LiteOS

V200R002C00

Developer Guide

Issue 01

Date 2018-07-28

HUAWEI TECHNOLOGIES CO., LTD.

Copyright © Huawei Technologies Co., Ltd. 2018. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

Trademarks and Permissions

HUAWEI 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: http://www.huawei.com

Email: support@huawei.com

Contents

1 Preface.....1

2 Overview.....3

2.1 Background Introduction.....3

2.2 Supported Cores.....6

2.3 Constraints.....6

3 Basic Kernel.....7

3.1 Task.....7

3.1.1 Overview.....7

3.1.2 Development Guidelines.....10

3.1.3 Precautions.....16

3.1.4 Programming Example.....16

3.2 Memory.....19

3.2.1 Overview.....19

3.2.2 Dynamic Memory.....21

3.2.2.1 Development Guidelines.....22

3.2.2.2 Precautions.....25

3.2.2.3 Programming Example.....26

3.2.3 Static Memory.....27

3.2.3.1 Development Guidelines.....27

3.2.3.2 Precautions.....28

3.2.3.3 Programming Example.....28

3.3 Interrupt.....29

3.3.1 Overview.....29

3.3.2 Development Guidelines.....31

3.3.3 Precautions.....33

3.3.4 Programming Example.....33

3.4 Queue.....34

3.4.1 Overview.....34

3.4.2 Development Guidelines.....36

3.4.3 Precautions.....40

3.4.4 Programming Example.....41

3.5 Event.....43

Copyright © Huawei Technologies Co., Ltd.

3.5.1 Overview.... 43
3.5.2 Development Guidelines.... 45
3.5.3 Precautions....48
3.5.4 Programming Example.... 48
3.6 Mutex....50
3.6.1 Overview.... 50
3.6.2 Development Guidelines.... 51
3.6.3 Precautions....54
3.6.4 Programming Example.... 54
3.7 Semaphore.... 57
3.7.1 Overview.... 57
3.7.2 Development Guidelines.... 59
3.7.3 Precautions....62
3.7.4 Programming Example.... 62
3.8 Time Management.... 64
3.8.1 Overview.... 65
3.8.2 Development Guidelines.... 65
3.8.3 Precautions....66
3.8.4 Programming Example.... 67
3.9 Software Timer.... 68
3.9.1 Overview.... 68
3.9.2 Development Guidelines.... 69
3.9.3 Precautions....72
3.9.4 Programming Example.... 73
3.10 Error Handling.... 75
3.10.1 Overview.... 75
3.10.2 Development Guidelines.... 76
3.10.3 Precautions....76
3.10.4 Programming Example.... 76
3.11 Doubly Linked List....77
3.11.1 Overview....77
3.11.2 Development Guidelines....77
3.11.3 Precautions....78
3.11.4 Programming Example.... 78
3.12 Bitwise Operation.... 79
3.12.1 Overview.... 79
3.12.2 Development Guidelines.... 79
3.12.3 Precautions....80
3.12.4 Programming Example.... 80

4 Extended Kernel....82

4.1 Dynamic Loading.... 82
4.1.1 Overview.... 82

Copyright © Huawei Technologies Co., Ltd.

4.1.2 Development Guidelines.... 83
4.1.3 Precautions....90
4.1.4 Programming Example.... 90
4.2 Scatter Loading....91
4.2.1 Overview.... 91
4.2.2 Development Guidelines.... 93
4.2.3 Precautions....96
4.2.4 FAQs.... 96
4.3 Exception Management.... 97
4.3.1 Overview.... 97
4.3.2 Development Guidelines.... 99
4.3.3 Precautions....99
4.3.4 Programming Example.... 99
4.4 CPU Utilization Percentage....100
4.4.1 Overview.... 100
4.4.2 Development Guidelines.... 101
4.4.3 Precautions....102
4.4.4 Programming Example.... 102
4.5 Linux Adaption....104
4.5.1 Completion.... 104
4.5.1.1 Overview.... 104
4.5.1.2 Development Guidelines.... 105
4.5.1.3 Precautions....105
4.5.1.4 Programming Example.... 106
4.5.2 Workqueue....107
4.5.2.1 Overview.... 107
4.5.2.2 Development Guidelines.... 108
4.5.2.3 Precautions....109
4.5.2.4 Programming Example.... 109
4.5.3 Interrupt....111
4.5.3.1 Overview....111
4.5.3.2 Development Guidelines....111
4.5.3.3 Precautions....112
4.5.3.4 Programming Example.... 112
4.5.4 High Resolution Timer.... 113
4.5.4.1 Overview....113
4.5.4.2 Development Guidelines....113
4.5.4.3 Precautions....114
4.5.4.4 Programming Example.... 114
4.5.5 Linux APIs....115
4.5.5.1 Linux Adaption APIs....116
4.5.5.2 Linux APIs Not Supported.... 129

Copyright © Huawei Technologies Co., Ltd.

4.6 C++ Support..... 138
4.6.1 Overview..... 139
4.6.2 Development Guidelines..... 139
4.6.3 Precautions..... 140
4.6.4 Programming Example..... 141
4.7 MMU..... 141
4.7.1 Overview..... 141
4.7.2 Development Guidelines..... 142
4.7.3 Precautions..... 144
4.7.4 Programming Example..... 144
4.8 Atomic Operation..... 146
4.8.1 Overview..... 146
4.8.2 Development Guidelines..... 147
4.8.3 Precautions..... 148
4.8.4 Programming Example..... 148
4.9 Run-Stop..... 149
4.9.1 Overview..... 149
4.9.2 Development Guidelines..... 150
4.9.3 Precautions..... 154
4.9.4 LOS_MakeImage Parameter Configurations..... 155
4.10 Tickless..... 155
4.10.1 Overview..... 155
4.10.2 Development Guidelines..... 156
4.10.3 Precautions..... 156
5 File System..... 157
5.1 Functions Overview..... 157
5.2 VFS..... 158
5.2.1 Overview..... 159
5.2.2 Development Guidelines..... 160
5.2.3 Precautions..... 161
5.2.4 Programming Example..... 162
5.3 NFS..... 162
5.3.1 Overview..... 162
5.3.2 Development Guidelines..... 163
5.3.3 Precautions..... 165
5.3.4 Programming Example..... 165
5.4 JFFS2..... 166
5.4.1 Overview..... 166
5.4.2 Development Guidelines..... 166
5.4.3 Precautions..... 168
5.4.4 Programming Example..... 169
5.5 FAT..... 169

Copyright © Huawei Technologies Co., Ltd.

5.5.1 Overview.... 169
5.5.2 Development Guidelines.... 170
5.5.3 Precautions....178
5.5.4 Programming Example.... 179
5.6 YAFFS2.... 179
5.6.1 Overview.... 179
5.6.2 Development Guidelines.... 180
5.6.3 Precautions....182
5.6.4 Programming Example.... 182
5.7 RAMFS....182
5.7.1 Overview.... 182
5.7.2 Development Guidelines.... 183
5.7.3 Precautions....184
5.7.4 Programming Example.... 184
5.8 PROC....184
5.8.1 Overview.... 184
5.8.2 Development Guidelines.... 184
5.8.3 Precautions....186
5.8.4 Programming Example.... 187
5.9 ROMFS....188
5.9.1 Overview.... 188
5.9.2 Development Guidelines.... 188
5.9.3 Precautions....189
5.9.4 Programming Example.... 189

6 Driver Development.... 190

6.1 Driver Framework.... 190
6.1.1 Overview.... 190
6.1.2 Development Guidelines.... 192
6.1.3 Precautions....195
6.1.4 Programming Example.... 195
6.2 File Adaptation.... 196
6.2.1 Overview.... 197
6.2.2 Development Guidelines.... 197
6.2.3 Precautions....199
6.2.4 Programming Example.... 199

7 Maintenance and Testing....200

7.1 Telnet.... 200
7.1.1 Overview.... 200
7.1.2 Development Guidelines.... 201
7.1.3 Precautions....201
7.1.4 Programming Example....202
7.2 Shell....202

Copyright © Huawei Technologies Co., Ltd.

7.2.1 Overview.... 202
7.2.2 Development Guidelines.... 203
7.2.3 Precautions.... 204
7.2.4 Programming Example.... 205
7.2.5 Command Reference.... 206
7.2.5.1 System Commands.... 206
7.2.5.1.1 task.... 206
7.2.5.1.2 sem.... 209
7.2.5.1.3 swtmr.... 210
7.2.5.1.4 hwi.... 212
7.2.5.1.5 cpup.... 213
7.2.5.1.6 memcheck.... 214
7.2.5.1.7 writereg.... 215
7.2.5.1.8 readreg.... 216
7.2.5.1.9 free.... 217
7.2.5.1.10 uname.... 218
7.2.5.1.11 systeminfo.... 219
7.2.5.1.12 log.... 220
7.2.5.1.13 help.... 221
7.2.5.1.14 date.... 221
7.2.5.1.15 dmesg.... 223
7.2.5.1.16 watch.... 224
7.2.5.2 File.... 225
7.2.5.2.1 ls.... 225
7.2.5.2.2 cd.... 226
7.2.5.2.3 pwd.... 227
7.2.5.2.4 cp.... 228
7.2.5.2.5 cat.... 229
7.2.5.2.6 touch.... 230
7.2.5.2.7 rm.... 231
7.2.5.2.8 sync.... 232
7.2.5.2.9 statfs.... 232
7.2.5.2.10 format.... 233
7.2.5.2.11 mount.... 235
7.2.5.2.12 umount.... 235
7.2.5.2.13 rmdir.... 236
7.2.5.2.14 mkdir.... 237
7.2.5.2.15 partition.... 238
7.2.5.2.16 writeproc.... 239
7.2.5.2.17 partinfo.... 240
7.2.5.2.18 virstatfs.... 240
7.2.5.2.19 lsfd.... 242

Copyright © Huawei Technologies Co., Ltd.

7.2.5.2.20 fat_fsck.... 242
7.2.5.3 Network.... 243
7.2.5.3.1 arp.... 243
7.2.5.3.2 ifconfig.... 244
7.2.5.3.3 ping.... 248
7.2.5.3.4 tftp.... 249
7.2.5.3.5 ntpdate.... 252
7.2.5.3.6 dns.... 252
7.2.5.3.7 netstat.... 253
7.2.5.3.8 telnet.... 255
7.2.5.3.9 pppoe.... 256
7.2.5.3.10 tcpdump.... 257
7.2.5.4 Dynamic Loading.... 259
7.2.5.4.1 mopen.... 259
7.2.5.4.2 findsym.... 259
7.2.5.4.3 call.... 260
7.2.5.4.4 mclose.... 261
7.2.5.4.5 lddrop.... 262

8 Debug Guidelines.... 263

8.1 Common Exception Locating Methods.... 263
8.1.1 Methods for Locating Illegal Memory Write.... 263
8.1.1.1 Locating the Exception Based on the Exception Information.... 263
8.1.1.2 Memory Integrity Check.... 265
8.1.1.3 Check of Usable memset and memcpy Length.... 267
8.1.1.4 Global Variable Check.... 268
8.1.1.5 Task Status Check.... 269
8.1.2 Method for Locating a Deadlock.... 269
8.2 Common Solutions.... 271
8.2.1 Solutions to Illegal Memory Access.... 271
8.2.1.1 Illegal Memory Access Caused by the Audio Library.... 271
8.2.1.2 Unreadable Audio Task Name.... 272
8.2.1.3 Illegal Memory Access Caused by a Global Variable.... 273
8.3 Debug Tools.... 274
8.3.1 LiteDebug.... 274

9 Standard Libraries.... 284

9.1 POSIX APIs.... 284
9.1.1 POSIX Adaption APIs.... 284
9.1.2 POSIX APIs Not Supported.... 302
9.2 Libc/Libm APIs.... 311
9.2.1 Libc Adaption APIs.... 311
9.2.2 Libc Open Source APIs.... 316
9.2.3 Libm Open Source APIs.... 331

Copyright © Huawei Technologies Co., Ltd.

9.2.4 Libc/Libm APIs Not Supported....346
9.3 C++ Compatibility Specifications....347
10 Configuration Reference....350
10.1 Configuration Tool Instructions....350
10.2 Time Management Configuration Parameters....359
10.3 Memory Management Configuration Parameters....360
10.4 Memory Maintenance & Testing Configuration Parameters....361
10.5 Task Configuration Parameters....362
10.6 Software Timer Configuration Parameters....363
10.7 Semaphore Configuration Parameters....363
10.8 Mutex Configuration Parameters....364
10.9 Hardware Interrupt Configuration Parameters....364
10.10 Queue Configuration Parameters....365
10.11 Module Compaction Configuration Parameters....365
11 Appendix....368
11.1 OS Memory Usage....368
11.2 Kernel Boot Process Introduction....370
11.3 Compatibility Modifications in 32-Bit or 64-Bit Environments....372

Copyright © Huawei Technologies Co., Ltd.

1 Preface

Purpose

This guide describes the structure of Huawei LiteOS Kernel. It also explains how to develop and debug the Kernel.

Intended Audience

This guide is primarily intended for Huawei LiteOS Kernel developers, and is also recommended for:

- Internet of Things (IoT) device software engineers
- IoT architects

Symbol Conventions

The symbols that may be found in this guide are defined as follows:

Symbol	Description
DANGER	Indicates an emergency hazardous situation that, if not avoided, could result in death or serious injury.
WARNING	Indicates a potentially hazardous situation that, if not avoided, could result in death or serious injury.
CAUTION	Indicates a potentially hazardous situation that, if not avoided, could result in minor or moderate injury.

Copyright © Huawei Technologies Co., Ltd.

Symbol	Description
NOTICE	Indicates a device or environmental safety alert that, if not avoided, could result in equipment damage, data loss, performance deterioration, or unanticipated consequences. "Notice" cannot result in injury.
NOTE	"Note" is not a safety alert, cannot result in personal, device or environmental injury.

Change History

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

Date	Version	Description
2015-10-28	1.0	Initial draft
2016-03-26	2.0	Manual optimization

Copyright © Huawei Technologies Co., Ltd.

2 Overview

About This Chapter

2.1 Background Introduction

2.2 Supported Cores

2.3 Constraints

2.1 Background Introduction

Huawei LiteOS Kernel is intended for lightweight real-time operating systems designed for IoT OS Kernel of Huawei.

Huawei Proprietary and Confidential
Copyright © Huawei Technologies Co., Ltd.

Figure 2-1 Huawei LiteOS Kernel framework

Copyright © Huawei Technologies Co., Ltd.

Huawei LiteOS Basic kernel of Huawei LiteOS is the most tidy code of operating system. It contains operating system components based task management, time management, communication mechanism, interrupt management, queue management, event management, timer, exception management, etc. It can run independently.

Highlights of Huawei LiteOS Kernel

- Highlight real-time and stable
- Ultra-small kernel, basic kernel size of less than 10 KB
- Low power consumption
- Capable of dynamic and scatter loading
- Capable of Static function compaction

Modules

Task

Creates, deletes, delays, suspends, and resumes tasks, and can lock or unlock task scheduling. High priority tasks preempt resources from low priority ones. Tasks of the same priority share resources in a round robin setup using time slicing.

Task Synchronization

- Semaphore: creates, deletes, pends on, and releases semaphores.
- Mutex: creates, deletes, pends on, and releases mutexes.

Hardware Related Functions

Provides the following functions:

- Interrupt: Creates, deletes, enables, and disables interrupts; clears interrupt request flags.
- Timer: Creates, deletes, starts, and stops timers.

Inter-Process Communication (IPC)

Provides the following functions:

- Event: Reads and writes events
- Message queue: Creates, deletes, reads from, and writes into message queues

Time Management

- System time: generated when an output pulse of a timer/counter triggers an interrupt.
- Tick time: the basic time unit used in OS scheduling. The tick length is user configurable. Typically, it is determined by the system clock speed and represented in the form of ticks per second.
- Software timer: The timer length is measured in ticks. The Timer_Callback function (a function used to process timer expiry) is called when a soft tick interrupt is generated.

Memory Management

- Provide two algorithms of dynamic memory and static memory. Allocates or frees memory statically using the Membox algorithm or dynamically using the DLINK algorithm.
- Provides memory statistics, cross-border detection memory.

Copyright © Huawei Technologies Co., Ltd.

Exception Handling

Exception handling means that when the operating system encounters an exception, in order to save the current OS state or print the information stored in the call stack of the erroneous function, it switches to the hook function responsible for exception handling.

The printed register information of Huawei LiteOS exception handling includes the erroneous task ID, stack size, and LR/PC pointer.

Dynamic Loading

Dynamic loading a software loading technology that loads and links only the required module files at runtime of an executable instead of loading all modules files of the executable.

Two file formats are supported: OBJ and SO.

Scatter Loading

Scatter loading preferentially loads key services by loading images of key services into memory. This accelerates system boot.

2.2 Supported Cores

Table 2-1 Cores supported by Huawei LiteOS

Core	Chip
ARM9, Cortex-A7, Cortex-A17, Cortex-A53	Hi3516A, Hi3911, Hi3518eV200, Hi3556, Hi3731, Hi3559A
Cortex-M0, Cortex-M3, Cortex-M4, Cortex-M7	STM32 series, NB-IoT series, Kirin series, NXP, GD32, Atmel, NordicNRF series, Nuvoton, MTK
Cortex-R7, Cortex-R8	Kirin series
Tensilica DSP LX7, LX4	Kirin series, Hi3559A, Hi3796MV200, Hi3751V550
Intel, RISC-V	M2S150_RV32

2.3 Constraints

- Both Huawei LiteOS interfaces and POSIX interfaces are supported, but hybrid use of them may lead to unpredictable results. (For example, a POSIX interface is used for requesting semaphores while a Huawei LiteOS interface is used for releasing semaphores.)
- Use only Huawei LiteOS interfaces for driver development. POSIX interfaces are recommended for app development.

Copyright © Huawei Technologies Co., Ltd.

3 Basic Kernel

About This Chapter

3.1 Task

3.2 Memory

3.3 Interrupt

3.4 Queue

3.5 Event

3.6 Mutex

3.7 Semaphore

3.8 Time Management

3.9 Software Timer

3.10 Error Handling

3.11 Doubly Linked List

3.12 Bitwise Operation

3.1 Task

3.1.1 Overview

Basic Concept

Task is the minimum running unit of competitive system resources from a system perspective. It can use or wait for CPU, use memory space, and can run independently of other tasks.

Task modules of Huawei LiteOS provide a lot of tasks to help users manage business process procedures. It makes switches and communications between tasks come true. Through this, users can devote more energies to the achievement of business function.

Copyright © Huawei Technologies Co., Ltd.

Huawei LiteOS is an operating system supported multi-task. In Huawei LiteOS, a task is same as a thread.

Task in Huawei LiteOS is preemptive scheduling mechanism, while supporting round-robin scheduling.

High-priority task can interrupt low-priority task, low-priority task can only be scheduled when the high-priority task blocked or completed.

A total of 32 priorities are defined, with priority 0 being the highest and 31 being the lowest.

Related Concepts

Task States

A task in Huawei LiteOS switches between different states. After the operating system is initialized, a created task is allowed to contend for system resources according to the scheduling procedure regulated by Huawei LiteOS Kernel.

There are usually four task states:

- Ready: The task is waiting for execution by a CPU.
- Running: The task is being executed.
- Blocked: The task is not on the list of ready tasks. For example, the task may be suspended, delayed, waiting for a semaphore, waiting to read from a queue, or reading from or writing into a queue.
- Dead: The task execution is complete, and resources are waiting to be reclaimed.

Figure 3-1 Task state schematic diagram

The state transition process is as follows:

- Ready → Running

A task enters Ready state once created. When a task switch occurs, the task with the highest priority is selected from ready tasks and enters Running state to be executed. Although the task is in Running state, it remains on the list of ready tasks.

- Running → Blocked

When a running task is blocked (for example, it is suspended, delayed, or waiting to read a semaphore), it will be deleted from the list of ready tasks and enters Blocked state. The state transition triggers a task switch where the task with the highest priority is selected from ready tasks.

- Blocked → Ready (Blocked → Running)

Copyright © Huawei Technologies Co., Ltd.

After a blocked task is recovered (for example, if the task is resumed, the task successfully reads a semaphore, or if the delay period or semaphore read period expires), the task will be added to the list of ready tasks and enters ready state. If the recovered task takes precedence over the running task, a task switch will occur to send the resumed task into running state.

- Ready → Blocked

If a ready task is blocked (suspended), it will be deleted from the list of ready tasks no longer participated in task scheduling and enter blocked state.

- Running → Ready

When a task is created or resumed with a higher priority than the running task, the created or resumed task enters running state and task scheduling will be occurred. Meanwhile, the original running task enters ready state but it remains on the list of ready tasks.

- Running → Stopped

When a task running is stopped, the status of it will change from running to stopped. Stopped status includes normal exit after the task is stopped and Impossible status. For example, while separation property(LOS_TASK_STATUS_DETACHED) is not set, the task will present Impossible status, which is stopped.

- Blocked → Stopped

If calling the delete API when the task is in blocked status, the task status will change from blocked to stopped.

Task ID

You will receive a task ID after successfully creating a task. You may suspend, resume, or query a task using its ID.

Task Priority

Tasks are executed based on their priority. In the event of a task switch, the task with the highest priority will be selected from ready tasks.

Task Entrypoint Function

Each task has a task entrypoint function, which is defined by the task creation structure at the time of task creation and is executed after the task is scheduled. You can design task entrypoint functions.

Task Control Block

Each task has a task control block (TCB). A TCB contains task information such as context stack pointer (SP), state, priority, ID, name, and stack size. TCB can reflect running conditions of each task.

Task Stack

Each task has a separate task stack. The task stack stores information such as local variables, registers, function parameters, and function return addresses. When a task switch occurs, the context information of the task that is replaced is saved to its task stack. When the task is resumed, its context information will be quickly retrieved from the task stack to help resume the task from where it was paused.

Task Context

Copyright © Huawei Technologies Co., Ltd.

Resources (such as registers) used by a running task are collectively known as task context, just like registers. After a task is suspended, other running tasks might modify the context of the suspended task. If the original context of the suspended task is not saved, the suspended task uses the modified context once resumed, incurring unpredictable errors.

Therefore, Huawei LiteOS will save the task context information of this task in its own task stack. This function is to resume context information after the task is resumed. There by continuing to execute the interrupted code when the task is suspended.

Task Switch

A task switch process involves a few activities, including selecting the ready task with the highest priority, saving the context of the task that will be replaced, and restoring the context of the task that is newly selected to be executed.

Operation Mechanism

Task management module of Huawei LiteOS provides functions such as task creating, delaying, suspending, resuming, locking and unlocking task scheduling, querying task ID according to TCB, and querying TCB information according to ID.

In the linking phase, Huawei LiteOS allocates memory to TCBs. Linking fails if the available memory space in the OS is smaller than the memory space required by the TCBs.

While creating a task, the operating system initializes the task stack and resets the context. The operating system also places the task entrypoint function in the correct position so that the function will be executed after the task is booted for the first time.

3.1.2 Development Guidelines

Usage Scenarios

After a task is created, Huawei LiteOS Kernel can perform operations such as unlocking task scheduling, scheduling/suspending/resuming/delaying a task, or assigning/acquiring a task priority. If the task state is Detached (**LOS_TASK_STATUS_DETACHED**) when the task ends, the task will be detached.

Functions

The task management module provides the following functions:

Function Category	API	Description
Task creation and deletion	LOS_TaskCreateOnly	Creates a task and suspends the task without scheduling it
	LOS_TaskCreate	Creates a task. The task enters Ready state and is scheduled
	LOS_TaskDelete	Deletes a particular task
Task state control	LOS_TaskResume	Resumes the suspended task

Copyright © Huawei Technologies Co., Ltd.

Function Category	API	Description
	LOS_TaskSuspend	Suspends a particular task
	LOS_TaskDelay	Delays the task
	LOS_TaskYield	Explicits decentralization, and adjusts the scheduling order of tasks with a particular priority
Task scheduling control	LOS_TaskLock	Locks task scheduling
	LOS_TaskUnlock	Unlocks task scheduling
Task priority control	LOS_CurTaskPriSet	Assigns a priority to the current task
	LOS_TaskPriSet	Set the priority of a particular task
	LOS_TaskPriGet	Gets the priority of a particular task
Task information acquisition	LOS_CurTaskIDGet	Gets the ID of the current task
	LOS_TaskInfoGet	Gets the information of the current task

Development Process

Task creation is used as an example to explain the development process.

1. Configure the task management module in the los_config.h file.

   LOSCFG_BASE_CORE_TSK_IDLE_STACK_SIZE IDLE: task stack size. Retain the default value unless otherwise required. You can config according to requirement.

   LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE: default task stack size. Specify the parameter value according to actual needs when users create tasks.

   LOSCFG_BASE_CORE_TIMESLICE: a switch to enable or disable the Time Slice. Set it to YES.

   LOSCFG_BASE_CORE_TIMESLICE_TIMEOUT: time slice. You can config according to actual situations.

   LOSCFG_BASE_CORE_TSK_MONITOR: a switch to enable or disable the task monitoring module.
2. Configure the number of tasks in the los_config.h file.

   LOSCFG_BASE_CORE_TSK_CONFIG: the maximum number of tasks allowed. You can config according to requirement.
3. Call the LOS_TaskLock API to lock task scheduling. Prohibits high-priority task scheduling.
4. Call the LOS_TaskCreate API to create a task.

Copyright © Huawei Technologies Co., Ltd.

5. Call the LOS_TaskUnlock API to unlock task scheduling. Schedules tasks in order of priority.
6. Schedules tasks in order of priority. Delays the task.
7. Call the LOS_TaskSuspend API to suspend the task. Suspends the task.
8. Call the LOS_TaskResume API to resume the suspended task. Resumes the suspended task.

Task State

In Huawei LiteOS, most task states are defined by the kernel. Only Detached state can be defined by users. Users need to define Detached state during task creation.

No.	Definition	Value	Description
1	LOS_TASK_STATUS_DETACHED	0x0100	The task is detached.

When creating a task by calling the LOS_TaskCreate API, set the **uwResved** field of the **TSK_INIT_PARAM_S** parameter of the task to **LOS_TASK_STATUS_DETACHED**. Then the task will be detached after its execution is complete.

NOTE

When creating a task by calling the LOS_TaskCreate API, the task state is set to **LOS_TASK_STATUS_DETACHED**. by default.

Task Error Codes

An error code is returned when attempting to create, delete, suspend, resume, or delay a task fails. The error code gives some insights into the possible cause of the failure.

SN	Error Code	Error ID Number	Description	Recommended Solution
1	LOS_ERRNO_TSK_NO_MEMORY	0x02000200	Insufficient memory	Allocate a larger memory area
2	LOS_ERRNO_TSK_PTR_NULL	0x02000201	Null task parameter	Check task parameters
3	LOS_ERRNO_TSK_STKSZ_NOT_ALIGN	0x02000202	Task stack size not aligned	Align the task stack size on the boundary
4	LOS_ERRNO_TSK_PRIOR_ERROR	0x02000203	Incorrect task priority	Check the task priority

Copyright © Huawei Technologies Co., Ltd.

SN	Error Code	Error ID Number	Description	Recommended Solution
5	LOS_ERRNO_TSK_ENTRY_NULL	0x02000204	Null task entrypoint function	Define a task entrypoint function
6	LOS_ERRNO_TSK_NAME_EMPTY	0x02000205	Task name unspecified	Specify the task name
7	LOS_ERRNO_TSK_STKSZ_TOO_SMALL	0x02000206	Too small task stack	Expand the task stack
8	LOS_ERRNO_TSK_ID_INVALID	0x02000207	Invalid task ID	Check task IDs
9	LOS_ERRNO_TSK_ALREADY_SUSPENDED	0x02000208	Task already suspended	Suspend the task after it is resumed
10	LOS_ERRNO_TSK_NOT_SUSPENDED	0x02000209	Task not suspended	Suspend the task
11	LOS_ERRNO_TSK_NOT_CREATED	0x0200020a	Task not created	Create the task
12	LOS_ERRNO_TSK_DELETE_LOCKED	0x0200020b	Attempt to delay the task while task scheduling is locked	Delete the task after task scheduling is unlocked
13	LOS_ERRNO_TSK_MSG_NONZERO	0x0200020c	Task information not zero	Do not use the error code
14	LOS_ERRNO_TSK_DELAY_IN_INT	0x0300020d	Attempt to delay the task while an interrupt is underway	Delay the task after the interrupt is finished
15	LOS_ERRNO_TSK_DELAY_IN_LOCK	0x0200020e	Attempt to delay the task while task scheduling is locked	Delay the task after task scheduling is unlocked

Copyright © Huawei Technologies Co., Ltd.

