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:	
mux_t (互斥锁类	型定义	ζ)	4	

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
	ᄼᅥᅡᅥᅥᅥᅥᅥᄓᅜᄊᅅᅜᅁᅬ

<u>cstr.c</u> (提供方便的字符串操作)	5
err.c (将系统的、库的、app 的错误码统一起来)	20
err.h (将系统的、库的、app 的错误码统一起来)	24
mux.c (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)	28
mux.h (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)	32

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 <u>mux</u> linux互斥锁
- pthread_mutexattr_t <u>attr</u> linux互斥锁属性

详细描述

互斥锁类型定义

在文件 <u>mux.h</u> 第 <u>19</u> 行定义.

该结构体的文档由以下文件生成:

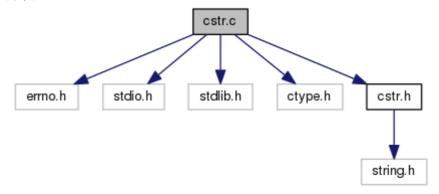
• mux.h

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 * <u>strlwr</u> (char *s) 字符串转为小写
- char * <u>strupr</u> (char *s) 字符串转为大写
- int <u>strstrip</u> (char *s) 删除给定字符串开始和结尾两处的所有空格
- int <u>bin2hex</u> (char *hex, const void *bin, size_t len)

 —进制转换为hex
- char * <u>abin2hex</u> (const void *bin, size_t len)二进制转换为hex,输出缓存由函数负责malloc
- int <u>hex2bin</u> (void *bin, const char *hex, size_t len)
 hex转换为二进制
- void * <u>ahex2bin</u> (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 {
       if (bin == NULL || len == 0) {
129
130
          errno = EINVAL;
131
          return NULL;
132
      }
133
      char *s = (char*) malloc((len * 2) + 1);
134
135
      if (s != NULL) {
          bin2hex (s, bin, len);
136
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 {
        if (hex == NULL) {
188
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 = \underline{\text{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 <u>cstr.c</u> 第 <u>102</u> 行定义.

```
103 {
        if (bin == NULL \mid \mid hex == NULL \mid \mid len == 0) {
104
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
        hex[i*2] = ' \setminus 0';
112
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
      size_t len2 = len;
164
      char buf[4] = \{0\};
165
166
     while (isxdigit(*hex) && len) {
167
         buf[0] = *hex++;
168
          buf[1] = *hex++;
169
          *((char *)bin) = (char)strtol(buf, NULL, 16);
          len--;
170
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:

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;
        for (size t i=0; i<num; \overline{i}++) {
2.44
            char *pin = (char *)in + i*section size;
char *pout = (char *)out + i*section_size;
size_t ss = 0;
245
246
247
            while (ss < section_size/2) {
   char c = pin[ss];</pre>
248
249
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 <u>cstr.c</u> 第 <u>25</u> 行定义.

```
26 {
      if (s == NULL) {
27
        errno = EINVAL;
28
29
         return NULL;
30
31
     char *p = s;
     while (*p) {
32
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 {
      char *last = NULL ;
68
69
      char *dest = s;
70
71
      if (s == NULL) {
         errno = EINVAL;
72
73
         return -1;
74
75
76
    last = s + strlen(s);
```

