206.2.0

Huawei LiteOS FAQ

Issue 3.0

Date 2022-03-23

HUAWEI

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: https://www.huawei.com

Email: support@huawei.com

Contents

1 Preface

Purpose

This document provides guidance for Huawei LiteOS developers to solve problems in their development process.

NOTE

This document uses Hi3516CV300 as an example. Unless otherwise specified, this document also applies to Hi3516EV100, Hi3516EV200, Hi3516EV300, Hi3518EV200、Hi3518EV300, Hi3559A and Hi3556A.

Product Version

The following table lists product versions described in this document.

Product Name	Product Version
Hi3516C	V300
Hi3516E	V100
Hi3516E	V200
Hi3516E	V300
Hi3518E	V200
Hi3518E	V300

Intended Audience

This document is intended for:

- Technical support engineers
- Software development engineers

Change History

Changes between document issues are cumulative. The latest document issue contains all the changes made in earlier issues.

Date	Version	Description
2018-08-22	1.0	First official release.
2021-11-25	2.0	206.1.0 version release
2022-3-23	3.0	206.2.0 version release

2 FAQs for Dynamic Library Loading

2.1 Functions in an Executive Library Are Abnormal After a Dynamic Library Is Successfully Loaded

- **Cause 1**: The **-nostdlib** compiler option is not added when a dynamic library is generated. As a result, the system uses the standard library provided by the compiler as the dynamic library.

  Solution: In principle, use the standard library provided by Huawei LiteOS instead of the system or compiler. Add the **-nostdlib** compiler option when a dynamic library is generated. Do not use **-l** to specify a standard library or its search path.

- **Cause 2**: The **-fPIC** and **-shared** compiler options that are mandatory for generating a shared dynamic library are not added when a dynamic library is generated.

  Solution: Add **-fPIC** and **-shared** compiler options when a dynamic library is generated. If the process of generating a dynamic library is a process of generating the **.c** file (or the **.cpp** file), **.o** file, and **.so** file, also add the **-fPIC** compiler option in the process of generating the **.o** file.

- **Cause 3**: When an image is generated, the **LITEOS_CFLAGS** or **LITEOS_CXXFLAGS** compiler option is not used. As a result, the system uses the standard library provided by the compiler, and the function addresses related to the standard library get malfunctional.

  Solution: Add **include config.mk** in the main directory of Huawei LiteOS to the **makefile** file for generating an image, and use the **LITEOS_CFLAGS** or **LITEOS_CXXFLAGS** compiler option.

- **Cause 4**: The **-mlong-calls** compiler option is not added. As a result, the **longjmp** function fails to be called.

Solution: Add the **-mlong-calls** compiler option. The generated target code allows functions in the same source file to be called. The distance between the calling point and the called functions can exceed 256 KB.

2.2 Symbols Cannot Be Readdressed Because a Dynamic Library Fails to Be Loaded

- **Cause 1**: The dynamic library to be loaded has an external dependent library. As a result, external symbols fail to be readdressed in the loading process of the dynamic library.

  Solution: LiteOS supports dynamic loading of .so and .o files. The .so files support the priority of loading dependent files. To load the .o file, you need to load the dependent library first and specify the loading path of the dependent library before loading the library, and set the loading path of the dependent library before loading the dynamic library.

- **Cause 2**: The dynamic library to be loaded is not processed by executing the **sym.sh** script in the **tools/scripts/dynload_tools** directory of Huawei LiteOS. As a result, external symbols provided by Huawei LiteOS in the dynamic library are not quoted but optimized when the system image is generated.

  Solution: Save the dynamic library to be loaded and its dependent library to the same path, and execute the **sym.sh** script containing the folder path input parameter to process them.

- **Cause 3:** There is no **kernel / extended / dynload / src** directory of Huawei LiteOS. As a result, the undefined symbol file **los_dynload_gsymbol.c** cannot be generated in this directory.

  Solution: Create the **kernel / extended / dynload / src** in the Huawei LiteOS home directory, and then resolve the problem by referring to **Cause 2.**

- **Cause 4:** The version of python in the compilation environment does not match. When the dynamic library is extracted, the execution of the python script with undefined symbols fails (**sym.sh** execution process error). The generated file **los_dynload_gsymbol.c** does not contain undefined symbols.

  Solution: Install **python 2.7** in the compilation environment to ensure that there are no errors during the execution of **sym.sh.**

- **Cause 5:** The undefined symbol file **los_dynload_gsymbol.c** is not involved in compiling and linking. The **los_dynload_gsymbol.c** file is in the LiteOS home directory: **kernel / extended / dynload / src**

  Solution: Modify the Makefile to ensure that **los_dynload_gsymbol.c** participates in compiling and linking when generating the image.

- **Cause 6:** The Makefile that generated the image failed to include **dynload_ld.mk** or did not include **dynload_ld.mk**, resulting in undefined symbols not participating in linking.

  Solution: Make sure that the Makefile file that generates the image contains **dynload_ld.mk** successfully.