SN	Error Code	Error ID Number	Description	Recommended Solution
16	LOS_ERRNO_TSK_YIELD_INVALID_TASK	0x0200020f	Invalid task to be scheduled	Check the task
17	LOS_ERRNO_TSK_YIELD_NOT_ENOUGH_TASK	0x02000210	No task or only one task available for scheduling	Add more tasks
18	LOS_ERRNO_TSK_TCB_UNAVAILABLE	0x02000211	No idle TCB	Add more TCBs
19	LOS_ERRNO_TSK_HOOK_NOT_MATCH	0x02000212	Task hook function mismatch	Do not use the error code
20	LOS_ERRNO_TSK_HOOK_IS_FULL	0x02000213	Maximum number of task hook functions is reached	Do not use the error code
21	LOS_ERRNO_TSK_OPERATE_IDLE	0x02000214	Idle task	Check the task ID and do not attempt to operate the task with the ID
22	LOS_ERRNO_TSK_SUSPEND_LOCKED	0x03000215	Attempt to suspend the task while task scheduling is locked	Suspend the task after task scheduling is unlocked
23	LOS_ERRNO_TSK_FREE_STACK_FAILED	0x02000217	Failed to free task stack	Do not use the error code
24	LOS_ERRNO_TSK_STKAREA_TOO_SMALL	0x02000218	Small task stack area	Do not use the error code
25	LOS_ERRNO_TSK_ACTIVE_FAILED	0x02000219	Failed to trigger the task	Create an idle task and trigger a task switch
26	LOS_ERRNO_TSK_CONFIG_TOO_MANY	0x0200021a	Too many task configuration options	Do not use the error code

Copyright © Huawei Technologies Co., Ltd.

SN	Error Code	Error ID Number	Description	Recommended Solution
27	LOS_ERRNO_TSK_CP_SAVE_AREA_NOT_ALIGN	0x0200021b	None	Do not use the error code
28	LOS_ERRNO_TSK_MSG_Q_TOO_MANY	0x0200021d	None	Do not use the error code
29	LOS_ERRNO_TSK_CP_SAVE_AREA_NULL	0x0200021e	None	Do not use the error code
30	LOS_ERRNO_TSK_SELF_DELETE_ERR	0x0200021f	None	Do not use the error code
31	LOS_ERRNO_TSK_STKSZ_TOO_LARGE	0x02000220	Large task stack	Reduce the task stack size
32	LOS_ERRNO_TSK_SUSPEND_SWTMR_NOT_ALLOWED	0x02000221	Suspension of a software timer task not allowed	Check the task ID. Do not attempt to suspend a software timer task.
33	LOS_ERRNO_TSK_DELAY_IN_SWTMR_TSK	0x03000222	LOS_TaskDelay function called in the software timer callback function.	Do not call the LOS_TaskDelay function in the software timer callback function.

Error Code Definition

An error code is 32 bits in length, where:

- Bits 31 – 24: error severity
- Bits 23 – 16: error flag
- Bits 15 – 8: module that encounters the error
- Bits 7 – 0: error ID number

```
#define LOS_ERRNO_OS_NORMAL(MID,ERRNO)  \
(LOS_ERRTYPE_NORMAL | LOS_ERRNO_OS_ID | ((UINT32)(MID) << 8) | (ERRNO))
LOS_ERRTYPE_NORMAL: Define the error level as critical
LOS_ERRNO_OS_ID: OS error code flag.
MID: OS_MOUDLE_ID
ERRNO: error ID number
```

For example:

Copyright © Huawei Technologies Co., Ltd.

```
LOS_ERRNO_TSK_NO_MEMORY  LOS_ERRNO_OS_FATAL(OS_MOD_TSK, 0x00)
```

NOTE

0x03000215 and 0x0300021c error codes are not defined and therefore cannot be used.

Platform Differences

None.

3.1.3 Precautions

- While a new task is being created, the task control blocks (TCBs) and task stacks of previously deleted tasks are reclaimed.
- A task name is a pointer and not allocated memory space. Do not set a task name to the address of a local variable when you set task name.
- If the task size is set to 0, the setting does not take effect. Instead, the default task size defined by the #LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE parameter is applied.
- Task stack size is aligned with the base address on the boundary of 8 bytes. Follow the "nothing more and nothing less" principle while determining the task stack size.
- A running task cannot be suspended while current task scheduling is locked.
- Idle and software timer tasks must not be suspended or deleted.
- In the interrupt handler function or in the case of the lock task, the operation that calls the LOS_TaskDelay API will fail.
- Locking task scheduling does not disable interrupts. Tasks can still be interrupted while task scheduling is locked.
- Locked task scheduling and unlocked task scheduling must be used in coordination.
- Task scheduling may occur while a task priority is being set.
- The maximum number of tasks (excluding idle tasks) able to be set by operating system is not equal to the total number of tasks available to users. For example, when a task is created for software timers, the number of available tasks is decreased by 1.
- Do not change the priority of a software timer task by calling the LOS_CurTaskPriSet API or the LOS_TaskPriSet API. Otherwise, system problems may occur.
- The LOS_CurTaskPriSet or LOS_TaskPriSet API must not be used when interrupts are being processed.
- If the corresponding task ID that LOS_TaskPriGet interface into the task is not created or exceed the maximum number of tasks, unified return 0xffff.
- Resources such as a mutex or a semaphore allocated to a task must have been released when the task is being deleted.

3.1.4 Programming Example

Example Description

Two tasks will be created: TaskHi and TaskLo. TaskHi has a higher priority than TaskLo.

There are some examples giving some basic insight into priority-based task scheduling and use cases of APIs, including create, delay, lock, unlock, suspend, resume, and query (task ID and information by task ID) a task.

Copyright © Huawei Technologies Co., Ltd.

1. Two tasks will be created: TaskHi and TaskLo.
2. TaskHi has a higher priority
3. TaskLo has a lower priority.

Example Code

```
UINT 32  g_uwTskLoID;
UINT 32 g_uwTskHiID;
#define TSK_PRIOR_HI 4
#define TSK_PRIOR_LO 5

UINT32 Example_TaskHi(VOID)
{
    UINT32 uwRet;
    UINT32 uwCurrentID;
    TSK_INFO_S stTaskInfo;

     printf("Enter TaskHi Handler.\r\n");

    /*Delay TaskHi for 2 ticks. The delayed TaskHi will be suspended. Meanwhile,
TaskLo will rise to the highest priority task among the remaining tasks and be
selected for execution(g_uwTskLoID task).*/
    uwRet = LOS_TaskDelay(2);
    if (uwRet != LOS_OK)
    {
        printf("Delay Task Failed.\r\n");
        return LOS_NOK;
    }

    /*Resume the task when 2 ticks elapse.*/
    printf("TaskHi LOS_TaskDelay Done.\r\n");

     /*Suspend the task.*/
    uwRet = LOS_TaskSuspend(g_uwTskHiID);
    if (uwRet != LOS_OK)
    {
        printf("Suspend TaskHi Failed.\r\n");
        return LOS_NOK;
    }
    printf("TaskHi LOS_TaskResume Success.\r\n");
}

/*Task entrypoint function for the TaskLo*/
UINT32 Example_TaskLo(VOID)
{
    UINT32 uwRet;
    UINT32 uwCurrentID;
    TSK_INFO_S stTaskInfo;

    printf("Enter TaskLo Handler.\r\n");

     /*Delay TaskLo for 2 ticks. The delayed TaskLo will be suspended. Meanwhile,
the background task will rise to the highest priority task among the remaining
tasks and be selected for execution.*/
    uwRet = LOS_TaskDelay(2);
    if (uwRet != LOS_OK)
    {
        printf("Delay TaskLo Failed.\r\n");
        return LOS_NOK;
    }

    printf("TaskHi LOS_TaskSuspend Success.\r\n");

    /*Resume the suspended task g_uwTskHiID.*/
    uwRet = LOS_TaskResume(g_uwTskHiID);
    if (uwRet != LOS_OK)
    {
        printf("Resume TaskHi Failed.\r\n");
```


Copyright © Huawei Technologies Co., Ltd.

```
        return LOS_NOK;
    }

    printf("TaskHi LOS_TaskDelete Success.\r\n");
}

/*Task test entrypoint function. Two tasks with different priorities will be
created.*/
UINT32 Example_TskCaseEntry(VOID)
{
    UINT32 uwRet;
    TSK_INIT_PARAM_S stInitParam;

    /*Lock task scheduling.*/
    LOS_TaskLock();

    printf("LOS_TaskLock() Success!\r\n");

    stInitParam.pfnTaskEntry = (TSK_ENTRY_FUNC)Example_TaskHi;
    stInitParam.usTaskPrio = TSK_PRIOR_HI;
    stInitParam.pcName = "HIGH_NAME";
    stInitParam.uwStackSize = 0x700;
    stInitParam.uwResved   = LOS_TASK_STATUS_DETACHED;
    /*Create a task with a high priority. The task will not be executed
immediately after being created, because task scheduling is locked.*/
    uwRet = LOS_TaskCreate(&g_uwTskHiID, &stInitParam);
    if (uwRet != LOS_OK)
    {
        LOS_TaskUnlock();

        printf("Example_TaskHi create Failed!\r\n");
        return LOS_NOK;
    }

    printf("Example_TaskHi create Success!\r\n");

    stInitParam.pfnTaskEntry = (TSK_ENTRY_FUNC)Example_TaskLo;
    stInitParam.usTaskPrio = TSK_PRIOR_LO;
    stInitParam.pcName = "LOW_NAME";
    stInitParam.uwStackSize = 0x700;
    stInitParam.uwResved   = LOS_TASK_STATUS_DETACHED;
    /*Create a task with a low priority. The task will not be executed
immediately after being created, because task scheduling is locked.*/
    uwRet = LOS_TaskCreate(&g_uwTskLoID, &stInitParam);
    if (uwRet != LOS_OK)
    {
        LOS_TaskUnlock();

        printf("Example_TaskLo create Failed!\r\n");
        return LOS_NOK;
    }

    printf("Example_TaskLo create Success!\r\n");

    /*Unlock task scheduling. Task scheduling will occur, selecting the task with
the highest priority from the list of ready tasks to be executed.*/
    LOS_TaskUnlock();

    while(1){};

    return LOS_OK;
}
```

Verification

The verification result is as follows:

Copyright © Huawei Technologies Co., Ltd.

```
 --- Test start---
LOS_TaskLock() Success!
Example_TaskHi create Success!
Example_TaskLo create Success!
Enter TaskHi Handler.
Enter TaskLo Handler.
TaskHi LOS_TaskDelay Done.
TaskHi LOS_TaskSuspend Success.
TaskHi LOS_TaskResume Success.
TaskHi LOS_TaskDelete Success.
```

Complete Code

sample_task.c

3.2 Memory

3.2.1 Overview

Basic Concept

The memory management module is one of the core modules of an operating system. Memory management primarily involves initializing, allocating, and freeing up memory.

While the operating system is running, the memory management module manages memory usage of users and the operating system by allocating and freeing up memory. This helps reduce memory fragments as much as possible.

Memory management is classified into static and dynamic memory management.

- Dynamic memory: a memory block of user-defined size
  - Advantage: on-demand memory allocation
  - Disadvantage: risk of memory fragments
- Static memory: a memory block whose size is predefined at the time of initialization
  - Advantages: no memory fragments; efficient memory allocation and freeing
  - Disadvantage: memory cannot be allocated on demand

Dynamic Memory Operation Mechanism

Dynamic memory management means taking a memory block of the required size out of the large pool of continuous memory whenever a user needs it, and reclaiming the memory block when the user no longer needs it.

Comparing with static memory, the advantage is to allocated a memory block of the required size, and the disadvantage is that memory pool prone to fragmentation.

If a user's request for memory is fulfilled, the user will be allocated a memory block of the requested size. The control header indicates the start address of the allocated memory block.

All control headers are recorded in a linked list and categorized by memory size. From the linked list, the operating system can quickly find which memory block has the required size.

Figure 3-2shows the dynamic memory management structure in Huawei LiteOS:

Copyright © Huawei Technologies Co., Ltd.

Figure 3-2

Part one indicates the start address and size of the heap memory (memory pool).

Part two is an array of which each element is a doubly linked list. Control headers of all free nodes are categorized and mounted to the doubly linked lists in this array.

If the smallest node allowed by the memory is 2^{min} bytes, the first doubly linked list in the array stores free nodes of the size that is bigger than 2^{min} and smaller than 2^{min+1}. The second doubly linked list in the array stores free nodes of the size that is bigger than 2^{min+1} and smaller than 2^{min+2}. The nth doubly linked list in the array stores free nodes of the size that is bigger than $2^{min+n-1}$ and smaller than 2^{min+n}. When memory is allocated, a free node of appropriate size (the size of the node being created) is located and memory is allocated to the free node. When memory is freed up, the freed memory is stored to the array as free nodes for later use.

Part three uses most space in the memory pool and is the actual area that stores nodes. The LOS_MEM_DYN_NODE node structure is described as follows:

```
typedef struct tagLOS_MEM_DYN_NODE
{
LOS_DL_LIST stFreeNodeInfo;
struct tagLOS_MEM_DYN_NODE *pstPreNode;
UINT32 uwSizeAndFlag;
}LOS_MEM_DYN_NODE;
```

Figure 3-3

Copyright © Huawei Technologies Co., Ltd.

If the LOSCFG_MEM_HEAD_BACKUP macro is enabled, the memory node backup function is enabled, and the LOS_MEM_DYN_NODE structure becomes bigger, as shown in **Figure 3**. The backup performs backup on the content of the node head to check whether there are illegal memory writes and to repair the node head when the magic is not corrupted.

Figure 3-4

NOTE

The backup mechanism is being optimized. The backup of a memory node subsequently may not back up the node head of this node. When this backup function is enabled, a node control header (including the backup) will use extra memory.

Static Memory Operation Mechanism

Static memory is in essence a static array. The size of a static memory block is defined at the time of initialization and cannot be changed since then.

A static memory pool consists of a control block and several memory blocks of same size. The control block is placed at the head of the static memory pool to manage memory blocks. The allocate and free up of memory block according to the size of the blocks.

Figure 3-5 Static memory

3.2.2 Dynamic Memory

Copyright © Huawei Technologies Co., Ltd.

3.2.2.1 Development Guidelines

Usage Scenarios

The main task of memory management is to dynamically partition and manage user allocated memory intervals.

Dynamic memory management is used when users have different demands on memory blocks.

When a user allocates a memory block of specified size, the operating system calls the LOS_AllocMem API to allocate the requested amount of memory. When the user no longer needs the memory block, the operating system calls the LOS_FreeMem API to free up the memory block.

Functions

The memory management module in Huawei LiteOS System provides the following functions. For details about the APIs, see the API reference.

Function Category	API	Description
Memory initialization	LOS_MemInit	Initializes a specific dynamic memory pool.
Memory pool deletion	LOS_MemDeInit	Deletes a specified memory pool and is valid only when the LOSCFG_MEM_MUL_POOL is enabled.
Dynamic memory allocation	LOS_MemAlloc	Allocates a specific dynamic memory pool block of specified size.
Dynamic memory free up	LOS_MemFree	Frees up the allocated dynamic memory block
Memory reallocation	LOS_MemRealloc	Reallocates memory block according to the size, and retains data in the previously allocated memory area.
Aligned memory allocation	LOS_MemAllocAlign	Takes the memory block of requested specific size out of the specific dynamic memory pool and aligns the head or tail of the memory address with a base address on the predefined boundary.
Checking memory size	LOS_MemPoolSizeGet	Gets the size of a specific dynamic memory pool.

Copyright © Huawei Technologies Co., Ltd.

Function Category	API	Description
Checking memory usage	LOS_MemTotalUsedGet	Gets the usage of a specific dynamic memory pool.
Checking the number of memory blocks	LOS_MemFreeBlksGet	Gets the number of free blocks in a specific dynamic memory pool.
Checking the number of memory blocks	LOS_MemUsedBlksGet	Gets the number of used blocks in a specific dynamic memory pool.
Checking a task ID	LOS_MemTaskIdGet	Gets the ID of the task to which specific memory is allocated.
Obtaining the last node address	LOS_MemLastUesdGet	Obtains the end address of the last used node in the memory pool.
Checking memory structure information	LOS_MemInfoGet	Gets the memory structure information of a specific memory pool.
Integrity check	LOS_MemIntegrityCheck	Checks the integrity of a specific memory pool.
Checking node size	LOS_MemNodeSizeCheck	Checks the total size of a node and the size of part of node that can be operated.
Setting memory check level	LOS_MemCheckLevelSet	Sets the memory check level.
Checking memory check level	LOS_MemCheckLevelGet	Gets the memory check level.
Listing memory pools	LOS_MemPoolList	Lists all memory pools that have been initialized in the OS and is valid only when the LOSCFG_MEM_MUL_POOL is enabled.
Dynamic memory allocation	LOS_MemMalloc	Takes memory out of a specified dynamic memory pool and allocates the memory to a specified module, and records the amount of memory allocated to the module.
Dynamic memory free up	LOS_MemMfree	Frees up a memory block that has been allocated to a specified module and records the amount of memory that is freed up.

Copyright © Huawei Technologies Co., Ltd.

Function Category	API	Description
Aligned memory allocation	LOS_MemMallocAlign	Allocates the memory that is of *size* and is aligned on a specified boundary from a specified dynamic memory pool and records the amount of the allocated memory.
Memory reallocation	LOS_MemMrealloc	Reallocates a memory block of *size* to a specified module, with the data in the memory block that is previously allocated the module retained, and records the amount of the reallocated memory.
Obtaining memory usage	LOS_MemMusedGet	Gets the memory usage of a specified module.

Development Process

1. Configuration:

   OS_SYS_MEM_ADDR: start address of the dynamic memory pool. In most cases, retain the default value.

   OS_SYS_MEM_SIZE: size (in bytes) of the dynamic memory pool. By default, the dynamic memory pool is the memory space that is left unused after DDR is allocated.

   LOSCFG_BASE_MEM_NODE_INTEGRITY_CHECK: a switch to enable or disable memory overwriting check. Default value: disabled. If enabled, the operating system carries out the memory overwriting check when a dynamic memory block is allocated or a static memory block is freed.

2. LOS_MemInit initialization

   The result of initializing a dynamic memory pool is shown as the figure bellow, generating a EndNode, and all the memory left signed to be FreeNode. Notice: EndNode as the last node in memory pool with size 0.

3. LOS_MemAlloc for allocating a dynamic memory block of any sizes

   Determines whether the required amount of memory is available. If available, it takes a dynamic memory block of requested size out of the large continuous memory and returns the pointer of the dynamic memory block to the user. If unavailable, it returns NULL to the user.

   Call the LOS_MemAlloc API three times to create three nodes. Assumes that their names are UsedA, UsedB, and UsedC. Their sizes are sizeA, sizeB, and sizfeC. Because there is only one large FreeNode in the memory pool when the pool is just initialized, these memory blocks cut from the FreeNode.

Copyright © Huawei Technologies Co., Ltd.

If malloc occurred when there are many FreeNodes in the memory pool, memory block will be created with the FreNode that malloced with the most benefited size to reduce memory fragmentation. If the size of the new one is not equal to the used one, the redundant memory block will be signed as a new FreeNode after creating a new memory block.

4. LOS_MemFree for free up dynamic memory

   Reclaims the dynamic memory block for using next time.

   Suppose that calling the LOS_MemFree to free up memory block UsedB, the memory block UsedB will be reclaimed and signed as FreeNode.

Platform Differences

None.

3.2.2.2 Precautions

- Dynamic memory management consumes the memory of the management control block structure. Therefore, the memory space available to users is smaller than the OS_SYS_MEM_SIZE defined in the los_config.h file.
- Calls to the LOS_MemAllocAlign API may consume a certain amount of memory and result in memory loss. When the memory used for alignment is freed up, the lost memory will be reclaimed.
- During memory reallocation to a user by using the LOS_MemRealloc API, the operating system determines whether sufficient continuous memory is adjacent to the memory area that has been allocated to the user. If adjacent memory is insufficient, the operating system frees up the previously allocated memory area and finds a new memory area for the user. If the reallocation fails, the previously allocated memory remains unchanged, and NULL will be returned. The use of pPtr = LOS_MemRealloc(pool, pPtr, uwSize) is not allowed, indicating that using the original pPtr to receive the returned value is forbidden.
- If the same memory block is repeatedly freed using the LOS_MemFree API, the first free-up operation receives an operation succeed message. However, subsequent free-up attempts lead to invalid operations on the pointer of the memory block and ultimately unpredictable results.
- The dynamic memory controller (DMC) structure uses the UINT32 data type, with the most significant two bits as flags. Therefore, the size of the initial memory pool cannot exceed 1 GB. Otherwise, unexpected results may occur.

Copyright © Huawei Technologies Co., Ltd.

3.2.2.3 Programming Example

Example Description

Memory is a scarce resource. If memory is frequently used while the operating system is running, program the memory management module to allocate and free up memory efficiently.

In the programming example, the following steps will be performed:

1. Initialize a dynamic memory pool.
2. Take a memory block out of the initialized memory pool and allocate it to a user.
3. Store data in the memory block.
4. Print the data in the memory block.
5. Free up the memory block.

Example Code

```
VOID los_memory_test() {
     UINT32 *p_num = NULL;
     UINT32 uwRet;
     uwRet = LOS_MemInit(m_aucSysMem0, 32);
     if (LOS_OK  == uwRet) {
         dprintf("Memory pool initialized successfully!\n");
     }
     else {
         dprintf("Failed to initialized the memory pool!\n");
         return;
     }
     /*Allocate a memory block.*/
     p_num = (int*)LOS_MemAlloc(m_aucSysMem0, 4);
     if (NULL == p_num) {
         dprintf("Failed to allocate the memory block!\n");
         return;
     }
     dprintf("Memory block allocated successfully!\n");
     /*Use the memory block.*/
     *p_num = 828;
     dprintf("*p_num = %d\n", *p_num);
     /*Free up the memory block.*/
     uwRet = LOS_MemFree(m_aucSysMem0, p_num);
     if (LOS_OK == uwRet) {
         dprintf("Memory block freed successfully!\n");
     }
     else {
         dprintf("Failed to free up the memory block!\n");
     }
         return;
 }
```

Verification

The verification result is as follows:

```
-----Test start-----
using new mem argorithm
*p_num = 828
```

Complete Code

sample_mem.c

Copyright © Huawei Technologies Co., Ltd.

3.2.3 Static Memory

3.2.3.1 Development Guidelines

Usage Scenarios

Static memory management is used when users demand memory of fixed size. When a user requests memory, the operating system calls the LOS_AllocBox API to allocate a static memory block. When the user no longer needs the memory, the operating system calls the LOS_FreeBox API to free up the memory block.

Functions

Static memory management of Huawei LiteOS provides the following functions:

Function Category	API	Description
Static memory initialization	LOS_MemboxInit	Initializes a static memory pool; defines the start address and total size of the static memory pool, as well as the size of each memory block.
Static memory clearing	LOS_MemboxClr	Clears data in a memory block of fixed size.
Static memory allocation	LOS_MemboxAlloc	Allocates a static memory block.
Memory free-up	LOS_MemboxFree	Frees up a static memory block.

Development Process

This section introduces the development process of static memory in typical scenarios:

1. Allocate continuous memory as a static memory pool.
2. Call the LOS_MemboxInit API

   Initializes the static memory pool; divides the memory pool that matches the input parameters into N memory blocks, where N depends on the total size of static memory pool and the size of each static memory block); adds all static memory blocks into a linked list of idle memory blocks; places a control header at the beginning of static memory pool.
3. Call the LOS_MemboxAlloc API

   Takes an idle memory block out of the linked list and returns the user space address of the memory block.
4. Call the LOS_MemboxFree API

   Adds the static memory block that has been freed up to the linked list.

Copyright © Huawei Technologies Co., Ltd.

5. Call the LOS_MemboxClr API

   Clears data in the static memory block that matches the input parameters.

Platform Differences

None.

3.2.3.2 Precautions

- The range of static memory pool can be acquired by using either a global variable array or the LOS_AllocMem API. In the latter case, to avoid memory leaks, free up a static memory block when the block is no longer in use.

3.2.3.3 Programming Example

Example Description

Memory is a scarce resource. If memory is frequently used while the operating system is running, program the memory management module to allocate and free up memory efficiently.

In the programming example, the following steps will be performed:

1. Initialize a static memory pool.
2. Take a static memory block out of the static memory pool.
3. Store data in the memory block.
4. Print the data in the memory block.
5. Clear the data in the memory block.
6. Free up the memory block.

Example Code

```
VOID los_membox_test(void) {
     UINT32 *p_num = NULL;
     UINT32 uwBlkSize = 10, uwBoxSize = 100;
     UINT32 uwRet;
     UINT32 pBoxMem[1000];
     uwRet = LOS_MemboxInit(&pBoxMem[0], uwBoxSize, uwBlkSize);
     if(uwRet != LOS_OK)
     {
         dprintf("Failed to initialized the memory pool!\n");
         return;
     }
     else {
         dprintf("Memory pool initialized successfully!\n");
     }

     /*Allocate a memory block.*/
     p_num = (int*)LOS_MemboxAlloc(pBoxMem);
     if (NULL == p_num) {
         dprintf("Failed to allocate the memory block!\n");
         return;
     }
     dprintf("Memory block allocated successfully!\n");
     /*Use the memory block.*/
     *p_num = 828;
     dprintf("*p_num = %d\n", *p_num);
      /*Clear data in the memory block.*/
```


Copyright © Huawei Technologies Co., Ltd.

```
      LOS_LOS_MemboxClr(pBoxMem, p_num);
      dprintf("Data in the memory block cleared successfully\n p_num = %d\n",
*p_num);
     /*Free up the memory block.*/
     uwRet = LOS_MemboxFree(pBoxMem, p_num);
     if (LOS_OK == uwRet) {
         dprintf("Memory block freed successfully!\n");
     }
     else {
         dprintf("Failed to free up the memory block!\n");
     }
         return;
 }
```

Verification

The verification result is as follows:

```
dist:1

---Test start---
*p_num = 828
 p_num = 0

---Test End---
```

Complete Code

sample_membox.c

3.3 Interrupt

3.3.1 Overview

Basic Concept

When a condition that needs immediate attention occurs, the CPU suspends current activities and switches to deal with the condition.

The CPU runs faster than external peripherals. When external peripherals are able to fulfill an activity alone, the CPU takes care of other activities.

When the CPU must be involved in fulfilling an activity, the interrupt mechanism enables an external peripheral to emit an interrupt signal to alert the CPU of the high-priority condition requiring the interruption of current activities. The CPU does not need to keep waiting for peripheral states, thereby improving CPU efficiency and accelerating system response.

The interrupt mechanism supports:

- Initialize
- Create
- Lock or unlock
- Restore
- Enable
- Disable

The interrupt mechanism of Huawei LiteOS is based on interrupt.

Copyright © Huawei Technologies Co., Ltd.

Introduce of Interrupt

The following three types of hardware are involved in the interrupt mechanism:

- Device: the interrupt source. When a device requests the help of the CPU, it emits an interrupt signal to the interrupt controller.
- Interrupt controller: a type of peripheral that sends an interrupt request to the CPU after receiving an interrupt signal from the interrupt pins of other peripherals. On the interrupt controller, you can prioritize, enable, or disable interrupt sources, as well as specify an interrupt trigger mode on each interrupt source. Common interrupt controllers include the Vector Interrupt Controller (VIC) and General Interrupt Controller (GIC, typically used in ARM Cortex-A7).
- CPU: executes an interrupt handler at the request of an interrupt source.

Terminology Associated With Interrupt

Interrupt ID: a unique identifier contained in all interrupt requests from a particular interrupt source.

Interrupt request (IRQ): an electrical pulse signal sent to alert the CPU of an urgent condition. The CPU suspends current activities and deals with the condition that needs immediate attention.

Interrupt priority: the priority of an interrupt source. Interrupt priority is determined based on importance and urgency. Priorities of all interrupt sources are the same in Huawei LiteOS. Interrupt nesting or preemption is not supported.

Interrupt handler: When an external peripheral generates an interrupt request, the CPU executes an interrupt handler to switch from current activities to the event that needs immediate attention.

Interrupt trigger: set to 1 when an interrupt source emits an interrupt signal.

Interrupt trigger type: the way in which an interrupt signal is sent to the interrupt controller. Typically, an interrupt signal is either level-triggered or edge-triggered.

Interrupt vector: starting address of interrupt service routine.

Interrupt vector table: a table where interrupt vectors are stored based on interrupt ID.

Interrupt sharing: If only a few external peripherals are present, each external peripheral is allocated a unique interrupt ID. However, if there are many external peripherals, consider sharing an interrupt ID among external peripherals. The interrupt handlers of the interrupts that share the same interrupt ID form a linked list. When an external peripheral generates an interrupt request, Huawei LiteOS Kernel traverses the linked list to find the interrupt handler of the interrupt request.

Interrupt top half and bottom half: If an interrupt is long, other interrupts that are more important may be blocked out. To balance the performance and workload of an interrupt handler, an interrupt handler is logically divided into two parts. The top half takes care of the urgent and critical part of the interrupt, and the bottom half deals with work, the longer yet less important part of the interrupt.