```
77
    while (isspace((int)*s) && *s)
78
        s++;
      while (last > s) {
79
80
      if (!isspace((int)*(last-1)))
81
           break ;
        last-- ;
82
83
84
     *last = (char)0;
85
     memmove(dest, s, last-s+1);
86
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) {
     *p = (char)toupper((int)(*p));
        p++;
56
57
58
     return s;
59 }
```

cstr.c

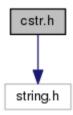
```
7 #include <errno.h>
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <ctype.h>
11 #include "cstr.h"
12
13 #ifdef __cplusplus
14 extern "C" {
15 #endif
16
25 char* strlwr(char *s)
26 {
27
      if (s == NULL) {
       errno = EINVAL;
28
         return NULL;
29
30
char *p = s;
32 while (*p) {
33 *p = (char)tolower((int)(*p));
     p++;
34
35
36
     return s;
37 }
38
47 char* strupr(char *s)
48 {
49
      if (s == NULL) {
      errno = EINVAL;
50
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 {
        char *last = NULL;
char *dest = s;
  68
  69
  70
  71
        if (s == NULL) {
  72
           errno = EINVAL;
           return -1;
  73
  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) {
           errno = EINVAL;
return -1;
 105
 106
 107
 108
        size_t i = 0;
        for (i = 0; i < len; i++) {
 109
           sprintf(&hex[i*2], "%02x", ((unsigned char *)bin)[i]);
 110
 111
 112
        hex[i*2] = ' \setminus 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
        char *s = (char*) malloc((len * 2) + 1);
 134
        if (s != NULL) {
 135
           bin2hex (s, bin, len);
 136
 137
        return s;
 138
 139 }
 140
 141
 157 int hex2bin(void *bin, const char *hex, size t len)
 158 {
 159
        if (bin == NULL \mid \mid hex == NULL \mid \mid len == 0) {
            errno = EINVAL;
 160
 161
           return -1;
 162
 163
 164
        size t len2 = len;
        \frac{1}{2} char buf[4] = {0};
 165
        while (isxdigit(*hex) && len) {
 166
 167
           buf[0] = *hex++;
 168
           buf[1] = *hex++;
 169
            *((char *)bin) = (char)strtol(buf, NULL, 16);
           len--;
 170
           bin = (char *)bin + 1;
 171
 172
 173
        return (int) (len2 - len);
 174 }
 175
 186 void* ahex2bin(const char *hex, size_t *bin_len)
 187 {
188 if (hex == NULL) {
```

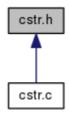
```
189
           errno = EINVAL;
190
           return NULL;
191
192
193
       size_t len = strlen(hex);
194
       unsigned char *b = (unsigned char*)malloc(len/2 + 1);
195
       if (b != NULL) {
196
           size_t ret = \frac{\text{hex2bin}}{\text{(b, hex, len/2 + 1)}};
197
           if (bin_len)
  *bin_len = ret;
198
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) {
           section_size = len;
232
233
234
235
       if (section size == 0 || out == NULL || in == NULL || len % section size) {
      errno = EINVAL;
236
           return -1;
237
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 \overline{\text{(size t i=0; i<num; i++)}} {
          char *pin = (char *)in + i*section size;
245
           char *pout = (char *)out + i*section_size;
246
           size_t ss = 0;
247
           while (ss < section size/2) {
248
              char c = pin[ss];
pout[ss] = pin[section_size - ss - 1];
249
250
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 <u>MAKE CSTR</u>(s) #s 构造字符串,如果s是宏,则结果为宏名本身
- #define <u>MAKE_CSTR(s) _MAKE_CSTR(s)</u> 构造字符串,如果s是宏,则结果为宏所表示的内容
- #define <u>CONCAT STRING(</u>l, r) l##r 连接字符串,如果l,r是宏,则结果为宏名本身
- #define <u>CONCAT_STRING(I, r) _CONCAT_STRING(I, r)</u> *连接字符串,如果I,r是宏,则结果为宏所表示的内容*

函数

- char * <u>strlwr</u> (char *s) *字符串转为小写*
- char * <u>strupr</u> (char *s) 字符串转为大写
- int <u>strstrip</u> (char *s) *删除给定字符串开始和结尾两处的所有空格*
- int <u>bin2hex</u> (char *hex, const void *bin, size_t len)二进制转换为hex
- char * <u>abin2hex</u> (const void *bin, size_t len)二进制转换为hex,输出缓存由函数负责malloc

- int <u>hex2bin</u> (void *bin, const char *hex, size_t len)
 hex转换为二进制
- void * <u>ahex2bin</u> (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)
 以字节为単位、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;
          return NULL;
131
132
      }
133
134
      char *s = (char*) malloc((len * 2) + 1);
       if (s != NULL) {
135
136
          bin2hex(s, bin, len);
137
138
       return s;
139 }
```

函数调用图:



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

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);
       if (b != NULL) {
195
196
           size_t ret = \underline{\text{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 <u>cstr.c</u> 第 <u>102</u> 行定义.

