YUNOS

YDK API

Doc. No. AP-00001-01-16

Rev 0.1

Release Date

Alibaba Confidential

By：

Revision History

Revision No.	Draft/Changes	Date	Author
0.1		2016.3.3	

Alibaba Confidential

 Copyright© 2016 Alibaba

Alibaba Confidential

Copyright© 2016 Alibaba

Alibaba Confidential

 Copyright© 2016 Alibaba

Alibaba Confidential

 Copyright© 2016 Alibaba

Alibaba Confidential

 Copyright© 2016 Alibaba

Alibaba Confidential

 Copyright© 2016 Alibaba

Alibaba Confidential

 Copyright© 2016 Alibaba

Alibaba Confidential

 Copyright© 2016 Alibaba

名词	解析
YoC	YunOS on Chip
YDK	YoC Development Kit
UART	串口设置
TIMER	时钟设置
PWM	脉宽调制设置
MTD	块设备设置
SYSTEM	系统设置
NETLIB	网络设置
GPIO	输入输出设置
OTA	系统更新设置
WDOG	定时器设置
DES	DES 加密设置
AES	AES 加解密设置
I2C	双向两线连续总线设置
Task Control Interfaces	任务控制接口设置
Task Scheduling Interfaces	任务调度接口设置
Task Switching Interfaces	任务切换接口设置
Named Message Queue Interfaces	信息队列接口设置
Counting Semaphore Interfaces	信号量接口设置
Clocks and Timers	时钟和定时器设置
Signal Interfaces	信号接口设置
Pthread Interfaces	线程接口设置
FIle System Interfaces	文件系统接口设置
Driver Operations	驱动操作
Directory Operations	目录操作
UNIX Standard Operations	UNIX 标准操作
StandardI/0	标准输入输出
Standard String Operations	标准字符串设置
Pipes and FIFOs	管道和 FIFO 操作
Network Interfaces	网络接口设置

Alibaba Confidential

Copyright© 2016 Alibaba

1 IDJS

1.1 Module Init

Function Prototype:

```
#include <nuttx/module.h>
coreapp_entry(IDJS_MOUDLE_INIT);
typedef void (*IDJS_MOUDLE_INIT)(void);
```

Description:

The module init function registered to the YunOS will be run when system start.

The type flow:

typedef void (*IDJS_MOUDLE_INIT)(void);

For a module init function:

void xxx_module_init()；

First in the C file:

1 add : #include <nuttx/module.h>

2 add : coreapp_entry(xxx_module_init);

Second int the Makefile

Add:

```
$(BIN): $(OBJS)
$(CROSSDEV)ld -r -o $(BIN) $(OBJS)
install:
```

1.2 Action

Function Prototype:

```
#include “idjs_action.h”
void idjs_action_register(const char *action_name,
                          YOC_ACTION_INFO * func_info);
```

Description:

Action register API

Input Parameters:

action_name[in]: name of the action to beregister

it should flow the type like:

Alibaba Confidential

 Copyright© 2016 Alibaba

module_name.action.func_name

func_info[in]: function info

its struct is:

```
typedef struct _yoc_action_is_info{
    uint8      type;
    const char* para[YOC_ACTION_PARA_NUM_MAX];
    YOC_ACTION_FUNC action;
}YOC_ACTION_INFO;
```

para: the parameter name array of the function，MAX array number is 4;

type: the function protype:

```
typedef enum {
        YOC_ACTION_TYPE_IS,
        YOC_ACTION_TYPE_I,
        YOC_ACTION_TYPE_II
}YOC_ACTION_TYPE;
```

The struct of YOC_ACTION_INFO is:

```
typedef union {
    YOC_ACTION_FUNC_IS    action_func_is;
    YOC_ACTION_FUNC_I     action_func_i;
    YOC_ACTION_FUNC_II    action_func_ii;
}YOC_ACTION_FUNC;
```

The name rule : YOC_ACTION_INFO + ii. The first I is return type, the second I is type of the first parameter，

Ex.:

YOC_ACTION_FUNC_IS

represents:

int action_is(const char*)；

Example:

```
YOC_ACTION_INFO* ptmp = &tmp;
ptmp->type = YOC_ACTION_TYPE_IS;
ptmp->para[0] = "url";
ptmp->action.action_func_is = yoc_play;
idjs_action_register("audioplayer.action.play",ptmp);
```

1.3 Event

1.3.1 idjs_event_register

Alibaba Confidential

 Copyright© 2016 Alibaba

Function Prototype:

#include “idjs_event.h”
void idjs_event_register(yoc_event_cb cb, void* cb_para);

Description:

event register API

Input Parameters:

cb[in]: the event call back function
typedef void (*yoc_event_cb)(struct input_event *event, void *cb_para);
cb_para[in]: the para of call back function

1.3.2 idjs_report_event

Function Prototype:

#include “idjs_event.h”

void idjs_report_event(char* module_name, char* event_name,
char* event_info, int len);

Description:

event report API

Input Parameters:

module_name[in]: module name of the event;
module_name[in]: name of the event
event_info[in]: the report information
len[in]: length of report information

Alibaba Confidential

 Copyright© 2016 Alibaba

2 Base64

2.1 base64_encode

Function Prototype:

```
#include <apps/netutils/base64.h>
unsigned char *base64_encode(const unsigned char *src, size_t len,
                             unsigned char *dst, size_t *out_len);
```

Description:

Base64 encode

Input Parameters:

src[in]: data to be encoded
len[in]: data length
dst[out]: encoded data
out_len[out]: pointer to output length variable, or NULL if not used

Returned Value:

Allocated buffer of out_len bytes of encoded data, or NULL on failure

2.2 base64_decode

Function Prototype:

```
#include <apps/netutils/base64.h>
unsigned char *base64_decode(const unsigned char *src, size_t len,
                             unsigned char *dst, size_t *out_len);
```

Description:

Base64 decode

Input Parameters:

src[in]: data to be decoded
len[in]: length of the data to be decoded
dst[out]: decoded data
out_len[out]: pointer to output length variable

Returned Value:

Alibaba Confidential

 Copyright© 2016 Alibaba

Allocated buffer of out_len bytes of decoded data, or NULL on failure

2. 3 base64w_encode

Refer to base64w_encode

2. 4 base64w_decode

Refer to base64w_decode

Alibaba Confidential

 Copyright© 2016 Alibaba

3 cJSON

3.1 cJSON_InitHooks

Function Prototype:

```
#include < netutils/cJSON.h >
void cJSON_InitHooks(cJSON_Hooks* hooks);
```

3.2 cJSON_Parse

Function Prototype:

```
#include < netutils/cJSON.h >
cJSON *cJSON_Parse(const char *value);
```

3.3 cJSON_Print

Function Prototype:

```
#include < netutils/cJSON.h >
char *cJSON_Print(cJSON *item);
```

3.4 cJSON_PrintUnformatted

Function Prototype:

```
#include < netutils/cJSON.h >
char *cJSON_PrintUnformatted(cJSON *item);
```

3.5 cJSON_Delete

Function Prototype:

```
#include < netutils/cJSON.h >
void cJSON_Delete(cJSON *c);
```

3.6 cJSON_GetArraySize

Function Prototype:

```
#include < netutils/cJSON.h >
```

Alibaba Confidential

 Copyright© 2016 Alibaba

int cJSON_GetArraySize(cJSON *array);

3.7 cJSON_GetArrayItem

Function Prototype:

#include < netutils/cJSON.h >
cJSON *cJSON_GetArrayItem(cJSON *array, int item);

3.8 cJSON_GetObjectItem

Function Prototype:

#include < netutils/cJSON.h >
cJSON *cJSON_GetObjectItem(cJSON *object, const char *string);

3.9 cJSON_GetErrorPtr

Function Prototype:

#include < netutils/cJSON.h >
const char *cJSON_GetErrorPtr(void);

3.10 cJSON_CreateNull

Function Prototype:

#include < netutils/cJSON.h >
cJSON *cJSON_CreateNull(void);

3.11 cJSON_CreateTrue

Function Prototype:

#include < netutils/cJSON.h >
cJSON *cJSON_CreateTrue(void);

3.12 cJSON_CreateFalse

Function Prototype:

#include < netutils/cJSON.h >
cJSON *cJSON_CreateFalse(void);

Alibaba Confidential

 Copyright© 2016 Alibaba

3.13 cJSON_CreateBool

Function Prototype:

```
#include < netutils/cJSON.h >
cJSON *cJSON_CreateBool(int b);
```

3.14 cJSON_CreateNumber

Function Prototype:

```
#include < netutils/cJSON.h >
cJSON *cJSON_CreateNumber(double num);
```

3.15 cJSON_CreateString

Function Prototype:

```
#include < netutils/cJSON.h >
cJSON *cJSON_CreateString(const char *string);
```

3.16 cJSON_CreateArray

Function Prototype:

```
#include < netutils/cJSON.h >
cJSON *cJSON_CreateArray(void);
```

3.17 cJSON_CreateObject

Function Prototype:

```
#include < netutils/cJSON.h >
cJSON *cJSON_CreateObject(void);
```

3.18 cJSON_CreateIntArray

Function Prototype:

```
#include < netutils/cJSON.h >
cJSON *cJSON_CreateIntArray(const int *numbers, int count);
```

3.19 cJSON_CreateFloatArray

Alibaba Confidential

 Copyright© 2016 Alibaba

Function Prototype:

```
#include < netutils/cJSON.h >
cJSON *cJSON_CreateFloatArray(const float *numbers, int count);
```

3.20 cJSON_CreateDoubleArray

Function Prototype:

```
#include < netutils/cJSON.h >
cJSON *cJSON_CreateDoubleArray(const double *numbers, int count);
```

3.21 cJSON_CreateStringArray

Function Prototype:

```
#include < netutils/cJSON.h >
cJSON *cJSON_CreateStringArray(const char **strings, int count);
```

3.22 cJSON_AddItemToArray

Function Prototype:

```
#include < netutils/cJSON.h >
void cJSON_AddItemToArray(cJSON *array, cJSON *item);
```

3.23 cJSON_AddItemToObject

Function Prototype:

```
#include < netutils/cJSON.h >
void cJSON_AddItemToObject(cJSON *object, const char *string,
                           cJSON *item);
```

3.24 cJSON_AddItemReferenceToArray

Function Prototype:

```
#include < netutils/cJSON.h >
void cJSON_AddItemReferenceToArray(cJSON *array, cJSON *item);
```

3.25 cJSON_AddItemReferenceToObject

Function Prototype:

Alibaba Confidential

 Copyright© 2016 Alibaba

```
#include < netutils/cJSON.h >
void cJSON_AddItemReferenceToObject(cJSON *object, const char *string,
                                    cJSON *item);
```

3.26 cJSON_DetachItemFromArray

Function Prototype:

```
#include < netutils/cJSON.h >
cJSON *cJSON_DetachItemFromArray(cJSON *array, int which);
```

3.27 cJSON_DeleteItemFromArray

Function Prototype:

```
#include < netutils/cJSON.h >
void cJSON_DeleteItemFromArray(cJSON *array, int which);
```

3.28 cJSON_DetachItemFromObject

Function Prototype:

```
#include < netutils/cJSON.h >
cJSON *cJSON_DetachItemFromObject(cJSON *object, const char *string);
```

3.29 cJSON_DeleteItemFromObject

Function Prototype:

```
#include < netutils/cJSON.h >
void cJSON_DeleteItemFromObject(cJSON *object, const char *string);
```

3.30 cJSON_ReplaceItemInArray

Function Prototype:

```
#include < netutils/cJSON.h >
void cJSON_ReplaceItemInArray(cJSON *array, int which,
                              cJSON *newitem);
```

3.31 cJSON_ReplaceItemInObject

Function Prototype:

```
#include < netutils/cJSON.h >
```

Alibaba Confidential

 Copyright© 2016 Alibaba

```
void cJSON_ReplaceItemInObject(cJSON *object, const char *string,
                               cJSON *newitem);
```

Alibaba Confidential

 Copyright© 2016 Alibaba

4 Device

4.1 UART

4.1.1 open

Function Prototype:

```
#include <fcntl.h>
int open(const char *path, int oflags, ...);
```

Description:

Open UART

Input Parameters:

path[in]: such as: /dev/ttyS0
oflags[in]: O_RDONLY, O_WRONLY, O_RDWR

Returned Value:

>=0: device file handle
-1: open error

4.1.2 close

Function Prototype:

```
#include <unistd.h>
int close(int fd);
```

Description:

Close UART

Input Parameters:

fd[in]: device file handle

Returned Value:

0: success
-1: error

4.1.3 write

Alibaba Confidential

 Copyright© 2016 Alibaba

Function Prototype:

#include <unistd.h>
ssize_t write(int fd, FAR const void *buf, size_t nbytes);

Description:

Write data

Input Parameters:

fd[in]: device file handle
buf[in]: data to be write
nbytes[in]: bytes of data

Returned Value:

>=0: writed bytes
-1: error

4.1.4 read

Function Prototype:

#include <unistd.h>
ssize_t read(int fd, FAR void *buf, size_t nbytes);

Description:

Read data

Input Parameters:

fd[in]: device file handle
buf[out]: buf to save data
nbytes[in]: bytes of data to be read

Returned Value:

>0: success
Other: error

4.2 TIMER

4.2.1 open

Function Prototype:

#include <fcntl.h>

Alibaba Confidential

 Copyright© 2016 Alibaba

int open(const char *path, int oflags, ...);

Description:

Open timer device

Input Parameters:

path[in]: such as: /dev/timer0
oflags[in]: O_RDONLY, O_WRONLY, O_RDWR

Returned Value:

>=0: device file handle
-1: open error

4.2.2 close

Function Prototype:

#include <unistd.h>
int close(int fd);

Description:

Close timer device

Input Parameters:

fd[in]: device file handle

Returned Value:

0: success
-1: error

4.2.3 ioctl

Function Prototype:

#include <sys/ioctl.h>
int ioctl(int fd, int req, ...);

Description:

Timer control interface

Input Parameters:

fd[in]: device file handle
req[in]: command

Alibaba Confidential

 Copyright© 2016 Alibaba

TCIOC_START
TCIOC_STOP
TCIOC_GETSTATUS
TCIOC_SETTIMEOUT
TCIOC_SETHANDLER

Returned Value:

0: success
-1: error

4.3 PWM

4.3.1 open

Function Prototype:

```
#include <fcntl.h>
int open(const char *path, int oflags, ...);
```

Description:

Open PWM

Input Parameters:

path[in]: such as: /dev/pwm0
oflags[in]: O_RDONLY, O_WRONLY, O_RDWR

Returned Value:

>=0: device file handle
-1: open error

4.3.2 close

Function Prototype:

```
#include <unistd.h>
int close(int fd);
```

Description:

Close PWM

Input Parameters:

fd[in]: device file handle

Alibaba Confidential

 Copyright© 2016 Alibaba

Returned Value:

0: success
-1: error

4.3.3 ioctl

Function Prototype:

```
#include <sys/ioctl.h>
int ioctl(int fd, int req, unsigned long arg);
```

Description:

PWM control interface

Input Parameters:

fd[in]: device file handle
req[in]: command
PWMIOC_SETCHARACTERISTICS
PWMIOC_GETCHARACTERISTICS
PWMIOC_START
PWMIOC_STOP
arg[in]:

Returned Value:

>=0: success
-1: error

4.4 MTD

4.4.1 open_blockdriver

Function Prototype:

```
#include <fs.h>
int open_blockdriver(FAR const char *pathname, int mountflags,
                     FAR struct inode **ppinode);
```

Description:

Open block device

Input Parameters:

pathname[in]: such as: /dev/mtdblock0

Alibaba Confidential

 Copyright© 2016 Alibaba

mouthflags[in]: O_WRONLY, O_RDWR
ppinode[out]: the inode of the block device

Returned Value:

>=0: file handle
EINVAL: pathname or pinode is NULL
ENOENT: device not exist
ENOTBLK: the device is not a block device
EACCESS: the device can't be writed although it is setted as O_RDWR

4.4.2 close_blockdriver

Function Prototype:

```
#include <fs.h>
int close_blockdriver(FAR struct inode *inode);
```

Description:

Close block device

Input Parameters:

Inode[in]: the inode of the block device

Returned Value:

0: success
EINVAL: inode is NULL
ENOTBLK: the device is not a block device

4.4.3 bread

Function Prototype:

```
#include <mtd.h>
ssize_t bread(FAR struct inode *inode, unsigned char *buffer,
              size_t start_sector, unsigned int nsectors);
```

Description:

Read data from given sector

Input Parameters:

inode: the inode of the block device
start_sector: the start sector of the device

Alibaba Confidential

 Copyright© 2016 Alibaba

nblocks: blocks
buffer: data buffer

Returned Value:

0: the end of the file or no data to be read
>0: the bytes of readed data
-1: error

4.4.4 bwrite

Function Prototype:

```
#include <mtd.h>
ssize_t bwrite(FAR struct inode *inode, const unsigned char *buffer,
               size_t start_sector, unsigned int nsectors) ;
```

Description:

将数据写入块设备的特定扇区

Input Parameters:

inode: 块设备节点
start_sector: 起始扇区
nblocks: 扇区个数
buffer: 数据缓存区

Returned Value:

0: 没有内容可以写
>0: 写入的字节数
-1 则为失败

4.4.5 ioctl

Function Prototype:

```
#include <mtd.h>
int ioctl(FAR struct inode *inode, int cmd, unsigned long arg);
```

Description:

块设备控制接口，是一个非标准的类 unix 接口

Input Parameters:

inode: 块设备节点

Alibaba Confidential

 Copyright© 2016 Alibaba

cmd: 命令
MTDIOC_GEOMETRY: 获取块设备的容量和物理特性
MTDIOC_XIPBASE: 把执行地址转换成物理地址
MTDIOC_BULKERASE: 擦处整块设备
arg: 参数

Returned Value:

>=0:成功执行
-1 : 执行失败

4.5 SYSTEM

4.5.1 yoc_sys_conf_set

Function Prototype:

```
#include <configs/sc6138a-xj-1/src/sys_conf.h>
int32_t yoc_sys_conf_set(const char *section, const char *key,
                         int32_t value_format, void *input);
```

Description:

将配置写入系统配置文件

Input Parameters:

const char *section: 配置存储的字段
const char *key: 配置存储的键
int32_t value_format: 写入数据的格式定义如下
FORMAT_INT 0
FORMAT_STRING 1
void *input: 写入的数据指针

Returned Value:

0 或错误号

4.5.2 yoc_sys_conf_get

Function Prototype:

```
#include <configs/sc6138a-xj-1/src/sys_conf.h>
int32_t yoc_sys_conf_get(const char *section, const char *key,
                         int32_t max_len, int32_t value_format, void *output);
```

Alibaba Confidential

 Copyright© 2016 Alibaba

Description:

向系统配置文件中读取对应的配置值

Input Parameters:

const char *section: 配置存储的字段
const char *key: 配置存储的键值
int32_t max_len: 获取配置字符串最大长度
int32_t value_format: 写入数据的格式
void *output: 输出的数据指针

Returned Value:

0 或错误号

4.6 NETLIB

4.6.1 netlib_ipaddrconv

Function Prototype:

```
#include <netlib.h>
bool netlib_ipaddrconv(FAR const char *addrstr, uint8_t *addr);
```

Description:

将正常的十进制 IP 地址解析成四字节的通用地址。

Input Parameters:

addrstr: ip 地址 例如: 192.168.0.1
addr: ip 整型地址 例如: 0xc0,0xa0,0x00,0x01

Returned Value:

false: 解析失败
true: 解析成功

4.6.2 netlib_hwmacconv

Function Prototype:

```
#include <netlib.h>
bool netlib_hwmacconv(FAR const char *hwstr, uint8_t *hw);
```

Description:

将正常的 mac 地址解析成四字节的通用地址。

Alibaba Confidential

 Copyright© 2016 Alibaba

Input Parameters:

hwstr: mac 地址　例如: 08:00:27:6c:b9:45
hw: mac 字符　例如: 0x08,0x00,0x27,0x6c,0xb9,0x45

Returned Value:

false: 解析失败
true: 解析成功

4.6.3 netlib_setmacaddr

Function Prototype:

```
#include <netlib.h>
int netlib_setmacaddr(FAR const char *ifname, const uint8_t *macaddr);
```

Description:

设置网卡驱动的 mac 地址

Input Parameters:

ifname: 要使用接口名称
macaddr: 要设置的 mac 地址，大小必须是 6

Returned Value:

0: 设置成功
-1: 设置失败

4.6.4 netlib_getmacaddr

Function Prototype:

```
#include <netlib.h>
int netlib_getmacaddr(FAR const char *ifname, uint8_t *macaddr);
```

Description:

获取网卡驱动的 mac 地址

Input Parameters:

ifname: 要使用接口的名称
macaddr: 要返回的 mac 地址的位置

Returned Value:

0: 获取成功

Alibaba Confidential

Copyright© 2016 Alibaba

-1: 获取失败

4.6.5 netlib_get_ipv4addr

Function Prototype:

#include <netlib.h>
int netlib_get_ipv4addr(FAR const char *ifname, FAR struct in_addr *addr);

Description:

获取网卡驱动 ipv4 地址

Input Parameters:

ifname: 要使用的接口名称
addr: 要返回 IP 地址的位置

Returned Value:

0: 获取成功
-1: 获取失败

4.6.6 netlib_set_ipv4addr

Function Prototype:

#include <netlib.h>
int netlib_set_ipv4addr(FAR const char *ifname, FAR const struct in_addr *addr);

Description:

设置网卡驱动 ipv4 地址

Input Parameters:

ifname: 要使用的接口名称
addr: 要设置的地址

Returned Value:

0: 设置成功
-1: 设置失败

4.6.7 netlib_set_dripv4addr

Function Prototype:

#include <netlib.h>

Alibaba Confidential

 Copyright© 2016 Alibaba

int netlib_set_dripv4addr(FAR const char *ifname,
FAR const struct in_addr *addr);

Description:

设置默认路由器的 ipv4 地址

Input Parameters:

ifname: 要使用的接口名称
addr: 要设置的地址

Returned Value:

0: 设置成功
-1: 设置失败

4.6.8 netlib_set_ipv4netmask

Function Prototype:

#include <netlib.h>
int netlib_set_ipv4netmask(FAR const char *ifname,
FAR const struct in_addr *addr);

Description:

设置 ipv4 网卡驱动

Input Parameters:

ifname: 要使用的接口名称
addr: 要设置的地址

Returned Value:

0: 设置成功
-1: 设置失败

4.6.9 netlib_parsehttpurl

Function Prototype:

#include <netlib.h>
int netlib_parsehttpurl(FAR const char *url, uint16_t *port, char *hostname,
int hostlen, char *filename, int namelen);

Description:

解析 httpurl 地址

Alibaba Confidential

 Copyright© 2016 Alibaba

Input Parameters:

url: url 地址
port: 端口
hostname: 主机名字
hostlen: 主机名字长度
filename: 文件名字
namelen: 文件名字长度

Returned Value:

EINVAL:无效的参数
E2BIG: 列表太长
ENOENT: 没有这样的文件或者目录
OK: 成功

4.6.10 netlib_listenon

Function Prototype:

```
#include <netlib.h>
int netlib_listenon(uint16_t portno);
```

Description:

实现基本的服务器监听

Input Parameters:

portno: 端口监听（在网络字节口）

Returned Value:

有效的声音端口。
-1 错误的端口

4.6.11 netlib_server

Function Prototype:

```
#include <netlib.h>
void netlib_server(uint16_t portno, pthread_startroutine_t handler,
                   int stacksize);
```

Description:

实现服务器的基本逻辑

Input Parameters:

Alibaba Confidential

 Copyright© 2016 Alibaba

portno: 端口监听（在网络字节口）
handler: 一个新的连接被接受时任务结果的切入点
stacksize: 产生任务所需要的堆栈大小

Returned Value:

不返回除非出现错误

4.6.12 netlib_getifstatus

Function Prototype:

```
#include <netlib.h>
int netlib_getifstatus(FAR const char *ifname, FAR uint8_t *flags);
```

Description:

获取网卡驱动上/下状态

Input Parameters:

ifname: 要使用的接口名称
flags:通过 SIOCGIFFLAGS 返回的接口标志

Returned Value:

0: 获取成功
-1: 获取失败

4.6.13 netlib_ifup

Function Prototype:

```
#include <netlib.h>
int netlib_ifup(FAR const char *ifname);
```

Description:

设置网络上升接口

Input Parameters:

ifname: 要使用的接口名称

Returned Value:

0: 设置成功
-1: 设置失败

4.6.14 netlib_ifdown

Alibaba Confidential

 Copyright© 2016 Alibaba

Function Prototype:

```
#include <netlib.h>
int netlib_ifdown(FAR const char *ifname);
```

Description:

设置网络下降接口

Input Parameters:

ifname: 要使用的接口名称

Returned Value:

0: 设置成功
-1: 设置失败

4.6.15 netlib_set_ipv4dnsaddr

Function Prototype:

```
#include <netlib.h>
int netlib_set_ipv4dnsaddr(FAR const struct in_addr *inaddr);
```

Description:

设置 DNS 服务器的 IPv4 地址

Input Parameters:

inaddr: 要设置的地址

Returned Value:

ZERO(OK): 设置成功
一个错误属性值: 设置失败

4.6.16 netlib_get_ipv4dnsaddr

Function Prototype:

```
#include <netlib.h>
int netlib_get_ipv4dnsaddr(FAR struct in_addr *inaddr);
```

Description:

获取 DNS 服务器的 IPv4 地址

Input Parameters:

inaddr: IPv4 地址的返回位置

Alibaba Confidential

 Copyright© 2016 Alibaba

Returned Value:

ZERO(OK): 获取成功
一个错误属性值: 获取失败

4.7 GPIO

4.7.1 gpio_init

Function Prototype:

```
#include <sc6138_gpio.h>
void gpio_init();
```

Description:

初始化 GPIO 模块

Input Parameters:

void

Returned Value:

void

4.7.2 gpio_irq_reg

Function Prototype:

```
#include <sc6138_gpio.h>
int gpio_irq_reg(unsigned pin, gpio_interrupt_t cb);
```

Description:

注册 GPIO 相应引脚的中断处理函数

Input Parameters:

pin: 引脚（0-63）
cb: 中断处理函数

Returned Value:

1: 注册成功
-1: 注册失败

4.7.3 gpio_irq_unreg

Function Prototype:

Alibaba Confidential

 Copyright© 2016 Alibaba

#include <sc6138_gpio.h>
int gpio_irq_unreg(unsigneedm pin);

Description:

注销 GPIO 相应引脚的中断

Input Parameters:

pin: 引脚（0-63）

Returned Value:

1: 注销成功
-1: 注销失败

4.7.4 gpio_get_value

Function Prototype:

#include <sc6138_gpio.h>
int gpio_get_value(unsigned pin);

Description:

读取 GPIO 相应引脚的电平值

Input Parameters:

pin: 引脚（0-63）

Returned Value:

电平的值 0: 低电平
1: 高电平

4.7.5 gpio_set_value

Function Prototype:

#include <sc6138_gpio.h>
void gpio_set_value(unsigned pin, int value);

Description:

设置 GPIO 相应引脚的输出电平

Input Parameters:

pin: 引脚（0-63）
value: 值

Alibaba Confidential

 Copyright© 2016 Alibaba

Returned Value:

void

4.7.6 gpio_direction_input

Function Prototype:

#include <sc6138_gpio.h>
int gpio_direction_input(unsigned pin);

Description:

设置 GPIO 相应引脚为输入模式

Input Parameters:

pin: 引脚（0-63）

Returned Value:

电平的值 0: 低电平
1: 高电平

4.7.7 gpio_direction_output

Function Prototype:

#include <sc6138_gpio.h>
int gpio_direction_output(unsigned pin, int value);

Description:

设置 GPIO 相应引脚为输出模式

Input Parameters:

pin: 引脚（0-63）
value: 电平值

Returned Value:

0: 成功

4.7.8 gpio_irq_mode

Function Prototype:

#include <sc6138_gpio.h>
int gpio_irq_mode(unsigned pin, enum irq_mode _irqmode);

Alibaba Confidential

 Copyright© 2016 Alibaba

Description:

设置 GPIO 相应引脚的中断触发模式

Input Parameters:

pin: 引脚（0-63）
_irqmode:中断触发模式
RISING_EDGE: 上升沿
FALLING_EDGE: 下降沿
DOUBLE_EDGE: 双边沿
LOW_LEVEL: 低电平
HIGH_LEVEL: 高电平

Returned Value:

-1: 设置失败
1: 设置成功

4.8 OTA

4.8.1 get_system_version

Function Prototype:

```
#include <nuttx/ota_interface.h>
char* get_system_version(void);
```

Description:

获取当前系统版本号

Input Parameters:

void

Returned Value:

包含系统版本的字符串

4.8.2 get_manifest_addr

Function Prototype:

```
#include <nuttx/ota_interface.h>
uint32_t get_manifest_addr(void);
```

Description:

Alibaba Confidential

 Copyright© 2016 Alibaba

获取 update 分区的 Manifest 的起始地址供 ota 升级使用

Input Parameters:

void

Returned Value:

Manifest 的地址

4.8.3 get_image_addr

Function Prototype:

```
#include <nuttx/ota_interface.h>
unit32_t get_image_addr(void);
```

Description:

获取 update 分区的 image 的起始地址供 ota 升级使用

Input Parameters:

void

Returned Value:

image 的地址

4.8.4 get_image_max_size

Function Prototype:

```
#include <nuttx/ota_interface.h>
uint32_t get_image_max_size();
```

Description:

获取 update 分区的 image 可烧写的最大大小供 ota 升级使用

Input Parameters:

void

Returned Value:

可烧写 image 的最大大小

4.8.5 flash_page_program

Function Prototype:

Alibaba Confidential

 Copyright© 2016 Alibaba

```
#include <nuttx/ota_interface.h>
int32_t flash_page_program(uint32_t start_addr, int32_t buf_len,
                           uint8_t *buf);
```

Description:

用于烧写 flash 的接口，实现了在将地址对齐后将原数据读出，修改相应部分，擦除对应的 flash 块并将新数据写入，并对写入正确性进行检查

Input Parameters:

start_addr: 写入页的起始地址
buf_len: 写入的数据的长度
buf: 写入数据的缓存区

Returned Value:

0 或者 flash 烧写出现错误的字节号

4.8.6 set_update_finished_flags

Function Prototype:

```
#include <nuttx/ota_interface.h>
int32_t set_update_finished_flags(void);
```

Description:

设置 ota 升级完成的标志位，并切换主备分区

Input Parameters:

void

Returned Value:

0 或者 flash 烧写出现错误的字节号

4.8.7 show_config_info

Function Prototype:

```
#include <nuttx/ota_interface.h>
void show_config_info(void);
```

Description:

打印主备切换标示，升级结果，和 Manifest 地址

Input Parameters:

Alibaba Confidential

 Copyright© 2016 Alibaba

void

Returned Value:

4.8.8 get_update_result

Function Prototype:

```
#include <nuttx/ota_interface.h>
int16_t get_update_result(void);
```

Description:

获取 ota 的升级结果

Input Parameters:

void

Returned Value:

0 时为更新完成状态 通常都处于该状态当升级后或升级失败后都会更新该区域为 0

1 时为下载后状态 当 ota 将镜像更新后状态从 0 变为 1 只有为 1 时 boot 才会更新该状态机

2 时为更新成功状态 当 boot2 发现状态为 1 image 检验成功则将状态设为 2

3 时为失败状态 当 boot1 或 boot2 发现状态为 1 引导错误则将状态设为 3 并切回之前分区引导

4.8.9 ota_start_udpate

Function Prototype:

```
#include <nuttx/ota_interface.h>
void ota_start_udpate(uint32_t start_addr, int32_t buf_len, void *buf,
                      ota_update_progress_t cb);
```

Description:

用于 ota 升级时异步烧写的接口，将 flash 烧写函数注册到工作队列

Input Parameters:

start_addr: 写入页的起始地址
buf_len: 写入的数据的长度
buf: 写入数据的缓存区
ota_update_progress_t: 相应进度的回调函数

Alibaba Confidential

 Copyright© 2016 Alibaba

Returned Value:

0 或者 flash 烧写出现的错误的字节号

4.8.10 reboot_system

Function Prototype:

```
#include <nuttx/ota_interface.h>
void reboot_system(void);
```

Description:

复位开发板

Input Parameters:

void

Returned Value:

4.9 WDOG

4.9.1 wd_create

Function Prototype:

```
#include <wdog.h>
WDOG_ID wd_create (void);
```

Description:

从空的 wdog 列表里创建一个新的 wdog

Input Parameters:

void

Returned Value:

wdog 的 ID，例如（i，e...）
NULL。

4.9.2 wd_delete

Function Prototype:

```
#include <wdog.h>
int wd_delete(WDOG_ID wdog);
```

Alibaba Confidential

 Copyright© 2016 Alibaba

Description:

用完 wdog 之后删除 wdog

Input Parameters:

wdog: 要被删除的 wdog ID(实际上是一个指向 watchdog 的指针)

Returned Value:

OK: 删除成功
ERROR: 删除失败

4.9.3 wd_start

Function Prototype:

```
#include <wdog.h>
int wd_start(WDOG_ID wdog, int delay, wdentry_t wdentry,  int argc, ...);
```

Description:

添加一个 watchdog 到定时器队列。在给定的时间走完以后，将会从中断中调用该 watchdog 回调函数。watchdog 定时器可能在中断中启动。

当 wd_start() 被调用时 watchdog 定时器开启会受地址环境影响。

watchdog 定时器只会执行一次。

用同一个 wdog 再次调用 wd_start()会延时操作；只有 wd_start()给的 watchdogID 才不会受到任何影响。

Input Parameters:

wdog: 要打开的 wdog 的 ID
delay: 时钟的延迟计数
wdentry: 函数调用超时
parml..4: 参数传递给 wdentry

Returned Value:

OK: 成功打开
ERROR: 打开失败

4.9.4 wd_cancel

Function Prototype:

```
#include <wdog.h>
int wd_cancel(WDOG_ID wdog);
```

Description:

Alibaba Confidential

 Copyright© 2016 Alibaba

取消目前正在运行的 watchdog 时钟。watchdog 时钟可能会从中断区域中取消。

Input Parameters:

wdog: 要取消打开 Wdog 的 ID

Returned Value:

OK: 取消成功
ERROR: 取消失败

4.9.5 wd_gettime

Function Prototype:

```
#include <wdog.h>
int wd_gettime(WDOG_ID wdog);
```

Description:

获取指定的 Wdog 到期之前的剩余时间

Input Parameters:

wdog: Wdog 的 ID

Returned Value:

系统时间直到 wdog 时间到期
0 表示要么 wdog 无效或者已经到期

4.9.6 work_queue

Function Prototype:

```
#include <wqueue.h>
int work_queue(int qid, FAR struct work_s *work, worker_t worker,
               FAR void *arg, uint32_t delay);
```

Description:

一段时间后形成工作队列，在工作线程(不是调用者)的执行下所有的工作队列都将执行。

这个工作结构体是由调用者分配的，但是却由工作队列逻辑完成。调用者没有修改过工作队列结构；直到前面的工作从工作队列中删除或者 work_cancel 被调用来删除工作队列上的工作时调用者才会调用 work_queue().