The top half of an interrupt typically reads the interrupt state from a register, clears the interrupt flag, and places the work in the workqueue.

Copyright © Huawei Technologies Co., Ltd.

Operation Mechanism

Interrupt mechanism of Huawei LiteOS supports interrupt sharing:

The implementation of interrupt sharing depends on the linked list. Each interrupt id create a linked list, the linked list node contains the interrupt handler function and the function input. When creating interrupt for many times to one same interrupt id, the interrupt handler function and the function input will be added to linked list. So when the hardware is interrupted, through the interrupt number to find its corresponding structure of the list, the implementation of the list of the interrupt handler.

Interrupt mechanism of Huawei LiteOS supports Interrupt bottom half:

The implementation of interrupt bottom half depends on workqueue, job is divided into interrupt top half and bottom half in interrupt handler. Handler int bottom half is associated with work, and mounted to legal workqueue. System executes bottom half program of work in workqueue while free.

3.3.2 Development Guidelines

Usage Scenarios

When an interrupt request is generated, the CPU responds by suspending current activities and calling the user-defined interrupt handler to deal with the condition that needs immediate attention.

Functions

The interrupt module provides the following functions:

API	Description
LOS_HwiCreate	Creates a hardware interrupt to register the corresponding interrupt handler
LOS_IntUnLock	Unlocks an interrupt
LOS_IntRestore	Restores an interrupt
LOS_IntLock	Locks an interrupt
hal_interrupt_mask	Disables an interrupt (A register is configured to prevent the CPU from responding to this interrupt.)
hal_interrupt_unmask	Enables an interrupt (A register is configured to prevent the CPU from responding to this interrupt.)

HWI Error Codes

Error codes are returned if errors occur during interrupt creation to facilitate fault locating.

Copyright © Huawei Technologies Co., Ltd.

No.	Definition	Error Code	Description	Solution
1	OS_ERRNO_HWI_NUM_INVALID	0x02000900	The interrupt ID is invalid.	Provide a valid interrupt ID.
2	OS_ERRNO_HWI_PROC_FUNC_NULL	0x02000901	The pointer to interrupt handler is null.	Pass in a non-null pointer to the interrupt handler.
3	OS_ERRNO_HWI_CB_UNAVAILABLE	0x02000902	Interrupts are unavailable.	Increase the number of available interrupts.
4	OS_ERRNO_HWI_NO_MEMORY	0x02000903	The memory is insufficient.	Enlarge the memory space.
5	OS_ERRNO_HWI_ALREADY_CREATED	0x02000904	The interrupt handler has already been created.	Check whether the interrupt handler corresponding to the passed-in interrupt ID has been created.
6	OS_ERRNO_HWI_PRIO_INVALID	0x02000905	The interrupt priority is invalid.	Pass in valid interrupt priority, which should fall in [0,31].
7	OS_ERRNO_HWI_MODE_INVALID	0x02000906	The interrupt mode is invalid.	Pass in a valid interrupt mode, which should fall in [0,1].
8	OS_ERRNO_HWI_FASTMODE_ALREADY_CREATED	0x02000907	The fast mode interrupt has already been created.	Check whether the interrupt handler corresponding to the passed-in interrupt ID has been created.
9	OS_ERRNO_HWI_INTERR	0x02000908	The API is called when an interrupt is underway.	Do not call this API when an interrupt is underway.
10	OS_ERRNO_HWI_NAME_TOOLONG	0x02000909	The name of the interrupt that is being registered is too long.	Decrease the length of the interrupt name or set **OS_HWI_MAX_NAMELEN** to a larger value.

Development Process

1. Configure the following parameters:

Copyright © Huawei Technologies Co., Ltd.

  - LOSCFG_PLATFORM_HWI: a switch to enable or disable the hardware interrupt module. Set to YES.
  - LOSCFG_PLATFORM_HWI_LIMIT: the maximum allowed number of hardware interrupts.
2. Call the LosHwiInit API to initialize the interrupt mechanism.
3. Call the LOS_HwiCreate API to create an interrupt.
4. Call the hal_interrupt_unmask API to enable an interrupt.
5. Call the hal_interrupt_mask API to disable an interrupt.

3.3.3 Precautions

- The register address of the LosHwiInit operation and the maximum allowed number of interrupts vary depending on hardware specifications.
- Interrupt sharing indicates that one interrupt handler can be mounted repeatedly. An interrupt request is accepted only when a unique **dev** parameter is passed in. For example, if you request an interrupt with a specified interrupt ID for twice, and at the second time you pass in the same interrupt handler and **dev** as you those you pass in at the first time, the interrupt request is rejected. If you pass in the same interrupt handler and a new **dev**, the interrupt request is accepted.
- Avoid long-running interrupt handlers because they have negative impact on CPU's response to interrupts.
- The function that leads to scheduling cannot be executed during the response to interrupts.
- The input parameter of the LOS_IntRestore() API must be the CPSR that is saved by the LOS_IntLock() API before locking the interrupt.
- In Cortex-A7, interrupts 0 – 31 are for internal use and it is not advisable to request or create them.
- The LOS_HwiCreate() API is not usually used to create an interrupt. Call the Linux adaption API request_irq to create an interrupt.

3.3.4 Programming Example

Example Description

The programming example will cover the following functions:

1. Disabling an interrupt
2. Creating an interrupt
3. Enabling an interrupt
4. Restoring an interrupt
5. Disabling an interrupt

Example Code

Prerequisite

- The LOSCFG_PLATFORM_HWI parameter in the los_config.h file is set to YES.
- The LOSCFG_PLATFORM_HWI_LIMIT parameter in the los_config.h file is set to the maximum number of hardware interrupts the operating system allows.

Copyright © Huawei Technologies Co., Ltd.

The code is as follows:

```
#include "los_hwi.h"
#include "los_typedef.h"
#define HWI_NUM_INT50 50
void uart_irqhandle(int irq,void *dev)
{
    printf("\n int the func uart_irqhandle \n");
}
void hwi_test()
{
    int a = 1;
    UINTPTR uvIntSave;
    uvIntSave = LOS_IntLock();
    LOS_HwiCreate(HWI_NUM_INT50, 0,0,uart_irqhandle,NULL);//Create an interrupt
    hal_interrupt_unmask(HWI_NUM_INT50);
    LOS_IntRestore(uvIntSave);
    hal_interrupt_mask(HWI_NUM_INT50);
}
```

Complete Code

sample_hwi.c

3.4 Queue

3.4.1 Overview

Basic Concept

A queue, also known as message queue, stores messages (also known as data) to be communicated between tasks. The length of message received by a queue is user defined. A queue receives messages of user-defined length from tasks or interrupts and determines whether to store a transferred message based on the interface through which the message is sent. A task reads messages from a queue. If the queue is empty, the task is suspended. When a new message is stored in the queue, the suspended task is woken up and processes the message.

A queue allows for asynchronous processing of messages, through which a message can be placed in a queue but left not processed immediately, and messages can be buffered.

The following features characterize queues:

- Messages in a queue are processed in the first in first out order. A message can be read and written asynchronously.
- Reading data from a queue and writing data into a queue support the timeout mechanism.
- The sender and the receiver agree on the type of message to be exchanged. The message length is variable, but cannot exceed the maximum message unit length.
- A task can choose any queue to send or receive messages.
- Multiple tasks can choose the same queue to send or receive messages.
- If a queue is allocated a dynamic memory block, the memory block can be reclaimed using the LOS_FreeMem API when the queue is no longer in use.

Copyright © Huawei Technologies Co., Ltd.

Operation Mechanism

Queue Control Block

```
/**
  * @ingroup los_queue
  * Queue information block structure
  */
typedef struct tagQueueCB
{
    UINT8       *pucQueue;      /**< pointer to the queue */
    UINT16      usQueueState;   /**< queue state */
    UINT16      usQueueLen;     /**< the number of messages in the queue */
    UINT16      usQueueSize;    /**< message node size */
    UINT16      usQueueID;             /**< queue ID */
    UINT16      usQueueHead;    /**< message head node position (array
subscript)*/
    UINT16      usQueueTail;    /**< message tail node position (array
superscript)*/
    UINT16      usReadWriteableCnt[2]; /**< the number of readable or writeable
messages in the queue,0: readable,1: writeable */

    LOS_DL_LIST stReadWriteList[2];    /**< read or write to the linked list of
waiting message tasks0: read the linked list,1: write to the linked list */
    LOS_DL_LIST stMemList;      /**< MailBox module usage */
} QUEUE_CB_S;
```

Each queue control block contains the element of queue state that indicates the usage of this queue:

- OS_QUEUE_UNUSED: The queue is not in use.
- OS_QUEUE_INUSED: The queue is in use.

Working Principles

During queue creation, memory is allocated to the queue based on the queue length and message node size and the queue ID is returned.

A message head node position (Head) and a message tail node position (Tail) are used in a queue control block to indicate the message storage in a queue. Head indicates the start position of an occupied message, and Tail indicates the start position of a vacant message. When a queue is first created, both Head and Tail point to the start position of the queue.

Data is written into the vacant message unit after the occupied message unit tail. If Tail points to the queue tail, the data is written into the start of the queue. The **usWritableCnt** parameter specifies whether the queue is fully occupied. Data cannot be written to a fully occupied queue (the **usWritableCnt** parameter value is **0**).

Data is read from the head of the occupied message units. If Head points to the queue tail, the data that is first written into the start of the queue is read. The **usReadableCnt** parameter specifies whether data is available for reading. A task of reading data from a vacant queue (the **usReadableCnt** parameter value is **0**) will be suspended.

During queue deletion, locate the queue that has a specified ID, set the queue state to be not in use, free up the memory allocated to the queue, and initialize the queue control head.

Copyright © Huawei Technologies Co., Ltd.

Figure 3-6 Read/write from/into a queue

3.4.2 Development Guidelines

Functions

The message processing module of Huawei LiteOS provides the following functions:

Function Category	API	Description
Queue creation	LOS_QueueCreate	Creates a queue.
Queue reading	LOS_QueueRead	Reads the specified queue header data (queue node data is an address).
Queue writing	LOS_QueueWrite	Writes the buffer address to the specified queue tail.
Queue reading	LOS_QueueReadCopy	Reads the specified queue header data.
Queue writing	LOS_QueueWriteCopy	Writes buffer data to the specified queue tail.
Queue writing	LOS_QueueWriteHead	Writes the buffer address to the specified queue header.
Queue writing	LOS_QueueWriteHeadCopy	Writes buffer data to the specified queue header.
Queue deletion	LOS_QueueDelete	Deletes a queue.

Copyright © Huawei Technologies Co., Ltd.

Function Category	API	Description
Queue information acquisition	LOS_QueueInfoGet	Gets information about a queue.

Development Process

The typical process of using the queue module is as follows:

1. Call the LOS_QueueCreate API to create a queue.

   Creates a queue and returns a queue ID.
2. Call the LOS_QueueWriteCopy API to write data into a queue.
3. Call the LOS_QueueReadCopy API to read data from a queue.
4. Call the LOS_QueueInfoGet API to get information about the queue.
5. Call the LOS_QueueDelete to delete a queue.

Queue Error Code

Error codes are returned if errors occur during queue operations, such as queue creation and queue deletion, to facilitate fault locating.

No.	Definition	Error Code	Description	Solution
1	LOS_ERRNO_QUEUE_MAXNUM_ZERO	0x02000600	The maximum number of queue resources is set to 0.	Set the maximum number of queue resources to be greater than 0. If the queue module is not used, disable the configuration of the maximum number of queue resources.
2	LOS_ERRNO_QUEUE_NO_MEMORY	0x02000601	The memory allocated to queue block fails to be initialized.	Allocate more memory to the queue block. Alternatively, decrease the maximum number of queue resources.
3	LOS_ERRNO_QUEUE_CREATE_NO_MEMORY	0x02000602	Memory fails to be allocated to the queue to be created.	Allocate more memory to the queue. Alternatively, decrease the length of the queue or the number of nodes in the queue to be created.
4	LOS_ERRNO_QUEUE_SIZE_TOO_BIG	0x02000603	The size of the largest message in the queue to be created exceeds the upper limit.	Change the size of the largest message to a size not exceeding the upper limit.

Copyright © Huawei Technologies Co., Ltd.

No.	Definition	Error Code	Description	Solution
5	LOS_ERRNO_QUEUE_CB_UNAVAILABLE	0x02000604	The number of created queues has exceeded the upper limit.	Increase the number of queue configuration resources.
6	LOS_ERRNO_QUEUE_NOT_FOUND	0x02000605	The queue is invalid.	Ensure the queue ID is valid.
7	LOS_ERRNO_QUEUE_PEND_IN_LOCK	0x02000606	The task must not be blocked on the queue when it is locked.	Unlock the task before the queue is used.
8	LOS_ERRNO_QUEUE_TIMEOUT	0x02000607	The wait time for processing a queue expires.	Set an appropriate expiry time.
9	LOS_ERRNO_QUEUE_IN_TSKUSE	0x02000608	The queue on which a task is blocked must not be deleted.	Enable the task to acquire resources rather than make the task blocked on the queue.
10	LOS_ERRNO_QUEUE_WRITE_IN_INTERRUPT	0x02000609	Writing data into a queue is not allowed when an interrupt is being processed.	Set the mode of writing data into a queue to non-blocking mode.
11	LOS_ERRNO_QUEUE_NOT_CREATE	0x0200060a	The queue is not created.	Pass in a valid handle.
12	LOS_ERRNO_QUEUE_IN_TSKWRITE	0x0200060b	Queue reading and writing are not synchronous.	Synchronize queue reading and writing.
13	LOS_ERRNO_QUEUE_CREAT_PTR_NULL	0x0200060c	A null pointer is passed in during queue creation.	Pass in a non-null pointer.
14	LOS_ERRNO_QUEUE_PARA_ISZERO	0x0200060d	The queue length or message node size passed in during queue creation is 0.	Pass in correct queue length and message node size.

Copyright © Huawei Technologies Co., Ltd.

No.	Definition	Error Code	Description	Solution
15	LOS_ERRNO_QUEUE_INVALID	0x0200060e	An invalid queue handle is passed in during queue reading or writing.	Pass in a valid handle.
16	LOS_ERRNO_QUEUE_READ_PTR_NULL	0x0200060f	A null pointer is passed in during queue reading.	Pass in a non-null pointer.
17	LOS_ERRNO_QUEUE_READSIZE_ISZERO	0x02000610	The buffer size passed in during queue reading is 0.	Pass in a correct buffer size.
18	LOS_ERRNO_QUEUE_WRITE_PTR_NULL	0x02000612	A null pointer passed in during queue writing.	Pass in a non-null pointer.
19	LOS_ERRNO_QUEUE_WRITESIZE_ISZERO	0x02000613	The buffer size passed in when data is being written into the queue is 0.	Pass in a correct buffer size.
20	LOS_ERRNO_QUEUE_WRITE_SIZE_TOO_BIG	0x02000615	The buffer size passed in during writing data into the queue is bigger than the queue size.	Decrease the buffer size. Alternatively, increase the node size.
21	LOS_ERRNO_QUEUE_ISFULL	0x02000616	Free nodes are unavailable during queue writing.	Ensure free nodes are available before writing data into the queue.
22	LOS_ERRNO_QUEUE_PTR_NULL	0x02000617	A null pointer is passed in when queue information is being acquired.	Pass in a non-null pointer.

Copyright © Huawei Technologies Co., Ltd.

No.	Definition	Error Code	Description	Solution
23	LOS_ERRNO_QUEUE_READ_IN_INTERRUPT	0x02000618	Reading data from a queue is not allowed when an interrupt is being processed..	Set the mode of reading data from a queue to non-blocking mode.
24	LOS_ERRNO_QUEUE_MAIL_HANDLE_INVALID	0x02000619	An invalid queue handle is passed in during releasing the memory allocated to the queue.	Pass in a valid handle.
25	LOS_ERRNO_QUEUE_MAIL_PTR_INVALID	0x0200061a	The passed-in pointer to the message memory pool is null.	Pass in a non-null pointer.
26	LOS_ERRNO_QUEUE_MAIL_FREE_ERROR	0x0200061b	Membox fails to be released.	Pass in a non-null pointer to membox.
27	LOS_ERRNO_QUEUE_ISEMPTY	0x0200061d	The queue is empty.	Ensure the queue contains messages when it is being read.
28	LOS_ERRNO_QUEUE_READ_SIZE_TOO_SMALL	0x0200061f	The buffer size passed in during queue reading is much smaller than the queue size.	Increase the buffer size. Alternatively, decrease the node size.

Platform Differences

On a 3516A platform, the data that is written into a queue does not need to be aligned on the boundary of 4 bytes. However, on a 3518e platform, the alignment is needed. On a 3559a platform, the data that is written into a queue needs to be aligned on the boundary of 8 bytes.

3.4.3 Precautions

- The maximum number of queues is not equal to the total number of queues available to users. When a queue is allocated to accommodate software timers, the number of available queues is decreased by 1.

Copyright © Huawei Technologies Co., Ltd.

- The queue name that is passed into the LOS_QueueCreate API is reserved for future use.
- The input parameter uwTimeOut of queue APIs must be set to relative time.
- LOS_QueueReadCopy, LOS_QueueWriteCopy, and LOS_QueueWriteHeadCopy are a group of APIs; LOS_QueueRead, LOS_QueueWrite, and LOS_QueueWriteHead are a group of APIs. The two groups of APIs must be used together.
- The LOS_QueueWrite, LOS_QueueWriteHead, and LOS_QueueRead APIs are called to operate data addresses. Ensure that the memory that is pointed to by the pointer obtained by calling the LOS_QueueRead API is not modified or released during the queue reading. Otherwise, unexpected results may be caused.
- The LOS_QueueWrite, LOS_QueueWriteHead, and LOS_QueueRead APIs are called to operate data addresses, which means the actual write or read data length is the same as that of a pointer. Therefore, before using these APIs, ensure that the length of an individual message is the same as that of a pointer during queue creation, to avoid unnecessary waste and read failures.
- The LOS_QueueWrite, LOS_QueueWriteHead, and LOS_QueueRead APIs are being discarded. The LOS_QueueWriteCopy, LOS_QueueWriteHeadCopy, and LOS_QueueReadCopy APIs are recommended.

3.4.4 Programming Example

Example Description

Two tasks are created in the programming example. Task 1 calls the send_Entry API to send messages. Task 2 calls the recv_Entry API to receive messages.

1. Call the LOS_TaskCreate API to create tasks 1 and 2.
2. Call the LOS_QueueCreate API to create a queue.
3. Call the send_Entry API to enable task 1 to send a message.
4. Call the rev_Entry API to enable task 2 to send a message.
5. Call the LOS_QueueDelete API to delete the queue.

Example Code

```
#include "los_task.h"
 #include "los_queue.h"
 static UINT32 g_uwQueue;
 CHAR abuf[] = "test is message x";

 /*Task 1 sends a message.*/
 void *send_Entry(void *arg)
 {
     UINT32 i = 0,uwRet = 0;
     UINT32 uwlen = sizeof(abuf);

     while (i <5)
     {
         abuf[uwlen -2] = '0' + i;
         i++;

         /*Task 1 writes data from abuf into the queue.*/
         uwRet = LOS_QueueWrite(g_uwQueue, abuf, uwlen, 0);
         if(uwRet != LOS_OK)
         {
             dprintf("send message failure,error:%x\n",uwRet);
         }
```


Copyright © Huawei Technologies Co., Ltd.

```
        LOS_TaskDelay(5);
    }
}

/*Task 2 receives a message.*/
void *recv_Entry(void *arg)
{
    UINT32 uwReadbuf;
    UINT32 uwRet = 0;

    while (1)
    {

        /*Task 2 reads data from the queue and stores it in uwReadbuf.*/
        uwRet = LOS_QueueRead(g_uwQueue, &uwReadbuf, 50, 0);
        if(uwRet != LOS_OK)
        {
            dprintf("recv message failure,error:%x\n",uwRet);
            break;
        }

        dprintf("recv message:%s\n", (char *)uwReadbuf);
        LOS_TaskDelay(5);
    }
    /*Delete the queue.*/
    while (LOS_OK != LOS_QueueDelete(g_uwQueue))
    {
        LOS_TaskDelay(1);
    }

    dprintf("queue successfully deleted!\n");
}

int Example_creat_task(void)
{
    UINT32 uwRet = 0;
    UINT32 uwTask1, uwTask2;
    TSK_INIT_PARAM_S stInitParam1;

    /*Create task 1.*/
    stInitParam1.pfnTaskEntry = send_Entry;
    stInitParam1.usTaskPrio = 9;
    stInitParam1.uwStackSize = 0x400;
    stInitParam1.pcName = "sendQueue";
    stInitParam1.uwResved = LOS_TASK_STATUS_DETACHED;
    LOS_TaskLock();//Lock task scheduling so that the newly created task will
not be executed even if it has a higher priority than the running task.
    uwRet = LOS_TaskCreate(&uwTask1, &stInitParam1);
    if(uwRet != LOS_OK)
    {
        dprintf("create task1 failed!,error:%x\n",uwRet);
        return uwRet;
    }

    /*Create task 2.*/
    stInitParam1.pfnTaskEntry = recv_Entry;
    uwRet = LOS_TaskCreate(&uwTask2, &stInitParam1);
    if(uwRet != LOS_OK)
    {
        dprintf("create task2 failed!,error:%x\n",uwRet);
        return uwRet;
    }

    /*Create the queue.*/
    uwRet = LOS_QueueCreate("queue", 5, &g_uwQueue, 0, 50);
    if(uwRet != LOS_OK)
    {
        dprintf("create queue failure!,error:%x\n",uwRet);
    }
```


Copyright © Huawei Technologies Co., Ltd.

```
    dprintf("create the queue success!\n");
    LOS_TaskUnlock();//Unlock task scheduling so that task scheduling will
happen after the queue is created.
 }
```

Verification

```
 --- Test start---
create the queue success!
recv message:test is message 0
recv message:test is message 1
recv message:test is message 2
recv message:test is message 3
recv message:test is message 4
recv message failure,error:200061d
delete the queue success!
```

Complete Code

sample_queue.c

3.5 Event

3.5.1 Overview

Basic Concept

Events are used for synchronization between tasks. A task or interrupt service routine can trigger an event (a synchronization signal) to another task through an event control block. One task is able to wait for several events to occur: whether while one event occurring or after several events occurred, both of these are sure to wake task up to do event handling.

In a multi-task environment, tasks must be synchronized. In the one-to-many synchronization model, a task waits for multiple events. In the many-to-many synchronization model, multiple tasks wait for multiple events.

Tasks trigger or wait for events through event control blocks.

Characteristics of events in Huawei LiteOS are as follows:

- Events are not associated with tasks and are independent from each other. A 32-bit unsigned integer variable is used to indicate the type of the event in which a task is interested. Each bit indicates one event type with 0 indicating that the event does not occur and 1 indicating that the event occurs.
- Events are used only for task synchronization, and not for data transport.
- Sending the same event type to a task for multiple times is equivalent to sending for only once.
- Multiple tasks are allowed to read or write the same event.
- Huawei LiteOS supports event reading and writing timeout.

Event control block

```
/**
 * @ingroup los_event
```


Copyright © Huawei Technologies Co., Ltd.

```
 * Event control structure
 */
typedef struct tagEvent
{
    UINT32 uwEventID;            /**bit that indicates an event type*/
    LOS_DL_LIST    stEventList;  /**linked list of  event reading tasks*/
} EVENT_CB_S, *PEVENT_CB_S;
```

uwEventID indicates the type of the event in which a task is interested. Each bit indicates one event type with 0 indicating that the event does not occur and 1 indicating that the event occurs. There are 31 bits that indicate event types (bit 25 is reserved).

Event reading mode

An event reading mode can be configured during event reading. Event reading modes are as follows:

LOS_WAITMODE_AND indicates that event of all event types specified by a mask need to be read. Event reading succeeds only when all events that are read occur.

LOS_WAITMODE_OR indicates that an event of an event type specified by a mask needs to be read. Event reading succeeds when the event that is read occurs.

LOS_WAITMODE_CLR (LOS_WAITMODE_AND| LOS_WAITMODE_CLR or LOS_WAITMODE_OR| LOS_WAITMODE_CLR) indicates that after successful event reading, the event types or event type that is read is automatically cleared.

Operation Mechanism

During event reading, one type or multiple types specified by uwEventMask are read. After event reading succeeds, the event type that is read is explicitly cleared if **LOS_WAITMODE_CLR** is configured in the event reading mode. The event type that is not cleared if **LOS_WAITMODE_CLR** is not configured. You can configure the event reading mode by passing in **LOS_WAITMODE_AND** to read all events of the event types specified by the event mask or by passing in **LOS_WAITMODE_OR** to read an event of an event type specified by the event mask.

During event writing, a specified event type is written into an event. Multiple event types can be written concurrently. Event writing may trigger task scheduling.

During event clearance, the bit that specifies the event type to be cleared is set to 0.

Copyright © Huawei Technologies Co., Ltd.

Figure 3-7 Tasks woken up by events

3.5.2 Development Guidelines

Usage Scenarios

Events are applicable in a variety of task synchronization scenarios and are partially similar to semaphore in some scenarios.

Functions

The event module provides the following functions:

Copyright © Huawei Technologies Co., Ltd.

Function Category	API	Description
Event initialization	LOS_EventInit	Initializes an event control block
Event reading	LOS_EventRead	Reads an event within N ticks
Event writing	LOS_EventWrite	Writes an event
Event clearance	LOS_EventClear	Clears an event
Event mask verification	LOS_EventPoll	Determines whether an event meets the expectations based on the passed-in event value, event mask, and verification mode
Event destroying	LOS_EventDestroy	Destroys a specified event control block

Development Process

The typical process of using the event module is as follows:

1. Call the LOS_EventInit API to initialize an event control block.
2. Call the LOS_EventInit to write an event.
3. Call the LOS_EventRead API to read an event.
4. Call the LOS_EventClear API to clear an event.

Event Error Code

Error codes are returned if errors occur during event operations, such as event initialization, event destroying, event reading, event writing, and event clearance, to facilitate fault locating.

No.	Definition	Error Code	Description	Solution
1	LOS_ERRNO_EVENT_SETBIT_INVALID	0x02001c00	Bit 25 of the event ID must not be set to 1 because it is reserved as an error code.	Set bit 25 of the event ID to 0.
2	LOS_ERRNO_EVENT_READ_TIMEOUT	0x02001c01	Event reading times out.	Increase the permitted wait time. Alternatively, re-read the event.

Copyright © Huawei Technologies Co., Ltd.

No.	Definition	Error Code	Description	Solution
3	LOS_ERRNO_EVENT_EVENTMASK_INVALID	0x02001c02	The passed-in event ID is invalid.	Pass in a valid event ID.
4	LOS_ERRNO_EVENT_READ_IN_INTERRUPT	0x02001c03	The event is being read when an interrupt is being processed.	Let a new task read the event.
5	LOS_ERRNO_EVENT_FLAGS_INVALID	0x02001c04	The mode of event reading is invalid.	Pass in a valid mode.
6	LOS_ERRNO_EVENT_READ_IN_LOCK	0x02001c05	The task is locked and fails to read the event.	Unlock the task, and then let the task read the event.
7	LOS_ERRNO_EVENT_PTR_NULL	0x02001c06	The passed-in pointer is null.	Pass in a non-null pointer.
8	LOS_ERRNO_EVENT_READ_IN_SWTMR_TSK	0x02001c07	Events are read from software timer tasks.	Start a new task and obtain events.
9	LOS_ERRNO_EVENT_SHOULD_NOT_DESTORY	0x02001c08	The linked list of events still contains tasks and cannot be destroyed.	Check whether the linked list of events is empty.

An error code is a 32-bit unsigned integer. Bit 24 to bit 31 indicate an error level; bit 16 to bit 23 indicate an error code flag; bit 8 to bit 15 indicate the ID of the module that reports the error code; bit 0 to bit 7 indicate an error code. The following is the example of an error code:

```
#define LOS_ERRNO_OS_ERROR(MID, ERRNO) \
(LOS_ERRTYPE_ERROR | LOS_ERRNO_OS_ID | ((UINT32)(MID) << 8) | (ERRNO))
```

LOS_ERRTYPE_ERROR: Define critical OS errors

LOS_ERRNO_OS_ID: OS error code flag

MID: OS_MOUDLE_ID

LOS_MOD_EVENT: Event module ID

ERRNO: error ID number

Copyright © Huawei Technologies Co., Ltd.

For example:

#define LOS_ERRNO_EVENT_READ_IN_LOCK
LOS_ERRNO_OS_ERROR(LOS_MOD_EVENT, 0x05)

Platform Differences

None.

3.5.3 Precautions

- Do not make calls to the LOS_EventRead and LOS_EventWrite APIs prior to the operating system being initialized. Otherwise, the operating system exhibits unexpected behavior.
- While an interrupt is underway, events can be written into an event control block but event reads are not allowed.
- Task blocking and event reading are not allowed while task scheduling is locked.
- The input parameter of LOS_EventClear is ~uwEvents (reverse code of event type).
- Bit 25 of the event mask is merely used to distinguish whether the LOS_EventRead API returns an event or error code.