```
103 {
104
        if (bin == NULL \mid \mid hex == NULL \mid \mid len == 0) {
105
           errno = EINVAL;
            return -1;
106
107
108
        size t i = 0;
109
       for \overline{(i = 0; i < len; i++)} {
           sprintf(&hex[i*2], "%02x", ((unsigned char *)bin)[i]);
110
111
112
        hex[i*2] = ' \setminus 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) {
           errno = EINVAL;
160
           return -1;
161
       }
162
163
164
       size t len2 = len;
165
       char buf[4] = \{0\};
166
       while (isxdigit(*hex) && len) {
           buf[0] = *hex++;
buf[1] = *hex++;
167
168
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:

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) {
          section_size = len;
232
233
234
235
       if (section size == 0 || out == NULL || in == NULL || len % section size) {
236
          errno = EINVAL;
          return -1;
237
238
239
      if (len <= 1 || section_size <= 1) // no need to swap
240
241
          return 0;
242
243
       size_t num = len/section_size;
      for \overline{\text{(size t i=0; i<num; i++)}} {
244
2.45
          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) {
             char c = pin[ss];
249
             pout[ss] = pin[section_size - ss - 1];
250
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) {
          errno = EINVAL;
return NULL;
28
29
30
      char *p = s;
while (*p) {
31
32
        *p = (char)tolower((int)(*p));
33
          p++;
34
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:

1 /	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 <u>cstr.c</u> 第 <u>66</u> 行定义.

```
67 {
      char *last = NULL ;
      char *dest = s;
69
70
71
      if (s == NULL) {
       errno = EINVAL;
return -1;
72
73
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 ;
         last-- ;
82
83
      *last = (char)0;
84
85
      memmove(dest, s, last-s+1);
return last - s;
86
87
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 {
      if (s == NULL) {
49
50
         errno = EINVAL;
51
         return NULL;
52
     char *p = s;
53
     while (*p) {
54
      *p = (char) toupper((int)(*p));
55
56
        p++;
57
      }
58
      return s;
59 }
```

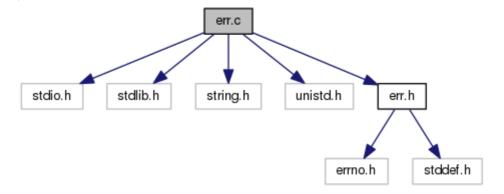
cstr.h

```
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(1, r) _CONCAT_STRING(1, r)
21
22 extern char* strlwr
23 extern char* strupr
                                   (char *s);
                                   (char *s);
24
25 extern int
                 strstrip
                                   (char *s);
26
                              (char *hex, const void *bin, size_t len);
                 bin2hex
abin2hex
27 extern int
28 extern char*
                                  (const void *bin, size_t len);
29
                 hex2bin
ahex2bin
                              (void *bin, const char *hex, size_t len);
(const char *hex, size_t *bin_len);
30 extern int
31 extern void*
32
33 extern int
                                 (void *out, const void *in, size_t len, size_t section_size);
                 memswap
34
35 #ifdef __cplusplus
36 }
37 #endif
38
39 #endif /* __C_STRING_H__ */
```

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 <u>err_init</u> (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 <u>unknown_str</u> [] = "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 ||
         errnum > (LIB ERRNO MAX NUM + LIB ERRNO BASE - 1) || str == NULL) {
63
         errno = EINVAL;
64
         return -1;
65
66
67
     if ((p = strdup(str)) == NULL)
         return -1;
68
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;
           ret += err add (LIB ERRNO QUE EMPTY, "Queue empty");
ret += err add (LIB ERRNO QUE FULL, "Queue full");
ret += err add (LIB ERRNO MEM ALLOC, "Malloc fail");
36
37
38
           ret += err add(LIB ERRNO MBLK SHORT, "Block size of memory-pool not enough");
ret += err add(LIB ERRNO BUF SHORT, "Local buffer not enough");
ret += err add(LIB ERRNO RES LIMIT, "Resources exceed limit");
39
40
41
            ret += err add (LIB ERRNO SHORT MPOOL, "Short of memory-pool");
ret += err add (LIB ERRNO NOT EXIST, "Objectives not exist");
42
43
44
         if (ret != 0)
45
                   return -1;
            else
46
47
                  return 0;
48 }
```

函数调用图:



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

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 {
       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)
             strncpy(buf, errtbl[ix], size);
98
99
          } else {
              snprintf(buf, size, "%s: %d", unknown str, errnum);
100
101
102
          buf[size-1] = ' \setminus 0';
103
104
       return buf;
105 }
```