Alibaba Confidential

 Copyright© 2016 Alibaba

Input Parameters:

qid:任务队列 ID
work:任务结构体
worker:任务的回调函数
arg:任务回调函数参数
delay:延时的时间(0 表示立即形成)

Returned Value:

0: 成功
一个错误的参数: 失败

4.9.7 work_cancel

Function Prototype:

```
#include <wqueue.h>
int work_cancel(int qid, FAR struct work_s *work);
```

Description:

取消前面的工作队列。这个被取消的任务会再次被 work_queue()调用。

Input Parameters:

qid: 任务队列 ID
work: 任务结构体

Returned Value:

0: 成功
一个错误的参数: 失败
ENOENT: 没有这样的任务队列
EINVAL: 指定了无效的任务队列

4.10 DES

4.10.1 des3_en

Function Prototype:

```
#include <nuttx/crypto/des.h>
int des3_en(unsigned char *key, int key_len, int mode, unsigned char *input,
    unsigned int input_len, unsigned char *output, unsigned int output_len);
```

Description:

Alibaba Confidential

 Copyright© 2016 Alibaba

DES 加密: 将明文加密成密文

Input Parameters:

key: 密钥
key_len: 密钥长度
mode:模式
input: 输入的明文
input_len:输入的明文长度
output:输出的密文
output_len: 输出的密文长度

Returned Value:

0: 加密成功
-1: 加密失败

4.11 AES

4.11.1 aes_encrypt

Function Prototype:

```
#include <nuttx/crypto/aes.h>
void aes_encrypt(FAR uint8_t *state, FAR const uint8_t *key);
```

Description:

AES EBC 加密: AES128 位密钥将 16 字节明文加密成 16 字节密文。

Input Parameters:

state: 输入 16 字节的明文和输出的 16 字节的密文
key: 16 字节大小的 AES128 密钥

Returned Value:

没有返回值

4.11.2 aes_decrypt

Function Prototype:

```
#include <nuttx/crypto/aes.h>
void aes_decrypt(FAR uint8_t *state, FAR const uint8_t *key);
```

Description:

Alibaba Confidential

 Copyright© 2016 Alibaba

AES EBC 解密: AES128 位密钥将 16 字节密文解密成 16 字节明文。

Input Parameters:

state: 输入 16 字节的密文和输出的 16 字节明文
key: 16 字节大小的 AES128 密钥

Returned Value:

没有返回值

4.12 I2C

4.12.1 I2C_SETFREQUENCY

Function Prototype:

#include <i2c.h>
#define I2C_SETFREQUENCY(FAR struct i2c_dev_s *dev, uint32_t frequency);

Description:

设置频率

Input Parameters:

dev: 专用设备状态结构体数据
frequency: 选定 I2C 的频率请求

Returned Value:

实际选择的频率

4.12.2 I2C_SETADDRESS

Function Prototype:

#include <i2c.h>
#define I2C_SETADDRESS(FAR struct i2c_dev_s *dev, int addr, int nbits);

Description:

设置 I2C 从机地址

Input Parameters:

dev: 专用设备状态结构体数据
addr: I2C 的从机地址

Alibaba Confidential

 Copyright© 2016 Alibaba

nbits: 提供地址位的数目（7 或 10）

Returned Value:

OK: 成功
一个错误信息: 失败

4.12.3 I2C_WRITE

Function Prototype:

```
#include <i2c.h>
#define I2C_WRITE(FAR struct i2c_dev_s *dev, const uint8_t *buffer,
                  int buflen);
```

Description:

从一个 I2C 发送一个阻塞数据到另一个 I2C

Input Parameters:

dev: 专用设备状态结构体数据
buffer:数据缓存区
buflen:数据长度

Returned Value:

0: 成功
<0: 一个错误的信息

4.12.4 I2C_READ

Function Prototype:

```
#include <i2c.h>
#define I2C_READ(FAR struct i2c_dev_s *dev, uint8_t *buffer, int buflen);
```

Description:

接收到从另一个 I2C 里发来的阻塞数据

Input Parameters:

dev: 专用设备状态结构体数据
buffer:数据缓存区
buflen:数据长度

Returned Value:

0: 成功

Alibaba Confidential

 Copyright© 2016 Alibaba

<0: 一个错误的信息

4.13 EVENT

4.13.1 open

Function Prototype:

#include <fcntl.h>
int open(const char *path, int oflags, ...);

Description:

打开 event 事件上报机制

Input Parameters:

path: UART 设备路径，如 ： “/dev/input/event”
oflags: 操作权限，如: O_RDONLY。

Returned Value:

>=0: 打开 EVENT 模块句柄
-1: 打开 EVENT 模块失败

4.13.2 read

Function Prototype:

#include <unistd.h>
ssize_t read(int fd, FAR void *buf, size_t nbytes);

Description:

读取事件数据

Input Parameters:

fd: EVENT 模块句柄
buf: 保存事件数据
nbytes: 事件数据字节数

Returned Value:

非零数据读取成功
为 0 或者-1 读取失败

4.13.3 input_add_event

Alibaba Confidential

 Copyright© 2016 Alibaba

Function Prototype:

#include <nuttx/input.h>

int input_add_event（struct input_event　*event

Description:

上报事件函数

Input Parameters:

Event: 事件信息

Type: 设备类型

在 linux 原有上报事件驱动设备类型基础上，添加针对 nuttx 上报事件驱动设备。

#define	EV_SYN	0x00	表示设备支持所有的事件
#define	EV_KEY	0x01	按键设备
#define	EV_REL	0x02	鼠标设备，表示一个相对的光标位置结果
#define	EV_ABS	0x03	手写板产生的值，其是一个绝对整数值
#define	EV_MSC	0x04	其他类型
#define	EV_LED	0x11	LED 灯设备
#define	EV_SND	0x12	蜂鸣器，输入声音
#define	EV_REP	0x14	允许重复按键类型
#define	EV_PWR	0x16	电源管理事件
#define	EV_KNOD	0x21	旋钮事件　　//nuttx 新增
#define	EV_SD	0x22	SD 卡插拔事件//nuttx 新增
#define	EV_CABLE	0x23	网线插拔事件　/nuttx 新增
#define	EV_WIFI	0x24	WIFI 事件　　//nuttx 新增

Code: code 含义根据 type 类型不同而不同

例如: 当 Type 为 EV_KNOD 时，0 表示内旋钮，1 表示外旋钮

Value: value 含义根据 type 和 code 类型不同而不同

例如:

1. Type 为 EV_KEY 时，0 表示按键短按弹起，1 表示按键短按按下，2 表示按键双击，3 表示按键长按弹起，4 表示按键长按按下

2. Type 为 EV_KNOD，code 值为 0 时，value 取值为 0-100，不同的值代表内 旋钮不同的位置；code 值为 1 时，value 取值为 0-7，不同的 value 值代表不同的 channel。

Alibaba Confidential

Copyright© 2016 Alibaba

3. Type 为 EV_REL 时，value: 表明移动的值和方向(正负值)
4. Type 为 EV_SD 时，0 表示 SD 卡拔出，1 表示 SD 卡插入
5. Type 为 EV_CABLE 时，0 表示网线拔出，1 表示网线插入
6. Type 为 EV_WIFI 时，0 表示 wifi 断开连接，1 表示 wifi 连接成功

Returned Value:

0: 执行成功
-1: 执行失败

4.14 WIFI

4.14.1 wifi_init

Function Prototype:

```
#include <nuttx/wifi.h>
int wifi_init();
```

Description:

wifi 初始化

Input Parameters:

void

Returned Value:

0: 成功
-1: 失败

4.14.2 wifi_get_macaddr

Function Prototype:

```
#include <nuttx/wifi.h>
int wifi_get_macaddr(char addr[6]);
```

Description:

获取无线网卡的 mac 地址

Input Parameters:

addr[6]: mac 地址

Returned Value:

Alibaba Confidential

 Copyright© 2016 Alibaba

0: 成功
-1: 失败

4.14.3 wifi_scan

Function Prototype:

```
#include <nuttx/wifi.h>
int wifi_scan(wifi_scan_t *scan_res, uint32_t size);
```

Description:

返回有效个数

Input Parameters:

scan_res: 属性
ssid[32]: 账号信息
bssid[6]:
channel: 通道
security: 模式
rssi: 用户信息
size: 大小

Returned Value:

有效个数的数目

4.14.4 wifi_sta_start

Function Prototype:

```
#include <nuttx/wifi.h>
int wifi_sta_start(char *sz_ssid, char * sz_pswd);
```

Description:

wifi 开启 sta 模式(设备连接路由器)。

Input Parameters:

sz_ssid: wifi 账号
sz_pswd: wifi 密码

Returned Value:

0: 开启成功
-1: 开启失败

Alibaba Confidential

Copyright© 2016 Alibaba

4.14.5 wifi_get_linkinfo

Function Prototype:

```
#include <nuttx/wifi.h>
int wifi_get_linkinfo(wifi_link_info_t * info);
```

Description:

获取无线路由器的属性

Input Parameters:

info: 无线路由器属性
status: 连接属性。0，断开；1，连接。
rssi: 用户信息
noise: 噪音

Returned Value:

0: 获取成功
-1: 获取失败

4.14.6 wifi_ap_start

Function Prototype:

```
#include <nuttx/wifi.h>
int wifi_ap_start(wifi_ap_param_t *param);
```

Description:

打开 ap 模式(手机当热点连接)

Input Parameters:

param: 属性
sz_ssid[36]: 账号信息
sz_password[36]: 密码信息
sz_ipaddr[16]: IP 地址
security_type: 加密方式。0，共享；1: wpa 密码；2: wpa2 密码
channel: 管道切换

Returned Value:

0: 打开成功
-1: 打开失败

Alibaba Confidential

Copyright© 2016 Alibaba

4.14.7 wifi_ap_stop

Function Prototype:

```
#include <nuttx/wifi.h>
int wifi_ap_stop();
```

Description:

停止 ap 模式

Input Parameters:

void

Returned Value:

0: 停止成功
-1: 停止失败

4.14.8 wifi_es_start

Function Prototype:

```
#include <nuttx/wifi.h>
int wifi_es_start();
```

Description:

开启智能模式(手机发给设备账号密码直接连接)

Input Parameters:

void

Returned Value:

0: 开启成功
-1: 开启失败

4.14.9 wifi_es_stop

Function Prototype:

```
#include <nuttx/wifi.h>
int wifi_es_stop();
```

Description:

停止智能模式

Alibaba Confidential

Copyright© 2016 Alibaba

Input Parameters:

void

Returned Value:

0: 停止成功
-1: 停止失败

4.14.10 wifi_es_get_info

Function Prototype:

```
#include <nuttx/wifi.h>
int wifi_es_get_info(char * ssid, char *pwd);
```

Description:

获取 wifi 信息

Input Parameters:

ssid: 账号
pwd: 密码

Returned Value:

0: 获取成功
-1: 获取失败

Alibaba Confidential

 Copyright© 2016 Alibaba

5 kernel

5.1 Task Control Interfaces

5.1.1 getpid

Function Prototype:

#include <unistd.h>
pid_t getpid(void);

Description:

获取当前进程的进程 ID

Input Parameters:

void

Returned Value:

当前进程的进程 ID

5.2 Task Scheduling Interfaces

5.2.1 sched_setparam

Function Prototype:

#include <sched.h>
int sched_setparam(pid_t pid, const struct sched_param *param);

Description:

根据 pid 参数设置进程 ID 的优先级

Input Parameters:

pid: 进程 ID（若是 0 则为当前进程）
param: 将要设置的任务优先级。有效的取值范围在 SCHED_PRIORITY_MIN 到 SCHED_PRIORITY_MAX。

Returned Value:

0：成功
-1 (错误) 错误函数：

Alibaba Confidential

 Copyright© 2016 Alibaba

- EINVAL. 输入的 param 参数无效
- EPERM. 调用的任务没有权限
- ESRCH. 任务 ID 无法被找到

5.2.2 sched_getparam

Function Prototype:

```
#include <sched.h>
int sched_getparam (pid_t pid, struct sched_param *param);
```

Description:

根据 pid 参数获取一个进程的优先级

Input Parameters:

- pid. 进程 ID。如果 pid 为 0，返回的是当前进程的优先级。
- param. 保存任务优先级的结构体。该进程的优先级将保存到此结构体的 sched_priority 中。

Returned Value:

- 0 (OK) 成功；

-1(ERROR)失败

5.2.3 sched_setscheduler

Function Prototype:

```
#include <sched.h>
int sched_setscheduler (pid_t pid, int policy,
                        const struct sched_param *param);
```

Description:

设置一个进程的调度策略和实时优先级。如果 pid 为 0，则设置正在被调用的进程。param 是在新的调度策略下的优先级。

Input Parameters:

- pid. 进程 ID。如果 pid 为 0，则设置正被调用的进程的优先级。
- policy. 调度策略请求（SCHED_FIFO or SCHED_RR).
- param. 将要设置任务的优先级。有效的取值范围在 SCHED_PRIORITY_MIN 到 SCHED_PRIORITY_MAX。

Returned Value:

Alibaba Confidential

 Copyright© 2016 Alibaba

OK (zero)：成功

-1：失败

- EINVAL ：调度策略不是公认的策略之一.
- ESRCH ：任务 ID 无法被找到

5.2.4 sched_getscheduler

Function Prototype:

```
#include <sched.h>
int sched_getscheduler (pid_t pid);
```

Description:

获取一个进程的调度策略。当 pid 为 0 时，获取的为正在调用的进程。

Input Parameters:

- pid. 进程 ID。如果 pid 为 0，则查询正在运行的进程。

Returned Value:

- (SCHED_FIFO or SCHED_RR)：成功

-1：失败

ESRCH ：进程 ID 没有被找到

5.2.5 sched_yield

Function Prototype:

```
#include <sched.h>
int sched_yield( void );
```

Description:

自愿放弃处理器而不阻塞

Input Parameters:

void

Returned Value:

0：成功

-1：失败

5.2.6 sched_get_priority_max

Function Prototype:

Alibaba Confidential

 Copyright© 2016 Alibaba

#include <sched.h>

int sched_get_priority_max (int policy)

Description:

获取调度策略的最大优先级值

Input Parameters:

policy. 调度策略请求

Returned Value:

最大优先级的值；

-1：失败

5.2.7 sched_get_priority_min

Function Prototype:

#include <sched.h>

int sched_get_priority_min (int policy);

Description:

获取调度策略的最小优先级值

Input Parameters:

- policy. 调度策略请求

Returned Value:

- 最小优先级的值；

-1：失败

5.2.8 sched_get_rr_interval

Function Prototype:

#include <sched.h>

int sched_get_rr_interval (pid_t pid, struct timespec *interval);

Description:

将时间片长度写入指向时间间隔的结构体进行保存，若 pid 为 0，则为当前结构。确认过程将在 SCHED_RR 模式下运行。

Input Parameters:

- pid. 进程 ID

Alibaba Confidential

 Copyright© 2016 Alibaba

- interval. 用于返回时间片的结构

Returned Value:

0：成功
-1：失败
- EFAULT：无法复制到间隔
- EINVAL：无效的进程 ID
- ENOSYS ：尚未实现的系统调用
- ESRCH：任务 ID 无法被找到

5.3 Task Switching Interfaces

5.3.1 sched_lock

Function Prototype:

```
#include <sched.h>
int sched_lock( void );
```

Description:

加锁进程，使这进程唯一能够运行。直到调用 sched_unlock 才能继续运行。

Input Parameters:

void

Returned Value:

OK：成功
ERROR：失败

5.3.2 sched_unlock

Function Prototype:

```
#include <sched.h>
int sched_unlock( void );
```

Description:

解锁进程，对应 sched_lock。一个 sched_lock 对应一个 sched_unlock。

Input Parameters:

void

Alibaba Confidential

 Copyright© 2016 Alibaba

Returned Value:

OK：成功
ERROR：失败

5.3.3 sched_lockcount

Function Prototype:

#include <sched.h>
int32_t sched_lockcount(void)

Description:

这个函数返回剩余 sched_lock 的次数。如果零，函数就可以运行；如果不为零，这个值表示剩余 sched_lock()的次数。

Input Parameters:

void

Returned Value:

lockcount 的值

5.4 Named Message Queue Interfaces

5.4.1 mq_open

Function Prototype:

#include <mqueue.h>
mqd_t mq_open(const char *mqName, int oflags, ...);

Description:

这个函数在信息队列和调用进程中直接建立了一个连接。在成功的调用 mq_open()之后，这个进程就可以返回到信息队列所用的地址。这个信息队列在 mq_close()出现之前一直有用。

Input Parameters:

- mqName. 要打开的队列名字
- oflags. 要打开的文件句柄

O_RDONLY. 只读
O_WRONLY. 只写
O_RDWR. 允许读写访问
O_CREAT. 创建新的消息队列

Alibaba Confidential

 Copyright© 2016 Alibaba

O_EXCL.如果名称存在函数会打开失败

O_NONBLOCK. 不等待数据

- ... 可选参数。当选择 O_CREAT 时，会出现第三和第三个参数 mode.模式。是 mode_t 模式中的一种。这个模式的值提供了对应的权限。

attr. 指向一个提供初始化的 qm_attr 结构。如果 attr 是 NULL，这个新创建的线程为默认属性。如果 attr 不是 NULL，新建的线程会从 mq_maxmsg 获取一个属性。mq_maxmsg 属性值确定了块模块失败前能发送的最大值，mq_msgsize 能够发送或者接受的最大值。其他属性可以忽略。

Returned Value:

- 消息队列描述符

-1：失败

5.4.2 mq_close

Function Prototype:

```
#include <mqueue.h>
int mq_close( mqd_t mqdes );
```

Description:

用于告诉进程调度信息队列已经完成。mq_close 释放了信息队列线程所使用的系统资源。如果这个调用线程通过 mqdes 额外的发送了一个通知请求给信息队列，那这个线程将会被删除，这个信息队列将会获取另一个进程的额外通知。

Input Parameters:

- mqdes. 消息队列描述符

Returned Value:

- 0：成功关闭

-1：关闭失败

5.4.3 mq_unlink

Function Prototype:

```
#include <mqueue.h>
int mq_unlink( const char *mqName );
```

Description:

该函数通过传入的"mqName"参数移除消息队列。当一个或多个任务在

Alibaba Confidential

 Copyright© 2016 Alibaba

my_unlink()被调用时打开了消息队列，直到所有相关的消息队列被关闭才能移除消息队列。

Input Parameters:

- mqName. 消息队列名字

Returned Value:

没有返回值

5.4.4 mq_send

Function Prototype:

```
#include <mqueue.h>
int mq_send(mqd_t mqdes, const void *msg, size_t msglen, int prio);
```

Description:

该函数向消息队列（mqdes）添加特定的消息(msg)，消息的字节长度由 msglen 参数指定。该长度不能超过在 mq_getattr()函数中定义的消息长度的最大值。如果该消息队列没有被填满，mq_send()将在消息队列的指定位置放置消息。高优先级的消息队列将被放置在低优先级的消息队列前面。消息的优先级不能超过 MQ_PRIO_MAX。如果指定的消息队列已满且已被设置成 O_NONBLOCK 模式， mq_send() 将被阻塞直到消息队列的空间足够用。如果消息队列已满且被设置为 NON_BLOCK 模式，返回 ERROR。

Input Parameters:

- mqdes. 消息队列描述符
- msg. 要发送的消息
- msglen. 消息的字节长度
- prio. 消息的优先级

Returned Value:

0：成功

-1：失败•

EAGAIN. 队列是空队列

- EINVAL. msg 或者 mqdes 中有一个是空的或无效参数
- EPERM. 消息队列打开了但不能写入
- EMSGSIZE. 消息长度超过了最大值
- EINTR. 被中断程序打断

5.4.5 mq_timedsend

Alibaba Confidential

 Copyright© 2016 Alibaba

Function Prototype:

```
#include <mqueue.h>
int mq_timedsend(mqd_t mqdes, const char *msg, size_t msglen, int prio,
                 const struct timespec *abstime);
```

Description:

在 mq_timedsend 的基础上添加了一个定时函数。定时向消息队列（mqdes）添加特定的消息(msg)，消息的字节长度由 msglen 参数指定。长度不能超过在 mq_getattr()函数中定义的消息长度的最大值。如果该消息队列没有被填满，mq_timedsend()将在消息队列的指定位置放置消息。高优先级的消息队列将被放置在低优先级的消息队列前面。消息的优先级不能超过 MQ_PRIO_MAX。如果指定的消息队列已满且已被设置成 O_NONBLOCK 模式，mq_send() 将被阻塞直到消息队列的空间足够用。如果消息队列已满，或者定时时间(这个时间为从 1970.1.1.0:00:00 开始的秒和纳秒计数)过了 mq_timedsend 将会立即返回。

Input Parameters:

- mqdes. 消息队列描述符
- msg. 要发送的消息
- msglen. 要发送的消息字节长度
- prio. 消息的优先级

abstime.时间

Returned Value:

0：成功

-1：失败

EAGAIN. 队列是空队列

- EINVAL. msg 或者 mqdes 中有一个是空的或无效参数
- EPERM. 消息队列打开了但不能写入
- EMSGSIZE. 消息长度超过了最大值
- EINTR. 被中断程序打断

5.4.6 mq_receive

Function Prototype:

```
#include <mqueue.h>
ssize_t mq_receive(mqd_t mqdes, void *msg, size_t msglen, int *prio);
```

Description:

Alibaba Confidential

 Copyright© 2016 Alibaba

该函数从 mqdes 指定的消息队列里接受优先级最高、最早的消息。如果消息的字节长度 msglen 比消息队列的 mq_mshsize 小，mq_receive()返回错误。

如果消息队列为空且设置为 O_NONBLOCK 模式，mq_receive()将被阻塞直到向消息队列中添加消息。当多个任务等待接受一个消息，只有优先级最高且等待时间最长的任务将不被阻塞。

如果队列为空且设置为 O_NONBLOCK 将会返回错误。

Input Parameters:

- mqdes. 消息队列描述符
- msg. 接收消息的缓存区
- msglen. 缓存区的字节大小
- prio. 存储消息的优先级的位置

Returned Value:.

返回字节长度则成功

-1：失败.

- EAGAIN 队列是空队列
- EPERM 信息打开后不能读取
- EMSGSIZE 缓存区大小小于要接收信息的大小
- EINTR 被中断程序打断
- EINVAL msg 或者 mqdes 无效

5.4.7 mq_timedreceive

Function Prototype:

```
#include <mqueue.h>
ssize_t mq_timedreceive(mqd_t mqdes, void *msg, size_t msglen, int *prio,
                        const struct timespec *abstime);
```

Description:

该函数定时的从 mqdes 指定的消息队列里接受优先级最高、最早的消息。如果消息的字节长度 msglen 比消息队列的 mq_mshsize 小，mq_timedreceive()返回错误。

如果消息队列为空且设置为 O_NONBLOCK 模式，mq_timedreceive()将被阻塞直到向消息队列中添加消息。当多个任务等待接受一个消息，只有优先级最高且等待时间最长的任务将不被阻塞。

如果队列为空且定时超时(这个时间为从 1970.1.1.0:00:00 开始的秒和纳秒计数)mq_timedreceive()将会返回错误。

Input Parameters:

Alibaba Confidential

 Copyright© 2016 Alibaba

- mqdes. 消息队列描述符
- msg. 接收消息的缓存区
- msglen. 缓存区的字节大小
- prio. 存储消息的优先级的位置
- abstime. 需要等待的时间.

Returned Value:

返回字节长度则成功
-1：失败
- EAGAIN: 队列是空队列
- EPERM: 消息打开后不能读取
- EMSGSIZE: 缓存区大小小于要接收消息的大小
- EINTR: 被中断消息打断
- EINVAL: msg 或者 mqdes 或者 abstime 无效
- ETIMEDOUT: 调用超时

5.4.8 mq_notify

Function Prototype:

```
#include <mqueue.h>
int mq_notify(mqd_t mqdes, const struct sigevent *notification);
```

Description:

如果输入的参数"notification"不为 NULL，一旦消息从空变为非空，该函数都将向任务发生信号。一个"notification"能绑定一个消息队列。

如果"notification"为 NULL，该通知与消息队列解绑，该消息队列能够与另外一个通知进行绑定。

当该通知被发送到被注册的任务中，原先的注册将被移除，该消息队列可以进行注册。

Input Parameters:

- mqdes. 消息队列描述符
- notification. 实时信号结构：

sigev_notify.应该是 SIGEV_SIGNAL (但是被忽略了)
sigev_signo.本该用于通知
sigev_value. 与信号相关联的值

Returned Value:

没有返回值

Alibaba Confidential

 Copyright© 2016 Alibaba

5.4.9 mq_setattr

Function Prototype:

```
#include <mqueue.h>
int mq_setattr( mqd_t mqdes, const struct mq_attr *mqStat,
                struct mq_attr *oldMqStat);
```

Description:

该函数对"mqdes"指定的消息队列设置属性。只有当该消息队列被设置为"O_NONBLOCK"，其属性才能被改变。

如果参数"oldMqStat"为 non-null，mq_setattr( )将保存先前的消息队列的属性。

Input Parameters:

- mqdes. 消息队列描述符
- mqStat. 消息队列新属性
- oldMqState. 消息队列旧属性

Returned Value:

- 0(OK)：成功

-1：失败

5.4.10 mq_getattr

Function Prototype:

```
#include <mqueue.h>
int mq_getattr( mqd_t mqdes, struct mq_attr *mqStat);
```

Description:

该函数获取 mqdes 指定的消息队列的状态信息和属性。

Input Parameters:

- mqdes. 消息队列描述符

mqStat. 返回属性的缓存区

mq_maxmsg. 队列中最大的值

mq_msgsize. 最大信息长度

mq_flags. 队列标志

mq_curmsgs. 目前队列中的信息数目

Returned Value:

Alibaba Confidential

 Copyright© 2016 Alibaba

- 0：成功

-1：失败

5.5 Counting Semaphore Interfaces

5.5.1 sem_init

Function Prototype:

```
#include <semaphore.h>
int sem_init ( sem_t *sem, int pshared, unsigned int value );
```

Description:

初始化一个匿名信号量 sem。当成功调用 sem_init( )，该信号量可以应用在 sem_wait( ), sem_post( ), sem_trywait( )等这些函数中，直到该信号量被销毁。

Input Parameters:

- sem. 指向信号量结构的一个指针
- pshared. 共享过程（若为 0，信号量将被进程内的线程共享，并且应该放在这个进程的所有线程都可见的地址上；非 0，信号量将在进程直接共享，并且应该定位共享内存区域）
- value. 可用资源的数目，即信号灯的数目

Returned Value:

- 0(OK):成功

-1：失败

5.5.2 sem_destroy

Function Prototype:

```
#include <semaphore.h>
int sem_destroy ( sem_t *sem );
```

Description:

注销一个由 sem 指定的匿名信号量，只有由 sem_init()创建的信号量才能调用 sem_destroy()进行注销。不能调用 sem_destroy()注销有名信号量。当调用 sem_destroy()注销该信号量后，该信号量不可用，直到进程重新调用 sem_init()进行初始化 sem 信号量。

Input Parameters:

Alibaba Confidential

 Copyright© 2016 Alibaba

- sem. 匿名的信号量

Returned Value:

- 0(OK)：注销成功

-1：注销失败

5.5.3 sem_open

Function Prototype:

```
#include <semaphore.h>
sem_t *sem_open ( const char *name, int oflag, ...);
```

Description:

这个函数在信号命名和线程直接建立了一个连接。在用信号名字调用 sem_open()的情况下，线程将会通过这个调用返回使用这个名字的信号的地址。这个信号还会被用于 sem_wait().sem_trywait(),sem_post()。这个信号一直存在直到在一次成功的调用 sem_close()之后关闭。

如果线程用同一个信号名字多次调用 sme_open()，将会返回到同一个地址（将不会提供调用给 sem_unlink()）

Input Parameters:

- name.有名信号量外部名字
- oflag.有名信号量创建选项，选择创建或打开一个现有的信号量

0: 只有当该信号量已经存在才连接

O_CREAT: 如果存在该信号量则连接，否则创建新的信号量

O_CREAT with O_EXCL (O_CREAT|O_EXCL): 创建一个新的信号

- ... 可选参数.

mode.模式

value.属性值

Returned Value:

- 信号描述符

-1：失败.

5.5.4 sem_close

Function Prototype:

```
#include <semaphore.h>
int sem_close ( sem_t *sem );
```

Description:

Alibaba Confidential

 Copyright© 2016 Alibaba

这个函数调用将会被用来说明指定名字的信号的进程已经结束。sem_close()将会释放分配给这个信号的系统资源。

如果在调用 sem_unlink()之后这个信号还没被删除，sem_close()对于同一个信号也不会有效果。当这些信号被完成拆开，当最后的进程关闭他们的时候他们就会完全消失。

Input Parameters:

- sem. 信号描述符

Returned Value:

- 0 (OK)：关闭成功

-1：关闭失败

5.5.5 sem_unlink

Function Prototype:

```
#include <semaphore.h>
int sem_unlink ( const char *name );
```

Description:

sem_unlink 会马上删除指定的信号量名，但要等到所有打开该信号量的进程关闭该信号量后才删除该信号。

Input Parameters:

- name. 信号量名

Returned Value:

- 0 (OK)：成功

-1：失败

5.5.6 sem_wait

Function Prototype:

```
#include <semaphore.h>
int sem_wait ( sem_t *sem );
```

Description:

减小(锁定)由 sem 指定的信号量的值.如果信号量的值比 0 大,那么进行减一的操作,函数立即返回.

如果信号量当前为 0 值,那么调用就会一直阻塞直到或者是信号量变

Alibaba Confidential

 Copyright© 2016 Alibaba

得可以进行减一的操作(例如,信号量的值比 0 大),或者是信号处理程序中断调用

Input Parameters:

- sem. 信号描述符

Returned Value:

- 0 (OK)：成功

-1：失败

- EINVAL: 输入的 sem 是无效的
- EINTR: 被中断程序打断

5.5.7 sem_trywait

Function Prototype:

```
#include <semaphore.h>
int sem_trywait ( sem_t *sem );
```

Description:

与 sem_wait()类似，只是在不能够对信号量立即减一时会返回错误。

Input Parameters:

- sem. 信息描述符

Returned Value:

- 0：成功

-1：失败

- EINVAL: 输入的 sem 是无效的.
- EAGAIN: 无法获取信号量

5.5.8 sem_post

Function Prototype:

```
#include <semaphore.h>
int sem_post ( sem_t *sem );
```

Description:

用来增加信号量的值。当有线程阻塞在这个信号量上时，调用这个函数

Alibaba Confidential

 Copyright© 2016 Alibaba

会使其中的一个线程不在阻塞，选择机制同样是由线程的调度策略决定的。如果信号当前值是 0，在调用 sem_wait()时阻塞的信号将会成功返回。

Input Parameters:

- sem.信号描述符

Returned Value:

- 0 (OK) ：成功

-1：失败

5.5.9 sem_getvalue

Function Prototype:

```
#include <semaphore.h>
int sem_getvalue ( sem_t *sem, int *sval );
```

Description:

把 sem 指向信号量当前值放置在 sval 指向的整数上。如果信号是被锁住的，sem_getvalue()得到的值将会是 0 或者负数。

Input Parameters:

- sem. 信号描述符
- sval. 返回值返回的缓存区

Returned Value:

- 0 (OK) ：成功

-1：失败

5.6 Clocks and Timers

5.6.1 clock_settime

Function Prototype:

```
#include <time.h>
int clock_settime(clockid_t clockid, const struct timespec *tp);
```

Description:

设置时钟时间

Input Parameters:

Alibaba Confidential

 Copyright© 2016 Alibaba

clockid：指定的时钟
tp：当前时间

Returned Value:

0(OK): 成功
非 0: 返回指定的错误

5.6.2 clock_gettime

Function Prototype:

```
#include <time.h>
int clock_gettime(clockid_t clockid, struct timespec *tp);
```

Description:

计算精度和纳秒

Input Parameters:

clockid：指定的时钟
tp：当前时间

Returned Value:

0(OK): 成功
非 0: 指定的错误

5.6.3 clock_getres

Function Prototype:

```
#include <time.h>
int clock_getres(clockid_t clockid, struct timespec *res);
```

Description:

获取对应时钟类型能够提供的时间精确度

Input Parameters:

clockid：指定的时钟
res：保存的精确度

Returned Value:

0(OK):成功
非 0 的值则为失败

Alibaba Confidential

 Copyright© 2016 Alibaba

5.6.4 mktime

Function Prototype:

```
#include <time.h>
time_t mktime(struct tm *tp);
```

Description:

将参数 tp 所指向的 tm 结构体数据转换为从公元 1970 年 1 月 1 日 0 时 0 秒算起至今的本地时间所经过的秒数

Input Parameters:

- tp：值之间连续两个非负的整数倍数

Returned Value:

0（OK）：转换成功
非 0:　　返回指向的错误

5.6.5 gmtime

Function Prototype:

```
#include <time.h>
struct tm *gmtime(const time_t *clock);
```

Description:

把日期和时间转换为格林威治(GMT)时间的函数

Input Parameters:

- clock. 当前时间。这是相当于当前时间的一个值，从 1970 年 1 月 1 日 00:00:00 到现在的秒数。

Returned Value:

转换后的时间

5.6.6 localtime

Function Prototype:

```
#include <time.h>
#define localtime(c) gmtime(c)
```

Description:

Alibaba Confidential

 Copyright© 2016 Alibaba

获取用户指定的时区的时间

Input Parameters:

Returned Value:

指定的时区的时间

5.6.7 gmtime_r

Function Prototype:

#include <time.h>
struct tm *gmtime_r(const time_t *clock, struct tm *result);

Description:

把日期和时间转换为格林威治(GMT)时间的函数并将数据存储到用户提供的结构体中

Input Parameters:

- clock. 当前时间。这是相当于当前时间的一个值，从 1970 年 1 月 1 日 00:00:00 到现在的秒数。

result：提供的存储时间结构

Returned Value:

0(OK):成功
非 0 的值则为失败

5.6.8 localtime_r

Function Prototype:

#include <time.h>
#define localtime_r(c,r) gmtime_r(c,r)

Description:

获取指定的时区的时间并将数据存储到用户提供的结构体中

Input Parameters:

Returned Value:

0(OK):成功
非 0 的值则为失败

Alibaba Confidential

 Copyright© 2016 Alibaba

5.6.9 timer_create

Function Prototype:

```
#include <time.h>
int timer_create(clockid_t clockid, struct sigevent *evp, timer_t *timerid);
```

Description:

进程可以通过调用 timer_create()创建特定的定时器，定时器是每个进程自己的，不是 fork 时继承的。clock_id 说明定时器是基于哪个时钟的，*timerid 装载的是被创建的定时器的 ID。该函数创建了定时器，并将他的 ID 放入 timerid 指向的位置中。参数 evp 指定了定时器到期要产生的异步通知。如果 evp 为 NULL，那么定时器到期会产生默认的信号，对 CLOCK_REALTIMER 来说，默认信号就是 SIGALRM.如果要产生除默认信号之外的其他信号，程序要将 sigevent 结构中的 sigev_notify 的值说明了定时器到期时应该采取的行动。通常，这个成员的值为 SIGEV_SIGNAL,这个值说明在定时器到期时，会产生一个信号。可以用 sigev_signo 来防止定时器到期时产生信号，sigev_value 来区分是哪个定时器产生了信号。

Input Parameters:

- clockid. 特定的时钟。必须是 CLOCK_REALTIME。
- evp. 异步通知。evp 可能为 NULL
- timerid. 用来调用 timer_create()创建的定时器 ID

Returned Value:

0(OK)成功

-1：失败

- EAGAIN. The system lacks sufficient signal queuing resources to honor the request.
- EAGAIN. 缺少足够的信号
- EINVAL. 未定义指定的时钟标示.
- ENOTSUP. 不支持创建时钟线程

5.6.10 timer_delete

Function Prototype:

```
#include <time.h>
int timer_delete(timer_t timerid);
```

Description:

Alibaba Confidential

 Copyright© 2016 Alibaba

销毁关联到 timerid 的定时器。

Input Parameters:

- timerid. 由调用 timer_create()产生的要销毁的定时器 ID。

Returned Value:

0(OK):成功
-1：失败
- EINVAL. 定时器无效

5.6.11 timer_settime

Function Prototype:

```
#include <time.h>
int timer_settime(timer_t timerid, int flags, const struct itimerspec *value,
                  struct itimerspec *ovalue);
```

Description:

初始化指定定时器。it_value 用于指定当前的定时器到期时间。当定时器到期，it_value 的值会被更新成 it_interval 的值。如果 it_interval 的值为 0，则定时器不是一个时间间隔定时器，一旦 it_value 到期就会回到未启动状态。如果 flags 的值为 TIMER_ABSTIME，则 value 所指定的时间值会被解读成绝对值（此值得默认的解读方式为相对于当前的时间）。这个经修改的行为可避免取得当前时间、计算“该时间”与“所期望的未来时间”的相对差额已经启动定时器期间造成的竞争条件。如果 ovalue 的值不是 NULL，则之前的定时器到期时间会被存于其所提供的 itimerspec。如果定时器之前处在未启动状态，则此结构的成员全都会被设定成 0.