3.5.4 Programming Example

Example Description

In the programming example, the Example_TaskEntry task is executed to create the Example_Event task. The Example_Event task is blocked from reading events. The Example_TaskEntry task writes an event in which the Example_Event task shows interest.

1. The Example_TaskEntry task is executed to create the Example_Event task. The Example_Event task takes a higher priority than the Example_TaskEntry task.
2. The Example_Event task is blocked from reading the event 0x00000001. After the Example_Event task is blocked, a task switch occurs to execute the task with a lower priority, namely, the Example_TaskEntry task.
3. The Example_TaskEntry task writes the event 0x00000001 toward the Example_Event task. The Example_Event task is interested in the event 0x00000001 and is therefore woken up to process the event.
4. The Example_Event task is executed.
5. The Example_TaskEntry task is executed.

Example Code

The order in which print-out is generated provides some clues into task switches that occur during event operations.

The code is as follows:

```
#include "los_event.h"
 #include "los_task.h"

 /*Task PID*/
 UINT32 g_TestTaskID01;
```


Copyright © Huawei Technologies Co., Ltd.

```
/*Event control structure*/
EVENT_CB_S  example_event;

/*Event that the Example_Event task is waiting for*/
#define event_wait 0x00000001

/*Task entrypoint function*/
VOID Example_Event()
{
    UINT32 uwRet;
    UINT32 uwEvent;

/*Wait for a completion in timeout mode, and the timeout interval is 100
ticks.
  If the event is not read within 100 ticks, the read operation expires and the
task is woken up.*/
    printf("Example_Event wait event 0x%x \n",event_wait);

    uwEvent = LOS_EventRead(&example_event, event_wait, LOS_WAITMODE_AND, 100);
    if(uwEvent == event_wait)
    {
        printf("Example_Event,read event :0x%x\n",uwEvent);
    }
    else
        printf("Example_Event,read event timeout\n");
    return;
}

UINT32 Example_TaskEntry()
{
    UINT32 uwRet;
    TSK_INIT_PARAM_S stTask1;

  /*Initialize the event.*/
    uwRet = LOS_EventInit(&example_event);
    if(uwRet != LOS_OK)
    {
        printf("init event failed .\n");
        return -1;
    }

    /*Create the task.*/
    memset(&stTask1, 0, sizeof(TSK_INIT_PARAM_S));
  stTask1.pfnTaskEntry = (TSK_ENTRY_FUNC)Example_Event;
    stTask1.pcName       = "EventTsk1";
    stTask1.uwStackSize  = OS_TSK_DEFAULT_STACK_SIZE;
    stTask1.usTaskPrio = 5;
    uwRet = LOS_TaskCreate(&g_TestTaskID01, &stTask1);
    if(uwRet != LOS_OK)
    {
        printf("Task creation failed .\n");
        return LOS_NOK;
    }

    /*Write the event type for which the task is waiting for.*/
    printf("Example_TaskEntry write event .\n");

    uwRet = LOS_EventWrite(&example_event, event_wait);
    if(uwRet != LOS_OK)
    {
        printf("Event write failed .\n");
        return LOS_NOK;
    }

    /*Clear the flag.*/
    printf("EventMask:%d\n",example_event.uwEventID);
    LOS_EventClear(&example_event, ~example_event.uwEventID);
    printf("EventMask:%d\n",example_event.uwEventID);
```


Copyright © Huawei Technologies Co., Ltd.

```
    /*Delete the task.*/
    uwRet = LOS_TaskDelete(g_TestTaskID01);
    if(uwRet != LOS_OK)
    {
        printf("Task deletion failed .\n");
        return LOS_NOK;
    }

    return LOS_OK;
}
```

Verification

The verification result is as follows:

```
Example_Event wait event 0x1
 Example_TaskEntry write event .
 Example_Event,read event :0x1
 EventMask:1
 EventMask:0
```

Complete Code

sample_event.c

3.6 Mutex

3.6.1 Overview

Basic Concept

A mutual exclusion (mutex) is a special binary semaphore designed to grant a task exclusive use of common resources.

At a given point in time, a mutex is either locked or unlocked. When a task acquires a mutex, the mutex is locked and the task has exclusive ownership of the mutex. When the task releases the mutex, the mutex is unlocked and the task loses exclusive ownership of the mutex. While a task has exclusive ownership of a mutex, other tasks are unable to acquire or release the mutex.

In a multi-task environment, it is common to see tasks competing for the same common resource. A mutex can avoid the task conflict problem without the trouble of priority inversion experienced with semaphores.

Mutex of Huawei LiteOS has characters as below:

- Solve the problem of priority inversion by using inheritance algorithm.

Operation Mechanism

Mutex Operation Principle

In a multi-task environment, multiple tasks may battle for the same common resource. If the common resource is not shareable, it must be used exclusively by a particular task.

When a task accesses a non-shareable common resource, the mutex is locked. Other tasks are blocked from accessing the resource until the task releases the mutex. In this way, only one

Copyright © Huawei Technologies Co., Ltd.

task accesses the non-shareable common resource at a given point in time, which ensures the integrity of the non-shareable common resources.

Figure 3-8 Working principle of a mutex

3.6.2 Development Guidelines

Usage Scenarios

A mutex is a good choice for preventing tasks from accessing the same shared resource at the same time.

Functions

The mutex module provides the following functions:

Function Category	API	Description
Mutex creation and deletion	LOS_MuxCreate	Creates a mutex
	LOS_MuxDelete	Deletes a mutex
Mutex request and release	LOS_MuxPend	Pends on a mutex
	LOS_MuxPost	Releases a mutex

Development Process

The typical mutex development process is as follows:

1. Call the LOS_MuxCreate API to create a mutex.
2. Call the LOS_MuxPend API to pend on a mutex.

   Takes actions depending on the mutex pend mode.

Copyright © Huawei Technologies Co., Ltd.

- Non-blocking mode: If no task has acquired the mutex or the task that has acquired the mutex is the same as the requesting task, the operating system grants the mutex to the requesting task.
- Permanent blocking mode: the requesting task waits endlessly for a mutex and enters Blocked state in the meantime. If the mutex has not been acquired by any task, the operating system grants the mutex to the requesting task. Otherwise, the operating system blocks the requesting task until the mutex is released. While the requesting task is blocked, the operating system selects the task with the highest priority among ready tasks to be executed.
- Temporary blocking mode: the requesting task waits for a specified period of time for a mutex and enters Blocked state in the meantime. If the mutex has not been acquired by any task, the operating system grants the mutex to the requesting task. Otherwise, the operating system blocks the requesting task until the mutex is released or the timeout period elapses. It then selects the ready task with the highest priority to be executed.

3. Call the LOS_MuxPost to release a mutex.
   - If there are tasks blocked from acquiring the mutex, the operating system wakes up the highest priority blocked task. The woken-up task then enters Ready state and is scheduled.
   - If there are no tasks blocked from acquiring the mutex, the operating system releases the mutex.
4. Call the LOS_MuxDelete API to delete a mutex.

Mutex Error Code

Error codes are returned if errors occur during mutex operations, such as mutex creation, mutex deletion, mutex pending, and mutex posting, to facilitate fault locating.

No.	Definition	Error Code	Description	Solution
1	LOS_ERRNO_MUX_NO_MEMORY	0x02001d00	The request for memory is rejected.	Lower the upper limit on the number of mutexes.
2	LOS_ERRNO_MUX_INVALID	0x02001d01	The mutex is not usable.	Pass in a valid mutex ID.
3	LOS_ERRNO_MUX_PTR_NULL	0x02001d02	The input parameter is null.	Pass in a non-null parameter.
4	LOS_ERRNO_MUX_ALL_BUSY	0x02001d03	No mutexes are available.	Raise the upper limit on the number of mutexes.

Copyright © Huawei Technologies Co., Ltd.

No.	Definition	Error Code	Description	Solution
5	LOS_ERRNO_MUX_UNAVAILABLE	0x02001d04	The mutex fails to be locked because it is locked by another thread.	Wait for another thread to release the mutex. Alternatively, set a timeout period.
6	LOS_ERRNO_MUX_PEND_INTERR	0x02001d05	Mutex pend occurs when an interrupt is being processed.	Do not call this API when an interrupt is being processed.
7	LOS_ERRNO_MUX_PEND_IN_LOCK	0x02001d06	When task scheduling is disabled, tasks are not allowed to pend on the mutex.	Set the mutex pend mode to the non-blocking mode. Alternatively, enable task scheduling.
8	LOS_ERRNO_MUX_TIMEOUT	0x02001d07	Mutex pend times out.	Increase the wait time. Alternatively, set the mutex pend mode to the permanent blocking mode.
9	LOS_ERRNO_MUX_OVERFLOW	0x02001d08	The error code is not in use.	N/A
10	LOS_ERRNO_MUX_PENDED	0x02001d09	The mutex being deleted is locked.	Delete the mutex after it is released.
11	LOS_ERRNO_MUX_GET_COUNT_ERR	0x02001d0a	The error code is not in use.	N/A
12	LOS_ERRNO_MUX_REG_ERROR	0x02001d0b	The error code is not in use.	N/A

An error code is a 32-bit storage unit. Bit 24 to bit 31 indicate an error level; bit 16 to bit 23 indicate an error code flag; bit 8 to bit 15 indicate the ID of the module that reports the error code; bit 0 to bit 7 indicate an error code. The following is the example of an error code:

```
#define LOS_ERRNO_OS_ERROR(MID, ERRNO) \
```


Copyright © Huawei Technologies Co., Ltd.

(LOS_ERRTYPE_ERROR | LOS_ERRNO_OS_ID | ((UINT32)(MID) << 8) | (ERRNO))

LOS_ERRTYPE_ERROR: Define critical OS errors

LOS_ERRNO_OS_ID: OS error code flag

LOS_MOD_MUX: Mutex module ID

MID: OS_MOUDLE_ID

ERRNO: error ID number

For example:

LOS_ERRNO_MUX_TIMEOUT LOS_ERRNO_OS_ERROR(LOS_MOD_MUX, 0x07)

Platform Differences

None.

3.6.3 Precautions

- Tasks are unable to lock the same mutex. If a task attempts to lock a mutex that has been locked by another task, the task will be blocked from locking the mutex until the mutex is unlocked.
- Do not use any mutex for interrupt service routines.
- Release a mutex immediately when the mutex is no longer in use. Otherwise, tasks will be blocked for a long time, slowing down task scheduling.
- Do not change the priority of a task by calling APIs such as LOS_TaskPriSet while the task has full ownership of a mutex.

3.6.4 Programming Example

Example Description

In the programming example, the following activities will happen:

1. The Example_TaskEntry task is executed to create a mutex. Task scheduling is locked. Two tasks Example_MutexTask1 and Example_MutexTask2 are created, where Example_MutexTask2 takes a higher priority than Example_MutexTask1. Then, task scheduling is unlocked.
2. Example_MutexTask2 is scheduled, granted a mutex, and then sent to sleep mode for 100 ticks. While Example_MutexTask2 is suspended, Example_MutexTask1 is woken up.
3. Example_MutexTask1 pends on the mutex and is willing to wait the mutex for 10 ticks to become free. At the time when Example_MutexTask1 requests the mutex, the mutex is held by Example_MutexTask2 and consequently Example_MutexTask1 is suspended. After the 10-tick wait period elapses, the mutex is still out of the reach of Example_MutexTask1, and Example_MutexTask1 is woken up, attempting to wait permanently for the mutex. The wait for the mutex switches Example_MutexTask1 to suspended state.
4. After 100 ticks, Example_MutexTask2 is woken up and releases the mutex. Example_MutexTask1 is scheduled, granted the mutex, and finally releases it.

Copyright © Huawei Technologies Co., Ltd.

5. 300 ticks after Example_MutexTask1 is finished, Example_TaskEntry is executed to delete the mutex.

Example Code

Prerequisites

- The LOSCFG_BASE_IPC_MUX parameter in the los_config.h file is set to YES.
- The LOSCFG_BASE_IPC_MUX_LIMIT parameter in the los_config.h file is set to the maximum number of mutexes that the operating system allows.

The code is as follows:

```
#include "los_mux.h"
 #include "los_task.h"

 /*Mutex handler ID*/
 MUX_HANDLE_T  g_Testmux01;
 /*Task ID*/
 UINT32 g_TestTaskID01;
 UINT32 g_TestTaskID02;

 VOID Example_MutexTask1()
 {
     UINT32 uwRet;

     printf("task1 try to get mutex, wait 10 Tick.\n");
     /*The task pends on a mutex.*/
     uwRet=LOS_MuxPend(g_Testmux01, 10);

     if(uwRet == LOS_OK)
     {
         printf("task1 get mutex g_Testmux01.\n");
         /*The task releases the mutex.*/
         LOS_MuxPost(g_Testmux01);
         return;
     }
     else if(uwRet == LOS_ERRNO_MUX_TIMEOUT )
     {
             printf("task1 timeout and try to get  mutex, wait forever.\n");
             /*The task pends on the mutex.*/
             uwRet = LOS_MuxPend(g_Testmux01, LOS_WAIT_FOREVER);
             if(uwRet == LOS_OK)
             {
                 printf("task1 wait forever,get mutex g_Testmux01.\n");
                 /*The task releases the mutex.*/
                 LOS_MuxPost(g_Testmux01);
                 return;
             }
     }
     return;
 }

 VOID Example_MutexTask2()
 {
     UINT32 uwRet;

     printf("task2 try to get mutex, wait forever.\n");
     /*The task pends on the mutex.*/
     uwRet=LOS_MuxPend(g_Testmux01, LOS_WAIT_FOREVER);

     printf("task2 get mutex g_Testmux01 and suspend 100 Tick.\n");

     /*Send the task to sleep mode for 100 ticks.*/
     LOS_TaskDelay(100);

     printf("task2 resumed and post the g_Testmux01\n");
```


Copyright © Huawei Technologies Co., Ltd.

```
    /*The task releases the mutex.*/
    LOS_MuxPost(g_Testmux01);
    return;

}

UINT32 Example_TaskEntry()
{
    UINT32 uwRet;
    TSK_INIT_PARAM_S stTask1;
    TSK_INIT_PARAM_S stTask2;

    /*Create the mutex.*/
    LOS_MuxCreate(&g_Testmux01);

    /*Lock task scheduling.*/
    LOS_TaskLock();

    /*Create task 1.*/
    memset(&stTask1, 0, sizeof(TSK_INIT_PARAM_S));
    stTask1.pfnTaskEntry = (TSK_ENTRY_FUNC)Example_MutexTask1;
    stTask1.pcName       = "MutexTsk1";
    stTask1.uwStackSize  = OS_TSK_DEFAULT_STACK_SIZE;
    stTask1.usTaskPrio = 5;
    uwRet = LOS_TaskCreate(&g_TestTaskID01, &stTask1);
    if(uwRet != LOS_OK)
    {
        printf("task1 create failed .\n");
        return LOS_NOK;
    }

    /*Create task 2.*/
    memset(&stTask2, 0, sizeof(TSK_INIT_PARAM_S));
    stTask2.pfnTaskEntry = (TSK_ENTRY_FUNC)Example_MutexTask2;
    stTask2.pcName       = "MutexTsk2";
    stTask2.uwStackSize  = OS_TSK_DEFAULT_STACK_SIZE;
    stTask2.usTaskPrio = 4;
    uwRet = LOS_TaskCreate(&g_TestTaskID02, &stTask2);
    if(uwRet != LOS_OK)
    {
        printf("task2 create failed .\n");
        return LOS_NOK;
    }

    /*Unlock task scheduling.*/
    LOS_TaskUnlock();
    /*Send the task to sleep mode for 300 ticks.*/
    LOS_TaskDelay(300);

    /*Delete the mutex.*/
    LOS_MuxDelete(g_Testmux01);

    /*Delete task 1.*/
    uwRet = LOS_TaskDelete(g_TestTaskID01);
    if(uwRet != LOS_OK)
    {
        printf("task1 delete failed .\n");
        return LOS_NOK;
    }
    /*Delete task 2.*/
    uwRet = LOS_TaskDelete(g_TestTaskID02);
    if(uwRet != LOS_OK)
    {
        printf("task2 delete failed .\n");
        return LOS_NOK;
    }

    return LOS_OK;
}
```


Copyright © Huawei Technologies Co., Ltd.

Verification

The verification result is as follows:

```
task2 try to get  mutex, wait forever.
 task2 get mutex g_Testmux01 and suspend 100 ticks.
 task1 try to get  mutex, wait 10 ticks.
 task1 timeout and try to get  mutex, wait forever.
 task2 resumed and post the g_Testmux01
 task1 wait forever,get mutex g_Testmux01.
```

Complete Code

sample_mutex.c

3.7 Semaphore

3.7.1 Overview

Basic Concept

A semaphore is a mechanism used for communication within a kernel, to achieve synchronization or mutual exclusion of critical resources between tasks.

In a multi-task system, it is necessary to synchronize one task with another or prevent tasks battling for critical resources. Semaphores are a good choice to serve that purpose.

Typically, a numerical value of a signal is used to correspond to the number of available resources. It means mutually exclusive resources remained that could be occupied. The meaning of its value is divided into two kinds of situations:

- 0, it means the post operation that is not accumulated, and it is possible to block tasks on this signal.
- Positive number, it means there is one or several release operations which are posted.

The differences to use between semaphore for the purpose of synchronization and semaphore for the purpose of mutex are:

- If a semaphore is used as a mutex, it is created with a full internal counter. Each time a task waits on critical resources, it is assigned the semaphore and the counter value is decreased by 1. When the counter value drops to 0, subsequent tasks are blocked from getting the semaphore.
- If a semaphore is used for task synchronization, it is created with an empty counter. When task 1 attempts to get the semaphore, it is blocked because the counter has reached the maximum value. Task 1 will enter Ready or Running state after task 2 releases the semaphore, thereby achieving task synchronization.

Operation Mechanism

Semaphore Control Block

```
/**
 * @ingroup los_sem
 * Semaphore control structure.
 */
```


Copyright © Huawei Technologies Co., Ltd.

```
typedef struct
{
    UINT8           usSemStat;          /**whether to use flag bit*/
    UINT16          uwSemCount;         /**semaphore count*/
    UINT32          usSemID;            /**semaphore quantity index*/
    LOS_DL_LIST     stSemList;          /**suspend the task blocked on the
semaphore*/
}SEM_CB_S;
```

Semaphore Operation Principle

During semaphore initialization, memory is allocated to N semaphores. N is configurable by users and limited by memory. For details, see section **10 "Configuration Reference."** All semaphores are initialized and added to the linked list of semaphores that are not in use.

During semaphore creation, a semaphore is obtained from the linked list of semaphores that are not in use and the initial value of the semaphore is set.

If the internal counter of a semaphore is more than 0 when the semaphore is pended, the counter value is decreased by 1 and the pending succeeds. If the counter value is 0, tasks are blocked from getting the semaphore and wait for other tasks to post the semaphore. The timeout interval of waiting on the semaphore can be configured. If a task is blocked from getting the semaphore, suspend the task to the tail of the queue of tasks waiting on the semaphore.

If no tasks are waiting on a semaphore, the counter value is increased by 1 and the semaphore is posted. Otherwise, wake up the first task in the queue of tasks waiting on the semaphore.

During semaphore deletion, the semaphore that is in use is set to be not in use and is added to the linked list of semaphores that are not in use.

A semaphore allows multiple tasks to access the same resource at the same time but sets a limit on the number of the tasks. Tasks are not allowed to access the resource if the maximum number of the tasks that can access the resource is reached and need to wait for one task to release the semaphore.

Copyright © Huawei Technologies Co., Ltd.

Figure 3-9 Working principle of semaphore

3.7.2 Development Guidelines

Usage Scenarios

Semaphores find their use in locking resources, counting resources, and maintaining synchronization between threads or between threads and interrupts.

Functions

The semaphore module provides the following functions:

Function Category	API	Description
Semaphore creation and deletion	LOS_SemCreate	Creates a semaphore
	LOS_SemDelete	Deletes a semaphore
Semaphore pend and post	LOS_SemPend	Pends on a semaphore
	LOS_SemPost	Posts a semaphore

Development Process

The typical semaphore development process is as follows:

Copyright © Huawei Technologies Co., Ltd.

1. Call the LOS_SemCreate API to create a semaphore.
2. Call the LOS_SemPend to pend on a semaphore.

   Huawei LiteOS takes actions depending on the semaphore pend mode.

   - Non-blocking mode: If the maximum number of tasks allowed by the semaphore is not reached, the request for the semaphore is fulfilled. Otherwise, the request for the semaphore is rejected.
   - Permanent blocking mode: The requesting task waits endlessly for a semaphore and the task enters Blocked state in the meantime. If the maximum number of tasks allowed by the semaphore is not reached, the request for the semaphore is fulfilled. Otherwise, the operating system blocks the requesting task until a task releases the semaphore. It then selects the ready task with the highest priority to be executed.

     If a task enters the Blocked state, this task will be re-executed when either of the following conditions is met:

     - Other tasks release semaphores before the specified period expires.
     - The specified period expires.
   - Temporary blocking mode: the requesting task waits for a specified period of time for a semaphore and enters Blocked state in the meantime. If the maximum number of tasks allowed by the semaphore is not reached, the request for the semaphore is fulfilled. Otherwise, the operating system blocks the requesting task until a task releases the semaphore or the timeout period elapses. It then selects the ready task with the highest priority to be executed.
3. Call the LOS_SemPost API to post a semaphore.
   - If there are tasks blocked from acquiring the semaphore, the operating system wakes up the first blocked task. The woken-up task then enters Ready state and is scheduled.
   - If there are no tasks blocked from acquiring the semaphore, the operating system posts the semaphore.
4. Call the LOS_SemDelete API to delete a semaphore.

Semaphore Error Codes

Error codes are returned if errors occur during semaphore operations, such as creating, pending, posting, and deleting semaphores, to facilitate fault locating.

No.	Definition	Error Code	Description	Solution
1	LOS_ERRNO_SEM_NO_MEMORY	0x02000700	The memory space is insufficient.	Allocate a larger memory space to the semaphore.
2	LOS_ERRNO_SEM_INVALID	0x02000701	The passed-in parameter is invalid.	Modify the parameter to a valid one.
3	LOS_ERRNO_SEM_PTR_NULL	0x02000702	The passed-in pointer is null.	Pass in a non-null pointer.

Copyright © Huawei Technologies Co., Ltd.

No.	Definition	Error Code	Description	Solution
4	LOS_ERRNO_SEM_ALL_BUSY	0x02000703	The semaphore control block is unavailable.	Post semaphore resources.
5	LOS_ERRNO_SEM_UNAVAILABLE	0x02000704	The scheduled time is invalid.	Modify the scheduled time to a correct one.
6	LOS_ERRNO_SEM_PEND_INTERR	0x02000705	When the CPU is processing interrupts, the LOS_SemPend API is called.	Do not call the LOS_SemPend API when the CPU is processing interrupts.
7	LOS_ERRNO_SEM_PEND_IN_LOCK	0x02000706	The task is locked and fails to obtain the semaphore.	Do not call the LOS_SemPend API when the task is locked.
8	LOS_ERRNO_SEM_TIMEOUT	0x02000707	The time for obtaining semaphores expires.	Set the time to a proper range.
9	LOS_ERRNO_SEM_OVERFLOW	0x02000708	The number of allowed semaphore pendings exceeds the maximum value.	Pass in a valid value.
10	LOS_ERRNO_SEM_PENDED	0x02000709	The task queue waiting for the semaphore is not empty.	Wake up all tasks that are waiting for the semaphore, and then delete the semaphore.

An error code is a 32-bit storage unit. Bit 24 to bit 31 indicate an error level; bit 16 to bit 23 indicate an error code flag; bit 8 to bit 15 indicate the ID of the module that reports the error code; bit 0 to bit 7 indicate an error code. The following is the example of an error code:

```
#define LOS_ERRNO_OS_NORMAL(MID,ERRNO)  \
(LOS_ERRTYPE_NORMAL | LOS_ERRNO_OS_ID | ((UINT32)(MID) << 8) | (ERRNO))
LOS_ERRTYPE_NORMAL: Define the error level as critical
LOS_ERRNO_OS_ID: OS error code flag.
MID: OS_MOUDLE_ID
ERRNO: error ID number
```

For example:

```
LOS_ERRNO_SEM_NO_MEMORY        LOS_ERRNO_OS_ERROR(LOS_MOD_SEM, 0x00))
```

NOTE

Error code IDs 0x16 and 0x1c are not defined and unavailable for use.

Copyright © Huawei Technologies Co., Ltd.

Platform Differences

None.

3.7.3 Precautions

- As interrupts cannot be blocked, permanent blocking and temporary blocking are not allowed for interrupts during the request for a semaphore.

3.7.4 Programming Example

Example Description

In the programming example, the following activities will happen:

1. The Example_TaskEntry task is executed to create a semaphore. Task scheduling is locked. Two tasks Example_SemTask1 and Example_SemTask2 are created, where Example_SemTask2 takes a higher priority than Example_SemTask1. Then, task scheduling is unlocked. Example_TaskEntry releases the semaphore.
2. Example_SemTask2 is granted the semaphore, scheduled, and sent to sleep mode for 20 ticks. While Example_SemTask2 is delayed, Example_SemTask1 is woken up.
3. Example_SemTask1 pends on the semaphore and is willing to wait the semaphore for 10 ticks to become free. At the time when Example_SemTask1 requests the semaphore, the semaphore is held by Example_SemTask2 and consequently Example_SemTask1 is suspended. After the 10-tick wait period elapses, the semaphore is still out of the reach of Example_SemTask1, and Example_SemTask1 is woken up, attempting to wait permanently for the semaphore. The wait for semaphore switches Example_SemTask1 to suspended state.
4. After 20 ticks, Example_SemTask2 is woken up and releases the semaphore. Example_SemTask1 is scheduled, granted the semaphore, and finally releases it.
5. 40 ticks after Example_SemTask1 is finished, Example_TaskEntry is woken up, deletes the semaphore and then the two tasks.

Example Code

Prerequisites

- The **LOSCFG_BASE_IPC_SEM** parameter in the **los_config.h** file is set to YES.
- The **LOSCFG_BASE_IPC_SEM_LIMIT** parameter in the **system_config.h** file is set to the maximum number (for example, 1024) of semaphores that the operating system allows.