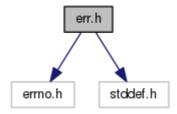
err.c

```
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <string.h>
15 #include <unistd.h>
16 #include "err.h"
18 #ifdef __cplusplus
19 extern "C" {
20 #endif
21
23 const char unknown str[] = "Unknown error";
2.4
26 static char* errtbl[LIB ERRNO MAX NUM] = {0};
27
33 int err init (void)
34 {
        int ret = 0;
3.5
      ret += err add(LIB ERRNO QUE EMPTY, "Queue empty");
ret += err add(LIB ERRNO QUE FULL, "Queue full");
ret += err add(LIB ERRNO MEM ALLOC, "Malloc fail");
ret += err add(LIB ERRNO MBLK SHORT, "Block size of memory-pool not enough");
ret += err add(LIB ERRNO BUF SHORT, "Local buffer not enough");
ret += err add(LIB ERRNO RES LIMIT, "Resources exceed limit");
36
37
38
39
40
41
       ret += err add(LIB ERRNO SHORT MPOOL, "Short of memory-pool");
42
       ret += err add (LIB ERRNO NOT EXIST, "Objectives not exist");
43
      if (ret != 0)
44
45
             return -1;
46
47
            return 0;
48 }
49
59 int err add(int errnum, const char *str)
60 {
        char *p;
61
62
        if (errnum < <u>LIB ERRNO BASE</u> ||
63
            errnum > (LIB ERRNO MAX NUM + LIB ERRNO BASE - 1) || str == NULL) {
64
             errno = EINVAL;
65
           return -1;
```