2.3 LOS_SoLoad Returns NULL Because a Dynamic Library Fails to Be Loaded

- **Cause 1**: The loading path of a dynamic library is incorrect or does not exist.

  Solution: Enter the correct loading path and name of the dynamic library.
- **Cause 2**: The number of loaded dynamic libraries has reached the upper limit.

  Solution: Check whether the number of loaded dynamic libraries exceeds the value of the **N_MAX_USER_MODULES** macro, and call the **LOS_ModuleUnload** function to uninstall unnecessary dynamic libraries or change the value of the **N_MAX_USER_MODULES** macro based on the available memory.
- **Cause 3**: The system memory is insufficient.

  Solution: Reserve sufficient memory space for loading the dynamic library.

3 FAQs for C++

3.1 Mathematical Functions Are Redefined Because the C++ Source Code Contains math.h

Cause: The C++ library includes some mathematical function libraries. These mathematical function libraries are quoted in the C++ header file. As a result, the quoted mathematical function libraries conflict with those defined in the C library of Huawei LiteOS.

Solution: Change **math.h** in the C++ source code to **cmath**, that is, change **#include <math.h>** to **#include <cmath>**.

3.2 stdlib.h Cannot Be Found When Compiling the C++ Source Code

Cause: The path in the C++ header file is incorrect. As a result, when the **#include_next <stdlib.h>** command is executed in the **cstdlib** function library, **stdlib.h** cannot be found in the **libc** function library.

Solution: Change the sequence of LITEOS_CXXINCLUDE and LITEOS_LIBC_INCLUDE in the los_config.mk file. The sequence of LITEOS_LIBC_INCLUDE should be followed by LITEOS_CXXINCLUDE.

NOTICE

If the problem still persists, check whether **LITEOS_CXXINCLUDE** is before **LITEOS_LIBC_INCLUDE** (or do not add **LITEOS_LIBC_INCLUDE**), or whether **LITEOS_CXXFLAGS** is before **LITEOS_CFLAGS**.

3.3 bits/c++config.h Cannot Be Found When Compiling the C++ Source Code

Cause: The path to the added C++ header file is missing when the **los_config.mk** file is synchronously modified in a BVT version.

Solution: When inheriting LiteOS code to your project, ensure that the compilation environment is consistent with the LiteOS compilation environment and that the C++ path is included. You can include the C++ compiler path in your compilation framework. For details, see the definitions of LITEOS_COMPILER_CXX_PATH and LITEOS_CXXINCLUDE in compiler_gcc.mk.

3.4 The System Crashes with the C++ Library Running

Cause: When C++ code is invoked before C++ initialization, an exception occurs when the address 0 is accessed. Check whether the exception is caused by uninitialization based on the exception information.

Solution: Initialize related global variables by calling the **LOS_CppSystemInit** function. For details, see the *Huawei LiteOS Developer Guide*.

3.5 The Function Symbol Cannot Be Found When C Calls a C++ Function

Cause: The called function does not comply with the calling conventions of C. As a result, the function symbol is compiled based on the naming modification rules of C++. However, the calling point is compiled based on the command modification rules of C.

Solution: Add the following C calling convention before the function definition and declaration:

```
#ifdef __cplusplus
#if __cplusplus
extern "C" {
#endif /* __cplusplus */
#endif /* __cplusplus */
/* code */
...
#ifdef __cplusplus
#if __cplusplus
}
#endif /* __cplusplus */
#endif /* __cplusplus */
```


NOTICE

The preceding problem also exists when C++ calls a C function. Add the preceding calling convention of C before the function declaration.

3.6 Information Cannot Be Printed Out Using cout

Cause: The output functions or devices are not initialized.

Solution: Before using cout, ensure that the following functions are called in sequence:

```
los_vfs_init();
uart_dev_init();
virtual_serial_init(TTY_DEVICE);
system_console_init(SERIAL);
```

3.7 Symbols _kernel_cmpxchg and _kernel_dmb Are Not Defined

Cause: When some classes in C++ are used, atomic operations are performed and the preceding two fixed memory address functions defined in the **libgcc.a** library file are called. The system crashes. Therefore, delete the function when the **libgcc.a** library file is generated.

Solution: Call the preceding two functions supported by the LiteOS, create the **atomic-linux.c** file in the **compat/linux/src** directory under the root directory, and add the following code to the file:

```
extern int LOS_AtomicCmpXchg32bits(volatile UINT32 *puwAddr, UINT32 uwNewVal, UINT32 uwOldVal);
extern void dmb(void);

/* Kernel helper for compare-and-exchange.  */
int __kernel_cmpxchg(int oldval,int newval,int *ptr)
{
    return LOS_AtomicCmpXchg32bits(ptr,newval,oldval);
}

/* Kernel helper for memory barrier.  */
void __kernel_dmb(void)
{
}
```


4 FAQs for File Systems

4.1 How Do I Locate the Files That Causes Failure to Open Files?

Answer: Run the **lsfd** command to list the files that have been opened. If file handles are leaked and files cannot be opened, run the **lsfd** command to check the files that are opened.