The code is as follows:

```
#include "los_sem.h"

 /*Task PID*/
 static UINT32 g_TestTaskID01,g_TestTaskID02;
 /*Task priority*/
 #define TASK_PRIO_TEST  5
 /*Semaphore structure ID*/
 static SEM_HANDLE_T g_usSemID;


 VOID Example_SemTask1(void)
 {
```


Copyright © Huawei Technologies Co., Ltd.

```
    UINT32 uwRet;

    printf("Example_SemTask1 try get sem g_usSemID ,timeout 10 ticks.\n");
    /*The task pends on the semaphore in timed blocking mode, with the wait
period being 10 ticks*/
    uwRet = LOS_SemPend(g_usSemID, 10);

    /*The task is granted the semaphore.*/
    if(LOS_OK == uwRet)
    {
         LOS_SemPost(g_usSemID);
         return;
    }
    /*The task does not get the semaphore within 10 ticks.*/
    if(LOS_ERRNO_SEM_TIMEOUT == uwRet)
    {
        printf("Example_SemTask1 timeout and try get sem g_usSemID wait forever.
\n");
        /*The task pends on the semaphore in permanent blocking mode.*/
        uwRet = LOS_SemPend(g_usSemID, LOS_WAIT_FOREVER);
        printf("Example_SemTask1 wait_forever and get sem g_usSemID .\n");
        if(LOS_OK == uwRet)
        {
            LOS_SemPost(g_usSemID);
          return;
        }
    }
    return;

 }
  VOID Example_SemTask2(void)
 {
    UINT32 uwRet;
    printf("Example_SemTask2 try get sem g_usSemID wait forever.\n");
    /*The task pends on the semaphore in permanent blocking mode.*/
    uwRet = LOS_SemPend(g_usSemID, LOS_WAIT_FOREVER);

    if(LOS_OK == uwRet)
    printf("Example_SemTask2 get sem g_usSemID and then delay 20ticks .\n");

    /*Send the task to sleep mode for 20 ticks.*/
    LOS_TaskDelay(20);

    printf("Example_SemTask2 post sem g_usSemID .\n");
    /*The task releases the semaphore.*/
    LOS_SemPost(g_usSemID);

    return;

 }
 UINT32 Example_TaskEntry()
 {
    UINT32 uwRet;
    TSK_INIT_PARAM_S stTask1;
    TSK_INIT_PARAM_S stTask2;

  /*Create the semaphore.*/
    LOS_SemCreate(0,&g_usSemID);

    /*Lock task scheduling.*/
    LOS_TaskLock();

    /*Create task 1.*/
    memset(&stTask1, 0, sizeof(TSK_INIT_PARAM_S));
    stTask1.pfnTaskEntry = (TSK_ENTRY_FUNC)Example_SemTask1;
    stTask1.pcName       = "MutexTsk1";
    stTask1.uwStackSize  = OS_TSK_DEFAULT_STACK_SIZE;
    stTask1.usTaskPrio = TASK_PRIO_TEST;
    uwRet = LOS_TaskCreate(&g_TestTaskID01, &stTask1);
```


Copyright © Huawei Technologies Co., Ltd.

```
    if(uwRet != LOS_OK)
    {
        printf("task1 create failed .\n");
        return LOS_NOK;
    }

    /*Create task 2.*/
    memset(&stTask2, 0, sizeof(TSK_INIT_PARAM_S));
    stTask2.pfnTaskEntry = (TSK_ENTRY_FUNC)Example_SemTask2;
    stTask2.pcName       = "MutexTsk2";
    stTask2.uwStackSize  = OS_TSK_DEFAULT_STACK_SIZE;
    stTask2.usTaskPrio = (TASK_PRIO_TEST - 1);
    uwRet = LOS_TaskCreate(&g_TestTaskID02, &stTask2);
    if(uwRet != LOS_OK)
    {
        printf("task2 create failed .\n");
        return LOS_NOK;
    }

    /*Unlock task scheduling.*/
    LOS_TaskUnlock();

    uwRet = LOS_SemPost(g_usSemID);

    /*Send the task to sleep mode for 40 ticks.*/
    LOS_TaskDelay(40);

    /*Delete the semaphore.*/
    LOS_SemDelete(g_usSemID);

    /*Delete task 1.*/
    uwRet = LOS_TaskDelete(g_TestTaskID01);
    if(uwRet != LOS_OK)
    {
        printf("task1 delete failed .\n");
        return LOS_NOK;
    }
    /*Delete task 2.*/
    uwRet = LOS_TaskDelete(g_TestTaskID02);
    if(uwRet != LOS_OK)
    {
        printf("task2 delete failed .\n");
        return LOS_NOK;
    }

    return LOS_OK;
}
```

Verification

The verification result is as follows:

```
 Example_SemTask2 try get sem g_usSemID wait forever.
  Example_SemTask1 try get sem g_usSemID,timeout 10 ticks.
  Example_SemTask2 get sem g_usSemID and then delay 20ticks .
  Example_SemTask1 timeout and try get sem g_usSemID wait forever.
  Example_SemTask2 post sem g_usSemID.
  Example_SemTask1 wait_forever and get sem g_usSemID.
```

Complete Code

sample_sem.c

3.8 Time Management

Copyright © Huawei Technologies Co., Ltd.

3.8.1 Overview

Basic Concept

Time management provides time services to applications and uses system time as the reference time.

System time is generated when an output pulse of a timer/counter triggers an interrupt, it is an integral number or long integral number of ticks. The interval between consecutive output pulses is a tick. The tick length is statically configured.

User time is measured in seconds or milliseconds, whereas the OS clock measures time in ticks. When a user initiates an operation to the operating system, such as suspending or delaying a task, the time management module converts user time between seconds/milliseconds and ticks.

The rule for conversion between ticks and seconds is user configurable.

The time management module of Huawei LiteOS provides time conversion, measurement, and deferral to satisfy what users need.

Related Concepts

- Cycle

Cycle is the minimal time unit of the operating system. The system clock speed is represented in the form of cycles per second.

- Tick

A tick is the basic time unit used in OS. The tick length is user configurable. Typically, it is determined by the system clock speed and represented in the form of ticks per second.

3.8.2 Development Guidelines

Usage Scenarios

Read the topic when you want to learn more about system time and conversion between ticks and seconds/milliseconds.

Functions

The time management module of Huawei LiteOS provides the following functions:

- Time conversion: converts the OS runtime from ticks to milliseconds or microseconds
- Time measurement: measures the system runtime in ticks

Function Category	API	Description
Time conversion	LOS_MS2Tick	Converts milliseconds into ticks
	LOS_Tick2MS	Converts ticks into milliseconds

Copyright © Huawei Technologies Co., Ltd.

Function Category	API	Description
Time measurement	LOS_CyclePerTickGet	Counts the number of cycles per tick
	LOS_TickCountGet	Measures the runtime in ticks

Time Management Error Codes

Error codes are returned if errors occur during time conversion to facilitate fault locating.

No.	Definition	Error Code	Description	Solution
1	LOS_ERRNO_SYS_PTR_NULL	0x02000010	The passed-in pointer is null.	Pass in a valid pointer.
2	LOS_ERRNO_SYS_CLOCK_INVALID	0x02000011	The system clock configuration is invalid.	Configure valid clock settings in the **los_config.h** file.
3	LOS_ERRNO_SYS_MAXNUMOFCORES_IS_INVALID	0x02000012	The error code is not in-use.	N/A
4	LOS_ERRNO_SYS_PERIERRCOREID_IS_INVALID	0x02000013	The error code is not in-use.	N/A
5	LOS_ERRNO_SYS_HOOK_IS_FULL	0x02000014	The error code is not in-use.	N/A

Development Process

The typical time management development process is as follows:

1. Set the LOSCFG_BASE_CORE_TICK_HW_TIME parameter in the los_config.h file to YES.
   - Set the LOSCFG_BASE_CORE_TICK_PER_SECOND parameter in the los_config.h file to a valid number of ticks per second.
2. Call the clock conversion API.
3. Get the system runtime that is measured in ticks
   - Call the LOS_TickCountGet API to measure the runtime in ticks.

3.8.3 Precautions

- The system runtime (measured in ticks) can be acquired only after the system clock is enabled.

Copyright © Huawei Technologies Co., Ltd.

- The time management module works only after the OS_SYS_CLOCK in los_config.h is enabled and the LOSCFG_BASE_CORE_TICK_PER_SECOND of the Tick module is specified.
- When measured in ticks, system runtime is not accurate, because it is not measured while interrupts are disabled.

3.8.4 Programming Example

Example Description

The programming example will cover the following functions:

1. Time conversion: from milliseconds to ticks, or conversely
2. Time measurement and deferral: measures the number of cycles per second, the number of ticks for which the system is running, and the number of ticks for which the deferral lasts

Example Code

Prerequisites

- The LOSCFG_BASE_CORE_TICK_PER_SECOND in the los_config.h file is set to a valid number of ticks per second.
- The OS_SYS_CLOCK (unit: Hz) is set.

Time conversion:

```
VOID Example_TransformTime(VOID)
 {
     UINT32 uwMs;
     UINT32 uwTick;

     uwTick = LOS_MS2Tick(10000);//Convert 10000 ms into ticks
      printf("uwTick = %d \n",uwTick);
     uwMs= LOS_Tick2MS(100);//Convert 100 ticks into ms
         printf("uwMs = %d \n",uwMs);
 }
```

Time measurement and deferral:

```
  VOID Example_GetTime(VOID)
   {
       UINT32 uwcyclePerTick;
       UINT64 uwTickCount;
       uwcyclePerTick = LOS_CyclePerTickGet();//Number of cycles per tick
      if(0 != uwcyclePerTick)
       {
           dprintf("LOS_CyclePerTickGet = %d \n", uwcyclePerTick);
       }
       uwTickCount = LOS_TickCountGet();//Get the count of ticks
       if(0 != uwTickCount)
       {
           dprintf("LOS_TickCountGet = %d \n", (UINT32)uwTickCount);
       }
       LOS_TaskDelay(200);//200-tick deferral
       uwTickCount = LOS_TickCountGet();
       if(0 != uwTickCount)
       {
           dprintf("LOS_TickCountGet after delay = %d \n", (UINT32)uwTickCount);
       }
   }
```


Copyright © Huawei Technologies Co., Ltd.

Verification

The verification result is as follows:

Time conversion:

```
uwTick = 1000
uwMs = 1000
```

Time measurement and deferral:

```
LOS_CyclePerTickGet = 495000
LOS_TickCountGet = 1
LOS_TickCountGet after delay = 201
```

Complete Code

sample_time.c

3.9 Software Timer

3.9.1 Overview

Basic Concept

A software timer is a timer simulated by software, and works based on system tick interrupts. When a predefined number of ticks elapse, a software timer triggers a user-defined callback function. The timer length is an integral number of ticks.

Only a limited number of hardware timers can be used due to hardware constraints. Software timers can fulfill the demand for more timers, allowing you to create more timing services.

The software timer module supports the following functions:

- Statically disable a software timer by macro
- Create a software timer
- Start a software timer
- Stop a software timer
- Delete a software timer
- Measure the number of ticks that must elapse prior to expiry of a software timer

Operation Mechanism

Software timers are system resources and are allocated continuous memory at the initialization of the timer module. The maximum number of software timers supported by the operating system is defined by LOSCFG_BASE_CORE_SWTMR_LIMIT in the los_config.h file.

Software timers are placed in a queue and a triggered in the first in first out order. The software timers with a short life cycle are placed at the beginning of queue so that they will be triggered earlier than those with a longer life cycle.

Copyright © Huawei Technologies Co., Ltd.

The software timer length is measured in ticks. When a software timer is actuated, Huawei LiteOS determines the timer expiry time based on the current system time (in ticks) and timer length (in ticks) and adds the timer control structure to the global timing list.

When a tick interrupt occurs, the tick interrupt handler scans the global timing list for expired timers. If such a timer is found, the timer is recorded.

After the tick interrupt handler finishes processing, the software timer task (a task exclusively used for software timers) is assigned the highest priority and then woken up to call the Timer_Callback function (callback function that handles software timer expiry) of the expired timer.

Software Timer States

- OS_SWTMR_STATUS_UNUSED

While the timer module is being initialized, the operating system initializes all timer resources in the system to OS_SWTMR_STATUS_UNUSED state.

- OS_SWTMR_STATUS_CREATED

If the LOS_SwtmrCreate API is called in OS_SWTMR_STATUS_UNUSED state or if the LOS_SwtmrStop API is called after timer start-up, the timer switches to OS_SWTMR_STATUS_CREATED state.

- OS_SWTMR_STATUS_TICKING

If the LOS_SwtmrStart API is called after the timer is created, the timer switches to OS_SWTMR_STATUS_TICKING state.

Software Timer Modes

Depending on timer mode, software timers are classified into three types:

- One-shot timer: The timer triggers the timer event only once after it is started. Then, the timer is automatically deleted.
- Periodic timer: The timer triggers the timer event periodically until the timer is manually stopped.
- One-shot timer: The timer differs from the other type of one-shot timer. It will not be automatically deleted after it expires. Call the LOS_SwtmrDelete API to delete this type of one-shot timer.

3.9.2 Development Guidelines

Usage Scenarios

- If you want to trigger a timer event only once, create a one-shot timer and define a Timer_Callback function for the timer. When the timer expires, the Timer_Callback function will be executed.
- If you want to trigger a timer event periodically, create a periodic timer and define a Timer_Callback function for the timer. When the timer expires, the Timer_Callback function will be executed.

Functions

The software timer module provides the following functions. For details about the APIs, see the API reference.

Copyright © Huawei Technologies Co., Ltd.

Function Category	API	Description
Timer creation and deletion	LOS_SwtmrCreate	Creates a software timer
	LOS_SwtmrDelete	Deletes a software timer
Timer start and stop	LOS_SwtmrStart	Starts a software timer
	LOS_SwtmrStop	Stops a software timer
Measurement of remaining ticks prior to timer expiry	LOS_SwtmrTimeGet	Measures the number of ticks that must elapse prior to expiry of a software timer

Development Process

The typical software timer development process is as follows:

1. Set software timer.
   - Set LOSCFG_BASE_CORE_SWTMR and LOSCFG_BASE_IPC_QUEUE to YES.
   - Set LOSCFG_BASE_CORE_SWTMR_CONFIG to the maximum number of software timers supported by the operating system.
   - Set OS_SWTMR_HANDLE_QUEUE_SIZE to the maximum size of the software timer queue.
2. Call the LOS_SwtmrCreate API to create a software timer.
   - Creates a software timer that has a user-defined timer length, Timer_Callback function, and trigger mode; returns the software timer handler after successful creation.
   - Returns the function execution result (successful or failed).
3. Call the LOS_SwtmrStart API to start a software timer.
4. Call the LOS_SwtmrTimeGet API to get left number of Ticks of software timer.
5. Call the LOS_SwtmrStop API to stop a software timer.
6. Call the LOS_SwtmrDelete API to delete a software timer.

Software Timer Error Codes

Error codes are returned if errors occur during software timer operations, such as creating, deleting, suspending, or restarting a software timer, to facilitate fault locating.

No.	Definition	Error Code	Description	Solution
1	LOS_ERRNO_SWTMR_PTR_NULL	0x02000300	The callback function of the software timer is null.	Define the callback function of the software timer.

Copyright © Huawei Technologies Co., Ltd.

No.	Definition	Error Code	Description	Solution
2	LOS_ERRNO_SWTMR_INTERVAL_NOT_SUITED	0x02000301	The timer length of the software timer is 0.	Redefine the timer length.
3	LOS_ERRNO_SWTMR_MODE_INVALID	0x02000302	The mode of the software timer is incorrect.	Modify the mode of the software timer. Range: [0, 3].
4	LOS_ERRNO_SWTMR_RET_PTR_NULL	0x02000303	The passed-in pointer to the software timer ID is null.	Pass in a non-null pointer.
5	LOS_ERRNO_SWTMR_MAXSIZE	0x02000304	The number of software timers exceeds the maximum value.	Redefine the maximum number of software timers, or wait until a software timer releases resources.
6	LOS_ERRNO_SWTMR_ID_INVALID	0x02000305	The passed-in software timer ID is incorrect.	Pass in a correct software timer ID.
7	LOS_ERRNO_SWTMR_NOT_CREATED	0x02000306	No software timer is created.	Create a software timer.
8	LOS_ERRNO_SWTMR_NO_MEMORY	0x02000307	The memory space is insufficient for creating the linked list of a software timer.	Apply for a larger memory space for the software timer.
9	LOS_ERRNO_SWTMR_MAXSIZE_INVALID	0x02000308	The maximum number of software timers is incorrect.	Redefine the maximum number of software timers.
10	LOS_ERRNO_SWTMR_HWI_ACTIVE	0x02000309	A timer is used when the CPU is processing interrupts.	Modify the source code to ensure that no timer is used when the CPU is processing interrupts.
11	LOS_ERRNO_SWTMR_HANDLER_POOL_NO_MEM	0x0200030a	The memory space allocated to the membox is insufficient.	Expand the memory space.

Copyright © Huawei Technologies Co., Ltd.

No.	Definition	Error Code	Description	Solution
12	LOS_ERRNO_SWTMR_QUEUE_CREATE_FAILED	0x0200030b	The software timer queue fails to be created.	Check whether the memory space is sufficient for creating the queue.
13	LOS_ERRNO_SWTMR_TASK_CREATE_FAILED	0x0200030c	The software timer task fails to be created.	Allocate sufficient memory space for creating the software timer task.
14	LOS_ERRNO_SWTMR_NOT_STARTED	0x0200030d	The software timer is not started.	Start the software timer.
15	LOS_ERRNO_SWTMR_STATUS_INVALID	0x0200030e	The software timer status is incorrect.	Check the software timer status.
16	LOS_ERRNO_SWTMR_SORTLIST_NULL	Null	The error code is not in use.	N/A
17	LOS_ERRNO_SWTMR_TICK_PTR_NULL	0x02000310	The passed-in pointer used for obtaining the number of software timer timeout ticks is null.	Pass in a non-null pointer.

An error code is a 32-bit storage unit. Bit 24 to bit 31 indicate an error level; bit 16 to bit 23 indicate an error code flag; bit 8 to bit 15 indicate the ID of the module that reports the error code; bit 0 to bit 7 indicate an error code. The following is the example of an error code:

```
#define LOS_ERRNO_OS_NORMAL(MID,ERRNO)  \
(LOS_ERRTYPE_NORMAL | LOS_ERRNO_OS_ID | ((UINT32)(MID) << 8) | (ERRNO))
LOS_ERRTYPE_NORMAL: Define the error level as critical
LOS_ERRNO_OS_ID: OS error code flag.
MID: OS_MOUDLE_ID
ERRNO: error ID number
```

For example:

```
#define LOS_ERRNO_SWTMR_PTR_NULL  \
LOS_ERRNO_OS_ERROR(LOS_MOD_SWTMR, 0x00)
```

3.9.3 Precautions

- Limit the number of operations contained in the callback function of a software timer. Do not use the API or perform any operation that may suspend or block tasks.
- Software timers are placed in a queue. A task is used exclusively to convey software timer information. The priority of a task in a software timer is set to 0, which is not allowed to be modified.

Copyright © Huawei Technologies Co., Ltd.

- The maximum number of software timer resources is not equal to the total number of software timer resources available to users. When a software timer occupies a software timer resource, the number of available software timer resources is decreased by 1.
- After the callback function of a one-shot software timer is executed, the software timer is automatically deleted and the resources allocated to the timer are reclaimed.
- A one-shot software timer that will not be automatically deleted after expiration needs to be deleted by calling the LOS_SwtmrDelete API. Resources allocated to the timer are reclaimed to avoid resource leaks.

3.9.4 Programming Example

Example Description

In the programming example, the following steps will be performed:

1. Create, delete, start, stop or restart a software timer.
2. Use a one-shot software timer and a periodical software timer.

Example Code

Prerequisites

- The LOSCFG_BASE_CORE_SWTMR parameter in the los_config.h file is set to YES.
- The LOSCFG_BASE_CORE_SWTMR_LIMIT parameter in the los_config.h file is set to the maximum number of software timers supported by the operating system.
- The OS_SWTMR_HANDLE_QUEUE_SIZE parameter in the los_config.h file is set to the maximum size of the software timer queue.

The code is as follows:

```
void Timer1_Callback(uint32_t arg);  // Callback function

 void Timer2_Callback(uint32_t arg);

 UINT32 g_timercount1 = 0;
 UINT32 g_timercount2 = 0;

 void Timer1_Callback(uint32_t arg)//Callback function 1
 {
   unsigned long tick_last1;
   g_timercount1++;
   tick_last1=(UINT32)LOS_TickCountGet();//Acquire the current number of ticks
   dprintf("g_timercount1=%d\n",g_timercount1);
   dprintf("tick_last1=%d\n",tick_last1);

 }

 void Timer2_Callback(uint32_t arg)//Callback function 2
 {
   unsigned long tick_last2;
   tick_last2=(UINT32)LOS_TickCountGet();
   g_timercount2 ++;
   dprintf("g_timercount2=%d\n",g_timercount2);
   dprintf("tick_last2=%d\n",tick_last2);
 }

 void Timer_example (void)
{
   UINT16 id1;
   UINT16 id2;// timer id
```


Copyright © Huawei Technologies Co., Ltd.

```
   UINT32 uwTick;
  /*Create a one-shot software timer that will execute callback function 1 when
the 1000-tick life cycle expires.*/
   LOS_SwtmrCreate (1000, LOS_SWTMR_MODE_ONCE,Timer1_Callback,&id1,1);
  /*Create a periodic software timer that will execute callback function 2 at a
regular interval of 100 ticks.*/
   LOS_SwtmrCreate(100,LOS_SWTMR_MODE_PERIOD,Timer2_Callback,&id2,1);
   dprintf("create Timer1 success\n");

   LOS_SwtmrStart (id1); //Start the one-shot software timer.
   dprintf("start Timer1 sucess\n");

   LOS_TaskDelay(200);//200-tick delay
   LOS_SwtmrTimeGet(id1,&uwTick);//Get the number of ticks that must elapse
before expiry of the one-shot software timer.
   dprintf("uwTick =%d\n",uwTick);

   LOS_SwtmrStop(id1);//Stop the software timer.
   dprintf("stop Timer1 sucess\n");

   LOS_SwtmrStart(id1);
   LOS_TaskDelay(1000);
   LOS_SwtmrDelete(id1);//Delete the software timer.
   dprintf("delete Timer1 sucess\n");

   LOS_SwtmrStart(id2);//Start the periodic software timer.
   dprintf("start Timer2\n");

   LOS_TaskDelay(1000);
   LOS_SwtmrStop(id2);
   LOS_SwtmrDelete(id2);
  }
```

Verification

The verification result is as follows:

```
 --- Test start---
create Timer1 success
start Timer1 sucess
uwTick =800
stop Timer1 sucess
g_timercount1=1
tick_last1=1201
delete Timer1 sucess
start Timer2
g_timercount2=1
tick_last2=1301
g_timercount2=2
tick_last2=1401
g_timercount2=3
tick_last2=1501
g_timercount2=4
tick_last2=1601
g_timercount2=5
tick_last2=1701
g_timercount2=6
tick_last2=1801
g_timercount2=7
tick_last2=1901
g_timercount2=8
tick_last2=2001
g_timercount2=9
tick_last2=2101
g_timercount2=10
tick_last2=2201

 --- Test End ---
```

Complete Code

sample_Timer.c

Copyright © Huawei Technologies Co., Ltd.

3.10 Error Handling

3.10.1 Overview

Basic Concept

In the event of code errors, the operating system calls APIs of the error handling module to report error information and calls user-defined hook functions to handle the errors.

Internal OS error codes cannot be conveyed via APIs. A solution to address this problem is reporting the error codes to the error handling module and processing them with the aid of user-defined hook functions. If the OS reports a fatal error, it initiates exception management to keep a record of what happened at the time of the fatal error.

Through error handling, invalid user input can be reported and controlled to avoid possible program crashes.

Operation Mechanism

Error handling is a mechanism to control invalid inputs into programs. When a program runs into a problem, an error code, is displayed and the error handler (if any) is executed.

Figure 3-10 Error handling

Copyright © Huawei Technologies Co., Ltd.

3.10.2 Development Guidelines

Functions

The error handling module provides the following functions:

Function Category	API	Description
Error handling	LOS_ErrHandle	Handles the error according to an error handler

3.10.3 Precautions

None.

3.10.4 Programming Example

Example Description

The programming example will cover the following functions:

1. Executing an error handler

Example Code

The code is as follows:

```
extern USER_ERR_FUNC_S      g_stUserErrFunc;
 void *err_handler(CHAR *pcFileName,UINT32 uwLineNo,
 UINT32 uwErrorNo,UINT32 uwParaLen,VOID  *pPara)
 {
     printf("err handel ok\n");
 }
 UINT32 Example_ErrCaseEntry(VOID)
 {
     /*Execute an error handler.*/
     LOS_ErrHandle(NULL, 0,0,0, NULL);
     return LOS_OK;
 }
```

Verification

The verification result is as follows:

```
 --- Test start---
seterrno success
errno address:0x830d5768
err handel ok

 --- Test End ---
```

Complete Code

sample_err.c

Copyright © Huawei Technologies Co., Ltd.

3.11 Doubly Linked List

3.11.1 Overview

Basic Concept

A doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Each node in a doubly linked list contains two pointers that reference to the previous and to the next node in the sequence of nodes. The head of the doubly linked list is deterministic and immediately accessible.

Any node of a doubly linked list, once obtained, can be used to begin a new traversal of the list in either direction (towards the beginning or end) from the given node. This allows a lot of data to be quickly traversed. Because of the symmetric nature of a doubly linked list, nodes can easily be inserted into or removed from the list.

3.11.2 Development Guidelines

Functions

The doubly linked list module provides the following functions:

Function Category	API	Description
List initialization	LOS_InitList	Initializes a doubly linked list
Node insertion	LOS_ListAdd	Inserts a node to a doubly linked list
	LOS_ListTailInsert	Inserts a node to the tail of a doubly linked list
Node insertion	LOS_ListHeadInsert	Inserts a node to the head of a doubly linked list
Node deletion	LOS_ListDelete	Deletes a node from a doubly linked list
List status determinatio n	LOS_ListEmpty	Determines whether a doubly linked list is empty
Node deletion and list initialization	LOS_ListDelInit	Deletes a node from a doubly linked list Uses the node to initialize a doubly linked list

Development Process

The doubly linked list development process is as follows:

Copyright © Huawei Technologies Co., Ltd.

1. Call the LOS_InitList API to initialize a doubly linked list.
2. Call the LOS_ListAdd API to insert a node into the list.
3. Call the LOS_ListTailInsert API to insert a node into the tail of the list.
4. Call the LOS_ListDelete API to delete a node from the list.
5. Call the LOS_ListEmpty API to determine whether the doubly linked list is empty.
6. Call the LOS_ListDelInit API to delete a node and use the node to initialize the doubly linked list.

3.11.3 Precautions

- While inserting or deleting a node from a doubly linked list, ensure that the direction of pointers of adjacent nodes is correct.

3.11.4 Programming Example

Example Description

Before using a doubly linked list, ensure that sufficient memory space is available to store the list. After deleting a node from the list, do not forget to free up the memory occupied by the node.

In the programming example, the following steps will be performed:

1. Initialize a doubly linked list.
2. Insert a node into the list.
3. Delete a node from the list.
4. Check whether the insertion and deletion was successful.

Example Code

The code is as follows:

```
#include "stdio.h"
#include "los_list.h"

#ifdef __cplusplus
#if __cplusplus
extern "C" {
#endif /* __cpluscplus */
#endif /* __cpluscplus */

static UINT32 DLlist_sample(VOID)
{
    LOS_DL_LIST DLlist = {NULL,NULL};
    LOS_DL_LIST DLlistNode01 = {NULL,NULL};
    LOS_DL_LIST DLlistNode02 = {NULL,NULL};
    LOS_DL_LIST DLlistNode03 = {NULL,NULL};

    PRINTK("Initial head\n");
    LOS_ListInit(&DLlist);

    LOS_ListAdd(&DLlist,&DLlistNode01);
    if (DLlistNode01.pstNext == &DLlist && DLlistNode01.pstPrev == &DLlist)
    {
        PRINTK("Add DLlistNode01 success \n");
    }

    LOS_ListTailInsert(&DLlist,&DLlistNode02);
```


Copyright © Huawei Technologies Co., Ltd.

```
    if (DLlistNode02.pstNext == &DLlist && DLlistNode02.pstPrev == &DLlistNode01)
    {
        PRINTK("Tail insert DLlistNode02 success \n");
    }

    LOS_ListHeadInsert(&DLlistNode02,&DLlistNode03);
    if (DLlistNode03.pstNext == &DLlist && DLlistNode03.pstPrev == &DLlistNode02)
    {
        PRINTK("Head insert DLlistNode03 success \n");
    }

    LOS_ListDelInit(&DLlistNode03);
    LOS_ListDelete(&DLlistNode01);
    LOS_ListDelete(&DLlistNode02);

    if (LOS_ListEmpty(&DLlist))
    {
        PRINTK("Delete success \n");
    }

    return LOS_OK;
}

#ifdef __cplusplus
#if __cplusplus
}
#endif /* __cpluscplus */
#endif /* __cpluscplus */
```

Verification

The verification result is as follows:

```
Initial head
Add DLlistNode01 success
Tail insert DLlistNode02 success
Head insert DLlistNode03 success
Delete success
```

3.12 Bitwise Operation

3.12.1 Overview

Basic Concept

A bitwise operation refers to the bit operation on binary numerals. A variable can be set to a status word (SW). Each marker bit of a status word is user-defined. Developers can set a marker bit to 1 or 0, change the marker bit content, and obtain the most significant bit (MSB) and the least significant bit (LSB) with a marker bit of 1 from the SW. Moreover, developers can also perform bitwise operations on registers.

3.12.2 Development Guidelines

Functions

The bitwise operation module in Huawei LiteOS provides the following interfaces for developers:

Copyright © Huawei Technologies Co., Ltd.

Function Category	Interface	Description
Setting a marker bit to 1	LOS_BitMapSet	Sets a marker bit of an SW to 1.
Setting a marker bit to 0	LOS_BitMapClr	Sets a marker bit of an SW to 0.
Obtaining the most significant bit with a marker bit of 1	LOS_HighBitGet	Obtains the most significant bit with a marker bit of 1 in the SW.
Obtaining the least significant bit with a marker bit of 1	LOS_LowBitGet	Obtains the least significant bit with a marker bit of 1 in the SW.

3.12.3 Precautions

None.

3.12.4 Programming Example

Example Description

In this example, the following functions will be implemented by performing bitwise operations on data:

1. Set a marker bit to 1.
2. Obtain the most significant bit with a marker bit of 1.
3. Set a marker bit to 0.
4. Obtain the least significant bit with a marker bit of 1.