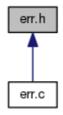
```
66
       if ((p = strdup(str)) == NULL)
  67
  68
         return -1;
      int ix = errnum - LIB ERRNO BASE;
if (errtbl[ix] != NULL)
  69
  70
  71
           free(errtbl[ix]);
  72
       errtbl[ix] = p;
  73
       return 0;
  74 }
  75
  87 char* err_string(int errnum, char *buf, size_t size)
  88 {
  89
        if (errnum < 0 || buf == NULL || size < 1) {
           errno = EINVAL;
return NULL;
  90
  91
      } if (errnum < <u>LIB_ERRNO_BASE</u>) {
  92
  93
  94
         strerror_r(errnum, buf, size);
       } else {
  95
        int ix = errnum - LIB ERRNO BASE;
if (errtbl[ix] != NULL) {
  96
  97
             strncpy(buf, errtbl[ix], size);
  98
 99
           } else {
              snprintf(buf, size, "%s: %d", unknown str, errnum);
 100
 101
 102
           buf[size-1] = ' \setminus 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 <u>LIB_ERRNO_BASE</u> 256
 系统错误码: 0 ~ 255, 函数库错误码: 256 ~ 511
- #define LIB ERRNO END 512
- #define <u>LIB_ERRNO_MAX_NUM</u> (1024 <u>LIB_ERRNO_BASE</u>) 除去系统错误码,剩下(1024-256)个由函数库负责管理
- #define <u>LIB ERRNO QUE EMPTY</u> (<u>LIB ERRNO BASE</u> + 0)
 以列空
- #define <u>LIB_ERRNO_QUE_FULL</u> (<u>LIB_ERRNO_BASE</u> + 1)
 以列满
- #define <u>LIB ERRNO MEM ALLOC</u> (<u>LIB ERRNO BASE</u> + 2) *malloc 分配失败*
- #define <u>LIB ERRNO MBLK SHORT</u> (<u>LIB ERRNO BASE</u> + 3) 内存池中的固定块大小低于想要分配的数据块
- #define <u>LIB ERRNO BUF SHORT</u> (<u>LIB ERRNO BASE</u> + 4) *缓存不足*
- #define <u>LIB ERRNO RES LIMIT</u> (<u>LIB ERRNO BASE</u> + 5)
 资源使用到达设定的上限
- #define <u>LIB_ERRNO_SHORT_MPOOL</u> (<u>LIB_ERRNO_BASE</u> + 6) 内存池数据块耗尽

● #define <u>LIB_ERRNO_NOT_EXIST</u> (<u>LIB_ERRNO_BASE</u> + 7) 目标不存在

函数

- int <u>err_init</u> (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:

1n

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 ||
         errnum > (LIB ERRNO MAX NUM + LIB ERRNO BASE - 1) || str == NULL) {
         errno = EINVAL;
64
         return -1;
65
66
     if ((p = strdup(str)) == NULL)
68
         return -1;
      int ix = errnum - LIB ERRNO BASE;
69
     if (errtbl[ix] != NULL)
70
71
         free(errtbl[ix]);
72
      errtbl[ix] = p;
7.3
      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;
         ret += err add(LIB ERRNO QUE EMPTY, "Queue empty");
ret += err add(LIB ERRNO QUE FULL, "Queue full");
ret += err add(LIB ERRNO MEM ALLOC, "Malloc fail");
36
37
38
         ret += err add(LIB ERRNO MBLK SHORT, "Block size of memory-pool not enough");
ret += err add(LIB ERRNO BUF SHORT, "Local buffer not enough");
ret += err add(LIB ERRNO RES LIMIT, "Resources exceed limit");
39
40
41
         ret += err add (LIB ERRNO SHORT MPOOL, "Short of memory-pool");
42
43
          ret += err add (LIB ERRNO NOT EXIST, "Objectives not exist");
         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	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 <u>err.c</u> 第 <u>87</u> 行定义.

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

err.h

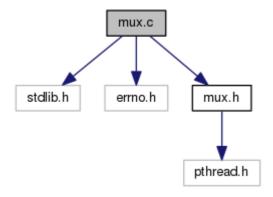
```
12 #ifndef __ERR_H_
13 #define __ERR_H_
14
15 #include <errno.h>
16 #include <stddef.h>
18 #ifdef __cplusplus
19 extern "C" {
20 #endif
21
22 #define LIB ERRNO BASE
23
30 #define LIB_ERRNO_END
                                           512
                                            (1024 - LIB_ERRNO_BASE)
31 #define LIB_ERRNO_MAX_NUM
                                           (LIB_ERRNO_BASE + 0)
(LIB ERRNO BASE + 1)
33 #define LIB ERRNO QUE EMPTY
34 #define LIB ERRNO QUE FULL
35 #define LIB ERRNO MEM ALLOC
                                            (LIB ERRNO BASE + 2)
                                             (LIB_ERRNO_BASE + 3)
(LIB_ERRNO_BASE + 4)
36 #define LIB ERRNO MBLK SHORT
37 #define LIB ERRNO BUF SHORT
38 #define LIB_ERRNO_RES_LIMIT
39 #define LIB_ERRNO_SHORT_MPOOL
                                            (LIB_ERRNO_BASE + 5)
                                             (LIB ERRNO BASE + 6)
40 #define LIB ERRNO NOT EXIST
                                           (LIB_ERRNO_BASE + 7)
41
                   err init (void);
42 extern int
                    err add(int errnum, const char *str);
err string(int errnum, char *buf, size_t size);
43 extern int
44 extern char*
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 <u>mux init</u> (<u>mux t</u>*mux) 初始化一个线程之间的、优先级继承的、可嵌套的互斥锁
- <u>mux t</u> * <u>mux new</u> (<u>mux t</u> **mux)
 创建一个线程之间的、优先级继承的、可嵌套的互斥锁
- void <u>mux_destroy</u> (<u>mux_t</u> *mux) *销毁互斥锁*
- int <u>mux lock</u> (<u>mux t</u>*mux) 加锁互斥锁
- int <u>mux_unlock</u> (<u>mux_t</u> *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)

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 <u>mux.c</u> 第 23 行定义.

```
24 {
      if (mux == NULL) {
25
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)
         return -1;
34
      if ((errno = pthread_mutexattr_settype(&mux->attr, PTHREAD_MUTEX_RECURSIVE)) != 0)
35
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 <u>mux.c</u> 第 <u>110</u> 行定义.