4.2 How Do I Improve the Write Performance of ramfs?

Answer: When data is frequently appended to ramfs, memory and copy are repeatedly applied for and the performance of ramfs becomes low. This is because a larger memory is needed for storing the greater ramfs. Set the memory to be allocated for each time through the **void los_set_ramfs_unit(off_t size)** API. For example, write 32 bytes into a file for 100 times. By default, new memory is applied for during each write operation. It is time-consuming to copy the original data to the new memory. Therefore, allocate the memory with a size of 32 bytes x 100 for the next write operation using the **los_set_ramfs_unit(32*100)** API. After the number of writes reaches 100, new memory does not need to be allocated for

each write operation. If the final size of the file to be written is unknown, use the preceding API to increase each allocated memory and reduce the number of memory allocations and the copy frequency. Restore the default value using the **los_set_ramfs_unit(0)** API. Otherwise, a large memory is allocated when a small file is written. As a result, excessive memory resources are consumed.

4.3 A File Cannot Be Created and the Error Message "no space" Is Displayed After the JFFS2 Is Mounted

Cause: The size of the mounting partition of Journalling Flash File System version 2 (JFFS2) must be not less than (3 x eraseblock + partitionsize/50).

Solution: Because JFFS2 is a log file system, the write and delete operations require extra erase blocks. In addition, JFFS2 includes the garbage collection (GC) mechanism. Additional erase block space is required for the GC operations. Therefore, some space in the flash partition needs to be reserved for GC operations and write and delete operations. The size of the space is determined by (3 x blocksize + partitionsize/50). Therefore, when adding a partition, ensure that the size of the partition is at least two erase blocks more than the reserved space; otherwise, JFFS2 cannot be used properly.

4.4 The Statfs Shows That There is Still Space Left in JFFS2 File System , but Fail to Write to File

Cause:

1. Write and delete require additional blocks in JFFS2 file system. For details, see en-us_topic_0144117688.xml.
2. When the dirty data in the jffs2 file system is less than one block, the gc will not reclaim the dirty data.

The value of statfs is the size of dirty_size + free_size. Even if the remaining space is displayed, the write may fail due to the preceding two causes. Currently, the available space of the jffs2 file system cannot be viewed.

4.5 Why Does Linux Write More Data Than LiteOS in the jffs2 Partition of the Same Size?

Cause: First, the jffs2 file system stores data in the flash memory in the form of node. One write operation is to write the information and data of a node to the flash memory. For example, if 10 bytes are written, the actual space occupied by the flash memory is about 72 bytes, which contains additional information about the node. Therefore, writing small data blocks to the jffs2 may result in a negative compression ratio. However, the flash usage will be higher if a large amount of data is written to the jffs2 at a time. Linux has an optimization mechanism to improve the flash usage by sacrificing memory. In Linux, data is cached in the memory every time while written to the flash memory. After the 4096 bytes are fully written, the system aggregates the data and compresses the data to form a node into the flash memory. The previous nodes are marked as dirty data and

wait for the gc to reclaim the data. This reduces the number of nodes and saves the flash space. However, in Linux, the amount of data written to the flash memory consumes the same size of memory. If the size is less than 4K, 4K is used. Considering the memory usage and memory fragmentation, Huawei LiteOS does not support this mechanism currently.

Solution: Write a large amount of data at a time to improve the flash usage.

4.6 NFS File System Mounts Subdirectory in Windows Environment, Prompt: No Such File or Directory

Cause: Nfs server shared path attribute configuration in Windows environment is missing the "**-alldirs**" option.

Solution: When configuring a shared path in NFS Server, include the "**-alldirs**" option.

Figure 4-1 Choose "Edit exports file" in NFS Server

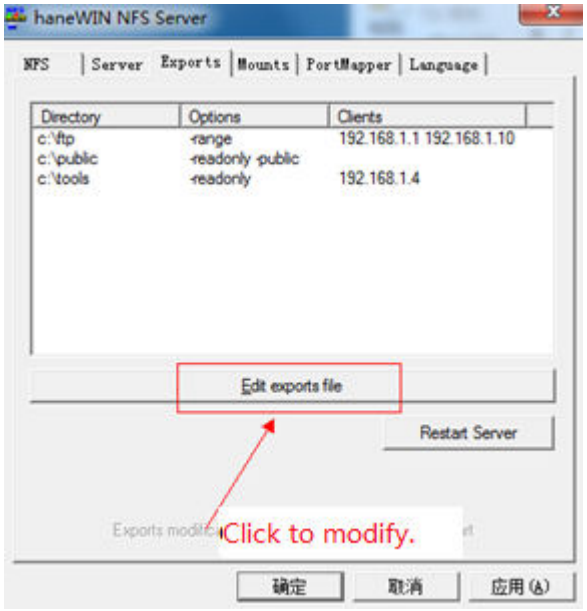
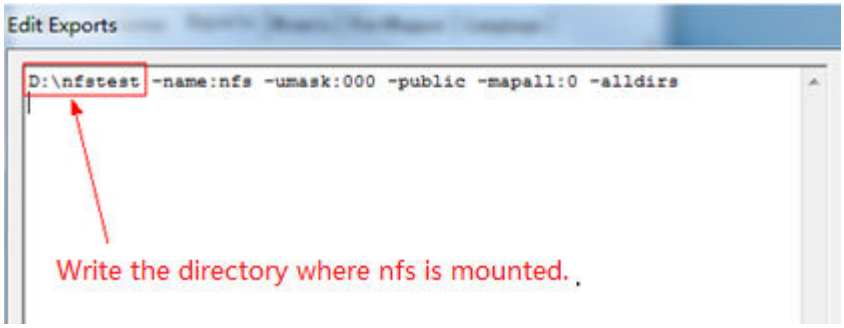