Example Code

```
#include "los_bitmap.h"
static UINT32 Bit_sample(VOID)
{
  UINT32 flag = 0x010101010;
  UINT16 usPos;

  dprintf("\nBitmap Sample!\n", flag);
  dprintf("The flag is 0x%8x\n", flag);

  usPos = 8;
  LOS_BitMapSet(&flag, usPos);
  dprintf("LOS_BitMapSet:\t Pos : %d, the flag is 0x%0+8x\n", usPos, flag);

  usPos = LOS_HighBitGet(&flag);
  dprintf("LOS_HighBitGet:\t The highest one bit is %d, the flag is 0x%0+8x\n",
usPos, flag);

  LOS_BitMapClr(&flag, usPos);
  dprintf("LOS_BitMapClr:\t Pos : %d, the flag is 0x%0+8x\n", usPos, flag);

  usPos = LOS_LowBitGet(&flag);
  dprintf("LOS_LowBitGet:\t The lowest one bit is %d, the flag is 0x%0+8x\n\n",
usPos, flag);
```


Copyright © Huawei Technologies Co., Ltd.

```
  return LOS_OK;
}
```

Verification

The verification result is as follows:

```
Bitmap Sample!
The flag is 0x10101010
LOS_BitMapSet: Pos : 8,  the flag is 0x10101110
LOS_HighBitGet:The highest one bit is 28, the flag is 0x10101110
LOS_BitMapClr: Pos : 28, the flag is 0x00101110
LOS_LowBitGet: The lowest one bit is 4, the flag is 0x00101110
```


Copyright © Huawei Technologies Co., Ltd.

4 Extended Kernel

About This Chapter

4.1 Dynamic Loading

4.2 Scatter Loading

4.3 Exception Management

4.4 CPU Utilization Percentage

4.5 Linux Adaption

4.6 C++ Support

4.7 MMU

4.8 Atomic Operation

4.9 Run-Stop

4.10 Tickless

4.1 Dynamic Loading

4.1.1 Overview

Basic Concept

Dynamic loading is a program loading technology.

Static linking is to link all module files of a program into an executable file, so that these files can be loaded into the memory as a whole. Dynamic loading enables developers to compile each module of a program into an independent file for dynamic loading into the memory, instead of linking all modules.

Static linking links all module files of a program together and loads them into memory at one time, featuring fast code loading. However, when a large program is involved and the modules in it need to be frequently changed and upgraded, static linking might waste the memory and disk space and make module upgrade difficult.

Copyright © Huawei Technologies Co., Ltd.

Dynamic loading is better than static linking in this scenario. An external module can be dynamically loaded or unloaded as required, helping share public code and smoothly upgrade modules.

In Huawei LiteOS, two file formats are supported: OBJ and SO.

SO files (or OBJ files) and the **bin** system image file required for dynamic loading are used together.

Related Concept

Symbol Tables

Symbol tables are arrays that record symbol names and the address information of the symbols in the memory. Symbol tables are loaded to the symbol management structure of the dynamic loading module when the dynamic linker is initialized. When a symbol needs to be relocated during module loading, the dynamic linker obtains the symbol address by searching the symbol management structure.

4.1.2 Development Guidelines

Functions

API	Description
LOS_LdDestroy	Destroys a dynamic loading module.
LOS_SoLoad	Dynamically loads an so module.
LOS_ObjLoad	Dynamically loads an obj module.
LOS_FindSymByName	Searches for a symbol address in a module or symbol table.
LOS_ModuleUnload	Unloads a module.
LOS_PathAdd	Adds a module search path.
LOS_DynParamReg	Sets dynamic loading parameters.
LOS_DynMemPoolSet	Set the memory pool address used for dynamic loading.

Copyright © Huawei Technologies Co., Ltd.

NOTE

The LOS_DynMemPoolSet interface entry must be the memory pool address initialized by LOS_MemInit, which is managed by the Huawei LiteOS memory management algorithm. And ensure that the memory pool does not coincide with the system memory pool. This interface needs to be used before loading .so or .o files.

Development Process

The implementation of dynamic loading involves the following steps:

1. **Preparing the Compilation Environment**
2. **Preparing the .o and .so Files and Compiling the System Image**
3. **Applying Dynamic Loading**
4. **Preparing the System Environment**

Preparing the Compilation Environment

Step 1 Add compilation options of the .o module and the .so module.

- Add -nostdlib -fno-PIC to the compilation options of .o modules.
- Add -nostdlib -fPIC -shared to the compilation options of .so modules.

NOTE

Determine whether to use the following compilation option based on your needs.

-z max-page-size=value

This compilation option is used to set loadable program segments to be aligned to **value**. This compilation option helps reduce the blank areas required for the alignment of the virtual addresses of adjacent loadable segments.

If this compilation option is not used, the default alignment is 0x10000.

NOTE

To implement the dynamic loading in an IP camera (IPC), the start addresses of all nodal regions of the LD_SHT_PROGBITS and LD_SHT_NOBITS types in the module files must be on the boundary of four bytes. Otherwise, the modules will not be loaded.

The following is an example of adding compilation options of .o and .so modules:

```
RM = -rm -rf

CC = arm-hisiv500-linux-gcc

SRCS = $(wildcard *.c)
OBJS = $(patsubst %.c,%.o,$(SRCS))
SOS = $(patsubst %.c,%.so,$(SRCS))

all: $(SOS)

$(OBJS): %.o : %.c
    @$(CC) -nostdlib -c $< -o $@
$(SOS): %.so : %.c
    @$(CC) -nostdlib $< -fPIC -shared -o $@

clean:
    @$(RM) $(SOS) $(OBJS)

.PHONY: all clean
```

Step 2 Compile the system image.

Copyright © Huawei Technologies Co., Ltd.

The makefile used to compile the **bin** system image file must include the **config.mk** file under the root directory. The LITEOS_CFLAGS or the LITEOS_CXXFLAGS compilation option in the makefile should be used. The code is similar to the following:

```
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
```

----End

Preparing the .o and .so Files and Compiling the System Image

Perform the following steps to compile the system image:

Step 1 Compile .o modules and .so modules and copy the .o and .so files required for system running to a directory, such as the following directory:

```
/home/wmin/customer/out/so
```

NOTE

- If **a.so** needs to call functions in **b.so**, or **a.so** uses data in **b.so**, **a.so** depends on **b.so.**
- When **a.so** depends on **b.so**, and **b.so** needs to be automatically loaded when **a.so** is being loaded, **b.so** should be a compilation parameter during the compilation of **a.so**. If **b.so** is not a compilation parameter during the loading of **a.so**, load **b.so** before loading **a.so**.

Step 2 Access the **Huawei_LiteOS/tools/scripts/dynload_tools** directory and run the **sym.sh** script as follows:

```
$ ./sym.sh /home/wmin/customer/out/so
```

NOTE

- Pass in the absolute path to the directory that stores the .o and .so files required for system running to the **sym.sh** script.
- If the required .o and .so files are updated, run the **sym.sh** script again. This script extracts all system symbols of the .o and .so files to be loaded. The compiler will calculate the addresses of those symbols when compiling the system image.
- Run this command under the **Huawei_LiteOS/tools/scripts/dynload_tools** directory.

Step 3 Compile the .bin system image file. For example, if the .bin file is saved in the **/home/wmin/customer/out/bin/** directory, the **sample** image file and the **sample.bin** file to be burnt into the Flash memory are generated in this directory after compilation.

Copyright © Huawei Technologies Co., Ltd.

NOTE

If the .o and .so files to be loaded contain undefined external symbols, which are neither defined in the .o and .so files nor valid global symbols, an error message will be prompted during the compilation of the .bin system image file. Troubleshoot the error to ensure the correct compilation.

----End

Applying Dynamic Loading

Step 1 Specify the loading policy for SO files.

- If you are loading OBJ files, ignore this step.
- In various application scenarios, SO files may be stored in storage media in ZIP or NOZIP format. Operations of reading and writing into ZIP files differ from those for NOZIP files. Therefore, the loading policy of SO files needs to be specified before the initialization of the dynamic loading module.

```
DYNLOAD_PARAM_S stDynloadParam = {ZIP}; //Specify the ZIP or NOZIP loading policy.

LOS_DynParamReg(&stDynloadParam);
```

The LOS_DynParamReg() function only specifies the loading policy. The specified loading policy will be adopted during subsequent loading of SO files.

The ZIP loading policy must be adopted for so files in the ZIP format. Common SO files can be successfully loaded when either one policy is adopted, and the NOZIP policy is recommended. If no policy is specified, the NOZIP policy is adopted by default.

NOTE

NOZIP policy: Common so files can be read through multiple calls to lseek() and read(). To read a so file, the size of the loadable segment is first calculated based on the **SegmentHeader** of the file. Then memory of the same size is allocated to the segment, and the segment is loaded into memory. In this way, a small segment of the file is read each time without wasting memory.

ZIP policy: A ZIP file must be loaded into memory (mem1) as a whole. However, the size of the loadable segment of the file is unknown during the first reading, indicating that memory (mem2) needs to be then reallocated to the loadable segment. mem2 also contains all information required for dynamic loading as mem1 does, and mem1 will not be used for dynamic loading. Therefore, the coexistence of mem1 and mem2 will cause memory waste and overhigh peak memory usage. To solve these problems, the ZIP policy is adopted, and the ZIP file are read for twice. At the first time, the size of the memory required for the loadable segment is calculated, after which the memory allocated to the calculation is immediately released. At the second time, the loadable segment is loaded into memory. In this way, the ZIP policy is used to avoid overhigh peak memory usage at the cost of using one more instruction.

Step 2 Load a module.

- The dynamic loading module of an IP camera (IPC) supports the dynamic loading of .o modules and .so modules. Call the LOS_ObjLoad API to dynamically load .obj files.

```
if ((handle = LOS_ObjLoad("/yaffs/bin/dynload/foo.o")) == NULL){
    printf("load module ERROR!!!!!!\n");
    return 1;
}
```

- Call the LOS_SoLoad API to dynamically load .so files.

```
if ((handle = LOS_SoLoad("/yaffs/bin/dynload/foo.so")) == NULL){
    printf("load module ERROR!!!!!!\n");
    return 1;
}
```

When module A of an .so file depends on module B, and the dependency is specified with **B.so** being a compilation parameter during the compilation of **A.so**, module B will be automatically loaded during the loading of module A. If the dependency is not

Copyright © Huawei Technologies Co., Ltd.

specified, ensure that module B has been successfully loaded during the loading of module A.

Step 3 Obtain the address of a symbol.

- Search for a symbol in a specified module.

  When searching for a symbol in a specified module, call the LOS_FindSymByName API and set the first argument value to the handle of the module in which the symbol is searched for.

```
if ((ptr_magic = LOS_FindSymByName(handle, "os_symbol_table")) == NULL) {
     printf("symbol not found\n");
     return 1;
 }
```

- Search for a symbol in the global symbol table.

  When searching for a symbol in the global symbol table (OS modules including your modules and other user modules), call the LOS_FindSymByName API and set the first argument value to **NULL**.

```
if ((pFunTestCase0 = LOS_FindSymByName(NULL, "printf")) == NULL) {
     printf("symbol not found\n");
     return 1;
 }
```

Step 4 Use the obtained symbol address: LOS_FindSymByName returns a symbol address (VOID *pointer). You can change the type of the symbol that is located at this address for different use. The following examples describe the use of two types of symbols:

- **Integer symbol (data symbol)**

  test.c needs to be loaded, and the global variable *g_uwTest* = 0 is available.

  Run the following code to obtain the address of *g_uwTest*:

```
const char *g_pscOsOSSymtblFilePath = "/yaffs/bin/dynload/test.so";
UINT32 * g_uwTestPtr = NULL;

 INT8 *pPtr = (INT8 *)NULL;
 if ((pOSSymtblHandler = LOS_SoLoad(g_pscOsOSSymtblFilePath)) == NULL) {
     return LOS_NOK;
 }
 if ((pPtr = LOS_FindSymByName(pOSSymtblHandler, g_uwTest)) == NULL) {
     printf("os_symtbl not found\n");
     return LOS_NOK;
 }
 g_uwTest = (UINT32 *)pPtr;/* Forcibly change the pointer type to a real
pointer type */
```

- Function symbol

  The test_0 function expecting no arguments and the test_2 function expecting two arguments are defined in foo.c. foo.o can be generated by compiling foo.c. The following code shows how to obtain and call the functions in the foo.o module in demo.c.

```
foo.c:
 int test_0(void) { return 0; }
 int test_2(int i, int j) { return 0; }
demo.c:
 typedef unsigned int (* TST_CASE_FUNC)();/* Declaration of the type of the
pointer to a function that expects no parameters */
 typedef unsigned int (* TST_CASE_FUNC1)(UINT32); /* Declaration of the type
of the pointer to a function that expects one parameter */
 typedef unsigned int (* TST_CASE_FUNC2)(UINT32, UINT32); /* Declaration of
the type of the pointer to a function that expects two parameters */
 TST_CASE_FUNC pFunTestCase0 = NULL;/* Definition of a pointer to a function
*/
 TST_CASE_FUNC2 pFunTestCase2 = NULL;
 handle = LOS_ObjLoad("/yaffs/bin/dynload/foo.o");
```


Copyright © Huawei Technologies Co., Ltd.

```
pFunTestCase0 = NULL;
pFunTestCase0 = LOS_FindSymByName(handle, "test_0");
if (pFunTestCase0 == NULL){
    printf("can not find the function name\n");
    return 1;
}
uwRet = pFunTestCase0();
pFunTestCase2 = NULL;
pFunTestCase2 = LOS_FindSymByName(NULL, "test_2");
if (pFunTestCase2 == NULL){
    printf("can not find the function name\n");
    return 1;
}
uwRet = pFunTestCase2(42, 57);
```

Step 5 Unload the module.

Call the LOS_ModuleUnload API to unload a module and use the handle of the module to be unloaded as the input parameter. Call the LOS_ModuleUnload API to unload an .obj file handle or an .so file handle that has been loaded.

```
uwRet = LOS_ModuleUnload(handle);
 if (uwRet != LOS_OK) {
     printf("unload module failed");
     return 1;
 }
```

Step 6 Destroy the dynamic loading module.

Call the LOS_LdDestroy API to destroy the dynamic loading module when it is no longer in need.

NOTE

All modules that have been loaded will be automatically unloaded when the dynamic loading module is destroyed.

```
LOS_LdDestroy();
```

NOTE

The dynamic loading module should be destroyed when services no longer need it.

Step 7 Use relative paths.

If you want to use relative paths, specifically, if you use mechanisms similar to environment variables, call the LOS_PathAdd API to add the relative paths.

```
uwRet = LOS_PathAdd("/yaffs/bin/dynload");
if (uwRet != LOS_OK) {
    printf("add relative path failed");
    return 1;
}
```

After the relative paths are added, pass in file names instead of absolute paths when you call the LOS_SoLoad and LOS_ObjLoad APIs. Then the modules with the passed-in file names will be automatically located in the added relative paths.

If the passed-in multiple paths contain modules with the same file name, the module in the first passed-in path will be loaded.

NOTE

- A relative path can be used only after it is added by calling the LOS_PathAdd API.
- You can add multiple relative paths by calling the LOS_PathAdd API for multiple times.

----End

Copyright © Huawei Technologies Co., Ltd.

Preparing the System Environment

SO files (or OBJ files) and the .bin system image file are used together.

The SO files (or OBJ files) must be stored in file systems such as JFFS2, YAFFS, and FAT.

Perform the following steps:

Step 1 Burn the .bin system image file to the Flash. This image does not enable the dynamic loading function.

Step 2 If the SO files (or OBJ files) are stored on a hot-swappable SD card, update the SO files (or OBJ files) to a specified path on the SD card.

Step 3 If the SO files (or OBJ files) are stored in JFFS2 or YAFFS, update the SO files (or OBJ files) in the following ways:

- Burn the file system image.
- After Huawei LiteOS is started, run the **tftp** command similar to the following to download the SO files (or OBJ files) and the **elf_symbol.so** file:

```
tftp -g -l /yaffs0/foo.so -r foo.so 10.67.211.235
```

Step 4 Enable dynamic loading.

----End

Shell Debugging

Some commands related to dynamic loading are encapsulated in Shell for debugging.

For details on Shell commands, see **Command Reference**.

- **Loading a Module**

Shell command: mopen

```
Huawei LiteOS# mopen /yaffs/bin/dynload/foo.o
 module handle: 0x80391928
 Huawei LiteOS#
```

NOTE

The path to the module to be loaded must be an absolute path.

- **Searching for a Symbol**

Shell command: findsym

```
Huawei LiteOS# findsym 0 printf
 symbol address:0x8004500c
 Huawei LiteOS#
Huawei LiteOS# findsym 0x80391928 test_0
 symbol address:0x8030f241
 Huawei LiteOS#
```

- **Calling a Symbol**

Shell command: call

```
Huawei LiteOS# call 0x8030f241
 test_0
 Huawei LiteOS#
```

- **Unloading a Module**

Shell command: mclose

Copyright © Huawei Technologies Co., Ltd.

```
Huawei LiteOS# mclose 0x80391928
 Huawei LiteOS#
```

- **Destroying a Dynamic Loading Module**

Shell command: lddrop

```
Huawei LiteOS# lddrop
 Huawei LiteOS#
```

NOTE

If no errors are returned, the dynamic linker is successfully destroyed.

4.1.3 Precautions

- The -nostdlib -fno-PIC option needs to be added to the compilation options of .o modules.
- The -nostdlib -fPIC -shared option needs to be added to the compilation options of .so modules.
- Before the compilation of the system image, the required .o and .so files must be available to ensure that the external symbols used by the files have been integrated into the system image when the image file is being compiled.
- Huawei LiteOS dynamic loading requires that the start addresses of LD_SHT_PROGBITS and LD_SHT_NOBITS nodal regions in the module file must be aligned by four bytes. Otherwise, the module will not be loaded.
- While loading a library file, once Huawei LiteOS detects that the external symbols referenced by this file are repeatly defined in multiple modules that have no dependency relationships with this file, the relocation of the symbol references will be rejected. Then the library file fails to be loaded. Ensure that no symbols (variables or functions) that are repeatly defined exist in the library file to be loaded.
- Ensure that the sources of the files to be loaded are reliable and secure.
- Problematic loaded files will result in problems including but not limited to device damage, data leakage, and data tampering, for which Huawei assumes no responsibility.
- Do not load files from high-risk media such as SD cards or USB flash drives. If such file loading is necessary, ensure that the files are reliable. Huawei is not liable for any loss or problems caused thereby.

4.1.4 Programming Example

Example Description

The test_0 function expecting no parameters and the test_2 function expecting two parameters are defined in foo.c. foo.o can be generated by compiling foo.c. The following code shows how to obtain and call the functions in the foo.o module in demo.c.

Example Code

The code is as follows:

```
foo.c:
 int test_0(void) { printf("test_0\n"); return 0; }
 int test_2(int i, int j) { printf("test_2: %d %d\n", i, j); return 0; }
 demo.c:
 typedef int (* TST_CASE_FUNC)(); /* Declaration of the type of the pointer to a
function that expects no parameters */
```


Copyright © Huawei Technologies Co., Ltd.

```
 typedef int (* TST_CASE_FUNC1)(UINT32); /* Declaration of the type of the
pointer to a function that expects one parameter */
 typedef int (* TST_CASE_FUNC2)(UINT32, UINT32); /* Declaration of the type of
the pointer to a function that expects two parameters */
 unsigned int uwRet;
 TST_CASE_FUNC pFunTestCase0 = NULL;/* Definition of a pointer to a function */
 TST_CASE_FUNC2 pFunTestCase2 = NULL;
 handle = LOS_ObjLoad("/yaffs/bin/dynload/foo.o");
 pFunTestCase0 = NULL;
 pFunTestCase0 = LOS_FindSymByName(handle, "test_0");
 if (pFunTestCase0 == NULL){
     printf("can not find the function name\n");
     return 1;
 }
 uwRet = pFunTestCase0(); /* Call the pointer to this function */
 pFunTestCase2 = NULL;
 pFunTestCase2 = LOS_FindSymByName(NULL, "test_2");
 if (pFunTestCase2 == NULL){
     printf("can not find the function name\n");
     return 1;
 }
 uwRet = pFunTestCase2(42, 57);  /* Call the pointer to this function */
 uwRet = LOS_ModuleUnload(handle);
 if (uwRet != LOS_OK) {
     printf("unload module failed");
     return 1;
 }
 uwRet = LOS_LD_Destroy();
 if (uwRet != LOS_OK) {
     printf("destroy dynamic loader failed");
     return 1;
 }
```

Verification

The verification result is as follows:

```
Huawei LiteOS#
 ****************************
 ****************************
 test_0
 test_2:42 57
 ****************************
 ****************************
```

Complete Code

sample_foo.c

sample_Dynamic_loading.c

4.2 Scatter Loading

4.2.1 Overview

Basic Concept

Scatter loading is a technology that achieves fast boot of specified code. It shortens the time between the boot of an OS and the execution of specified code by preferentially loading specified code into memory. In this way, scatter loading can be used to fast boot key services.

Copyright © Huawei Technologies Co., Ltd.

An embedded OS loads image files on the flash memory into memory through uboot. Image files are probably large and the speed of flash reading is limited. Therefore, the requirements on the boot speed of time-sensitive services probably cannot be met if all images are executed after they are loaded.

Scatter loading enables the fast boot of key services by preferentially loading and executing some images containing time-sensitive services.

Scatter Loading in Huawei LiteOS

The scatter loading in Huawei LiteOS consists of two phases. Images of key services are loaded into memory through uboot and executed in the first phase, and the remaining images are loaded into memory and executed in the second phase. By properly organizing image loading, loading part of the images containing key services in the first phase is faster than loading all images, which shortens the time between the boot of an OS and the running of key services.

On Huawei LiteOS implemented on an IP camera (IPC), it takes 1s from powering on camera to starting preview, far shorter than the time (3s to 4.5s) taken on Linux because the scatter loading technology is applied.

Operation Mechanism

Scatter loading is used to preferentially load and execute the time-sensitive services by putting the data and code segments related to these services at the front-end of image files and loading the images at the front-end in the first phase of scatter loading, which enables time-sensitive services to run within the shortest time.

After these services are executed, the scatter loading API is called in the code used in the first phase to load the rest images and run the services specified by the rest images.

Figure 4-1 Scatter loading process

The internal conceptual diagram of scatter loading is shown in figure 2.

Copyright © Huawei Technologies Co., Ltd.

Figure 4-2 Internal conceptual diagram of scatter loading

Scatter loading enables the key services to be loaded and executed first, after which the non-key services are loaded.

4.2.2 Development Guidelines

Usage Scenarios

The scatter loading is applied in the scenario where fast boot of time-sensitive services is needed.

On an embedded OS, some services require short boot time. For example, in Huawei LiteOS on an IPC, the time required from powering on the camera and starting preview needs to be short, and the scatter loading technique can be adopted to enable fast boot of the recording service.

Functions

The scatter loading module of Huawei LiteOS provides the following function:

Type	API	Description
Scatter loading API	LOS_ScatterLoad	This API is called in the last phase of scatter loading to load the rest non-urgent services from images.

Development Process

The following figure shows the operation process of scatter loading:

Copyright © Huawei Technologies Co., Ltd.

Step 1 Call the LOS_ScatterLoad API and write service code.

The code entry is the app_init function contained in the **os_adapt.c** file. Following the code for loading urgent services, write the code used to call the LOS_ScatterLoad API for scatter loading. Enable the MAKE_SCATTER_IMAGE macro to control the compilation of the code for non-urgent services. The example code is as follows:

```
void app_init() {
    proc_fs_init();
    hi_uartdev_init();
    system_console_init("/dev/uartdev-0");
    LOS_CppSystemInit((unsigned long)&__init_array_start__, (unsigned
long)&__init_array_end__, BEFORE_SCATTER);
    LOS_ScatterLoad(0x100000, flash_read, NAND_READ_ALIGN_SIZE);

#ifndef MAKE_SCATTER_IMAGE /* The following are non-urgent services. */
    LOS_CppSystemInit((unsigned long)&__init_array_start__, (unsigned
long)&__init_array_end__, AFTER_SCATTER);

    extern unsigned int osShellInit(void);
    osShellInit();
    rdk_fs_init();
    SDK_init();

    hi_product_driver_init();
    char *apszArgv[3]={"vs_server","./higv.bin","-i"};
    vs_server(3, apszArgv);
#endif /* MAKE_SCATTER_IMAGE */
}
```

NOTE

The **os_adapt.c** file can be found in the **platform/bsp/board/hi3516a/os_adapt** directory in the Huawei LiteOS code package.

Copyright © Huawei Technologies Co., Ltd.

Step 2 Configure the **SCATTER_SRC** variable.

Run the following command to set the **SCATTER_SRC** variable in build/mk/wow_scatter.mk under the root directory to the source file of the services that call the scatter loading function. **LITEOSTOPDIR** represents the root directory of Huawei LiteOS code.

```
SCATTER_SRC := $(LITEOSTOPDIR)/platform/bsp/board/$(LITEOS_PLATFORM)/os_adapt/
os_adapt.c
```

Step 3 Run the make command to compile all images.

- Run the following command under the root directory to compile all service code.

  ```
  Huawei_LiteOS$ make
  ```

  After compilation, information about the size of urgent service images will be returned, which is similar to the following:

  ```
  ###### ###### ##### ##### ##### ##### #####
  #######Calculate the size of scatter#######
  the size is 0x4E0000
  ####################end#####################
  ```

- View the image segment allocation. If there are scatter loading segments in the images, the scatter loading is successfully started. In the directory where system images are generated, for example, **out/hi3516a** of the hi3516a platform, run the ***readelf -S vs_server*** command to open the system image file **vs_server**. Information similar to the following will be displayed, including segment name, start address, and offset. In the following figure, .fast_rodata, .fast_text, and .fast_data indicate the read-only segment, code segment, and data segment of scatter loading respectively.

  ```
  [15] .fast_rodata      PROGBITS        80159000 088868 0a6adc
  [16] .fast_text        PROGBITS        80200000 12f868 27ece0
  [17] .fast_data        PROGBITS        8047f000 3ae868 067b78
  [18] .got              PROGBITS        804e6b78 4163e0 000504
  [19] .text             PROGBITS        804e8000 417868 1b1b90
  [20] .rodata           PROGBITS        8069a000 5c9868 09f4d8
  [21] .data             PROGBITS        8073a000 669868 01cd7c
  [22] .bss              NOBITS          80757000 6865e4 6d74e0
  ```

  View the .text segment in the link script of scatter loading. scatter.o(*.text*) is added, as shown in the following figure, indicating that symbols related to the fast booted code of scatter loading are placed in the same area.

  ```
      .fast_text ALIGN (0x1000): { __fast_text_start = ABSOLUTE(.); . = .;
  scatter.o(*.text*);
      }

      . = (ABSOLUTE (.) + (0x1000 - 1)) & ~ (0x1000 - 1);
      __fast_text_end = ABSOLUTE (.);
  ```

  NOTE

  The path to the link script of the scatter loading is **Huawei_LiteOS/tools/scripts/ld/scatter.ld**.

Step 4 Run the **tftp *0x82000000 vs_server.bin*;nand erase *0x100000 0x700000*;nand write *0x82000000 0x100000 0x700000*;** command to burn all images into the Flash.

In the serial port tool interface, enter the following command to burn all images into the Flash at the address of 0x100000.

```
tftp 0x82000000 vs_server.bin;nand erase 0x100000 0x700000;nand write 0x82000000
0x100000 0x700000;
```

vs_server.bin in the command is the name of system image file. Burn this file into memory at the address of 0x82000000. Then burn it into the Flash starting at the address of 0x100000.

Copyright © Huawei Technologies Co., Ltd.

The size of file to be burnt is 0x700000, indicating that the size of the burnt image file must not exceed 7 MB, adjust to actual size.

Step 5 Run the **nand read *0x80008000 0x100000 0x4E0000*; go *0x80008000*;** command to read the urgent service images of 0x4E0000 from the address of 0x100000 in the Flash and load urgent services to the address of 0x80008000.

```
nand read 0x80008000 0x100000 0x4E0000; go 0x80008000;
```

Step 6 Restart Huawei LiteOS, and urgent services will be loaded first, and then non-urgent services will be automatically loaded. Huawei LiteOS automatically restarts and loads the urgent service images at the address of 0x80008000.

----End

4.2.3 Precautions

- The OS will be abnormal if the data copied in the first phase is not sufficient or the offset addresses are not aligned based on different storage media. Therefore, the size of images to be loaded by uboot needs to be the size returned when the compilation ends.
- The library file list to be extracted needs to be the superset that supports the running of the key services. Otherwise, the code used in the first phase of scatter loading will access the code or data that will be loaded into memory in the second phase, and the OS will be abnormal.
- During scatter loading, a variable value might be changed after the variable is run in the first phase, but after the variable is loaded and run in the second phase the value is changed into an uninitialized value. This problem occurs when the variable is used in the first phase but it is not put into the fast-booted segment together with other variables used in the first phase. The solution is to put this variable into the fast-booted segment and ensure that all data used in the first phase is in the fast-booted segment.

4.2.4 FAQs

This section describes problems encountered during using scatter loading and solutions.

- Lacking the .o file.

  ```
  arm-hisiv300-linux-ld: cannot find libscatter.O
   make: *** [vs_server] Error 1
  ```

  This problem occurs because the .o file is not generated after the link script is modified. The solution is to generate the .o file and save it in the object directory.

