB_ERRNO_MAX_NUM (1024 - LIB_ERRNO_BASE)
  除去系统错误码，剩下(1024-256)个由函数库负责管理

- #define LIB_ERRNO_QUE_EMPTY (LIB_ERRNO_BASE + 0)
  队列空

- #define LIB_ERRNO_QUE_FULL (LIB_ERRNO_BASE + 1)
  队列满

- #define LIB_ERRNO_MEM_ALLOC (LIB_ERRNO_BASE + 2)
  malloc分配失败

- #define LIB_ERRNO_MBLK_SHORT (LIB_ERRNO_BASE + 3)
  内存池中的固定块大小低于想要分配的数据块

- #define LIB_ERRNO_BUF_SHORT (LIB_ERRNO_BASE + 4)
  缓存不足

- #define LIB_ERRNO_RES_LIMIT (LIB_ERRNO_BASE + 5)
  资源使用到达设定的上限

- #define LIB_ERRNO_SHORT_MPOOL (LIB_ERRNO_BASE + 6)
  内存池数据块耗尽

- #define LIB_ERRNO_NOT_EXIST (LIB_ERRNO_BASE + 7)
  目标不存在

函数

- int err_init (void)
  err模块初始化

- int err_add (int errnum, const char *str)
  注册一个错误码和它所对应的字符串

- char * err_string (int errnum, char *buf, size_t size)
  输出给定错误码所对应的字符串，线程安全，用法类似标准库函数strerror_r()

详细描述

将系统的、库的、app 的错误码统一起来

作者:

ln

```
模块初始化(err_init())完成后，需要先调用err_add()增加自定义的错误码和对应字符串，
之后就可以通过err_string()输出错误码所对应的字符串。

函数err_string()是线程安全的，和它用法类似的标准库函数是strerror_r()
```

在文件 err.h 中定义.

宏定义说明

#define LIB_ERRNO_END 512

app错误码: 512 ~ 1023

举例:

```
#define APP_ERRNO_XXX    (LIB_ERRNO_END + [0~511])
```

在文件 err.h 第 30 行定义.

函数说明

int err_add (int *errnum*, const char * *str*)

注册一个错误码和它所对应的字符串

参数:

errnum	错误码
str	错误码所对应的字符串，函数内部会使用strdup()对该字符串进行拷贝

返回值:

0	成功
-1	失败并设置errno

在文件 err.c 第 59 行定义.

```
60 {
61     char *p;
62     if (errnum < LIB_ERRNO_BASE ||
63         errnum > (LIB_ERRNO_MAX_NUM + LIB_ERRNO_BASE - 1) || str == NULL) {
64         errno = EINVAL;
65         return -1;
66     }
67     if ((p = strdup(str)) == NULL)
68         return -1;
69     int ix = errnum - LIB_ERRNO_BASE;
70     if (errtbl[ix] != NULL)
71         free(errtbl[ix]);
72     errtbl[ix] = p;
73     return 0;
74 }
```

这是这个函数的调用关系图:

int err_init (void)

err模块初始化

返回值:

0	成功
-1	失败并设置errno

在文件 err.c 第 33 行定义.

```
34 {
35     int ret = 0;
36     ret += err_add(LIB_ERRNO_QUE_EMPTY, "Queue empty");
37     ret += err_add(LIB_ERRNO_QUE_FULL, "Queue full");
38     ret += err_add(LIB_ERRNO_MEM_ALLOC, "Malloc fail");
39     ret += err_add(LIB_ERRNO_MBLK_SHORT, "Block size of memory-pool not enough");
40     ret += err_add(LIB_ERRNO_BUF_SHORT, "Local buffer not enough");
41     ret += err_add(LIB_ERRNO_RES_LIMIT, "Resources exceed limit");
42     ret += err_add(LIB_ERRNO_SHORT_MPOOL, "Short of memory-pool");
43     ret += err_add(LIB_ERRNO_NOT_EXIST, "Objectives not exist");
44     if (ret != 0)
45         return -1;
46     else
47         return 0;
48 }
```

函数调用图:

char* err_string (int *errnum*, char * *buf*, size_t *size*)

输出给定错误码所对应的字符串，线程安全，用法类似标准库函数strerror_r()

参数:

errnum	错误码
buf	字符串的输出buf，函数会确保输出的字符串含有结束符('\0')； 所以如果buf的size不足，输出的字符串会被截断。
size	输出buf的大小

返回值:

0	成功

-1	失败并设置errno

在文件 err.c 第 87 行定义.

```
 88 {
 89     if (errnum < 0 || buf == NULL || size < 1) {
 90         errno = EINVAL;
 91         return NULL;
 92     }
 93     if (errnum < LIB_ERRNO_BASE) {
 94         strerror_r(errnum, buf, size);
 95     } else {
 96         int ix = errnum - LIB_ERRNO_BASE;
 97         if (errtbl[ix] != NULL) {
 98             strncpy(buf, errtbl[ix], size);
 99         } else {
100             snprintf(buf, size, "%s: %d", unknown_str, errnum);
101         }
102         buf[size-1] = '\0';
103     }
104     return buf;
105 }
```

err.h

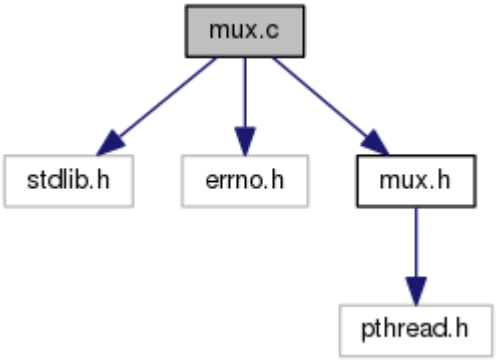
```
  1
 12 #ifndef __ERR_H__
 13 #define __ERR_H__
 14
 15 #include <errno.h>
 16 #include <stddef.h>
 17
 18 #ifdef __cplusplus
 19 extern "C" {
 20 #endif
 21
 22 #define LIB_ERRNO_BASE            256
 23
 30 #define LIB_ERRNO_END             512
 31 #define LIB_ERRNO_MAX_NUM          (1024 - LIB_ERRNO_BASE)
 32
 33 #define LIB_ERRNO_QUE_EMPTY        (LIB_ERRNO_BASE + 0)
 34 #define LIB_ERRNO_QUE_FULL         (LIB_ERRNO_BASE + 1)
 35 #define LIB_ERRNO_MEM_ALLOC        (LIB_ERRNO_BASE + 2)
 36 #define LIB_ERRNO_MBLK_SHORT       (LIB_ERRNO_BASE + 3)
 37 #define LIB_ERRNO_BUF_SHORT        (LIB_ERRNO_BASE + 4)
 38 #define LIB_ERRNO_RES_LIMIT        (LIB_ERRNO_BASE + 5)
 39 #define LIB_ERRNO_SHORT_MPOOL      (LIB_ERRNO_BASE + 6)
 40 #define LIB_ERRNO_NOT_EXIST        (LIB_ERRNO_BASE + 7)
 41
 42 extern int     err_init(void);
 43 extern int     err_add(int errnum, const char *str);
 44 extern char*   err_string(int errnum, char *buf, size_t size);
 45
 46 #ifdef __cplusplus
 47 }
 48 #endif
 49
 50 #endif
 51
```

mux.c 文件参考

创建并使用一个线程之间的、优先级继承的、可嵌套的互斥锁

```
#include <stdlib.h>
#include <errno.h>
#include "mux.h"
```

mux.c 的引用(Include)关系图:



函数

- int mux_init (mux_t *mux)
  初始化一个线程之间的、优先级继承的、可嵌套的互斥锁
- mux_t * mux_new (mux_t **mux)
  创建一个线程之间的、优先级继承的、可嵌套的互斥锁
- void mux_destroy (mux_t *mux)
  销毁互斥锁
- int mux_lock (mux_t *mux)
  加锁互斥锁
- int mux_unlock (mux_t *mux)
  解锁互斥锁

详细描述

创建并使用一个线程之间的、优先级继承的、可嵌套的互斥锁

作者:

ln

在文件 mux.c 中定义.

函数说明

void mux_destroy (mux_t * *mux*)

销毁互斥锁

参数:

mux	互斥锁指针

返回:

void

注意:

mux_destroy并不能free由mux_new返回的动态内存

示例:

```
mux_t *mux = NULL;
mux_new(&mux);
//...
mux_destroy(mux);
free(mux);
mux = NULL;
```

在文件 mux.c 第 94 行定义.

```
 95 {
 96     if (mux) {
 97         pthread_mutexattr_destroy(&mux->attr);
 98         pthread_mutex_destroy(&mux->mux);
 99     }
100 }
```

int mux_init (mux_t * *mux*)

初始化一个线程之间的、优先级继承的、可嵌套的互斥锁

参数:

mux	未初始化的互斥锁

返回值:

0	成功
-1	失败并设置errno

在文件 mux.c 第 23 行定义.

```
 24 {
 25     if (mux == NULL) {
 26         errno = EINVAL;
 27         return -1;
 28     }
 29 
 30     pthread_mutexattr_init(&mux->attr);
 31     if ((errno = pthread_mutexattr_setpshared(&mux->attr, PTHREAD_PROCESS_PRIVATE)) != 0)
 32         return -1;
 33     if ((errno = pthread_mutexattr_setprotocol(&mux->attr, PTHREAD_PRIO_INHERIT)) != 0)
 34         return -1;
 35     if ((errno = pthread_mutexattr_settype(&mux->attr, PTHREAD_MUTEX_RECURSIVE)) != 0)
 36         return -1;
 37 
 38     return pthread_mutex_init(&mux->mux, &mux->attr);
 39 }
```

这是这个函数的调用关系图:

int mux_lock (mux_t * *mux*)

加锁互斥锁

参数:

mux	互斥锁指针

返回值:

0	成功
!0	错误码

在文件 mux.c 第 110 行定义.

```
111 {
112     return pthread_mutex_lock(&mux->mux);
113 }
```

mux_t* mux_new (mux_t ** *mux*)

创建一个线程之间的、优先级继承的、可嵌套的互斥锁

参数:

mux	互斥锁指针的指针

返回:

返回新建的互斥锁，并将该互斥锁赋给*mux（如果mux不为NULL的话）

返回值:

!NULL	成功
NULL	失败并设置errno

示例:

```
mux_t *mux = mux_new(NULL);
```

或者

```
mux_t *mux = NULL;
mux_new(&mux);
```

注意:

返回的互斥锁需要free，mux_destroy并不能free互斥锁本身

在文件 mux.c 第 62 行定义.

```
63 {
64     mux_t *p = (mux_t *)malloc(sizeof(mux_t));
65     if (p && (mux_init(p) != 0)) {
66         free(p);
67         p = NULL;
68     }
69 
70     if (mux != NULL)
71         *mux = p;
72     return p;
73 }
```

函数调用图:

int mux_unlock (mux_t * *mux*)

解锁互斥锁

参数:

mux	互斥锁指针

返回值:

0	成功

!0	错误码

在文件 mux.c 第 123 行定义.

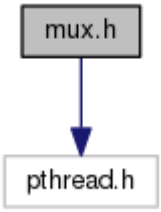
```
124 {
125     return pthread_mutex_unlock(&mux->mux);
126 }
```

mux.c

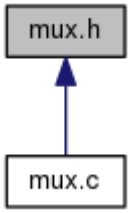
```
  1
  7 #include <stdlib.h>
  8 #include <errno.h>
  9 #include "mux.h"
 10
 11 #ifdef __cplusplus
 12 extern "C" {
 13 #endif
 14
 23 int mux_init(mux_t *mux)
 24 {
 25     if (mux == NULL) {
 26         errno = EINVAL;
 27         return -1;
 28     }
 29
 30     pthread_mutexattr_init(&mux->attr);
 31     if ((errno = pthread_mutexattr_setpshared(&mux->attr, PTHREAD_PROCESS_PRIVATE)) != 0)
 32         return -1;
 33     if ((errno = pthread_mutexattr_setprotocol(&mux->attr, PTHREAD_PRIO_INHERIT)) != 0)
 34         return -1;
 35     if ((errno = pthread_mutexattr_settype(&mux->attr, PTHREAD_MUTEX_RECURSIVE)) != 0)
 36         return -1;
 37
 38     return pthread_mutex_init(&mux->mux, &mux->attr);
 39 }
 40
 62 mux_t* mux_new(mux_t **mux)
 63 {
 64     mux_t *p = (mux_t *)malloc(sizeof(mux_t));
 65     if (p && (mux_init(p) != 0)) {
 66         free(p);
 67         p = NULL;
 68     }
 69
 70     if (mux != NULL)
 71         *mux = p;
 72     return p;
 73 }
 74
 94 void mux_destroy(mux_t *mux)
 95 {
 96     if (mux) {
 97         pthread_mutexattr_destroy(&mux->attr);
 98         pthread_mutex_destroy(&mux->mux);
 99     }
100 }
101
110 int mux_lock(mux_t *mux)
111 {
112     return pthread_mutex_lock(&mux->mux);
113 }
114
123 int mux_unlock(mux_t *mux)
124 {
125     return pthread_mutex_unlock(&mux->mux);
126 }
127
128 #ifdef   cplusplus
129 }
130 #endif
131
```

mux.h 文件参考

创建并使用一个线程之间的、优先级继承的、可嵌套的互斥锁
`#include <pthread.h>`
mux.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



类

- struct mux_t
  互斥锁类型定义

函数

- int mux_init (mux_t *mux)
  初始化一个线程之间的、优先级继承的、可嵌套的互斥锁
- mux_t * mux_new (mux_t **mux)
  创建一个线程之间的、优先级继承的、可嵌套的互斥锁
- void mux_destroy (mux_t *mux)
  销毁互斥锁
- int mux_lock (mux_t *mux)
  加锁互斥锁
- int mux_unlock (mux_t *mux)
  解锁互斥锁

详细描述

创建并使用一个线程之间的、优先级继承的、可嵌套的互斥锁

作者:

ln

在文件 mux.h 中定义.