Figure 4-2 Config a shared path with "d:\NFS -name:nfs -umask:000 -public -mapall:0 -alldirs" .



Property interpretation:

- **d:\NFS** : The directory where the specified window is actually mounted.
- **name:nfs**: The name of the mount, which is the name used when the client is mounted.
- **-umask:000**: Set the default permission for users to create files in the NFS file system. The default format is 022, which is set to 000 to ensure maximum permissions. Users can set it by themselves.
- **-public**: Enable WebNFS access.
- **-mapall:0**: Map all Unix user ids and group ids to the specified user id and group id.
- **-alldirs**: Support for mounting any directory of the host file system.

4.7 Remove the SD Card While Writing, the Last Written File Does Not Exist or the Length is 0 After Re-Inserting the Card.

Cause: Since bcache uses the LRU(least recently used) mechanism, the directory entry needs to be modified each time a file is written. Therefore, the cache block in which the directory entry is located has not been written back to the disk. After re-inserting the card, accessing the directory entry does not get the correct result when viewing the file information.

Solution:

1. Users are required to actively invoke sync according to their own scene requirements to ensure that the contents of the directory entries are written to disk.
2. The LOSCFG_FS_FAT_CACHE_SYNC_THREAD option can be configured in menuconfig. System will create a task brush cache while this option is enabled. The proportion of dirty data blocks in the cache is checked every 5 seconds in default. If the proportion is more than 80%, the synchronization operation is performed, and the dirty data in the cache will all be written back to disk. The task priority, the refresh interval, and the dirty block ratio threshold can be set by the interfaces LOS_SetSyncThreadPrio, LOS_SetSyncThreadInterval and LOS_SetDirtyRatioThreshold respectively. For details, see "Huawei LiteOS Development Guide and API Reference.chm".

5 Network FAQs

5.1 How Do I Obtain a Gateway Address?

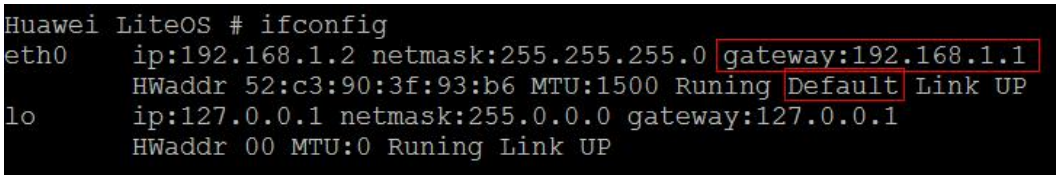
- **Linux**: The gateway address refers to the next hop address corresponding to the default route. In the service code, call the **system** function to run the **route** command (or open the **/proc/net/route** file), and then analyze the command output to obtain the gateway address.

  Figure 5-1 Obtaining the gateway address from Linux

  ```
  g00400460@ubuntu:~$ route
  Kernel IP routing table
  Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
  default         10.175.100.1    0.0.0.0         UG    0      0        0 enp2s0f0
  localnet        *               255.255.252.0   U     0      0        0 enp2s0f0
  172.17.0.0      *               255.255.0.0     U     0      0        0 docker0
  192.168.122.0   *               255.255.255.0   U     0      0        0 virbr0
  ```

- **Huawei LiteOS**: Both the **route** command and the **system** function are not supported. The **/proc/net/route** file does not exist in the **proc** file system. Therefore, the gateway address cannot be obtained using the preceding method. By default, an NIC is deployed in the LiteOS. Such NIC functions as the default route in Linux mode. The gateway address in the default NIC corresponds to the default gateway address.

Figure 5-2 Obtaining the gateway address from LiteOS



For example:

```
#include "lwip/netifapi.h"
 #include "lwip/ip_addr.h"

 struct in_addr gw;
 struct netif *netif = NULL;
 netif = netifapi_netif_get_default();
 if (NULL == netif) {
    printf("no default netif...\n");
 } else {
    gw.s_addr = netif->gw.addr;
    printf("gateway: %s\n", inet_ntoa(gw));
 }
```

5.2 How Do I Set a Gateway Address?

- **Linux:** Run the **route** command to manage system routes. During service processing, call the **system("route add default gw 192.168.1.1")** function to set system gateways. When running the **route** command to set a system gateway, run the **SIOCADDRT** command (ioctl is called) at the underlying layer.
- **Huawei LiteOS:** The **SIOCADDRT** command is also supported. However, compared with Linux, Huawei LiteOS has the following two different points in using the **SIOCADDRT** command:
  a. Huawei LiteOS supports only the setting of system gateways but does not support the setting of subnet and host routes.
  b. When setting system gateways on Huawei LiteOS, set the value of **rt_flags** to **RTF_GATEWAY**. Other flags are not supported.