- Some symbols are not defined.

  ```
  /usr1/xxxxx/gerrit_code/modify-debug/liteos_ipc/out/lib/
  libar6003.a(ar6000_drv.o): In function `ar6000_avail_ev':
   /usr1/xxxxx/gerrit_code/modify-debug/liteos_ipc/vendor/ar6k3_wifi/AR6003/
  host/qca/source/ar6000_drv.c:1553: undefined reference to
  `wireless_init_event'
   /usr1/xxxxx/gerrit_code/modify-debug/liteos_ipc/out/lib/
  libar6003.a(drv_config.o): In function `ar6000_tkip_micerr_event':
   /usr1/xxxxx/gerrit_code/modify-debug/liteos_ipc/vendor/ar6k3_wifi/AR6003/
  host/qca/source/drv_config.c:1856: undefined reference to
  `wireless_send_event'
   make: *** [vs_server] Error 1
  ```

  This problem occurs because some useful .a files are removed when the link script is modified. Run the grep command to search for undefined variables under the out/lib directory. Add the .a files that contain these variables and are not in the link script to the link script.

Copyright © Huawei Technologies Co., Ltd.

- Instruction exception.

  If the PC position is beyond the range of files that are loaded in the first phase of scatter loading when the exception occurs, this problem occurs when the library file list used in the first phase does not cover all files to be loaded, and some symbols are not put into the code and data segments that are loaded in the first phase. Use the image disassembly file to locate the name of the function where the abnormal PC is located, find the library where this function belongs, and add the library to the library file list.

4.3 Exception Management

4.3.1 Overview

Basic Concept

Exception management is a set of actions taken to handle an exception. For example, when the operating system encounters an exception, it prints information about the CPU condition, task stacks, and function call stacks.

Exception management is a useful debugging approach. It provides the exception information required for fault diagnosis. The information includes the exception type and system state at the time of exception.

When an exception occurs, Huawei LiteOS displays the CPU condition and the task information including task name, task ID, and stack size.

Operation Mechanism

Stacks Analysis

- R11 is used as a general register or as a frame pointer (FP) register with backtrace stacks if specified compilation options are enabled.

  By default, R11 is used by the GNU Compiler Collection (GCC) as a general register with store variables and cannot backtrace stacks. To use R11 as an FP register with analyze call stacks, enable the -fno-omit-frame-pointer compilation option.
- An FP register can trace the sequence of functions called by a program.

  An FP register points to the stack backtracing structure of the current function. The returned value is the pointer to the stack backtracing structure established by the parent function that has called the current function. Others may be deduced by analogy. The calling relationships among functions can be traced using an FP register.

  If an exception occurs during the runtime, the operating system prints the contents of an FP register, helping you to diagnose the exception by backtracing the sequence of function calls.

  The following figure describes the stack analysis process.

Copyright © Huawei Technologies Co., Ltd.

Figure 4-3 Stack analysis process

Registers in different colors represent different functions. The FP register backtraces the parent function of the erroneous function to cast light on the sequence of function calls.

Relationships between Call Stacks

The preceding figure illustrates the working process of stack frame register. The function call sequence can be deduced by analyzing the stacks step by step. The procedure is as follows:

1. Obtain the value of the current FP register.
2. Subtract the current FP register by 4 bytes to get the current PC value. Check the execution condition of instructions in the system image by searching the system file (ELF file) or the image disassembly file (ASM file) based on the PC value.
3. Subtract the FP register by 24 bytes to get the start address of the call stack frame of the previous function. Subtract the FP register by 16 bytes to get the end SP address of the previous function. The stack between FP and SP is the call stack frame of the function.
4. The sequence of function calls can be obtained from the PC pointer of the stack frame at each layer.

Copyright © Huawei Technologies Co., Ltd.

4.3.2 Development Guidelines

Exception Types

Exception management provides following exception types:

Table 4-1

Exception Name	Description	Value
OS_EXCEPT_UNDEF_INSTR	Undefined instruction exception	1
OS_EXCEPT_SWI	Software interrupt exception	2
OS_EXCEPT_PREFETCH_ABORT	Instruction pre-fetch exception	3
OS_EXCEPT_DATA_ABORT	Data abort exception	4
OS_EXCEPT_FIQ	FIQ exception	5

Development Process

In general, exception management fault location process is as follows:

1. Open the .asm file that is generated after compilation.
2. Search the .asm file for the location of PC pointer.
3. Search for a called function by the value of LR.
4. Repeat step 3 to find the abnormal task function.

For details about the fault location process, see **Programming Example**.

4.3.3 Precautions

- Before querying information about call stacks, add the compilation option -fno-omit-frame-pointer. If the option is not added, stack frames are not supported and the FP register is disabled by default.

4.3.4 Programming Example

Example Description

The panic command triggers a software interrupt exception and the abnormal function is LOS_Panic. The test_panic code is used to trigger exceptions, and the Huawei LiteOS# panic code is used to print information about the abnormal call stack.

uwExcType 2 is the software interrupt exception.

The fault location process is as follows:

1. Open the .asm file that is generated by the compiler. Default is vs_server.asm
2. Search the .asm file for the location of PC pointer 80121234.
3. Search for a called function by the value of LR.

Copyright © Huawei Technologies Co., Ltd.

4. Determine that g_RunningTask is the task function at the time of exception.

```
UINT32 test_panic(UINT32 argc, CHAR **args)
 {
     LOS_Panic("*****Trigger an exception\n");
     return;
 }
Huawei LiteOS# panic
 *****Trigger an exception
 uwExcType = 2
 puwExcBuffAddr pc = 0x80121234
 puwExcBuffAddr lr = 0x80121234
 puwExcBuffAddr sp = 0x80e63400
 puwExcBuffAddr fp = 0x80e6340c
 *******backtrace begin*******
 traceback 0 -- lr = 0x80138d04
 traceback 0 -- fp = 0x80e635d4
 traceback 1 -- lr = 0x80138d88
 traceback 1 -- fp = 0x80e635e4
 traceback 2 -- lr = 0x801247d4
 traceback 2 -- fp = 0x80e635f4
 traceback 3 -- lr = 0x801217c4
 traceback 3 -- fp = 0x11111111
 R0         = 0x1c
 R1         = 0x800dba3a
 R2         = 0x1b
 R3         = 0xfe
 R4         = 0x80e634a0
 R5         = 0x0
 R6         = 0x800cc7f8
 R7         = 0x7070707
 R8         = 0x8080808
 R9         = 0x9090909
 R10        = 0x10101010
 R11        = 0x80e6340c
 R12        = 0x1b
 SP         = 0x80e63400
 LR         = 0x80121234
 PC         = 0x80121234
 CPSR       = 0x60000013
 g_pRunningTask->pcTaskName = shellTask
 g_pRunningTask->uwTaskPID = 6
 g_pRunningTask->uwStackSize = 12288
```

4.4 CPU Utilization Percentage

4.4.1 Overview

Basic Concept

CPU usage is classified into system CPU usage and task CPU usage.

System CPU usage refers to the percentage of CPU resources occupied by the operating system during the measurement period. It is an important way to quantify the workload of the operating system. System CPU usage ranges from 0% to 100%. The precision is represented as a percentage and can be adjusted. System CPU usage of 100% indicates the operating system is fully loaded.

Task CPU usage refers to the ratio of CPU resources occupied by a particular task during the measurement period. From task CPU usage, you can determine whether the task is busy or idle. Task CPU usage ranges from 0% to 100%. Task CPU usage of 100% indicates the operating system keeps the task running throughout the measurement period. Ratios can be expressed out of various bases, like out of 1,100,1000, etc.

Copyright © Huawei Technologies Co., Ltd.

System CPU usage is an important metric to determine whether the operating system is on the verge of overload.

Query the CPU usage of tasks to determine if they meet the CPU usage requirements you have laid out during the design phase.

Operation Mechanism

System CPU usage (CPU Percent, CPUP) is broken down into task CPU usage. Each task switch will generate a record of when the task was started and when it was exited. When the task is exited, the operating system measures the total runtime of the task.

You can configure the CPU usage control function in the kernel module of menuconfig.

Huawei LiteOS enables you to query the following CPU usage information:

- System CPU usage
- Task CPU usage

CPU usage measurement formula:

System CPU usage = runtime of tasks except idle task in the operating system/system total runtime

Task CPU usage = runtime of a particular task/system total runtime

4.4.2 Development Guidelines

Usage Scenarios

Query system CPU usage regularly to check whether the operating system is on the verge of overload.

Query thread CPU usage to learn each thread meets the CPU usage requirements you have laid out during the design phase.

Functions

The CPU usage module of Huawei LiteOS provides the following functions:

Function Category	API	Description
System CPU usage query	LOS_SysCpuUsage	Acquires current system CPU usage
	LOS_HistorySysCpuUsage	Acquires historical system CPU usage
Task CPU usage query	LOS_TaskCpuUsage	Acquires current CPU usage of a particular task
	LOS_HistoryTaskCpuUsage	Acquires historical CPU usage of a particular task
	LOS_AllTaskCpuUsage	Acquires CPU usage of all tasks

Copyright © Huawei Technologies Co., Ltd.

Function Category	API	Description
CPU usage reset	LOS_CpupReset	Reset data of CPU usage

Development Process

The typical CPU usage development process is as follows:

1. Call the LOS_SysCpuUsage API to get CPU usage now.
2. Call the LOS_HistorySysCpuUsage API to get system historical CPU usage.
   - Disables interrupts, acquires the task end time, measures the historical CPU usage of the current task.
3. Call the LOS_TaskCpuUsage API to get particular task CPU usage.
   - If a particular task is ready, the operating system disables interrupts, acquires the task end time, and measures the CPU usage of the task.
   - If the task has not been created or is not ready, the operating system returns an error code.
4. Call the LOS_HistoryTaskCpuUsage API to get historical CPU usage of the particular task.
   - If a particular task is ready, the operating system disables interrupts, acquires the task end time, and measures the historical CPU usage of the task.
   - If the task has not been created or is not ready, the operating system returns an error code.
5. Call the LOS_AllTaskCpuUsage API to get historical CPU usage of all tasks.
   - If CPUP is initialized, interrupt will be breaking off. Acquired information according to different modules, and then interrupt recovery.
   - If CPUP is not initialized or it has illegal parameters insert, returning error code.

Platform Differences

None.

4.4.3 Precautions

- Only product designers need to learn CPU usage of each task. To prevent measurement of thread CPU usage from adversely impacting system performance, set LOSCFG_KERNEL_CPUP to NO before releasing your product.
- The value returned by the interface above is permillage. This value can be divided by LOS_CPUP_PRECISION_MULT to obtain the corresponding percentage.

4.4.4 Programming Example

Example Description

In the programming example, the following steps will be performed:

1. Create a CPUP test task.

Copyright © Huawei Technologies Co., Ltd.

2. Acquire current system CPUP.
3. Acquire historical CPUP of the operating system.
4. Acquire the CPUP of the CPUP test task.
5. Acquire the CPUP of the CPUP test task in different modes.

Example Code

Prerequisite

- The OS_INCLUDE_CPUP parameter in the los_config.h is set to YES.

The code is as follows:

```
#include "los_task.h"
#include "los_cpup.h"

#define  MODE  4

 UINT32 cpupUse;

 UINT16 pusMaxNum = 0;
 UINT32 g_CpuTestTaskID;

 VOID Example_cpup(VOID)
 {

     printf("entry cpup test example\n");
     while(1) {
         usleep(100);
     }
 }

 UINT32 it_cpup_test(VOID)
 {
     UINT32 uwRet;
     TSK_INIT_PARAM_S CpupTestTask;

     /*Create a CPUP test task.*/
     memset(&CpupTestTask, 0, sizeof(TSK_INIT_PARAM_S));
     CpupTestTask.pfnTaskEntry = (TSK_ENTRY_FUNC)Example_cpup;
     CpupTestTask.pcName       = "TestCpupTsk";    /*Test task name*/
     CpupTestTask.uwStackSize  = 0x800;
     CpupTestTask.usTaskPrio = 5;
     CpupTestTask.uwResved = LOS_TASK_STATUS_DETACHED;
     uwRet = LOS_TaskCreate(&g_CpuTestTaskID, &CpupTestTask);
     if(uwRet != LOS_OK)
     {
         printf("CpupTestTask create failed .\n");
         return LOS_NOK;
     }
          usleep(100);

     /*Acquire current system CPU usage*/
     cpupUse = LOS_SysCpuUsage();
     printf("the current system cpu usage is: %d\n",cpupUse);

     /*Acquire historical CPU usage of the operating system within the 1-second
measurement period. There are three types of measurement period: 10s, 1s, and
less than 1s.*/
     //cpupUse = LOS_HistorySysCpuUsage(MODE1);
     //printf("the history system cpu usage in 10s: %d\n",cpupUse);

     //cpupUse = LOS_HistorySysCpuUsage(MODE2);
     //printf("the history system cpu usage in 1s:  %d\n",cpupUse);

     cpupUse = LOS_HistorySysCpuUsage(MODE);
```


Copyright © Huawei Technologies Co., Ltd.

```
    printf("the history system cpu usage in <1s: %d\n",cpupUse);

    /*Acquire CPU usage of a particular task (CPUP test task in the programming
example).*/
    cpupUse = LOS_TaskCpuUsage(g_CpuTestTaskID);
    printf("cpu usage of the CpupTestTask:\n TaskID: %d\n usage: %d
\n",g_CpuTestTaskID,cpupUse);

   /*Acquire historical CPU usage of a particular task (CPUP test task in the
programming example) within the measurement period of less than 1 second.*/
    cpupUse = LOS_HistoryTaskCpuUsage(g_CpuTestTaskID, MODE);
    printf("cpu usage of the CpupTestTask in <1s:\n TaskID: %d\n usage:%d
\n",g_CpuTestTaskID,cpupUse);
    return LOS_OK;
 }
```

Verification

The verification result is as follows:

```
--- Test start---
entry cpup test example

Huawei LiteOS# the current system cpu usage is :       15
the history system cpu usage in <1s:     3
cpu usage of the CpupTestTask:
TaskID:10
usage:0
cpu usage of the CpupTestTask in <1s:
TaskID:10
usage:0

 ---Test End ---
```

Complete Code

sample_cpup.c

4.5 Linux Adaption

4.5.1 Completion

4.5.1.1 Overview

Basic Concept

Completion is a supplement to semaphore. It is a lightweight mechanism for task synchronization. In Linux, up() and down() functions can be executed concurrently in the same semaphore and in the multi-CPU environment, up() may mistakenly access a semaphore data structure that does not exist. Completion is designed to prevent up() from accessing a non-existent semaphore data structure.

In the scenario in which task A can be executed only after task B has completed a particular operation, the completion enables task B to wake up task A at completion of the particular operation, thereby achieving task synchronization.

In the multi-task environment, tasks need to be synchronized with each other. The completion mechanism can well serve the purpose.

Copyright © Huawei Technologies Co., Ltd.

The realization of the completion mechanism of Huawei LiteOS is similar to the signal mechanism in system. By calling the kernel function to achieve the function of the completion mechanism. The completion mechanism of Huawei LiteOS has characters similar to signal mechanism.

4.5.1.2 Development Guidelines

Usage Scenarios

Use the completion mechanism in the multi-task environment to synchronize one task with another.

At the core of the completion mechanism is the wait for a completion and task wakeup.

Functions

Function Category	API	Description
Initialize a completion	init_completion	Initializes a completion
Wait for a completion	wait_for_completion	Waits for a completion until the completion occurs
Wait for a completion in timeout mode	wait_for_completion_timeout	Waits for a completion for a specified number of ticks
Wake a completion	complete	Wakes up the first task waiting for a completion
Wake a completion	complete_all	Wakes up all tasks waiting for a completion

Development Process

The typical development process of the completion mechanism is as follows:

1. Call the init_completion API to initialize a completion structure.
   - Create a completion
2. Call the wait_for_completion_timeout API to wait for a completion in timeout mode.
3. Call the wait_for_completion API to wait for a completion until the completion occurs.
4. Call the complite/complite_all API to wake a completion.
   - complete: Wakes up a task waiting for the completion
   - complete_all: Wakes up all tasks waiting for the completion

4.5.1.3 Precautions

- The completion mechanism is similar to the semaphore mechanism. Permanent blocking and timed blocking are not allowed while an interrupt is underway, since interrupts cannot be blocked.
- The input parameter to the completion APIs must be a valid completion pointer. For example, a task is not allowed to wait for a completion while an interrupt is underway.

Copyright © Huawei Technologies Co., Ltd.

4.5.1.4 Programming Example

Example Description

In the programming example, the Example_TaskEntry task is executed to create the Example_Completion task. The Example_Completion task is blocked while waiting for a completion. Then, the Example_TaskEntry task wakes up the completion. Based on the information printed on the screen, you can learn the task switching that occurs along with the completion operation.

1. The Example_TaskEntry task is executed to create the Example_Completion task. The Example_Completion task takes a higher priority than the Example_TaskEntry task.
2. The Example_Completion task is blocked while waiting for the completion. After the Example_Completion task is blocked, a task switch occurs and the task with a lower priority, namely, the Example_TaskEntry task will be executed.
3. The Example_TaskEntry task wakes up the completion. Then, a task switch occurs and the Example_Completion task will be executed.
4. The Example_Completion task is executed.
5. The Example_TaskEntry task is executed.

Example Code

The code is as follows:

```
##include "linux/completion.h"
 #include "los_task.h"
 //#include "osTest.h"

 /*Task PID*/
 UINT32 g_TestTaskID01;

 /*Completion*/
 struct completion example_completion;

 /*Example task entrypoint function*/
 VOID Example_Completion()
 {
     UINT32 uwRet;

     /*Wait for a completion in timeout mode, and the timeout interval is 100
ticks*/
     printf("Example_Completion wait completion\n");
     uwRet = wait_for_completion_timeout(&example_completion,100);

     if(uwRet == 0)
     {
         printf("Example_Completion,wait completion timeout\n");
     }
     else
         printf("Example_Completion,wait completion success\n");
     return;
 }

 UINT32 Example_TaskEntry()
 {
     UINT32 uwRet;
     TSK_INIT_PARAM_S stTask1;

     /*Initialize completion*/
     init_completion(&example_completion);
```


Copyright © Huawei Technologies Co., Ltd.

```
    /*Create a task*/
    memset(&stTask1, 0, sizeof(TSK_INIT_PARAM_S));
    stTask1.pfnTaskEntry = (TSK_ENTRY_FUNC)Example_Completion;
    stTask1.pcName       = "EventTsk1";
    stTask1.uwStackSize  = OS_TSK_DEFAULT_STACK_SIZE;
    stTask1.usTaskPrio = 8;
    uwRet = LOS_TaskCreate(&g_TestTaskID01, &stTask1);
    if(uwRet != LOS_OK)
    {
        printf("task create failed \n");
        return LOS_NOK;
    }

    /*Wake up completion*/
    printf("Example_TaskEntry complete\n");

    complete(&example_completion);

    printf("Delete Task.\n");
    /*Delete a task*/
    uwRet = LOS_TaskDelete(g_TestTaskID01);
    if(uwRet != LOS_OK)
    {
        printf("task delete failed \n");
        return LOS_NOK;
    }

    return LOS_OK;
}
```

Verification

The verification result is as follows:

```
Example_Completion wait completion
 Example_TaskEntry complete
 Example_Completion,wait completion success
 Delete Task.
```

Complete Code

sample_completion.c

4.5.2 Workqueue

4.5.2.1 Overview

Basic Concept

Any system modules can place works into a workqueue. Works are processed by the workqueue (a type of tasks) in the first in first out (FIFO) order. The workqueue processing task can be re-scheduled or sent to sleep mode.

Works in a workqueue can be processed by a single task, freeing you from the cumbersome burden of repeatedly creating a task for each work. Moreover, the workqueue processing task can be re-scheduled or sent to sleep mode instead of being kept running, which reduces the demand for system resources significantly.

The workqueue mechanism can process works using a single task and provides abundant external APIs for managing a workqueue. When a work is added to a workqueue, the

Copyright © Huawei Technologies Co., Ltd.

workqueue processing task is woken up to process the work. After all works in the workqueue are processed, the workqueue processing task is sent to sleep mode.

4.5.2.2 Development Guidelines

Usage Scenarios

A workqueue is applicable only in delay-tolerant scenarios because works in the workqueue can be processed only after the workqueue processing task is executed. Many factors are considered while determining whether the workqueue processing task can be woken up or scheduled. For example, the workqueue processing task may be blocked because another task with a higher priority is waiting to be executed or because an interrupt is underway.

Functions

Function Category	API	Description
Creating a workqueue	create_workqueue	Creates a workqueue
Destroying a workqueue	destroy_workqueue	Destroys a workqueue
Initiating a work	INIT_WORK	Binds a work to the processing function
Initiating a delayed work	INIT_DELAYED_WORK	Binds a delayed work to the processing function
Placing a work in a workqueue	queue_work	Places a work in a specified workqueue
Placing a delayed work in a workqueue	queue_delayed_work	Places a delayed work in a specified workqueue
Placing a work in a workqueue	schedule_work	Places a work in the default workqueue
Placing a delayed work in a workqueue	schedule_delayed_work	Places a delayed work in the default workqueue
Querying work status	work_busy	Queries the current state of a work
Processing a delayed work	flush_delayed_work	Takes the work out of delay and processes it immediately
Canceling a delayed work	cancel_delayed_work	Cancels a delayed work that is not executed
Processing a work	flush_work	Executes a work immediately
Canceling a work	cancel_work_sync	Cancels a work after the work is executed

Copyright © Huawei Technologies Co., Ltd.

Development Process

The typical workqueue development process is as follows:

1. Call the create_workqueue API to create the workqueue processing task.
   - Huawei LiteOS initializes a semaphore used for waking up the workqueue processing task or sending it to sleep mode; initializes important structures; builds a workqueue linked list.
2. Call the queue_work/queue_delayed_work API to place a work (either normal or delayed work) in a specified workqueue.
   - A normal work can be added to the workqueue immediately, whereas a delayed work has to wait for a period of time specified by a delay parameter before being added to the workqueue.
3. Call the cancel_delayed_work/cancel_work_sync API to cancel the normal or delayed work from the workqueue without processing it.
4. Call the flush_work/flush_delayed_work API to process the work in the workqueue immediately.
5. Call the destroy_workqueue API to delete the workqueue.
   - Huawei LiteOS locks resources, deletes the workqueue processing task, releases the semaphore and memory, and unlocks resources.

4.5.2.3 Precautions

- Do not use a workqueue in delay-sensitive scenarios because there is much uncertainty in the time when a task can be scheduled to process a work in a workqueue.
- Workqueues are identified by their names. Therefore, each workqueue must have a unique name.
- The schedule_work or schedule_delayed_work API can be used to place a work in the default workqueue without first creating a workqueue.

4.5.2.4 Programming Example

Example Description

The programming example will cover the following functions:

1. Creating a workqueue named wq_test
2. Allocating and initializing the work memory
3. Placing a work in the work queue
4. Processing a work immediately
5. Destroy the workqueue

Example Code

The code is as follows:

```
#include "los_config.h"
 #include  "linux/workqueue.h"

 static void work_func(struct work_struct *work)
 {
```


Copyright © Huawei Technologies Co., Ltd.

```
   int i;
   for (i = 0; i < 2; i++)
   {
     printk("workqueue function is been called!..%d..%d..\n",i,work-
>work_status);
   }
 }

 UINT32 It_workqueue_1008()
 {
   struct workqueue_struct *wq;
   struct work_struct *work;
   UINT32 uwRet = LOS_OK;

   wq = create_workqueue("test1008");
   dprintf("create the workqueue successfully!\n\n");

   work =(struct work_struct *)malloc(sizeof(struct work_struct));
   if (!work)
   {
      uwRet = LOS_FAIL;
   }
   dprintf("create work ok!\n\n");

   INIT_WORK(work, work_func);
   dprintf("init the work ok!\n\n");

   uwRet = queue_work(wq, work);
   dprintf("mount the work into workqueue successfully!\n\n");

   uwRet = flush_work(work);
   dprintf("flush the work ok!\n\n");

   destroy_workqueue(wq);
   dprintf("destroy the work ok!\n\n");

   return uwRet;
 }
```

Verification

The verification result is as follows:

```
 --- Test start---
create the workqueue successfully!

create work ok!

init the work ok!

mount the work into workqueue successfully!

workqueue function is been called!..0..3..
workqueue function is been called!..1..3..
flush the work ok!

destroy the work ok!

  [Passed] It_workqueue_1008

 --- Test End ---
```

Complete Code

sample_workqueue.c

Copyright © Huawei Technologies Co., Ltd.

4.5.3 Interrupt

4.5.3.1 Overview

Basic Concept

Linux kernel has APIs specially used for interrupts. The interrupt mechanism of Huawei LiteOS adapts to the interrupt-related Linux APIs, making Huawei LiteOS more user-friendly.

Huawei LiteOS has the following interrupt-related functions:

- Requesting an interrupt
- Deleting an interrupt
- Enabling an interrupt
- Masking an interrupt
- Interrupt bottom half (based on workqueues)

4.5.3.2 Development Guidelines

Functions

The following table lists Linux APIs to which the interrupt module in Huawei LiteOS adapts.

API	Description
request_irq	Requests an interrupt
free_irq	Frees an interrupt
enable_irq	Enables an interrupt
disable_irq	Disables an interrupt
irq_bottom_half	Places the work of interrupt bottom half in the workqueue

Development Process

1. Call the request_irq API to request an interrupt.

   Registers the interrupt handler into the linked list of the interrupt ID based on the input parameter. Each interrupt ID is allowed to register multiple interrupt handlers.
2. Call the irq_bottom_half to place the interrupt bottom half in the workqueue.

   Associates the interrupt bottom half with the time-consuming yet less important work and places the work of interrupt bottom half in the workqueue. The operating system executes the interrupt bottom half when idle.
3. Call the enable_irq API to enable an interrupt.
4. Call the disable_irq API to disable an interrupt.
5. Call the free_irq to free an interrupt.

Copyright © Huawei Technologies Co., Ltd.

4.5.3.3 Precautions

- While calling the request_irq() API, ensure that the input parameter of the interrupt handler is in the (int, void*) format. While calling the LOS_HwiCreate() API, note that the input parameter of the interrupt handler can be NULL.
- An interrupt handler is not allowed to call the request_irq() and free_irq() APIs.
- If an interrupt ID is shared among external peripherals, interrupts with the interrupt ID cannot be created using the LOS_HwiCreate() API and the input parameter dev pointer of the request_irq() API must match the interrupt handler instead of being NULL. Interrupts created using the LOS_HwiCreate() API cannot be deleted using the free_irq() API.
- The work_queue pointer input by the irq_bottom_half() function cannot be an invalid pointer.
- The input parameter of the interrupt bottom half is a work pointer. The work is dynamically requested during the call to the irq_bottom_half() and must be freed while the interrupt bottom half is being executed. Otherwise, a memory leak occurs.

4.5.3.4 Programming Example

Example Description

The programming example will cover the following functions:

1. Requesting an interrupt
2. Freeing an interrupt

Example Code

Prerequisite

- The OS_INCLUDE_HWI parameter in the los_config.h file is set to YES.
- The OS_HWI_MAX_USED_NUM parameter in the los_config.h file is set to the maximum number of hardware interrupts the operating system allows.

The code is as follows:

```
#include "los_hwi.h"
 #define HWI_NUM_INT50 50

 void uart_irqhandle_1(int irq,void *dev)
 {
     printf("\nuart0:the function1 \n");
 }

 void hwi_test()
 {
     int a = 1;
     void *dev = &a;
     unsigned long flags = 0;
     const char * name = "hwiTest";
     request_irq(HWI_NUM_INT50,uart_irqhandle_1,flags,name,dev);//Create an
interrupt

     free_irq(HWI_NUM_INT50,dev);//Delete the interrupt
 }
```


Copyright © Huawei Technologies Co., Ltd.

Complete Code

sample_irq.c

4.5.4 High Resolution Timer

4.5.4.1 Overview

Basic Concept

Using hardware timer resources and algorithms, high resolution timers satisfy the requirement for precise time.

A high resolution timer in Huawei LiteOS is designed with a software architecture and provides microsecond timing resolution, satisfying applications and kernel drivers' urgent need for precise time.

4.5.4.2 Development Guidelines

Usage Scenarios

The kernel's software timers provide millisecond timing resolution, which is not high and not able to meet the high resolution time requirements in some scenarios. High resolution timers provide timing APIs of microsecond resolution to satisfy preceding-mentioned scenarios.

Functions

Function Category	API	Description
High resolution timer initialization	hrtimer_init	Initializes the resources for a high resolution timer. This function has been implemented during kernel initialization. This API is only an adapted Linux API and is not implemented in Huawei LiteOS.
High resolution timer creation	hrtimer_create	Sets the timing period and creates the callback function.
High resolution timer start	hrtimer_start	Starts a high resolution timer.
High resolution timer cancellation	hrtimer_cancel	Cancels the high resolution timers that have not expired.
Timing period change	hrtimer_forward	Changes the timing periods of a high resolution timer that has not expired.