函数说明

void mux_destroy (mux_t * *mux*)

销毁互斥锁

参数:

mux	互斥锁指针

返回:

void

注意:

mux_destroy并不能free由mux_new返回的动态内存

示例:

```
mux_t *mux = NULL;
mux_new(&mux);
//...
mux_destroy(mux);
free(mux);
mux = NULL;
```

在文件 mux.c 第 94 行定义.

```
 95 {
 96     if (mux) {
 97         pthread_mutexattr_destroy(&mux->attr);
 98         pthread_mutex_destroy(&mux->mux);
 99     }
100 }
```

int mux_init (mux_t * *mux*)

初始化一个线程之间的、优先级继承的、可嵌套的互斥锁

参数:

mux	未初始化的互斥锁

返回值:

0	成功
-1	失败并设置errno

在文件 mux.c 第 23 行定义.

```
 24 {
 25     if (mux == NULL) {
 26         errno = EINVAL;
 27         return -1;
 28     }
 29 
 30     pthread_mutexattr_init(&mux->attr);
 31     if ((errno = pthread_mutexattr_setpshared(&mux->attr, PTHREAD_PROCESS_PRIVATE)) != 0)
 32         return -1;
 33     if ((errno = pthread_mutexattr_setprotocol(&mux->attr, PTHREAD_PRIO_INHERIT)) != 0)
 34         return -1;
 35     if ((errno = pthread_mutexattr_settype(&mux->attr, PTHREAD_MUTEX_RECURSIVE)) != 0)
 36         return -1;
 37 
 38     return pthread_mutex_init(&mux->mux, &mux->attr);
 39 }
```

这是这个函数的调用关系图:

int mux_lock (mux_t * *mux*)

加锁互斥锁

参数:

mux	互斥锁指针

返回值:

0	成功
!0	错误码

在文件 mux.c 第 110 行定义.

```
111 {
112     return pthread_mutex_lock(&mux->mux);
113 }
```

mux_t* mux_new (mux_t ** *mux*)

创建一个线程之间的、优先级继承的、可嵌套的互斥锁

参数:

mux	互斥锁指针的指针

返回:

返回新建的互斥锁，并将该互斥锁赋给*mux（如果mux不为NULL的话）

返回值:

!NULL	成功
NULL	失败并设置errno

示例:

```
mux_t *mux = mux_new(NULL);
```

或者

```
mux_t *mux = NULL;
mux_new(&mux);
```

注意:

返回的互斥锁需要free，mux_destroy并不能free互斥锁本身

在文件 mux.c 第 62 行定义.

```
63 {
64     mux_t *p = (mux_t *)malloc(sizeof(mux_t));
65     if (p && (mux_init(p) != 0)) {
66         free(p);
67         p = NULL;
68     }
69
70     if (mux != NULL)
71         *mux = p;
72     return p;
73 }
```

函数调用图:

int mux_unlock (mux_t * *mux*)

解锁互斥锁

参数:

mux	互斥锁指针

返回值:

0	成功
!0	错误码

在文件 mux.c 第 123 行定义.

```
124 {
125     return pthread_mutex_unlock(&mux->mux);
126 }
```

mux.h

```
 1
 7 #ifndef   THR_MUX
 8 #define __THR_MUX__
 9
10 #include <pthread.h>
11
12 #ifdef __cplusplus
13 extern "C" {
14 #endif
15
19 typedef struct {
20    pthread_mutex_t    mux;
21    pthread mutexattr t attr;
22 } mux_t;
23
24 extern int     mux_init    (mux_t *mux);
25 extern mux_t*   mux_new     (mux_t **mux);
26 extern void    mux_destroy (mux_t *mux);
27
28 extern int     mux_lock    (mux_t *mux);
29 extern int     mux_unlock  (mux_t *mux);
30
31 #ifdef __cplusplus
32 }
33 #endif
34
35 #endif // __THR_MUX__
36
```

索引

INDEX