  For example:

```
#include "lwip/netifapi.h"
#include "lwip/ip_addr.h"

struct sockaddr_in *saddr;
struct rtentry rt;
int sock;
int ret;

memset(&rt, 0, sizeof(rt));
saddr = (struct sockaddr_in*)&(rt.rt_gateway);
saddr->sin_family = AF_INET;
saddr->sin_addr.s_addr = inet_addr("10.173.210.1");
rt.rt_flags = RTF_GATEWAY;

sock = socket(AF_INET, SOCK_DGRAM, 0);
if (sock < 0) {
    perror("socket");
    return;
}
ret = ioctl(sock, SIOCADDRT, &rt);
if (ret < 0) {
    perror("ioctl SIOCADDRT");
```



```
} else {
    printf("default gateway changed to 10.173.210.1\n");
}

close(sock);
```

5.3 How Do I Set and Obtain a DNS Server Address?

- **Linux:** Modify the **/etc/ resolv.conf** file to configure a DNS server.
- **Huawei LiteOS:** Compared with Linux, Huawei LiteOS provides APIs for you to configure or obtain a DNS server.

  For example:

```
#include "lwip/netifapi.h"
#include "lwip/ip_addr.h"

  ip_addr_t dns1, dns2;
  ip_addr_t dns1_g, dns2_g;
  struct in_addr d1, d2;
  struct in_addr d1_g, d2_g;
  int ret;

  inet_aton("8.8.8.8", &d1);
  inet_aton("114.114.114.114", &d2);

  dns1.addr = d1.s_addr;
  dns2.addr = d2.s_addr;

  ret = lwip_dns_setserver(0, &dns1);
  if (ret < 0) {
      printf("set dns server1 failed %d\n", ret);
  }

  ret = lwip_dns_setserver(1, &dns2);
  if (ret < 0) {
      printf("set dns server2 failed %d\n", ret);
  }

  lwip_dns_getserver(0, &dns1_g);
  lwip_dns_getserver(1, &dns2_g);

  d1_g.s_addr = dns1_g.addr;
  d2_g.s_addr = dns2_g.addr;
  printf("dns server1 is %s\n", inet_ntoa(d1_g));
  printf("dns server2 is %s\n", inet_ntoa(d2_g));
```

NOTICE

Currently, Huawei LiteOS supports two DNS servers by default. To configure more DNS servers, modify the **DNS_MAX_SERVERS** macro and re-compile the lwIP library. Moreover, the system provides the **dns** shell command for you to query or set a DNS server.

5.4 How Do I Use the DHCP Client Supported by Huawei LiteOS?

- **Linux:** BusyBox contains udhcpc, which can be used to enable the DHCP client.

- **Huawei LiteOS:** The system does not provide udhcpc. Therefore, the DHCP client cannot be enabled using the preceding method. However, the protocol stack for Huawei LiteOS supports the DHCP client implementation. You can call the corresponding API provided by the protocol stack to enable the DHCP client supported by Huawei LiteOS.

  For example:

```
#include "lwip/netifapi.h"

int ret, i;
struct netif *netif = netifapi_netif_find_by_name("eth0");

/* start dhcp client on eth0 */
ret = netifapi_dhcp_start(netif);
if (ret < 0) {
    printf("dhcp satrt failed %d\n", ret);
} else {
    for (i = 0; i < 10; i++) {
        ret = netifapi_dhcp_is_bound(netif);
        if (ret < 0) {
            sleep(1);
            continue;
        } else {
            printf("dhcp ok...\n");
            break;
        }
    }

    if (i == 10) {
         printf("dhcp failed after 10s, stop...\n");
         /* stop dhcp client on eth0 */
         netifapi_dhcp_stop(netif);
         /* clear dhcp client resource */
         netifapi_dhcp_cleanup(netif);
    }
}
```

NOTICE

If you need to add private options to the DHCP client packets, port the DHCP client. Huawei LiteOS does not require that you use the DHCP client supported by the protocol stack.

5.5 How Do I Use the DHCP Server Supported by Huawei LiteOS?

- **Linux:** BusyBox contains udhcpd, which can be used to enable the DHCP server.
- **Huawei LiteOS:** The system does not provide udhcpd. However, the protocol stack for Huawei LiteOS supports the DHCP server implementation. You can call the corresponding API provided by the protocol stack to enable the DHCP server supported by Huawei LiteOS.