Input Parameters:

- timerid. 由调用 timer_create()产生的要设置的定时器 ID。
- flags. 定时器类型
- value. 指定定时器的到期时间
- ovalue. 之前定时器到期时间的返回位置

Returned Value:

0(OK)：成功
-1：失败
- EINVAL. 这个 ID 不是由 timer_create()创建但是没有被 timer_delete()删除• EINVAL. 一个值指定结构的纳秒值小于零或大于或等于 10 亿，并且该结构的 it_value 成员没有指定零秒和纳秒。

Alibaba Confidential

 Copyright© 2016 Alibaba

5.6.12 timer_gettime

Function Prototype:

```
#include <time.h>
int timer_gettime(timer_t timerid, struct itimerspec *value);
```

Description:

这个函数会将 timerid 定时器剩余时间储存到 value 中。它的结构成员 it_value 会在定时器到期或者为 0 之前储存剩余时间。返回值为活动定时器的剩余时间。value 中的 it_interval 会包含 timer_settime()在重置前的最后一次设置。由于这个函数的异步操作，这个函数报道的时间，可能在任何时间都会大大超过活动定时器的剩余时间。

Input Parameters:

- timerid. 指定定时器 id

value：指定定时器的到期时间

Returned Value:

0(OK):成功

-1：失败

- EINVAL. 这个 ID 不符合 timer_create()但是没有被 timer_delete()删除

5.6.13 timer_getoverrun

Function Prototype:

```
#include <time.h>
int timer_getoverrun(timer_t timerid);
```

Description:

有可能一个定时器到期了，而同一定时器上一次到期时产生的信号还处于挂起状态。在这种情况下，其中的一个信号可能会丢失。这就是定时器超限。程序可以通过调用 timer_getoverrun 来确定一个特定的定时器出现这种超限的次数。定时器超限只能发生在同一个定时器产生的信号上。由多个定时器，甚至是那些使用相同的时钟和信号的定时器，所产生的信号都会排队而不会丢失。执行成功时，timer_getoverrun()会返回定时器初次到期与通知进度（例如通过信号）定时器已到期之间额外发生的定时器到期次数。如果超限运行的次数等于或大于 DELAYTIMER_MAX，则此调用会返回 DELAYTIMER_MAX。

Input Parameters:

Alibaba Confidential

 Copyright© 2016 Alibaba

- timerid. 特定定时器 id

Returned Value:

定时器超限的次数

- EINVAL. 这个 ID 不符合 timer_create()但是没有被 timer_delete()删除

5.6.14 gettimeofday

Function Prototype:

```
#include <sys/time.h>
int gettimeofday(struct timeval *tp, void *tzp);
```

Description:

获取当前精确时间,或者为执行计时。把获取的时间保存在 timeval 结构体中。

Input Parameters:

- tp. 保存获取时间结构的结构体
- tzp. 保存时区结果

Returned Value:

0(OK):成功
非 0 的值为失败

5.7 Signal Interfaces

5.7.1 sigemptyset

Function Prototype:

```
#include <signal.h>
int sigemptyset(sigset_t *set);
```

Description:

将参数 set 信号集初始化并清空

Input Parameters:

- set. 要初始化的信号集

Returned Value:

- 0 (OK)：成功

Alibaba Confidential

 Copyright© 2016 Alibaba

-1：失败

5.7.2 sigfillset

Function Prototype:

#include <signal.h>
int sigfillset(sigset_t *set);

Description:

将参数 set 信号集初始化，然后把所有的信号加入到此信号集里即将所有的信号标志位置为 1，屏蔽所有的信号

Input Parameters:

- set. 要初始化的信号集

Returned Value:

- 0 (OK)：成功

-1：失败
EFAULT：参数 set 指针地址无法存取

5.7.3 sigaddset

Function Prototype:

#include <signal.h>
int sigaddset(sigset_t *set, int signo);

Description:

将参数 signo 代表的信号加入到 set 信号集里

Input Parameters:

- set. 要添加信号的信号集
- signo.要添加的信号

Returned Value:

- 0 (OK)：成功

-1：失败

5.7.4 sigdelset

Function Prototype:

#include <signal.h>

Alibaba Confidential

 Copyright© 2016 Alibaba

int sigdelset(sigset_t *set, int signo);

Description:

将参数 signo 代表的信号从 set 信号集里删除

Input Parameters:

- set. 要删除信号的信号集
- signo. 要删除的信号

Returned Value:

- 0 (OK)：成功

-1：失败

5.7.5 sigismember

Function Prototype:

#include <signal.h>

int sigismember(const sigset_t *set, int signo);

Description:

测试 signo 代表的信号是否已经加入到参数 set 信号集里

Input Parameters:

- set. 要加入的信号集
- signo.要测试的信号

Returned Value:

- 1(TURE)：指定信号是该组成员•

0 (OK or FALSE), 指定信号不是该组成员

- -1 (ERROR) 信号数目是无效的

5.7.6 sigaction

Function Prototype:

#include <signal.h>

int sigaction(int signo, const struct sigaction *act, struct sigaction *oact);

Description:

查询或设置信号处理方式。信号结构体中有以下成员：

sa_u.sa_handler 信号处理函数

su_u.sa_sigaction 替代信号处理函数

Alibaba Confidential

 Copyright© 2016 Alibaba

sa_mask 用来设置在处理信号时是否要阻塞其他信号

sa_flags 用来设置信号处理的其他相关操作

如果参数 act 不是 NULL 指针，它会指向相关的信号处理方式结构。如果参数 oact 不是 NULL 指针，则原来的信号处理方式将由此结构返回。如果 act 是 NULL 指针，信号处理函数不会因为这个调用改变，所以这个调用可以用于查询信号处理函数。

如果设定 sa_sigaction 为一个处理函数，那么这个函数被调用的时候，不但可以得到编号，而且可以获悉被调用的原因以及产生问题的上下文的相关信息。

Input Parameters:

- sig. 要查询或设置的信号
- act. 新处理程序的位置
- oact. 老处理程序存储的位置

Returned Value:

- 0 (OK)：成功

-1：sig 是无效参数

5.7.7 sigprocmask

Function Prototype:

```
#include <signal.h>
int sigprocmask(int how, const sigset_t *set, sigset_t *oset);
```

Description:

用于改变进程的当前阻塞信号集,也可以用来检测当前进程的信号掩码。如果 set 不是 NULL 指针，则参数 how 指示如何修改当前信号屏蔽字。

每个进程都有一个用来描述哪些信号递送到进程时将被阻塞的信号集，该信号集中的所有信号在递送到进程后都将被阻塞。

如果 sigprocmask()失败，信号掩码将不会被改变。

Input Parameters:

- how. 如何修改：

SIG_BLOCK. 该值代表的功能是将 newset 所指向的信号集中所包含的信号加到当前的信号掩码中，作为新的信号屏蔽字。

SIG_UNBLOCK. 将参数 newset 所指向的信号集中的信号从当前的信号掩码中移除

SIG_SETMASK. 设置当前信号掩码为参数 newset 所指向的信号集中所包含的信号•

Alibaba Confidential

 Copyright© 2016 Alibaba

set. 新信号掩模的位置

- oset. 旧信号掩模存储的位置

Returned Value:

- 0 (OK)：成功

-1：失败

5.7.8 sigpending

Function Prototype:

```
#include <signal.h>
int sigpending( sigset_t *set );
```

Description:

函数返回在送往进程的时候被阻塞挂起的信号集合。如果进程接收了来自 sigprocmask 的信号，这个信号会被阻塞直到他不在被遮挡。只有一个阻塞信号会被系统保留。

Input Parameters:

- set. 待处理的信号集.

Returned Value:

- 0 (OK) ：成功

-1：失败

5.7.9 sigsuspend

Function Prototype:

```
#include <signal.h>
int sigsuspend( const sigset_t *set );
```

Description:

用于在接收到某个信号之前，临时用 mask 替换进程的信号掩码，并暂停进程执行，直到收到信号为止。如果设置参数的效果是非阻塞一个挂起信号，那就不会有等待执行。当 sigsuspend()返回时，那个原始信号掩码恢复。等待一个空信号集去停止一个进程将不会释放任何资源。

Input Parameters:

- set. 信号掩模在暂停时使用的值。

Returned Value:

Alibaba Confidential

 Copyright© 2016 Alibaba

- 始终返回-1

5.7.10 sigwaitinfo

Function Prototype:

```
#include <signal.h>
int sigwaitinfo(const sigset_t *set, struct siginfo *info);
```

Description:

此功能选择参数集指定的挂起的信号集

Input Parameters:

- set. 等待的信号集
- info. 返回的信号值

Returned Value:

导致等待终止的信号,
-1：失败

5.7.11 sigtimedwait

Function Prototype:

```
#include <signal.h>
int sigtimedwait( const sigset_t *set, struct siginfo *info,
                  const struct timespec *timeout );
```

Description:

此功能清除挂起的信号，并把传来的信号值赋给参数 info。如果在 set 里有多个信号被挂起，它会删除这些信号并返回到最小的一个信号上。如果在调用中没有信号在 set 里被挂起，这个调用进程会暂停直到有一个信号被挂起或者直到进程被一个非阻塞信号中断或者直到中断时间到了。如果中断时间是 NULL 指针，那就永远不会停止。

如果参数 info 为非 NULL 指针，被选择的信号数字会被储存在 si_signo 里，剩余的信号会被保存在 si_code 里，si_code 定义在 signal.h 里:

SI_USER.从 kill，raise，abort 发来的信号
SI_QUEUE sigqueue 发来的信号
SI_TIMER 请求超时的信号
SI_ASYNCIO 异步 IO 完成后的结果
SI_MESGQ 在一个空的消息队列上生成一个新的消息

Input Parameters:

Alibaba Confidential

 Copyright© 2016 Alibaba

- set. 等待的信号集
- info. 返回的信号值
- timeout. 等待的时间

Returned Value:

导致等待终止的信号,
-1：失败

5.7.12 sigqueue

Function Prototype:

#include <signal.h>
int sigqueue (int tid, int signo, union sigval value);

Description:

在队列中向指定进程发送一个信号和数据

Input Parameters:

- tid. 目标进程的进程号
- signo. 信号代号
- value. 信号附带的数据

Returned Value:

- 0(OK)：成功

-1：失败
EGAIN. 已达到信号的限制.
EINVAL. 无效的 singo
EPERM. 任务没有权限将信号发送给接收程序
ESRCH. 进程没有一个匹配的进程号

5.7.13 kill

Function Prototype:

#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int sig);

Description:

可以让系统发送任何信号到任何任务

Input Parameters:

Alibaba Confidential

 Copyright© 2016 Alibaba

- pid. 接收信号任务的代号(0 和负数不支持，非 0 的值才支持)
- sig. 要发送的信号数目

Returned Value:

- OK：成功

ERROR：失败

5.8 Pthread Interfaces

5.8.1 pthread_attr_init

Function Prototype:

```
#include <pthread.h>
int pthread_attr_init(pthread_attr_t *attr);
```

Description:

初始化一个线程对象的属性

Input Parameters:

- attr：属性

Returned Value:

0：成功

错误代码：失败

5.8.2 pthread_attr_destroy

Function Prototype:

```
#include <pthread.h>
int pthread_attr_destroy(pthread_attr_t *attr);
```

Description:

销毁一个目标结构，并且使它在重新初始化之前不能重新使用

Input Parameters:

- attr：属性

Returned Value:

0：成功

错误代码：失败

Alibaba Confidential

 Copyright© 2016 Alibaba

5.8.3 pthread_attr_setschedpolicy

Function Prototype:

```
#include <pthread.h>
int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```

Description:

设置线程调度策略

Input Parameters:

- attr：属性

policy：参数

Returned Value:

0：成功
错误代码：失败

5.8.4 pthread_attr_getschedpolicy

Function Prototype:

```
#include <pthread.h>
int pthread_attr_getschedpolicy(pthread_attr_t *attr, int *policy);
```

Description:

获取线程调度策略

Input Parameters:

- attr：属性

policy：参数

Returned Value:

0：成功
错误代码：失败

5.8.5 pthread_attr_getschedpolicy

Function Prototype:

```
#include <pthread.h>
int pthread_attr_setschedparam(pthread_attr_t *attr,
                               const struct sched_param *param);
```

Alibaba Confidential

 Copyright© 2016 Alibaba

Description:

获取线程调度策略

Input Parameters:

- attr：属性

param:参数

Returned Value:

0：成功
错误代码：失败

5.8.6 pthread_attr_getschedparam

Function Prototype:

```
#include <pthread.h>
int pthread_attr_getschedparam(pthread_attr_t *attr,
                               struct sched_param *param);
```

Description:

设置和获取 schedparam 属性

Input Parameters:

- attr：属性

param:来储存线程优先级的位置

Returned Value:

0：成功
错误代码：失败

5.8.7 pthread_attr_setinheritsched

Function Prototype:

```
#include <pthread.h>
int pthread_attr_setinheritsched(pthread_attr_t *attr, int inheritsched);
```

Description:

设置继承的调度策略

Input Parameters:

- attr：属性

Alibaba Confidential

 Copyright© 2016 Alibaba

Inheritsched：新建的线程

Returned Value:

0：成功
错误代码：失败

5.8.8 pthread_attr_getinheritsched

Function Prototype:

#include <pthread.h>
int pthread_attr_getinheritsched(const pthread_attr_t *attr, int *inheritsched);

Description:

获取继承的调度策略

Input Parameters:

- attr：属性

Inheritsched：新建的线程

Returned Value:

0：成功
错误代码：失败

5.8.9 pthread_attr_setstacksize

Function Prototype:

#include <pthread.h>
int pthread_attr_setstacksize(pthread_attr_t *attr, long stacksize);

Description:

设置堆栈大小

Input Parameters:

- attr：线程属性变量

stacksize：设置的堆栈大小

Returned Value:

0：成功
错误代码：失败

5.8.10 pthread_attr_getstacksize

Alibaba Confidential

 Copyright© 2016 Alibaba

Function Prototype:

#include <pthread.h>
int pthread_attr_getstacksize(pthread_attr_t *attr, long *stackaddr);

Description:

获取堆栈大小

Input Parameters:

- attr：线程属性变量

stackaddr：设置的堆栈地址

Returned Value:

0：成功
错误代码：失败

5.8.11 pthread_create

Function Prototype:

#include <pthread.h>
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
pthread_startroutine_t startRoutine, pthread_addr_t arg);

Description:

创建一个线程。

Input Parameters:

- thread: 保存新创建线程的线程 ID

attr：线程属性
startRoutine：线程运行函数的起始地址
arg：创建的线程函数的参数

Returned Value:

0：成功
错误代码：失败

5.8.12 pthread_detach

Function Prototype:

#include <pthread.h>
int pthread_detach(pthread_t thread);

Alibaba Confidential

 Copyright© 2016 Alibaba

Description:

线程运行结束后自动释放所有资源

Input Parameters:

- thread:指向线程的标示符的指针

Returned Value:

0：成功
错误代码：失败

5.8.13 pthread_exit

Function Prototype:

```
#include <pthread.h>
void pthread_exit(pthread_addr_t pvValue);
```

Description:

终止调用它的线程并返回一个指向某个对象的指针

Input Parameters:

- pvValue：指向线程的标示符的指针

Returned Value:

0：成功
错误代码：失败

5.8.14 pthread_cancel

Function Prototype:

```
#include <pthread.h>
int pthread_cancel(pthread_t thread);
```

Description:

主动取消线程函数

Input Parameters:

- thread. 确定要取消的线程

Returned Value:

0：成功
- ESRCH. 没有找对对应的线程

Alibaba Confidential

 Copyright© 2016 Alibaba

5.8.15 pthread_setcancelstate

Function Prototype:

#include <pthread.h>
int pthread_setcancelstate(int state, int *oldstate);

Description:

设置取消方式函数

Input Parameters:

state：取消方式 PTHREAD_CANCEL_ENABLE or PTHREAD_CANCEL_DISABLE

- oldstate. 返回先前取消方式的地址

Returned Value:

0：成功
ESRCH. 没有找到对应的线程

5.8.16 pthread_testcancelstate

Function Prototype:

#include <pthread.h>
int pthread_setcancelstate(void);

Description:

设置取消方式函数

Input Parameters:

- void

Returned Value:

0：成功
-1：失败

5.8.17 pthread_join

Function Prototype:

#include <pthread.h>
int pthread_join(pthread_t thread, pthread_addr_t *ppvValue);

Description:

Alibaba Confidential

 Copyright© 2016 Alibaba

等待一个线程的结束

Input Parameters:

- thread：线程标示符

ppvValue：用户定义的指针，用来存储被等待线程的返回值

Returned Value:

0：成功
错误的代码：失败

5.8.18 pthread_yield

Function Prototype:

```
#include <pthread.h>
void pthread_yield(void);
```

Description:

使当前的线程自动放弃剩余的 CPU 时间从而让另一个线程运行

Input Parameters:

- void

Returned Value:

0：成功
错误代码：失败

5.8.19 pthread_self

Function Prototype:

```
#include <pthread.h>
pthread_t pthread_self(void);
```

Description:

获取线程自身的 ID

Input Parameters:

- void

Returned Value:

0：成功
错误代码：失败

Alibaba Confidential

 Copyright© 2016 Alibaba

5.8.20 pthread_getschedparam

Function Prototype:

#include <pthread.h>

int pthread_getschedparam(pthread_t thread, int *policy, struct sched_param *param);

Description:

获取线程的权限

Input Parameters:

- thread. 使用 pthread_create 所获得的线程 ID
- policy. 存储线程调度策略的位置
- param. 存储线程优先级的位置

Returned Value:

0：成功

-1：失败

5.8.21 pthread_setschedparam

Function Prototype:

#include <pthread.h>

int pthread_setschedparam(pthread_t thread, int policy, const struct sched_param *param);

Description:

设置线程的权限

Input Parameters:

- thread. 使用 pthread_create 所获得的线程 ID
- policy. 线程的调度有三种策略：SCHED_OTHER、SCHED_RR 和 SCHED_FIFO。policy 用于指明使用哪种策略

param. 存储线程优先级的位置.

Returned Value:

0：成功

- EINVAL. 参数是无效的
- ENOTSUP. 不支持策略
- EPERM. 权限不足

Alibaba Confidential

 Copyright© 2016 Alibaba

- ESRCH. 指定的值不引用现有的线程

5.8.22 pthread_key_create

Function Prototype:

#include <pthread.h>

int pthread_key_create(pthread_key_t *key, void (*destructor)(void*))

Description:

创建线程私有数据

Input Parameters:

- key.指向一个键值的指针
- destructor .指明了一个 destructor 函数，如果这个参数不为空，那么当每个线程结束时，系统将调用这个函数来释放绑定在这个键上的内存块。

Returned Value:

0：成功

- EAGAIN. 数目过多
- ENONMEM 内存不足.

5.8.23 pthread_setspecific

Function Prototype:

#include <pthread.h>

int pthread_setspecific(pthread_key_t key, void *value)

Description:

对键值进行设置

Input Parameters:

- key. 设置绑定的数据键
- value. 绑定到密钥的值

Returned Value:

0：成功•

ENOMEM. 内存不足

- EINVAL. 参数无效

5.8.24 pthread_getspecific

Alibaba Confidential

 Copyright© 2016 Alibaba

Function Prototype:

#include <pthread.h>
void *pthread_getspecific(pthread_key_t key)

Description:

获取键值

Input Parameters:

- key. 要获取绑定的数据键

Returned Value:

键值
NULL

5.8.25 pthread_key_delete

Function Prototype:

#include <pthread.h>
int pthread_key_delete(pthread_key_t key)

Description:

删除线程私有数据

Input Parameters:

- key. 要删除的键值

Returned Value:

- 一直返回 EINVAL

5.8.26 pthread_mutexattr_init

Function Prototype:

#include <pthread.h>
int pthread_mutexattr_init(pthread_mutexattr_t *attr);

Description:

初始化互斥锁属性对象

Input Parameters:

- attr：互斥锁属性值

Alibaba Confidential

 Copyright© 2016 Alibaba

Returned Value:

0：成功
错误代码：失败

5.8.27 pthread_mutexattr_destroy

Function Prototype:

```
#include <pthread.h>
int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
```

Description:

销毁互斥锁属性对象

Input Parameters:

- attr：互斥锁属性值

Returned Value:

0：成功
错误代码：失败

5.8.28 pthread_mutexattr_getpshared

Function Prototype:

```
#include <pthread.h>
int pthread_mutexattr_getpshared(pthread_mutexattr_t *attr, int *pshared);
```

Description:

获取互斥锁范围

Input Parameters:

- attr：互斥锁属性值

pshared：属性范围

Returned Value:

0：成功
错误代码：失败

5.8.29 pthread_mutexattr_setpshared

Function Prototype:

Alibaba Confidential

 Copyright© 2016 Alibaba

#include <pthread.h>
int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr, int pshared);

Description:

设置互斥锁范围

Input Parameters:

- attr：互斥锁属性值

pshared：属性范围

Returned Value:

0：成功
错误代码：失败

5.8.30 pthread_mutexattr_gettype

Function Prototype:

#include <pthread.h>
#ifdef CONFIG_MUTEX_TYPES
int pthread_mutexattr_gettype(const pthread_mutexattr_t *attr, int *type);
#endif

Description:

获取互斥锁的类型属性

Input Parameters:

- attr. 互斥锁的属性
- type. 参数指定互斥锁的类型

Returned Value:

0：成功•
EINVAL. 无效的参数

5.8.31 pthread_mutexattr_settype

Function Prototype:

#include <pthread.h>
#ifdef CONFIG_MUTEX_TYPES
int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
#endif

Alibaba Confidential

 Copyright© 2016 Alibaba

Description:

设置互斥锁的类型属性

Input Parameters:

- attr. 互斥锁的属性
- type. 参数指定互斥锁的类型

PTHREAD_MUTEX_NORMAL. 此类型的互斥锁不会检测死锁。如果线程在不首先解除互斥锁的情况下尝试重新锁定该互斥锁，则会产生死锁。尝试解除由其他线程锁定的互斥锁会产生不确定的行为。如果尝试解除锁定的互斥锁未锁定，则会产生不确定的行为。

PTHREAD_MUTEX_ERRORCHECK. 此类型的互斥锁可提供错误检查。如果线程在不首先解除锁定互斥锁的情况下尝试重新锁定该互斥锁，则会返回错误。如果线程尝试解除锁定的互斥锁已经由其他线程锁定，则会返回错误。如果线程尝试解除锁定的互斥锁未锁定，则会返回错误。

PTHREAD_MUTEX_RECURSIVE.如果线程在不首先解除锁定互斥锁的情况下尝试重新锁定该互斥锁，则可成功锁定该互斥锁。 与 PTHREAD_MUTEX_NORMAL 类型的互斥锁不同，对此类型互斥锁进行重新锁定时不会产生死锁情况。多次锁定互斥锁需要进行相同次数的解除锁定才可以释放该锁，然后其他线程才能获取该互斥锁。如果线程尝试解除锁定的互斥锁已经由其他线程锁定，则会返回错误。 如果线程尝试解除锁定的互斥锁未锁定，则会返回错误。

PTHREAD_MUTEX_DEFAULT. 如果尝试以递归方式锁定此类型的互斥锁，则会产生不确定的行为。对于不是由调用线程锁定的此类型互斥锁，如果尝试对它解除锁定，则会产生不确定的行为。对于尚未锁定的此类型互斥锁，如果尝试对它解除锁定，也会产生不确定的行为。允许在实现中将该互斥锁映射到其他互斥锁类型之一。对于 Solaris 线程，PTHREAD_PROCESS_DEFAULT 会映射到 PTHREAD_PROCESS_NORMAL。

Returned Value:

0：成功

错误代码：失败

5.8.32 pthread_mutex_init

Function Prototype:

```
#include <pthread.h>
int pthread_mutex_init(pthread_mutex_t *mutex, pthread_mutexattr_t *attr);
```

Alibaba Confidential

 Copyright© 2016 Alibaba

Description:

互斥锁的初始化

Input Parameters:

- mutex：互斥锁

attr：互斥锁属性

Returned Value:

0：初始化成功

错误代码：失败

5.8.33 pthread_mutex_destroy

Function Prototype:

```
#include <pthread.h>
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Description:

销毁互斥锁

Input Parameters:

- mutex：要销毁的互斥锁

Returned Value:

0：销毁成功

错误代码：销毁失败

5.8.34 pthread_mutex_lock

Function Prototype:

```
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

Description:

给互斥锁上锁

Input Parameters:

- mutex. 互斥锁

Returned Value:

0：成功

Alibaba Confidential

 Copyright© 2016 Alibaba

错误代码：失败(但是不会返回 EINTR)

5.8.35 pthread_mutex_trylock

Function Prototype:

#include <pthread.h>
int pthread_mutex_trylock(pthread_mutex_t *mutex);

Description:

非阻塞的锁定互斥锁

Input Parameters:

- mutex. 要锁定的互斥锁

Returned Value:

0：成功
错误代码：失败(但是不会返回 EINTR)

5.8.36 pthread_mutex_unlock

Function Prototype:

#include <pthread.h>
int pthread_mutex_unlock(pthread_mutex_t *mutex);

Description:

释放互斥锁

Input Parameters:

- mutex：要释放的互斥锁

Returned Value:

0：成功
错误代码：失败(但是不会返回 EINTR)

5.8.37 pthread_condattr_init

Function Prototype:

#include <pthread.h>
int pthread_condattr_init(pthread_condattr_t *attr);

Description:

Alibaba Confidential

 Copyright© 2016 Alibaba

初始化条件变量属性

Input Parameters:

- attr：属性

Returned Value:

0：成功
错误代码：失败

5.8.38 pthread_condattr_destroy

Function Prototype:

```
#include <pthread.h>
int pthread_condattr_destroy(pthread_condattr_t *attr);
```

Description:

销毁条件变量属性

Input Parameters:

- attr：属性

Returned Value:

0：成功
错误代码：失败

5.8.39 pthread_cond_init

Function Prototype:

```
#include <pthread.h>
int pthread_cond_init(pthread_cond_t *cond, pthread_condattr_t *attr);
```

Description:

初始化条件变量

Input Parameters:

- cond：变量

attr：变量属性

Returned Value:

0：成功
错误代码：失败

Alibaba Confidential

 Copyright© 2016 Alibaba

5.8.40 pthread_cond_destroy

Function Prototype:

```
#include <pthread.h>
int pthread_cond_destroy(pthread_cond_t *cond);
```

Description:

销毁条件变量

Input Parameters:

- cond：变量

Returned Value:

0：成功
错误代码：失败

5.8.41 pthread_cond_broadcast

Function Prototype:

```
#include <pthread.h>
int pthread_cond_broadcast(pthread_cond_t *cond);
```

Description:

将所有等待该条件变量的线程解锁

Input Parameters:

- cond：变量

Returned Value:

0：成功
错误代码：失败

5.8.42 pthread_cond_signal

Function Prototype:

```
#include <pthread.h>
int pthread_cond_signal(pthread_cond_t *cond);
```

Description:

发送一个信号给另外一个正在处于阻塞等待状态的线程,使其脱离阻塞

Alibaba Confidential

 Copyright© 2016 Alibaba

状态,继续执行

Input Parameters:

- cond：变量

Returned Value:

0：成功
错误代码：失败

5.8.43 pthread_cond_wait

Function Prototype:

```
#include <pthread.h>
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
```

Description:

等待条件变量

Input Parameters:

- cond：条件变量

mutex：互斥锁

Returned Value:

0：成功
错误代码：失败

5.8.44 pthread_cond_timedwait

Function Prototype:

```
#include <pthread.h>
int pthread_cond_timedwait(pthread_cond_t *cond,
                           pthread_mutex_t *mutex,
                           const struct timespec *abstime);
```

Description:

计时等待条件变量

Input Parameters:

- cond：条件变量

mutex：互斥锁
abstime：时间

Alibaba Confidential

 Copyright© 2016 Alibaba

Returned Value:

0：成功
错误代码：失败

5.8.45 pthread_barrierattr_init

Function Prototype:

#include <pthread.h>
int pthread_barrierattr_init(FAR pthread_barrierattr_t *attr);

Description:

初始化计数锁属性

Input Parameters:

- attr. 计数锁属性

Returned Value:

0 成功
EINVAL ： 无效参数

5.8.46 pthread_barrierattr_destroy

Function Prototype:

#include <pthread.h>
int pthread_barrierattr_destroy(FAR pthread_barrierattr_t *attr);

Description:

释放 init 申请的资源

Input Parameters:

- attr. 计数锁属性

Returned Value:

0 ： 成功
EINVAL ： 无效的参数

5.8.47 pthread_barrierattr_setpshared

Function Prototype:

#include <pthread.h>

Alibaba Confidential

 Copyright© 2016 Alibaba

int pthread_barrierattr_setpshared(FAR pthread_barrierattr_t *attr,
int pshared);

Description:

设置计数锁的范围属性

Input Parameters:

- attr. 属性
- pshared. 计数锁范围

Returned Value:

0 ： 成功
EINVAL ： 无效的参数

5.8.48 pthread_barrierattr_getpshared

Function Prototype:

#include <pthread.h>
int pthread_barrierattr_getpshared(FAR const pthread_barrierattr_t *attr,
FAR int *pshared);

Description:

获取计数锁范围属性

Input Parameters:

- attr. 属性
- pshared. 计数锁范围

Returned Value:

0 ： 成功
EINVAL：无效的参数

5.8.49 pthread_barrier_init

Function Prototype:

#include <pthread.h>
int pthread_barrier_init(FAR pthread_barrier_t *barrier,
FAR const pthread_barrierattr_t *attr, unsigned int count);

Description:

初始化计数锁

Alibaba Confidential

 Copyright© 2016 Alibaba

Input Parameters:

- barrier. 要初始化的计数锁
- attr. 计数锁属性
- count. 要等待的线程个数

Returned Value:

0 ：成功
错误代码：失败

- EAGAIN. 没有资源初始化
- EINVAL. 无效的参数
- ENOMEM. 内存不足
- EBUSY. 在使用是初始化

5.8.50 pthread_barrier_destroy

Function Prototype:

```
#include <pthread.h>
int pthread_barrier_destroy(FAR pthread_barrier_t *barrier);
```

Description:

释放计数器申请的资源

Input Parameters:

- barrier. 要释放的计数器

Returned Value:

0 ：成功

- EBUSY. 正在使用
- EINVAL. 无效的参数

5.8.51 pthread_barrier_wait

Function Prototype:

```
#include <pthread.h>
int pthread_barrier_wait(FAR pthread_barrier_t *barrier);
```

Description:

检查是否所以函数都已经完成

Input Parameters:

Alibaba Confidential

 Copyright© 2016 Alibaba

- barrier. 要等待的计数器

Returned Value:

0 ： 成功
EINVAL ： 无效的计数器

5.8.52 pthread_once

Function Prototype:

```
#include <pthread.h>
int pthread_once(FAR pthread_once_t *once_control,
                 CODE void (*init_routine)(void));
```

Description:

仅实现一次

Input Parameters:

- once_control. 实行一次的函数
- init_routine. 初始化例程

Returned Value:

0 ： 成功
EINVAL ： 无效的参数

5.8.53 pthread_kill

Function Prototype:

```
#include <signal.h>
#include <pthread.h>
int pthread_kill(pthread_t thread, int signo)
```

Description:

向某个线程传递一个信号

Input Parameters:

- thread. 线程
- signo. 要传递的信号

Returned Value:

0：成功
错误代码：失败

Alibaba Confidential

 Copyright© 2016 Alibaba

- EINVAL. 一个无效的信号
- EPERM. 没有足够的权限
- ESRCH. 找不到线程
- ENOSYS. 不支持向进程发送信号

5.8.54 pthread_sigmask

Function Prototype:

```
#include <signal.h>
#include <pthread.h>
int pthread_sigmask(int how, FAR const sigset_t *set, FAR sigset_t *oset);
```

Description:

在主调线程里控制信号掩码

Input Parameters:

- how. 如何修改

SIG_BLOCK: 结果集是当前集合参数集的并集集的差集
SIG_UNBLOCK: 结果集是当前集合参数集的差集
SIG_SETMASK: 结果集是由参数集指向的集•
set. 新信号掩模的位置

- oset. 旧信号掩模存储的位置

Returned Value:

0 ： 成功
EINVAL ： how 是无效的参数

5.9 File System Interfaces

5.9.1 Driver Operations

1. **Function Prototype:**

```
#include <fcntl.h>
int open(const char *path, int oflag, ...);
```

2. **Function Prototype:**

```
#include <unistd.h>
int close(int fd);
int dup(int fildes);
int dup2(int fildes1, int fildes2);
```

Alibaba Confidential

 Copyright© 2016 Alibaba

off_t lseek(int fd, off_t offset, int whence);
ssize_t read(int fd, void *buf, size_t nbytes);
int unlink(const char *path);
ssize_t write(int fd, const void *buf, size_t nbytes);

3. **Function Prototype:**

#include <sys/ioctl.h>
int ioctl(int fd, int req, unsigned long arg);

4. **Function Prototype:**

#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);

Description:

把当前的文件指针挂到设备内部定义的等待队列中

Input Parameters:

- fds. 存放需要检测其状态的 Socket 描述符
- nfds. 用于标记数组 fds 中的结构体元素的总数量
- timeout. poll 函数调用阻塞的时间，单位：毫秒

Returned Value:

文件描述符
-1：失败•
EBADF. 又一个无效的文件
- EFAULT. 文件描述符地址是无效的
- EINTR. 在请求发生之前有信号产生.
- EINVAL. 值太大
- ENOMEM. 内存不足
- ENOSYS. 文件不支持

10.1.5. sys/select.h
10.1.5.1. select

5. **Function Prototype:**

#include <sys/select.h>
int select(int nfds, FAR fd_set *readfds, FAR fd_set *writefds,
FAR fd_set *exceptfds, FAR struct timeval *timeout);

Description:

Alibaba Confidential

 Copyright© 2016 Alibaba

确定一个或多个套接口的状态，如：需要则等待

Input Parameters:

- nfds. 是一个整数值，是指集合中所有文件描述符的范围，即所有文件描述符的最大值加 1，不能错！在 Windows 中这个参数的值无所谓，可以设置不正确
- readfds. （可选）指针，指向一组等待可读性检查的套接口• writefds. （可选）指针，指向一组等待可写性检查的套接口
- exceptfds. （可选）指针，指向一组等待错误检查的套接口
- timeout. select()最多等待时间，对阻塞操作则为 NULL.

Returned Value:

- 0: 超时
- >0: 3 个描述符集中的一个
- -1: 所有描述符集清 0

5.9.2 Directory Operations

Function Prototype:

```
#include <dirent.h>
int closedir(DIR *dirp);
FAR DIR *opendir(const char *path);
FAR struct dirent *readdir(FAR DIR *dirp);
int readdir_r(FAR DIR *dirp, FAR struct dirent *entry,
              FAR struct dirent **result);
void rewinddir(FAR DIR *dirp);
void seekdir(FAR DIR *dirp, int loc);
int telldir(FAR DIR *dirp);
```

5.9.3 UNIX Standard Operations

Function Prototype:

```
#include <unistd.h>
pid_t getpid(void);
void _exit(int status) noreturn_function;
unsigned int sleep(unsigned int seconds);
void usleep(unsigned long usec);
int close(int fd);
int dup(int fd);
```

Alibaba Confidential

 Copyright© 2016 Alibaba

```
int dup2(int fd1, int fd2);
int fsync(int fd);
off_t lseek(int fd, off_t offset, int whence);
ssize_t read(int fd, FAR void *buf, size_t nbytes);
ssize_t write(int fd, FAR const void *buf, size_t nbytes);
int pipe(int filedes[2]);
int chdir(FAR const char *path);
FAR char *getcwd(FAR char *buf, size_t size);
int unlink(FAR const char *pathname);
int rmdir(FAR const char *pathname);
int getopt(int argc, FAR char *const argv[], FAR const char *optstring);
```

5.9.4 Standard I/O

Function Prototype:

```
#include <stdio.h>
int fclose(FILE *stream);
int fflush(FILE *stream);
FILE *fdopen(int fd, const char *type);
int feof(FILE *stream);    /* Prototyped but not implemented */
int ferror(FILE *stream); /* Prototyped but not implemented */
int fileno(FAR FILE *stream);
int fgetc(FILE *stream);
int fgetpos(FILE *stream, fpos_t *pos);
char *fgets(char *s, int n, FILE *stream);
FILE *fopen(const char *path, const char *type);
int fprintf(FILE *stream, const char *format, ...);
int fputc(int c, FILE *stream);
int fputs(const char *s, FILE *stream);
size_t fread(void *ptr, size_t size, size_t n_items, FILE *stream);
int fseek(FILE *stream, long int offset, int whence);
int fsetpos(FILE *stream, fpos_t *pos);
long ftell(FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t n_items, FILE *stream);
char *gets(char *s);
int printf(const char *format, ...);
int puts(const char *s);
int rename(const char *source, const char *target);
```

Alibaba Confidential

 Copyright© 2016 Alibaba

```
int snprintf(FAR char *buf, size_t size, const char *format, ...);
int sprintf(char *dest, const char *format, ...);
int sscanf(const char *buf, const char *fmt, ...);
int ungetc(int c, FILE *stream);
int vprintf(const char *s, va_list ap);
int vfprintf(FILE *stream, const char *s, va_list ap);
int vsnprintf(FAR char *buf, size_t size, const char *format, va_list ap);
int vsscanf(char *buf, const char *s, va_list ap);
int vsprintf(char *buf, const char *s, va_list ap);

#include <sys/stat.h>
int mkdir(FAR const char *pathname, mode_t mode);
int mkfifo(FAR const char *pathname, mode_t mode);
int stat(const char *path, FAR struct stat *buf);
int fstat(int fd, FAR struct stat *buf);

#include <sys/statfs.h>
int statfs(const char *path, struct statfs *buf);
int fstatfs(int fd, struct statfs *buf);
```

5.9.5 Standard String Operations

Function Prototype:

```
#include <string.h>
char *strchr(const char *s, int c);
FAR char *strdup(const char *s);
const char *strerror(int);
size_t strlen(const char *);
char *strcat(char *, const char *);
char *strncat(char *, const char *, size_t);
int strcmp(const char *, const char *);
int strncmp(const char *, const char *, size_t);
int strcasecmp(const char *, const char *);
int strncasecmp(const char *, const char *, size_t);
char *strcpy(char *dest, const char *src);
char *strncpy(char *, const char *, size_t);
char *strpbrk(const char *, const char *);
char *strchr(const char *, int);
```

Alibaba Confidential

 Copyright© 2016 Alibaba

```
char *strrchr(const char *, int);
size_t strspn(const char *, const char *);
size_t strcspn(const char *, const char *);
char *strstr(const char *, const char *);
char *strtok(char *, const char *);
char *strtok_r(char *, const char *, char **);
void *memset(void *s, int c, size_t n);
void *memcpy(void *dest, const void *src, size_t n);
int memcmp(const void *s1, const void *s2, size_t n);
void *memmove(void *dest, const void *src, size_t count);
# define bzero(s,n) (void)memset(s,0,n)
```

5.9.6 Pipes and FIFOs

1. **Function Prototype:**

   ```
   #include <unistd.h>
   int pipe(int filedes[2]);
   ```

   Description:

   会建立管道,并将文件描述词由参数 filedes 数组返回

   Input Parameters:

   - filedes[2]. 管道

   Returned Value:

   0 ：成功
   -1：失败

2. **Function Prototype:**

   ```
   #include <sys/stat.h>
   int mkfifo(FAR const char *pathname, mode_t mode);
   ```

   Description:

   依参数 pathname 建立特殊的 FIFO 文件

   Input Parameters:

   - pathname. 对 FIFO 的实例的完整路径连接或创建
   - mode. 该文件权限

   Returned Value:

Alibaba Confidential

 Copyright© 2016 Alibaba

0：成功
-1 ： 失败

5.9.7 FAT File System Support

Function Prototype:

```
#include <nuttx/mkfatfs.h>
int mkfatfs(FAR const char *pathname, FAR struct fat_format_s *fmt);
```

Description:

通过路径格式化一个文件系统映像

Input Parameters:

- pathname .对 FIFO 的实例的完整路径连接或创建
- fmt .一个提供的结构实例的引用

Returned Value:

0：成功
-1：失败
ENOENT：路径不是指向系统
ENOTBLK 路径不是指向一个块设备驱动程序
EACCESS 块设备模块不允许被写入

5.9.8 mmap() and eXecute In Place (XIP)

Function Prototype:

```
#include <sys/mman.h>
FAR void *mmap(FAR void *start, size_t length, int prot, int flags,
               int fd, off_t offset)
```

Description:

将一个文件或者其它对象映射进内存

Input Parameters:

- start 映射区的开始地址，设置为 0 时表示由系统决定映射区的起始地址。
- length 映射区的长度。//长度单位是 以字节为单位，不足一内存页按一内存页处理
- prot 期望的内存保护标志，不能与文件的打开模式冲突

PROT_NONE - 页不可访问

Alibaba Confidential

 Copyright© 2016 Alibaba

PROT_READ - 页内容可以被读取

PROT_WRITE - 页可以被写入.

PROT_EXEC - 页内容可以被执行

- flags 指定映射对象的类型，映射选项和映射页是否可以共享

MAP_SHARED - 与其它所有映射这个对象的进程共享映射空间

MAP_PRIVATE - 建立一个写入时拷贝的私有映射

MAP_FIXED - 使用指定的映射起始地址，如果由 start 和 len 参数指定的内存区重叠于现存的映射空间，重叠部分将会被丢弃。如果指定的起始地址不可用，操作将会失败。并且起始地址必须落在页的边界上

MAP_FILE - 兼容标志，被忽略

MAP_ANONYMOUS - 匿名映射，映射区不与任何文件关联

MAP_ANON - MAP_ANONYMOUS 的别称，不再被使用

MAP_GROWSDOWN - 用于堆栈，告诉内核 VM 系统，映射区可以向下扩展

MAP_DENYWRITE - 这个标志被忽略

MAP_EXECUTABLE - 这个标志被忽略

MAP_LOCKED - 锁定映射区的页面，从而防止页面被交换出内存

MAP_NORESERVE - 不要为这个映射保留交换空间

MAP_POPULATE - 为文件映射通过预读的方式准备好页表

MAP_NONBLOCK - 仅和 MAP_POPULATE 一起使用时才有意义

- fd 有效的文件描述词
- offset 被映射对象内容的起点.

Returned Value:

返回被映射区的指针

错误代码

EACCES：访问出错

EAGAIN：文件已被锁定，或者太多的内存已被锁定

EBADF：fd 不是有效的文件描述词

EINVAL：一个或者多个参数无效

ENFILE：已达到系统对打开文件的限制

ENODEV：指定文件所在的文件系统不支持内存映射

ENOMEM：内存不足，或者进程已超出最大内存映射数量

EPERM：权能不足，操作不允许

ETXTBSY：已写的方式打开文件，同时指定 MAP_DENYWRITE 标志

SIGSEGV：试着向只读区写入

SIGBUS：试着访问不属于进程的内存区

Alibaba Confidential

 Copyright© 2016 Alibaba

5.10 Network Interfaces

5.10.1 socket

Function Prototype:

#include <sys/socket.h>
int socket(int domain, int type, int protocol);

Description:

socket()创建了一个能够进程网络通信的套接字。

Input Parameters:

domain: 一个地址描述
type: 指定 socket 类型
protocol:指定协议

Returned Value:

若无错误发生，socket()返回引用新套接口的描述字
EACCES:协议被拒绝
EAFNOSUPPORT：指定的地址不被支持
EINVAL：未知的协议
EMFILE：进程文件表溢出
ENFILE：系统对已打开的文件总数限制
ENOBUFS or ENOMEM ：内存不足
EPROTONOSUPPORT：协议不支持这个领域

5.10.2 bind

Function Prototype:

#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

Description:

bind()给了套接口 sockfd 一个地址 addr。addr 的长度为 addrlen。这就是一个套接口绑定一个名字。当用 socket()新建一个套接口时，他就会存在于一个空间(address family)，但没有指定名称。

Input Parameters:

sockfd: 标识一未捆绑套接口的描述字

Alibaba Confidential

 Copyright© 2016 Alibaba

addr: 赋予套接口的地址
addrlen: 地址长度

Returned Value:

0：成功
-1：失败
EACCES：使用者不是超级用户
EADDRINUSE：给的地址已经被使用
EBADF：套接口不是一个有效地描述符
EINVAL：套接口已经绑定到一个地址上了
ENOTSOCK：socked 不是一个套接口

5.10.3 connect

Function Prototype:

```
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Description:

建立与指定 socket 的连接。通过 addr 和指定的文件描述符 sockfd 跟 socket 建立连接，addrlen 表示指定 addr 的长度。如果指定文件描述符 sockfd 是 SOCK_DGRAM，将会被默认发送 addr，只有指定的地址才能接收到。如果 socket 是 SOCK_STREAM 或者 SOCK_SEQPACKET 之一的类型，将会通过 addr 在 socket 和地址之间直接建立一个连接。基于连接协议套接口只会成功连接一次。未连接的套接口将会通过连接一个地址和 sa_family 成员的 sockaddr 设置成 AF_UNSPEC.

Input Parameters:

sockfd: 标识一个未连接 socket
addr: 指向要连接套接字的 sockaddr 结构体的指针
addrlen: sockaddr 结构体的字节长度

Returned Value:

0：成功
-1：失败
EACCES or EPERM：被防火墙阻拦
EADDRINUSE：本地地址已经被使用
EAFNOSUPPORT：没有正确的地址。
EAGAIN：缓存不足
EALREADY：一个阻塞套接口调用正在运行中。

Alibaba Confidential

 Copyright© 2016 Alibaba

EBADF：文件描述符不是一个有效的。
ECONNREFUSED：连接尝试被强制拒绝。
EFAULT：一个参数指定一个无效的用户空间地址。
EINPROGRESS：套接非阻塞连接不能完成。
EINTER：被别的中断程序打断。
EISCONN：套接口早已连接。
ENETUNREACH：当前无法从本主机访问网络。
ENOTSOCK：描述字不是一个套接口。
ETIMEOUT：超时时间到。

5.10.4 listen

Function Prototype:

#include <sys/socket.h>
int listen(int sockfd, int backlog);

Description:

listen()函数使用主动连接 socket 变为被连接 socket，使得一个进程可以接受其他进程的请求，从而成为一个服务器进程。他一般在调用 bind 之后-调用 accept 之前调用。listen()调用仅适用于 SOCK_STREAM 或者 SOCK_SEQPACKET.

Input Parameters:

sockfd: 用于标识一个已捆绑未连接套接口的描述字
backlog: 等待连接队列的最大长度

Returned Value:

0：成功
-1：失败
EADDRINUSE: 套接口正在运行
EBADF: 无效的描述符
ENOTSOCK: 不是一个套接口
EOPNOTSUPP: 不允许被访问

5.10.5 accept

Function Prototype:

#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

Alibaba Confidential

 Copyright© 2016 Alibaba

Description:

accept()用在在一个 socket 上建立连接(SOCK_STREAM, SOCK_SEQPACKET, SOCK_RDM)。从 socket 的等待连接队列中抽取第一个连接，创建一个与 socket 同类的新的 socket 并返回句柄。已接收连接的 socket 不能用于接受新的连接，原 socket 仍保持开放。

socket 参数是 socket()建立的一个 socket 描述符，通过 bind()绑定到本地地址，在调用 listen()后监听连接。

addr 参数的实际格式由通讯时参数的地址确定，addrlen 在调用时初始化为 addr 所指的地址空间；在调用结束时它包含了实际返回的地址长度。

如果队列中无等待连接，且 socket 为非阻塞方式，则 accept 阻塞调用进程直至新的连接出现。如果 socket 为非阻塞方式且队列中无等待连接，accept 返回 EAGAIN。

Input Parameters:

sockfd: 套接字描述符，该套接口在 listen()后监听连接

addr:(可选）指针，指向一缓冲区，其中接收为通讯层所知的连接实体的地址。Addr 参数的实际格式由套接口创建时所产生的地址族确定

addrlen:（可选）指针，输入参数，配合 addr 一起使用，指向存有 addr 地址长度的整型数。

Returned Value:

如果没有错误产生，则 accept()返回一个描述所接受包的 SOCKET 类型的值

EAGAIN or EWOULDBLOCK：请求被阻止

EBADF 指定了无效的文件描述符

ENOTSOCK 参数不是一个套接口

EOPNOTSUPP 该套接口类型不支持面向连接服务。EINTR 数据传输前有信号发送

ECONNABORTED：已终止连接。

EINVAL 无效的参数传输

EMFILE：打开文件有限制

EFAULT 一个参数指定一个无效的用户空间地址。

ENOBUFS or ENOMEM：没有足够的内存

EPROTO：协议错误

EPERM：防火墙禁止连接

5.10.6 send

Function Prototype:

Alibaba Confidential

 Copyright© 2016 Alibaba

#include <sys/socket.h>
ssize_t send(int sockfd, const void *buf, size_t len, int flags);

Description:

向一个已经连接的 socket 发送数据。跟 write()的区别就在于 flags，当 flags 为 0 时，send()就等效于 write()了，send(s, buf, len, flags)等效于 sendto(s, buf, len, flags, NULL, 0)

Input Parameters:

sockfd:一个用于标识已连接套接口的描述字。
buf: 包含待发送数据的缓冲区。
len: 缓冲区中数据的长度。
flags: 调用执行方式.

Returned Value:

成功则返回实际传送出去的字符数，失败返回－1。
EAGAIN or EWOULDBLOCK 请求被阻止
EBADF 指定了无效的文件描述符
ECONNRESET 连接复位
EDESTADDRREQ 没有对应的地址设置
EFAULT 一个参数指定一个无效的用户空间地址。
EINTR 数据传输前有信号发送
EINVAL 无效的参数传输
EISCONN 找不到指定的地址
EMSGSIZE 大小不可能
ENOBUFS 没有足够的系统资源
ENOMEM 没有足够的内存
ENOTCONN 套接口没有连接
ENOTSOCK 参数不是一个套接口
EOPNOTSUPP 该套接口类型不支持面向连接服务。
EPIPE 本地段已经关闭

Alibaba Confidential

5.10.7 sendto

Function Prototype:

#include <sys/socket.h>
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
const struct sockaddr *to, socklen_t tolen);

Description:

 Copyright© 2016 Alibaba

向 socket 发送数据。如果 sendto()用在(SOCK_STREAM, SOCK_SEQPACKET)连接模式上，参数 to 和 tolen 就可以忽略了。(当返回值不是 NULL 或 0 时，返回的错误值是 EISCONN)，当 socket 不能连接时返回的错误值为 ENOTCONN。

Input Parameters:

sockfd: 套接口字符
buf: 待发送数据的缓冲区
len: 缓冲区长度
flags: 调用方式标志位, 一般为 0, 改变 Flags，将会改变 Sendto 发送的形式

- to: （可选）指针，指向目的套接字的地址•
  tolen: 地址长度

Returned Value:

成功则返回实际传送出去的字符数，失败返回－1。
EAGAIN or EWOULDBLOCK 请求被阻止
EBADF 指定了无效的文件描述符
ECONNRESET 连接复位
EDESTADDRREQ 没有对应的地址设置
EFAULT 一个参数指定一个无效的用户空间地址。
EINTR 数据传输前有信号发送
EINVAL 无效的参数传输
EISCONN 找不到指定的地址
EMSGSIZE 大小不可能
ENOBUFS 没有足够的系统资源
ENOMEM 没有足够的内存
ENOTCONN 套接口没有连接
ENOTSOCK 参数不是一个套接口
EOPNOTSUPP 参数不合适
EPIPE 本地段已经关闭

5.10.8 recv

Function Prototype:

```
#include <sys/socket.h>
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

Description:

Alibaba Confidential

 Copyright© 2016 Alibaba

recv()从 socket 接收信息，不管是否定向连接他都可以在 socket 上接收数据。

Input Parameters:

- sockfd: 套接口字符

buf: 待发送数据的缓冲区

len: 缓冲区长度

flags: 调用方式标志位，一般为 0，改变 flags，将会改变 Sendto 发送的形式

Returned Value:

若无错误发生，recv()返回读入的字节数。如果连接已中止，返回 0。

错误代码：

EAGAIN：超时

EBADF：socked 不是一个有效地描述符。

ECONNREFUSED：远程主机拒绝网络连接

EFAULT：缓存区指针在地址空间外部。

EINTR：被别的中断程序打断

EINVAL：无效的套接传递。

ENOMEM:内存空间不足

ENOTCONN：协议没有被连接

ENOTSOCK:socked 不是一个套接口

5.10.9 recvfrom

Function Prototype:

```
#include <sys/socket.h>
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *from, socklen_t *fromlen);
```

Description:

recvfrom()从 socket 接收信息，不管是否定向连接他都可以在 socket 上接收数据。如果 from 不是 NULL，并且底层协议提供了源地址，这个源地址会将 from 填满。这个 fromlen 初始化了 from 的缓存区大小，多个实际地址长度储存在这个缓存区里。

Input Parameters:

- sockfd:标识一个已连接套接口的描述字•

buf: 接收数据缓冲区

- len: 缓冲区长度

Alibaba Confidential

 Copyright© 2016 Alibaba

- flags: 调用操作方式。
- from: （可选）指针，指向装有源地址的缓冲区。•

fromlen: （可选）指针，指向 from 缓冲区长度值

Returned Value:

若无错误发生，recvfrom()返回读入的字节数。如果连接已中止，返回 0。
错误代码：
EAGAIN：超时
EBADF：socked 不是一个有效地描述符。
ECONNREFUSED：远程主机拒绝网络连接
EFAULT：缓存区指针在地址空间外部。
EINTR：被别的中断程序打断
EINVAL：无效的套接传递。
ENOMEM:内存空间不足
ENOTCONN：协议没有被连接
ENOTSOCK:socked 不是一个套接口

5.10.11 setsockopt

Function Prototype:

```
#include <sys/socket.h>
int setsockopt(int sockfd, int level, int option, const void *value,
               socklen_t value_len);
```

Description:

setsockopt()通过 option 参数设置特殊的选项，通过 level 参数决定特殊的协议下，通过 value 参数指向选项值的缓存区，通过 sockfd 参数标示一个套接字的描述符。

level 参数指定了在特定协议下的选项。要在 socket 的缓存区设置选项，level 参数就要是 SOL_SOCKET。

Input Parameters:

- sockfd: 标识一个套接字的描述符
- level: 选项定义的层次；目前仅支持 SOL_SOCKET 和 IPPROTO_TCP 层次。• option: 需设置的选项
- value: 指针，指向存放选项值的缓冲区•

value_len: 属性长度

Returned Value:

若无错误发生，setsockopt（）返回 0。否则的话，

Alibaba Confidential

 Copyright© 2016 Alibaba

错误代码：
BADF：sockfd 不是一个有效地套接口描述符。
DOM：发送和接收的超时值太大
INVAL：指定的选项无效
ISCONN：无法在套接字连接时设置。
NOPROTOOPT：该选项不支持该协议
NOTSOCK：socked 不是一个套接口。
NOMEM：系统内存不够
NOBUFS：系统资源不够

5.10.12 getsockopt

Function Prototype:

```
#include <sys/socket.h>
int getsockopt(int sockfd, int level, int option, void *value,
                    socklen_t *value_len);
```

Description:

getsockopt()获取任意类型、任意状态套接口的选项当前值。如果选项值得长度大于 valen_len 被选中选项的值会放在 value 缓存区中。value_len 所指向的整形数在初始时包含缓存区的长度，在调用返回时被置为实际 value 的长度。

level 参数指定了特定协议下的选项。要在 socket 下获取选项，就得使用 SOL_SOCKET。

Input Parameters:

- sockfd 一个标识套接口的描述字。
- level 选项定义的层次。支持的层次仅有 SOL_SOCKET 和 IPPROTO_TCP。• option 需获取的套接口选项• value 指向存放所获得选项值的缓冲区。
- value_len 指向缓存区的长度值

Returned Value:

若无错误发生，getsockopt()返回 0。
错误代码：
BADF：socked 不是一个有效地套接字描述符
INVAL：指定的选项无效
NOPROTOOPT：该选项不支持该协议
NOTSOCK：描述字不是一个套接口。

Alibaba Confidential

 Copyright© 2016 Alibaba

NOBUFS：系统中资源不足

Alibaba Confidential

 Copyright© 2016 Alibaba

6 Posix

Interface	YoC posix (*)
#include <unistd.h>	
getpid( )	*
close( )	*
chdir( )	*
dup( )	*
dup2( )	*
execl( )	*
execv( )	*
lseek( )	*
read( )	*
unlink( )	*
write( )	*
_exit( )	*
sleep( )	*
usleep( )	*
sync( )	
fsync( )	*
fdatasync( )	
pipe( )	*
pause( )	*
pread( )	*
pwrite( )	*
chdir( )	
getcwd( )	*
rmdir( )	*
getopt( )	*
getoptargp( )	*
getoptindp( )	*
getoptoptp( )	*
gethostname( )	*
sethostname( )	*
vfork( )	*
fork( )	
access( )	*

 Copyright© 2016 Alibaba

#include <poll.h>	
poll( )	*
#include <fcntl.h>	
open ( )	*
fcntl( )	*
creat( )	*
#include <sys/select.h>	
select( )	*
#include <syslog.h>	
syslog()	*
#include <time.h>	
asctime( )	*
asctime_r( )	*
ctime( )	*
ctime_r( )	*
clock_gettime( )	*
clock_settime( )	*
clock_getres( )	*
mktime( )	*
nanosleep( )	*
gmtime( )	*
localtime( )	*
localtime_r( )	
gmtime( )	*
gmtime_r( )	*
strftime( )	*
time( )	*
timer_create( )	*
timer_delete( )	*
timer_settime( )	*

Alibaba

Copyright© 2016 Alibaba

timer_gettime( )	*
timer_getoverrun( )	*
times( )	
tzset( )	*
#include <mqueue.h>	
mq_close( )	*
mq_getattr( )	*
mq_notify( )	*
mq_open	*
mq_receive( )	*
mq_send( )	*
mq_setattr( )	*
mq_timedreceive( )	*
mq_timedsend( )	
mq_unlink( )	*
mq_inode_release( )	*
#include <pthread.h>	
pthread_atfork( )	
pthread_attr_destroy( )	*
pthread_attr_getdetachstate( )	
pthread_attr_getguardsize( )	
pthread_attr_getinheritsched( )	*
pthread_attr_getschedparam( )	*
pthread_attr_getschedpolicy( )	*
pthread_attr_getscope( )	
pthread_attr_getstack( )	
pthread_attr_getstackaddr( )	
pthread_attr_getstacksize( )	*
pthread_attr_init( )	*
pthread_attr_setdetachstate( )	
pthread_attr_setcancelstate( )	*
pthread_attr_setguardsize( )	
pthread_attr_setinheritsched( )	*
pthread_attr_setschedparam( )	*
pthread_attr_setschedpolicy( )	*

Alibaba Confidential

Copyright© 2016 Alibaba

pthread_attr_setscope( )	
pthread_attr_setstack( )	
pthread_attr_setstackaddr( )	
pthread_attr_setstacksize( )	*
pthread_barrier_destroy( )	*
pthread_barrier_init( )	*
pthread_barrier_wait( )	*
pthread_barrierattr_destroy( )	*
pthread_barrierattr_getpshared( )	*
pthread_barrierattr_init( )	*
pthread_barrierattr_setpshared( )	*
pthread_cancel( )	*
pthread_cleanup_pop( )	
pthread_cleanup_push( )	
pthread_cond_broadcast( )	*
pthread_cond_destroy( )	*
pthread_cond_init( )	*
pthread_cond_signal( )	*
pthread_cond_timedwait( )	*
pthread_cond_wait( )	*
pthread_condattr_destroy( )	
pthread_condattr_getclock( )	
pthread_condattr_getpshared( )	
pthread_condattr_init( )	*
pthread_condattr_setclock( )	
pthread_condattr_setpshared( )	
pthread_create( )	*
pthread_detach( )	*
pthread_equal( )	*
pthread_exit( )	*
pthread_getconcurrency( )	
pthread_getspecific( )	*
pthread_getschedparam( )	*
pthread_join( )	*
pthread_key_create( )	*
pthread_key_delete( )	*
pthread_kill( )	*
pthread_mutex_destroy( )	*
pthread_mutex_getprioceiling( )	
pthread_mutex_init( )	*
pthread_mutex_lock( )	*
pthread_mutex_setprioceiling( )	
pthread_mutex_timedlock( )	

Alibaba Confidential

Copyright© 2016 Alibaba

pthread_mutex_trylock( )	*
pthread_mutex_unlock( )	*
pthread_mutexattr_destroy( )	*
pthread_mutexattr_getprioceiling( )	
pthread_mutexattr_getprotocol( )	
pthread_mutexattr_getpshared( )	*
pthread_mutexattr_gettype( )	*
pthread_mutexattr_init( )	*
pthread_mutexattr_setprioceiling( )	
pthread_mutexattr_setprotocol( )	
pthread_mutexattr_setpshared( )	*
pthread_mutexattr_settype( )	*
pthread_once( )	*
pthread_self( )	*
pthread_setcancelstate( )	*
pthread_setcanceltype( )	
pthread_setconcurrency( )	
pthread_setspecific( )	*
pthread_setschedparam( )	*
pthread_setschedprio( )	*
pthread_sigmask( )	*
pthread_testcancel( )	*
pthread_yield( )	*
#include <sched.h>	
sched_get_priority_max( )	*
sched_get_priority_min( )	*
sched_getparam( )	*
sched_getscheduler( )	*
sched_fork( )	
sched_dead( )	
sched_rr_get_interval( )	*
sched_setparam( )	*
sched_setscheduler( )	*
sched_setscheduler_nocheck( )	
sched_yield( )	*
sched_lock( )	*
sched_unlock( )	*
sched_lockcount( )	*
sched_note_start( )	*

Alibaba Confidential

 Copyright© 2016 Alibaba

sched_note_stop( )	*
sched_note_switch( )	*
#include <semaphore.h>	
sem_close( )	*
sem_destroy( )	*
sem_getvalue( )	*
sem_init( )	*
sem_open( )	*
sem_timedwait( )	*
sem_post( )	*
sem_trywait( )	*
sem_unlink( )	*
sem_wait( )	*
semctl( )	
semget( )	
semop( )	
#include <signal.h>	
sigaction( )	*
sigaddset( )	*
sigaltstack( )	
sigdelset( )	*
sigemptyset( )	*
sigfillset( )	*
sighold( )	*
sigignore( )	
siginterrupt( )	
sigismember( )	*
signal( )	
signbit( )	
signgam( )	
signpause( )	*
sigpending( )	*
sigprocmask( )	*
sigqueue( )	*
sigrelse( )	*
sigset( )	
sigsetjmp( )	

Alibaba Confidential

 Copyright© 2016 Alibaba

sigsuspend( )	*
sigtimedwait( )	*
sigwait( )	
sigwaitinfo( )	*
kill( )	*
#include <sys/socket.h>	
socket ( )	*
bind ( )	*
connect ( )	*
listen ( )	*
accept ( )	*
send( )	*
sendto ( )	*
shutdown( )	*
sendmsg( )	
recv ( )	*
recvfrom ( )	*
setsockopt ( )	*
getsockopt ( )	*
getsockname( )	*
Interface	**YoC posix (*)**
#include <unistd.h>	
getpid( )	*
close( )	*
chdir( )	*
dup( )	*
dup2( )	*
execl( )	*
execv( )	*
lseek( )	*
read( )	*
unlink( )	*
write( )	*
_exit( )	*
sleep( )	*
usleep( )	*
sync( )	
fsync( )	*

Confidential

Copyright© 2016 Alibaba

fdatasync( )	
pipe( )	*
pause( )	*
pread( )	*
pwrite( )	*
chdir( )	
getcwd( )	*
rmdir( )	*
getopt( )	*
getoptargp( )	*
getoptindp( )	*
getoptoptp( )	*
gethostname( )	*
sethostname( )	*
vfork( )	*
access( )	*
#include <poll.h>	
poll( )	*
#include <fcntl.h>	
open ( )	*
fcntl( )	*
creat( )	*
#include <sys/select.h>	
select( )	*
#include <time.h>	
asctime( )	*
asctime_r( )	*
ctime( )	*

 Copyright© 2016 Alibaba

ctime_r( )	*
clock_gettime( )	*
clock_settime( )	*
clock_getres( )	*
mktime( )	*
nanosleep( )	*
gmtime( )	*
localtime( )	*
localtime_r( )	
gmtime( )	*
gmtime_r( )	*
strftime( )	*
time( )	*
timer_create( )	*
timer_delete( )	*
timer_settime( )	*
timer_gettime( )	*
timer_getoverrun( )	*
times( )	
tzset( )	*
#include <mqueue.h>	
mq_close( )	*
mq_getattr( )	*
mq_notify( )	*
mq_open	*
mq_receive( )	*
mq_send( )	*
mq_setattr( )	*
mq_timedreceive( )	*
mq_timedsend( )	
mq_unlink( )	*
mq_inode_release( )	*
#include <pthread.h>	
pthread_atfork( )	
pthread_attr_destroy( )	*

Alibaba Confidential

Copyright© 2016 Alibaba

pthread_attr_getdetachstate( )	
pthread_attr_getguardsize( )	
pthread_attr_getinheritsched( )	*
pthread_attr_getschedparam( )	*
pthread_attr_getschedpolicy( )	*
pthread_attr_getscope( )	
pthread_attr_getstack( )	
pthread_attr_getstackaddr( )	
pthread_attr_getstacksize( )	*
pthread_attr_init( )	*
pthread_attr_setdetachstate( )	
pthread_attr_setcancelstate( )	*
pthread_attr_setguardsize( )	
pthread_attr_setinheritsched( )	*
pthread_attr_setschedparam( )	*
pthread_attr_setschedpolicy( )	*
pthread_attr_setscope( )	
pthread_attr_setstack( )	
pthread_attr_setstackaddr( )	
pthread_attr_setstacksize( )	*
pthread_barrier_destroy( )	*
pthread_barrier_init( )	*
pthread_barrier_wait( )	*
pthread_barrierattr_destroy( )	*
pthread_barrierattr_getpshared( )	*
pthread_barrierattr_init( )	*
pthread_barrierattr_setpshared( )	*
pthread_cancel( )	*
pthread_cleanup_pop( )	
pthread_cleanup_push( )	
pthread_cond_broadcast( )	*
pthread_cond_destroy( )	*
pthread_cond_init( )	*
pthread_cond_signal( )	*
pthread_cond_timedwait( )	*
pthread_cond_wait( )	*
pthread_condattr_destroy( )	
pthread_condattr_getclock( )	
pthread_condattr_getpshared( )	
pthread_condattr_init( )	*
pthread_condattr_setclock( )	
pthread_condattr_setpshared( )	
pthread_create( )	*

Alibaba Confidential

Copyright© 2016 Alibaba

pthread_detach( )	*
pthread_equal( )	*
pthread_exit( )	*
pthread_getconcurrency( )	
pthread_getspecific( )	*
pthread_getschedparam( )	*
pthread_join( )	*
pthread_key_create( )	*
pthread_key_delete( )	*
pthread_kill( )	*
pthread_mutex_destroy( )	*
pthread_mutex_getprioceiling( )	
pthread_mutex_init( )	*
pthread_mutex_lock( )	*
pthread_mutex_setprioceiling( )	
pthread_mutex_timedlock( )	
pthread_mutex_trylock( )	*
pthread_mutex_unlock( )	*
pthread_mutexattr_destroy( )	*
pthread_mutexattr_getprioceiling( )	
pthread_mutexattr_getprotocol( )	
pthread_mutexattr_getpshared( )	*
pthread_mutexattr_gettype( )	*
pthread_mutexattr_init( )	*
pthread_mutexattr_setprioceiling( )	
pthread_mutexattr_setprotocol( )	
pthread_mutexattr_setpshared( )	*
pthread_mutexattr_settype( )	*
pthread_once( )	*
pthread_self( )	*
pthread_setcancelstate( )	*
pthread_setcanceltype( )	
pthread_setconcurrency( )	
pthread_setspecific( )	*
pthread_setschedparam( )	*
pthread_setschedprio( )	*
pthread_sigmask( )	*
pthread_testcancel( )	*
pthread_yield( )	*
#include <sched.h>	

Alibaba Confidential

Copyright© 2016 Alibaba

sched_get_priority_max( )	*
sched_get_priority_min( )	*
sched_getparam( )	*
sched_getscheduler( )	*
sched_fork( )	
sched_dead( )	
sched_rr_get_interval( )	*
sched_setparam( )	*
sched_setscheduler( )	*
sched_setscheduler_nocheck( )	
sched_yield( )	*
sched_lock( )	*
sched_unlock( )	*
sched_lockcount( )	*
sched_note_start( )	*
sched_note_stop( )	*
sched_note_switch( )	*
#include <semaphore.h>	
sem_close( )	*
sem_destroy( )	*
sem_getvalue( )	*
sem_init( )	*
sem_open( )	*
sem_timedwait( )	*
sem_post( )	*
sem_trywait( )	*
sem_unlink( )	*
sem_wait( )	*
semctl( )	
semget( )	
semop( )	
#include <signal.h>	
sigaction( )	*
sigaddset( )	*
sigaltstack( )	
sigdelset( )	*

Alibaba Confidential

Copyright© 2016 Alibaba

sigemptyset( )	*
sigfillset( )	*
sighold( )	*
sigignore( )	
siginterrupt( )	
sigismember( )	*
signal( )	
signbit( )	
signgam( )	
signpause( )	*
sigpending( )	*
sigprocmask( )	*
sigqueue( )	*
sigrelse( )	*
sigset( )	
sigsetjmp( )	
sigsuspend( )	*
sigtimedwait( )	*
sigwait( )	
sigwaitinfo( )	*
kill( )	*
#include <sys/socket.h>	
socket ( )	*
bind ( )	*
connect ( )	*
listen ( )	*
accept ( )	*
send( )	*
sendto ( )	*
shutdown( )	*
sendmsg( )	
recv ( )	*
recvfrom ( )	*
setsockopt ( )	*
getsockopt ( )	*
getsockname( )	*

Alibaba Confidential

Copyright© 2016 Alibaba