```
111 {
    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 <u>mux.c</u> 第 <u>62</u> 行定义.

```
63 {
         \underline{\text{mux t}} *p = (\underline{\text{mux t}} *) \text{ malloc (size of } (\underline{\text{mux t}}));
64
         if (p && (<u>mux init</u>(p) != 0)) {
65
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	
ì	返回值:		
	0	成功	

```
在文件 <u>mux.c</u> 第 <u>123</u> 行定义.
```

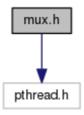
```
124 {
125 return pthread_mutex_unlock(&mux->mux);
126 }
```

mux.c

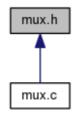
```
7 #include <stdlib.h>
  8 #include <errno.h>
  9 #include "mux.h"
 10
 11 #ifdef __cplusplus
12 extern "C" {
 13 #endif
 14
 23 int mux init(mux t *mux)
 24 {
 25
        if (mux == NULL) {
 26
          errno = EINVAL;
 27
           return -1;
 2.8
 29
 30
       pthread mutexattr init(&mux->attr);
       if ((errno = pthread_mutexattr_setpshared(&mux->attr, PTHREAD_PROCESS_PRIVATE)) != 0)
 31
 32
           return -1;
      if ((errno = pthread_mutexattr_setprotocol(&mux->attr, PTHREAD_PRIO_INHERIT)) != 0)
 33
          return -1;
 34
 35
       if ((errno = pthread mutexattr settype(&mux->attr, PTHREAD MUTEX RECURSIVE)) != 0)
           return -1;
 36
 37
 38
        return pthread_mutex_init(&mux->mux, &mux->attr);
 39 }
 40
 62 mux t* mux new(mux t **mux)
 63 {
 64
        \underline{mux \ t} \ *p = (\underline{mux \ t} \ *) malloc(sizeof(\underline{mux \ t}));
        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 {
        return pthread_mutex_lock(&mux->mux);
112
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 <u>mux t</u> 互斥锁类型定义

函数

- int <u>mux_init</u> (<u>mux_t</u> *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 <u>mux_destroy</u> (<u>mux_t</u>*mux) 销毁互斥锁
- int <u>mux_lock</u> (<u>mux_t</u> *mux)
 加锁互斥锁
- int <u>mux_unlock</u> (<u>mux_t</u>*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 <u>mux.c</u> 第 23 行定义.

```
24 {
25
      if (mux == NULL) {
26
         errno = EINVAL;
27
         return -1;
28
29
30
      pthread_mutexattr_init(&mux->attr);
      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 <u>mux.c</u> 第 <u>110</u> 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 <u>mux.c</u> 第 <u>62</u> 行定义.

```
63 {
        \frac{\text{mux t}}{\text{if (p && (mux init(p) != 0)) }} *p = (mux t *) malloc(sizeof(mux t));
64
65
             free(p);
             p = NULL;
67
68
69
        if (mux != NULL)
70
             *mux = p;
71
        return p;
72
73 }
```

函数调用图:



int mux_unlock (mux t * mux)

参数:

mux	互斥锁指针
-----	-------

返回值:

0	成功	
!0	错误码	

在文件 <u>mux.c</u> 第 <u>123</u> 行定义.

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

mux.h

```
7 #ifndef THR MUX
8 #define __THR_MUX_
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 } <u>mux_t</u>;
23
27
28 extern int \underline{\text{mux lock}} (\underline{\text{mux t}} *mux); 29 extern int \underline{\text{mux unlock}} (\underline{\text{mux t}} *mux);
30
31 #ifdef __cplusplus
32 }
33 #endif
35 #endif // ___THR_MUX___
36
```

索引

INDEX