  For example:

```
#include "lwip/netifapi.h"

int ret;
struct netif *netif = netifapi_netif_find_by_name("eth0");
```



```
/* start dhcp server */
ret = netifapi_dhcps_start(netif, NULL, 0);
if (ret < 0) {
    printf("dhcp server start failed %d\n", ret);
}

sleep(10);
/* stop dhcp server */
netifapi_dhcps_stop(netif);
```

NOTICE

If you need to add private options to the DHCP server packets, port the DHCP server. Huawei LiteOS does not require that you use the DHCP server carried by the protocol stack.

5.6 How Do I Use the SNTP Supported by Huawei LiteOS for Network Time Synchronization?

- **Linux**: The **ntpdate** system command is supported. You can run the **system("ntpdate 192.168.1.1")** command to synchronize the network time.
- **Huawei LiteOS**: The preceding system command is not supported. Huawei LiteOS provides the **lwip_sntp_start** API for network time synchronization. Note that the API cannot be concurrently called. In addition, Huawei LiteOS provides the **ntpdate** shell command. For details, see the *Huawei LiteOS lwIP Developer Guide* in the **..\osdrv\opensource\liteos\liteos\doc** directory.

5.7 Differences Between Huawei LiteOS and Linux in the Routing Mechanism

- **Linux**: The system provides a routing table for routing IP packets. The routing table is maintained by the kernel. When you configure an IP address, the corresponding route option will be automatically added to the routing table. You can also run the **route** or **ip route** command to manually add the corresponding route option. The routing table is the core function of the entire IP layer. You need to query it when receiving, sending, and forwarding packets. When you send an IP packet, if there are multiple route options to route the IP packet, you can determine the wanted route option based on the longest prefix match algorithm. In addition, Linux supports policy routing, multicast routing, and packet forwarding.
- **Huawei LiteOS**: The system does not provide routing tables and the **route** or **ip route** command. Therefore, users cannot manually add route options. However, the system gateway can be modified by running the **SIOCADDRT** command (ioctl is called) in Huawei LiteOS, which is similar to modify the default route in Linux OS. All NICs in Huawei LiteOS will be linked to form a single linked list. The IP packets to be sent traverse the single linked list. Only if the address of an NIC and the destination address are in the same subnet, the NIC can be determined to send the packets. Therefore, Huawei LiteOS

differs from Linux in routing. In addition, Huawei LiteOS does not support policy routing, multicast routing, and packet forwarding.

5.8 Ioctl Commands, Socket Attributes, and Socket Types Supported by Huawei LiteOS

Some frequently encountered problems are listed as follows:

- The **SO_LINGER** and **SO_REUSEPORT** socket attributes are not supported.
- The UDP or RAW socket sending cache cannot be set. The TCP socket sending cache can be set. The setting range must be [2MSS, 256K].
- The TCP socket receiving cache cannot be set. The UDP or RAW socket receiving cache can be set.

Huawei LiteOS only supports the **SOCK_RAW** link layer socket instead of **SOCK_DGRAM** and **SOCK_PACKET**.

6 FAQs for USB Modules

6.1 What Specifications Are Supported by the LiteOS USB Driver?

Answer: The specifications of the LiteOS USB driver vary with the chip hardware resources. For details, see the manual *Peripheral Driver Operation Guide (Huawei LiteOS).*

6.2 What Chip Models Are Supported by USB-to-Ethernet?

Answer: Currently, USB-to-Ethernet supports five types of chip models: Ax88178a, Ax88178, Ax88772, Ax88772A, and Ax88772B.

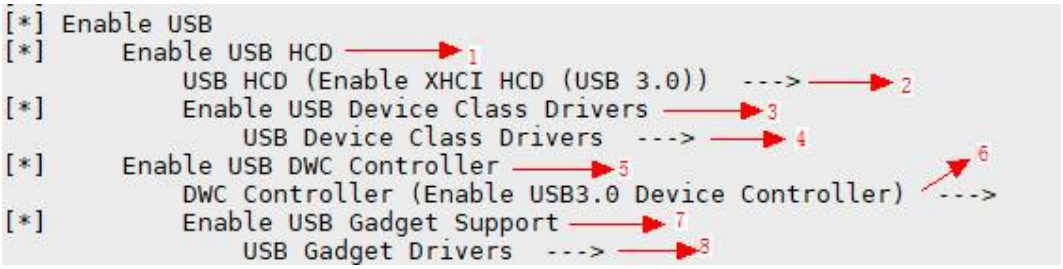
6.3 Does the LiteOS USB Driver Support Dynamic Switching?

Answer: The USB driver supports dynamic switching between Device protocols and does not support switching between Host mode and Device mode. When the protocol to be loaded specified is dynamically switched through the usb_init interface, the usb_deinit command is used to unload the currently used protocol, and then reload the protocol to be switched by usb_init. Some protocols do not support uninstallation (such as DFU, UVC, Camera,HID). You need to disable the current service before calling usb_deinit to uninstall.

6.4 How Do I Configure the USB Driver on the LiteOS menuconfig Page?

Answer: The following figure shows the configurations of the USB driver on the **LiteOS menuconfig** page. You can enable the USB driver functions by selecting **Enable USB** or disable them by deselecting **Enable USB**.

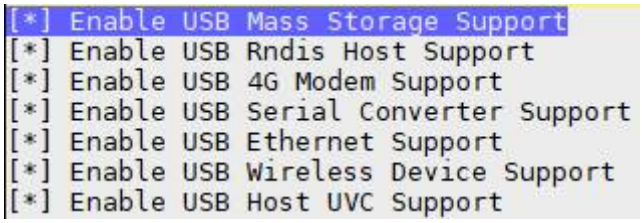
Figure 6-1 Configuring the USB driver on the **LiteOS menuconfig** page



- At the host side:

  As shown in the preceding figure, **1** indicates whether to enable the host function of the USB driver. **2** indicates the host controller type, which can be 2.0 or 3.0 (depending on the USB port type of the development board). **3** indicates whether to enable the USB protocol function in host mode. **4** indicates the USB protocol function in host mode. The configurations of the USB driver at the host side are as follows:

  Figure 6-2 Configuring the USB driver

  The configuration items are as follows: 1. USB flash drive; 2. Host-RNDIS; 3. 4G modem; 4. USB-to-serial; 5. USB-to-Ethernet; 6. USB-Wi-Fi;7.Host UVC.

  NOTICE

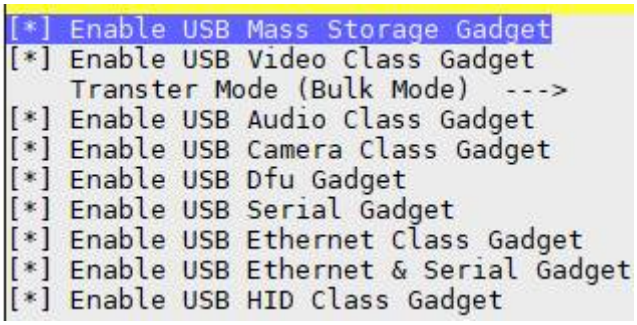
  In host mode, the USB driver can be initialized by calling the **usb_init(HOST, 0)** API. You can select partial or all protocols based on service requirements. The protocol stack automatically loads the corresponding USB driver based on the type of the inserted device. However, the dependent peripheral components need to be initialized in advance.

- At the device side:

  As shown in the preceding figure, **5** indicates whether to enable the device function of the USB driver. **6** indicates the device controller type, which can be

Dwc2.0 or Dwc3.0 (depending on the USB port type of the development board). **7** indicates whether to enable the USB protocol function in device mode. **8** indicates the USB protocol function in device mode. The configurations of the USB driver at the device side are as follows:

Figure 6-3 Configuring the USB driver



The configuration items are as follows: 1. USB flash drive; 2. UVC; 3. UAC; 4. UAV (UVC&UAC); 5. DFU upgrade; 6. Virtual serial port; 7. Virtual network port; 8. Composite device (virtual serial port & virtual network port),9.HID.

NOTICE

In device mode, the USB driver can be initialized by calling the **usb_init(DEVICE, xxx)** API. You can enable the required or all protocols based on service requirements. Compared with the host mode, this mode needs to statically specify the protocol to be loaded during the initialization. For example, **xxx** indicates the protocol to be loaded, which can be declared by the definition of the **usb_init.h** macro. However, the same point is that the dependent peripheral components also need to be initialized in advance.

6.5 What Frame Rate Can the LiteOS USB YUV 1080P Support?

Answer: Currently, the LiteOS USB YUV 720P is at most supported because YUV frame storage occupies much memory (the memory usage for USB YUV 1080P is over 5 times more than that for USB YUV 720P). If the memory usage is not limited, increase the corresponding frame memory. The frame rate measured for USB YUV 1080P is approximately 4.2 frames/second.

7 FAQs for Compiler

7.1 Backtrace function provided by glibc under ARM platform has less backtrace information

Reason: The arm-linux-gcc cross-compilation on ARM platform differs from the gcc default compilation options on x86 platform. And some optimizations such as stack pointer optimization and tail call optimization can also result in backtrack information reduced.

Solution: Add four compile options when compiling a file or module that uses the backtrace function:

```
-rdynamic -funwind-tables -fno-omit-frame-pointer -fno-optimize-sibling-calls
```

7.2 The Target File Generated by Link Cannot be Found in Shell Command and the File System Mount Failed

Reason: The link parameter is missing when the Makefile is linked to the target file.

Solution: The compile option macros **LITEOS_LDFLAGS**, **LITEOS_TABLES_LDFLAGS** must be added at compile time.

8 Other FAQs

8.1 How Do I View the Created Interrupts and Triggers in the System?

Answer: Using the **hwi** command, you can view the created interrupts and triggers in the system and check whether the created interrupts are successfully triggered.

8.2 Error Message "Warning:DO NOT call xxx in software timer callback" and Call Stack Exceptions Are Cyclically Displayed During System Running

Cause: To ensure the accuracy of software timer task timing, the LiteOS does not allow calling APIs (for example, LOS_SemPend and LOS_MuxPend) that may cause

task blocking in the software timer callback process. Therefore, when this happens, the system displays the warning and call stack exceptions.

Solution: You are advised to find the calling point based on call stack information and modify the code.

8.3 What Can I Do If the System Does Not Support the system Function?

Answer: If the LiteOS does not support the **system** function for related operations, directly call the **system** function API to enable the corresponding function.

8.4 Shell Task Stack Overflows

Cause: During the system running or test, you may add a **shell** command. However, the shell task stack configured in the system is not large. Therefore, if you perform all operations using the **shell** command, it is easy to cause the shell task stack to overflow.

Solution: Create tasks in the entry function of the **shell** command to complete operations.

8.5 Third-Party or Private Library Fails to Run

Cause: The LiteOS uses its own C library header file for compilation.

Solution: Save the **config.mk** file in the root directory to the **makefile** file used to compile the **bin** file for the system image, and use the **LITEOS_CFLAGS** or **LITEOS_CXXFLAGS** compiler option in the **config.mk** file. For example:

```
LITEOSTOPDIR ?= ../..
SAMPLE_OUT = .
include $(LITEOSTOPDIR)/config.mk
RM = -rm -rf
LITEOS_LIBDEPS := --start-group $(LITEOS_LIBDEP) --end-group
SRCS = $(wildcard sample.c)
OBJS = $(patsubst %.c,$(SAMPLE_OUT)/%.o,$(SRCS))
all: $(OBJS)
clean:
@$(RM) *.o sample *.bin *.map *.asm
$(OBJS): $(SAMPLE_OUT)/%.o : %.c
$(CC) $(LITEOS_CFLAGS) -c $< -o $@
$(LD) $(LITEOS_LDFLAGS) -uinit_jffspar_param --gc-sections -Map=$(SAMPLE_OUT)/sample.map -o $
(SAMPLE_OUT)/sample ./$@ $(LITEOS_LIBDEPS) $(LITEOS_TABLES_LDFLAGS)
$(OBJCOPY) -O binary $(SAMPLE_OUT)/sample $(SAMPLE_OUT)/sample.bin
$(OBJDUMP) -d $(SAMPLE_OUT)/sample >$(SAMPLE_OUT)/sample.asm
```


8.6 When the Code is Simultaneously Commissioned by the Serial Port and the Telnet, If Both of Them are Used to Obtain the Stdin in the Commissioning Codes, the Input of One Port May not Respond.

Cause: LiteOS is a real-time operating system with no process concept. When obtaining stdin on two ports at the same time, it is equivalent to two threads listening to stdin under the same process. Since getchar and other input functions have a competitive relationship, the system will lock the stdin. Therefore, when the serial port and the telnet port acquire the stdin at the same time, the system will only respond to the port that requested the input first.

Solution: Avoid using stdin interfaces such as getchar, gets, scanf, fgets, vfscanf on the serial port and telnet.

8.7 LiteOS Does not Support TLS, Using the Compiler Built-In Keyword __thread Will Trigger an Exception

Cause： TLS，full name Thread Local Storage. The LiteOS system runs in kernel mode after startup, but the __thread keyword is generally used in user mode programs. The compiler will pile at the point of use and access the registers in el0 or pl0 mode. LiteOS does not configure this register, its content is invalid, and adding this keyword to the Linux kernel code will also cause a running exception.

Solution： Avoid scenarios using TLS.

8.8 After a User Transplants an Algorithm Program to LiteOS, the Runtime of the Algorithm Program on LiteOS Is Longer Than That on the Linux OS

Cause: By default, LiteOS contains a compilation option **-fno-builtin** that is used to prevent the LiteOS system function from being identified as a built-in function and replaced by special code. With this option, only functions starting with **_builtin_** can be identified as a built-in function. GNU compiler collection (GCC) on Linux may generate special code to process built-in functions more efficiently, reducing the runtime of the algorithm.

Solution: To identify a function (typically a mathematical function) as a built-in function based on the algorithm program, use the following macro definition in the algorithm source file (funcA is used as an example):

```
#define funcA ( x ) __builtin_funcA ( x )
```


8.9 When a Large Amount of Data Is Printed by Running the cat Command, the Printed Data May Be Incomplete

Cause: When the console is enabled, the print functions such as **dprintf** are output through the console. The console buffers data through ringbuf. The size of ringbuf is configurable and is 4 KB by default. The actual size is determined by the configuration item **LOSCFG_CONSOLE_RINGBUFF_SIZE**, which can meet common print requirements. The possible causes of incomplete data printing are as follows:

1. If a large amount of data is printed by running the **cat** command, ensure that the ringbuf can store the printed data. Otherwise, the data that exceeds the buffer size will be discarded. As a result, the printed data is incomplete.
2. The console print output is implemented by task scheduling. If the priority of the print task is higher than that of the console print task, when a large amount of data is printed, the data in the buffer may overflow because the console task cannot be scheduled in time. As a result, the printed data is incomplete.

Solution: Increase the ringbuf size to reduce the risk of buffer overflow when a large amount of data is printed.