Copyright © Huawei Technologies Co., Ltd.

Function Category	API	Description
High resolution timer query	hrtimer_is_queued	Checks whether a high resolution timer is created. **true**: The specified high resolution timer is created. **false**: The high resolution timer is not created.

Development Process

The typical process of using a high resolution timer is as follows:

1. Declare a hrtimer variable and a ktime variable
2. Set the values of the **sec** and **usec** fields for the ktime variable according to the timing period.
3. Call the hrtimer_create API to set the timing period and create the callback function.
4. Call the hrtimer_start API to start the high resolution timer.
5. The hrtimer_cancel and hrtimer_forward APIs are provided as extended functions. Use them as required.
6. The hrtimer_is_queued API helps you find out whether a high resolution timer is created or expires.

4.5.4.3 Precautions

- The hrtimer variable required by high resolution timer APIs should be provided by users. The APIs do not request any resources for the variable.
- The hrtimer_init API is not implemented and is only an adapted Linux API. Therefore, this API does not need to be called when using high resolution timers.
- Currently, high resolution timers in Huawei LiteOS support one-off timing mode rather than periodic timing mode. The periodic timing mode can be achieved by recreating a high resolution timer.

4.5.4.4 Programming Example

Example Description

The programming example will cover the following functions:

1. Users need to request the *pstSwtmr0* and *time0* variables.
2. The hrtimer_create API is called to set the timing period and create the callback function.
3. The hrtimer_start API is called to start a high resolution timer.
4. The hrtimer_forward API is called to change the timing period from 80 ms to 40 ms.
5. The LOS_TaskDelay API is called to delay a task for 50 ms.
6. The hrtimer_cancel is called to cancel a timer. Then the timer is found to have expired and no longer exists.

Copyright © Huawei Technologies Co., Ltd.

Example Code

The code is as follows:

```
#include "hrtimer.h"

static enum hrtimer_restart hrtimer_func(struct hrtimer *arg)
{
    dprintf("The hrtimer is timeout!!!\n");
}

static UINT32 testcase(VOID)
{
    struct hrtimer pstSwtmr0;
    struct ktime time0;
    struct ktime interval = {0};
    interval.tv.sec = 0;
    interval.tv.usec = 40000;
    int ret;

    dprintf("----->test start<-----\n");

    time0.tv.sec =0;
    time0.tv.usec = 80000;
    ret = hrtimer_create(&pstSwtmr0, time0, hrtimer_func);
    if (ret == 0)
        dprintf("Hrtimer ctreate successfully!\n");

    ret = hrtimer_start(&pstSwtmr0,time0,MODE_ONCE);
    if (ret == 0)
        dprintf("Hrtimer start successfully!\n");

    ret = hrtimer_forward(&pstSwtmr0,  interval);

    LOS_TaskDelay(5);

    ret = hrtimer_cancel(&pstSwtmr0);
    if (ret == 0)
        dprintf("Hrtimer already timeout!\n");

    dprintf("----->test end<-----\n");

    return LOS_OK;
}
```

Verification

The verification result is as follows:

```
----->test start<-----
Hrtimer ctreate successfully!
Hrtimer start successfully!
The hrtimer is timeout!!!
Hrtimer already timeout!
----->test end<-----
```

Complete Code

sample_hrtimer.c

4.5.5 Linux APIs

Copyright © Huawei Technologies Co., Ltd.

4.5.5.1 Linux Adaption APIs

Huawei LiteOS adapts to the following Linux APIs:

NOTE

"Compatible" indicates that the functions of the API are inherited from Linux, but the error code returned by the API depends on Huawei's actual code. "Partially compatible" indicates that some functions of the API are inherited from Linux.

Header File	API	Function	Compatibility
timer.h	add_timer	Add a timer.	Compatible
atomic.h	atomic_add	Add an integer to an atomic variable.	Compatible
atomic.h	atomic_add_return	Add an integer to an atomic variable and return the new variable value.	Compatible
atomic.h	atomic_dec	Decrement an atomic variable by one.	Compatible
atomic.h	atomic_dec_and_test	Decrement an atomic variable by one and test whether the result is zero.	Compatible
atomic.h	atomic_dec_return	Decrement an atomic variable by one and return the new variable value.	Compatible
atomic.h	atomic_inc	Increment an atomic variable by one.	Compatible
atomic.h	atomic_inc_return	Increment an atomic variable by one and return the new variable value.	Compatible
atomic.h	atomic_read	Read the value of an atomic variable.	Compatible
atomic.h	atomic_sub	Subtract an integer from an atomic variable.	Compatible

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Function	Compatibility
string.h	bzero	Set the first n bytes of a string to zero, including \0.	Compatible
workqueue.h	cancel_delayed_work	Cancel a pending delayed work.	Compatible
workqueue.h	cancel_delayed_work_sync	Cancel a pending delayed work and wait for its execution to finish.	Compatible
workqueue.h	cancel_work_sync	Cancel a work and wait for its execution to finish.	Compatible
completion.h	complete	Wake up a thread waiting on a completion.	Compatible
completion.h	complete_all	Wake up all threads waiting on a completion.	Compatible
crc32.h	crc32	Calculate crc.	Compatible
crc32.h	crc32_accumulate	Calculate crc.	Compatible
workqueue.h	create_singlethread_workqueue	Create a workqueue.	Compatible
workqueue.h	create_workqueue	Create a workqueue.	Compatible
checksum.h	csum_partial	Data check.	Compatible
timer.h	del_timer	Delete a timer.	Compatible
timer.h	del_timer_sync	Delete a timer.	Compatible
workqueue.h	destroy_workqueue	Destroy a workqueue.	Compatible
interrupt.h	disable_irq	Disable an interrupt.	Compatible
kernel.h	div64_u64	Do a division.	Compatible

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Function	Compatibility
kernel.h	div64_u64_rem	Do a division.	Compatible
kernel.h	div64_s64	Do a division.	Compatible
kernel.h	div_u64	Do a division.	Compatible
kernel.h	div_u64_rem	Do a division.	Compatible
kernel.h	div_s64	Do a division.	Compatible
kernel.h	div_s64_rem	Do a division.	Compatible
dlfcn.h	dlclose	Close a dynamic linking library that has a specified handle.	Compatible
dlfcn.h	dlopen	Open a dynamic linking library and return a handle for the dynamic loading library.	Compatible
dlfcn.h	dlsym	Take the handle and symbol for the dynamic linking library and return the address of the symbol.	Compatible
kernel.h	do_div_imp	Do a division.	Compatible
kernel.h	do_div_s64_imp	Do a division.	Compatible
rwsem.h	down_read	Hold a semaphore for reading.	Compatible
rwsem.h	down_read_trylock	Hold a semaphore for reading.	Compatible
rwsem.h	down_write	Hold a semaphore for writing.	Compatible
rwsem.h	down_write_trylock	Hold a semaphore for writing.	Compatible
tzdst.h	dst_enable	Enable DST.	This API is a customized API of Huawei LiteOS.

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Function	Compatibility
tzdst.h	dst_disable	Disable DST.	This API is a customized API of Huawei LiteOS.
tzdst.h	dst_inquire	Check DST.	This API is a customized API of Huawei LiteOS.
interrupt.h	enable_irq	Enable an interrupt.	Compatible
kernel.h	ERR_PTR	Return an error code.	Compatible
fs.h	fb_alloc_cmap	Allocate memory for a color map.	Compatible
fs.h	fb_cmap_to_user	Copy a colormap.	Compatible
fs.h	fb_copy_cmap	Copy a colormap.	Compatible
fs.h	fb_dealloc_cmap	Deallocate a color map that was previously allocated.	Compatible
fs.h	fb_default_cmap	Set the default colormap.	Compatible
fs.h	fb_pan_display	Refresh the operation screen.	Compatible
fs.h	fb_set_cmap	Set a colormap.	Compatible
fs.h	fb_set_user_cmap	Set a colormap.	Compatible
fs.h	fb_set_var	Set the display mode of fbinfo and the variadic arguments.	Compatible
workqueue.h	flush_delayed_work	Wait for a delayed work to finish executing.	Compatible
workqueue.h	flush_work	Wait for a work to finish executing.	Compatible
fs.h	framebuffer_alloc	Apply to the kernel for space.	Compatible

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Function	Compatibility
fs.h	framebuffer_release	Release space to the kernel.	Compatible
interrupt.h	free_irq	Free an interrupt.	Compatible
kernel.h	hi_sched_clock	Return time.	Compatible
completion.h	init_completion	Initialize a completion.	Compatible
workqueue.h	INIT_DELAYED_WORK	Initialize a delayed work.	Compatible
list.h	INIT_LIST_HEAD	Initialize a linked list.	Compatible
timer.h	init_timer	Initialize a timer.	Compatible
wait.h	init_waitqueue_head	Initialize an existing wait queue head.	Compatible
workqueue.h	INIT_WORK	Initialize a work.	Compatible
interrupt.h	irq_bottom_half	Interrupt bottom half API.	This API is available in Huawei LiteOS but is unavailable in Linux.
kernel.h	IS_ERR	Test whether a returned pointer is an error code.	Compatible
rtc.h	is_leap_year	Determine whether a specified year is a leap year.	Compatible
jiffies.h	jiffies_to_msecs	Convert the number of ticks that have elapsed since kernel boot to milliseconds.	Compatible
kernel.h	jiffies_to_tick	Convert jiffies to ticks.	Compatible
slab.h	kfree	Free memory.	Compatible
slab.h	kmalloc	Allocate memory.	Compatible

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Function	Compatibility
slab.h	kzalloc	Allocate memory and initialize the memory.	Compatible
list.h	list_add	Add a linked list.	Compatible
list.h	list_add_tail	Add a linked list.	Compatible
list.h	list_del	Deleting a linked list.	Compatible
list.h	list_entry	Return a pointer to a structure.	Compatible
list.h	list_first_entry	Acquire the first element from a linked list.	Compatible
list.h	list_for_each	Traverse a linked list.	Compatible
list.h	list_for_each_entry	Traverse a linked list.	Compatible
list.h	list_for_each_entry_reverse	Traverse a linked list.	Compatible
list.h	list_for_each_entry_safe	Traverse a linked list.	Compatible
list.h	list_for_each_safe	Traverse a linked list.	Compatible
list.h	LIST_HEAD	Initialize a linked list.	Compatible
list.h	LIST_HEAD_INIT	Initialize a linked list.	Compatible
list.h	list_is_last	Test whether an element is the last element in a linked list.	Compatible
list.h	list_move	Move an element from a linked list to another linked list.	Compatible

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Function	Compatibility
string.h	memchr	Search a memory area for a character.	Compatible
string.h	memcmp	Compare the first n bytes of two memory areas.	Compatible
string.h	memcpy	Copy content from a memory area to a destination memory area. The memory areas must not overlap.	Compatible
string.h	memmove	Copy content from a memory area to a destination memory area. The memory areas may overlap.	Compatible
string.h	memset	Set the n bytes of a memory area to a specific value.	Compatible
timer.h	mod_timer	Add a timer.	Compatible
mount.h	mount	Mount a specified system partition to a folder.	Partially compatible
adp.c	msecs_to_jiffies	Convert the number of milliseconds that have elapsed since kernel boot to ticks.	Partially compatible
delay.h	msleep	Cause a program sleep for some milliseconds.	Compatible

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Function	Compatibility
prctl.h	prctl	Specify operations on a process. In Linux, this API supports multiple parameters, whereas in Huawei LiteOS, only PR_SET_NAME can be set to the thread name. The thread name in LiteOS is a pointer without allocating space. When setting the thread name, the local variable address is prohibited as a function of parameters. When creating a thread by using the POSIX interface, you are advised to set the thread name while entering the thread.	Partially compatible
kernel.h	PTR_ERR	Return an error code.	Compatible
workqueue.h	queue_delayed_work	Add a delayed work to a specified workqueue.	Compatible
workqueue.h	queue_work	Add a work to a specified workqueue.	Compatible
rbtree.h	rb_erase	Remove a node.	Compatible
rbtree.h	rb_first	Return the first node.	Compatible

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Function	Compatibility
rbtree.h	rb_insert_color	Color the inserted node.	Compatible
rbtree.h	rb_next	Return the next node.	Compatible
rbtree.h	rb_prev	Return the previous node.	Compatible
rbtree.h	rb_replace_node	Replace a node.	Compatible
io.h	readb	Read one byte from I/O.	Compatible
io.h	readl	Read four bytes from I/O.	Compatible
uio.h	readv	Read the input data into buffers following the input order.	Compatible
io.h	readw	Read two bytes from I/O.	Compatible
fs.h	register_framebuffer	Register fbinfo into the kernel.	Compatible
interrupt.h	request_irq	Allocate an interrupt.	Compatible
rtc.h	rtc_time_to_tm	Convert absolute time to year, month, day, hour, minute, and second.	Compatible
rtc.h	rtc_tm_to_time	Convert year, month, day, hour, minute, and second to absolute time.	Compatible
workqueue.h	schedule_delayed_work	Add a delayed work to the default workqueue.	Compatible
kernel.h	schedule_timeout	Schedule a task.	Compatible
kernel.h	schedule_timeout_interruptible	Schedule a task.	Compatible

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Function	Compatibility
kernel.h	schedule_timeout_uninterruptible	Schedule a task.	Compatible
workqueue.h	schedule_work	Add a work to the default workqueue.	Compatible
seq_file.h	seq_lseek	Deviate the pointer to a sequential file.	Compatible
seq_file.h	seq_open	Open a sequential file.	Compatible
seq_file.h	seq_printf	Write formatted information to a sequential file.	Compatible
seq_file.h	seq_read	Read a sequential file.	Compatible
seq_file.h	seq_release	Free the dynamic memory allocated by sequential files.	Compatible
tzdst.h	settimezone	Set a time zone.	This API is a customized API of Huawei LiteOS.
scatterlist.h	sg_init_one	Initialize a scatter list.	Compatible
scatterlist.h	sg_init_table	Initialize a scatter list.	Compatible
scatterlist.h	sg_mark_end	Mark the end of a scatter list.	Compatible
scatterlist.h	sg_set_buf	Set a scatter list.	Compatible
seq_file.h	single_open	Opens a sequential file.	Compatible
seq_file.h	single_release	Free the dynamic memory allocated by sequential files.	Compatible
string.h	strcasecmp	Compare two strings, ignoring the case of the characters.	Compatible

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Function	Compatibility
string.h	strcasestr	Locate the first occurrence of a substring in a string, ignoring the case of the strings.	Compatible
string.h	strcat	Append string A to string B.	Compatible
string.h	strchr	Locate the first occurrence of a character in a string.	Compatible
string.h	strcmp	Compare two strings.	Compatible
string.h	strcoll	Compare two strings in a specific locale.	Compatible
string.h	strcpy	Copy a string.	Compatible
string.h	strcspn	Return the number of bytes in the initial segment of a string, which does not consist of a specified character.	Compatible
string.h	strdup	Copy a string to a new location.	Compatible
string.h	strerror	Return information about system errors or user program errors.	Compatible
string.h	strlcpy	Copy a string of specified length.	Compatible
string.h	strlcpy	Copy a string.	Compatible
string.h	strlen	Calculate the length of a string.	Compatible

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Function	Compatibility
string.h	strncasecmp	Compare the first n bytes of two strings, ignoring the case of the characters.	Compatible
string.h	strncat	Append n bytes from string A to string B.	Compatible
string.h	strncmp	Compare the first n bytes of two strings.	Compatible
string.h	strncpy	Copy a string containing specified bytes.	Compatible
string.h	strpbrk	Locate the first occurrence in a string of any of the bytes in another string.	Compatible
string.h	strrchr	Locate the last occurrence of a character in a string.	Compatible
string.h	strsep	Separate a string into a set of strings.	Compatible
string.h	strspn	Return the subscript of the first character in a string that is not contained in a specified string.	Compatible
string.h	strstr	Locate the first occurrence of a substring in a string.	Compatible
string.h	strtok	Parse a string.	Compatible
string.h	strtok_r	Parse a string.	Compatible
string.h	strtoul	Convert a string to an unsigned long integer.	Compatible

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Function	Compatibility
string.h	strxfrm	String transformation.	Compatible
mount.h	umount	Unmount a file system.	Compatible
fs.h	unregister_frame buffer	Release fbinfo.	Compatible
rwsem.h	up_read	Release a semaphore held for reading.	Compatible
rwsem.h	up_write	Release a semaphore held for writing.	Compatible
slab.h	vfree	Free memory.	Compatible
slab.h	vmalloc	Allocate memory.	Compatible
wait.h	wait_event	Wait for an event.	Compatible
wait.h	wait_event_interr uptible	Wait for an event.	Compatible
wait.h	wait_event_interr uptible_timeout	Wait for an event for a limited period of time.	Compatible
completion.h	wait_for_comple tion	Wait for completion.	Compatible
completion.h	wait_for_comple tion_timeout	Wait for completion for a specified period of time. If the time expires, the waiting ends.	Compatible
wait.h	waitqueue_active	Test whether a wait queue is empty.	Compatible
wait.h	wake_up	Wake up a task.	Compatible
wait.h	wake_up_interru ptible	Wake up a task that has been put to interruptible sleep.	Compatible

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Function	Compatibility
workqueue.h	work_busy	Test the status of a work.	Compatible
io.h	writeb	Write one byte to I/O.	Compatible
io.h	writel	Write four bytes to I/O.	Compatible
uio.h	writev	Store data in multiple non-contiguous buffers and writes out the data.	Compatible
io.h	writew	Write two bytes to I/O.	Compatible
zlib.h	zlib_deflate	Compress data.	Compatible
zlib.h	zlib_deflateEnd	Free the dynamic data structure allocated to the current stream.	Compatible
zlib.h	zlib_deflateInit	Initialize the zlib status.	Compatible
zlib.h	zlib_inflate	Decompress data.	Compatible
zlib.h	zlib_inflateEnd	Free the dynamic data structure allocated to the current stream.	Compatible
zlib.h	zlib_inflateInit	Initialize the internal stream status for decompression.	Compatible
zlib.h	zlib_inflateInit2	Decompress data.	Compatible

4.5.5.2 Linux APIs Not Supported

Some Linux APIs are not supported in Huawei LiteOS. The following table lists the detailed specifications:

Copyright © Huawei Technologies Co., Ltd.

File	API	Description	Supported/Not Supported
adp.c	__assert	Assertion. This API is used to determine whether program execution is correct.	Not supported
adp.c	__cxa_atexit	The process exits.	Not supported
adp.c	__tls_get_addr	Get the address of a thread-local variable.	Not supported
wait.h	add_wait_queue	Add a process to the tail of a queue.	Not supported
adp.c	alarm	Arrange for a SIGALRM signal to be delivered to the calling process in **seconds** seconds. If **seconds** is zero, the previously set alarm is canceled. This API returns the number of seconds remaining until any previously scheduled alarm was due to be delivered.	Not supported
atomic.h	atomic_set	Set the value of an atomic variable.	Not supported
bug.h	BUG	Provide assertions and dumps information.	Not supported
adp.c	chroot	Change the root directory to the directory specified by **path**. Only a superuser is allowed to change the root directory. All children of the calling process will inherit the new root directory.	Not supported
adp.c	closelog	Close the opened connection to a system logger.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Description	Supported/Not Supported
sched.h	cond_resched	Schedule a new process for running.	Not supported
kernel.h	copy_from_user	Copy data from user space to kernel space.	Not supported
kernel.h	copy_to_user	Copy data from kernel space to user space.	Not supported
adp.c	daemon	Create a daemon.	Not supported
wait.h	DECLARE_WAITQUEUE	Define and initialize a wait queue.	Not supported
semaphore.h	down	Attempt to acquire a semaphore. If the semaphore is not available, the process waits on the semaphore and cannot be woken up.	Not supported
semaphore.h	down_interruptible	Attempt to acquire a semaphore. If the semaphore is not available, the process waits on the semaphore. When the semaphore is freed up, the process is woken up.	Not supported
semaphore.h	down_trylock	Attempt to acquire a semaphore. If the semaphore is not available, down_trylock immediately returns with a nonzero return value, and the caller will not wait on the semaphore.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Description	Supported/Not Supported
adp.c	execve	Execute a file specified by **filename**. The second parameter is an array of character pointers. The final parameter is an array of character pointers to the new environment.	Not supported
adp.c	fchown	Change the owner and group of the file specified by **fd** to **owner** and **group** respectively. If **owner** or **group** is specified as －1, the owner or group is changed. The parameter **fd** is an open file descriptor. When root calls fchown() to change the owner or group of a file, the S_ISUID or S_ISGID permission bits are cleared.	Not supported
adp.c	fork	Create a process that is almost the same as the calling process.	Not supported
adp.c	fs_fssync	File synchronization.	Not supported
adp.c	getdtablesize	Return the maximum number of open files a process can have.	Not supported
adp.c	gethostname	Get the hostname.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Description	Supported/Not Supported
adp.c	getpwnam	Search for account names specified by **name** and return the data of each account as a passwd structure. For details on the passwd structure, see the description of getpwent().	Not supported
adp.c	getrlimit	Get the upper limit on resources.	Not supported
semaphore.h	init_MUTEX	Initialize a semaphore and set the semaphore's value to one.	Not supported
semaphore.h	init_MUTEX_LOCKED	Initialize a semaphore and set the semaphore's value to zero.	Not supported
adp.c	initgroups	Read group data from the group database **/etc/group**. If **user** is a member of the group data, the **group** parameter is added to the group data.	Not supported
kernel.h	ioremap_cached	Map physical memory to kernel virtual address space.	Not supported
kernel.h	ioremap_nocached	Map physical memory to kernel virtual address space.	Not supported
kernel.h	iounmap	Cancel the mapping made by ioremap.	Not supported
compiler.h	likely	Conditional statement that indicates the value is likely to be true.	Not supported
adp.c	linux_module_init	Load a module.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Description	Supported/Not Supported
list.h	list_empty	Test whether a linked list is empty.	Not supported
kernel.h	misc_deregister	Unregister a miscellaneous device.	Not supported
kernel.h	misc_register	Register a miscellaneous device.	Not supported
adp.c	mmap	Map files or other objects into memory. Accessing the memory area means reading and writing a file. The **start** parameter specifies the start address of the memory area where files or other objects are mapped, and is normally set to **NULL**, indicating that the address is automatically determined by the OS. Upon successful completion, this address is returned. The **length** parameter specifies the length of file to be mapped into memory.	Not supported
module.h	module_put	Decrease the number of times a module is referenced by one.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Description	Supported/Not Supported
adp.c	munmap	Delete the mappings of files or other objects in the memory. The **length** parameter specifies the size of the memory to be deleted. If the process ends, or other programs are executed by using exec functions, the mapped memory will be automatically deleted. However, the mapping will not be deleted when the corresponding file descriptor is closed.	Not supported
adp.c	nice	Change the execution priority of a process. A greater **inc** value means a lower priority. Only a superuser is allowed to set a negative **inc** value, in which case a greater value means a higher priority.	Not supported
adp.c	pipe	Create a pipe and place two file descriptors, one each into **filedes[0]** and **filedes[1]**. **filedes[0]** refers to the read end of the pipe, and **filedes[1]** the write end.	Not supported
kernel.h	printtime	Print the system time.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Description	Supported/Not Supported
adp.c	readlink	Place the contents of the symbolic link **path** in the buffer **buf**. The returned content is not NULL-terminated but the number of bytes placed in the buffer. If the buffer size **bufsiz** is smaller than the size of the contents of the symbolic path, the contents will be truncated.	Not supported
adp.c	recvmsg	Receive a message from a socket specified by a remote host. The **s** parameter specifies a connected socket. If the user datagram protocol (UDP) is used, the socket does not need to be connected. The **msg** parameter points to the message structure to be connected. The **flags** parameter is normally set to **0**. For details about **flags**, see the description of send(). For the definition of the msghdr structure, see the description of sendmsg().	Not supported
wait.h	remove_wait_queue	Remove a wait queue from the wait queue linked list pointed to by the wait queue head associated with the wait queue to be removed.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Description	Supported/Not Supported
adp.c	setgroups	Set the supplementary group IDs in the array specified by **list** for the calling process. The **size** parameter specifies the number of **gid_t** in **list**. The maximum **size** value is **NGROUP(32)**.	Not supported
sched.h	signal_pending	Test whether the current process is processes by a signal.	Not supported
adp.c	sigset	Change the disposition of the signal specified by **sig** to the disposition specified by **disp**.	Not supported
string.h	simple_strtol	Convert a string to an unsigned long data.	Not supported
adp.c	syscall	Invoke a system call.	Not supported
timer.h	timer_pending	Test whether a timer has been activated.	Not supported
module.h	try_module_get	Increase the number of modules in use.	Not supported
rwsem.h	try_module_get	Downgrade writers to readers.	Not supported
mount.h	umount2	Unmount a file system.	Not supported
compiler.h	unlikely	Conditional statement that indicates the value is likely to be false.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Description	Supported/Not Supported
semaphore.h	up	Release a semaphore, wake up the first process in the queue of processes waiting on the semaphore.	Not supported
adp.c	waitpid	Suspend execution of the calling process until the delivery of a signal, or until a child process terminates. If a child process has terminated at the time wait() is called, wait() will immediately return the termination status value of the child process. The termination status value stored in the **status** parameter and the child process ID are returned. If the termination status value is unnecessary, **status** can be set to **NULL**.	Not supported
wakelock.h	wake_lock	Activate a lock.	Not supported
wakelock.h	wake_lock_active	Test whether a lock is active.	Not supported
wakelock.h	wake_lock_init	Initialize a lock.	Not supported
wakelock.h	wake_unlock	Unlock and deactivates a lock.	Not supported
watchdog.h	watchdog_init	Initialize a watch dog.	Not supported

4.6 C++ Support

Copyright © Huawei Technologies Co., Ltd.

4.6.1 Overview

Basic Concept

C++ is one of the most widely used programming languages. It is an object-oriented programming language with features including classes, encapsulation, and reloading.

Operation Mechanism

STL is a collection of some "containers", as well as a collection of algorithms and other components. The objective is to develop a standardized component that can be used without developing again, directly using an off the shelf component.

4.6.2 Development Guidelines

Function

Function Category	API	Description
Initialize C++ constructors	LOS_CppSystemInit	Initializes C++ constructors

Initializing C++

C++ initialization functions vary depending on whether scatter loading is enabled or disabled. This is because the use of scatter loading affects program code and the data segment loading time.

Scatter loading is disabled

Before calling C++ code, call the LOS_CppSystemInit API with the NO_SCATTER parameter.

```
LOS_CppSystemInit((unsigned long)&__init_array_start__, (unsigned
long)&__init_array_end__, NO_SCATTER);
```

Table 4-2 Description

Parameter	Description
__init_array_start__	Start array
__init_array_end__	End array
NO_SCATTER	Scatter loading is disabled

Scatter loading is enabled

- If C++ code needs to be called at the fast startup phase of scatter loading, first call the LOS_CppSystemInit API with the BEFORE_SCATTER parameter.
  ```
  LOS_CppSystemInit((unsigned long)&__init_array_start__, (unsigned
  long)&__init_array_end__, BEFORE_SCATTER);
  ```


Copyright © Huawei Technologies Co., Ltd.

Table 4-3 Description

Parameter	Description
__init_array_start__	Start array
__init_array_end__	End array
BEFORE_SCATTER	LOS_CppSystemInit is called at the fast startup phase of scatter loading

Then, at the non-fast startup phase of scatter loading, call the LOS_CppSystemInit API with the AFTER_SCATTER parameter.

```
LOS_CppSystemInit((unsigned long)&__init_array_start__, (unsigned
long)&__init_array_end__, AFTER_SCATTER);
```

Table 4-4 Description

Parameter	Description
__init_array_start__	Start array
__init_array_end__	End array
AFTER_SCATTER	LOS_CppSystemInit is called at the non-fast startup phase of scatter loading

- If C++ code does not need to be called at the fast startup phase of scatter loading, you can try any of the approaches: (1) call the LOS_CppSystemInit API twice, first at the fast startup phase of scatter loading and then at the non-fast startup phase; (2) call the LOS_CppSystemInit API twice at the non-fast startup phase of scatter loading, first with the BEFORE_SCATTER parameter and then with the AFTER_SCATTER parameter.

  ```
  LOS_CppSystemInit((unsigned long)&__init_array_start__, (unsigned
  long)&__init_array_end__, BEFORE_SCATTER);

   LOS_CppSystemInit((unsigned long)&__init_array_start__, (unsigned
  long)&__init_array_end__, AFTER_SCATTER);
  ```

  Or, alternatively, call the LOS_CppSystemInit API once with the NO_SCATTER parameter.

  ```
  LOS_CppSystemInit((unsigned long)&__init_array_start__, (unsigned
  long)&__init_array_end__, NO_SCATTER);
  ```

Call C Language Functions

To call a C language function in C++ language, add the following statement to the function declaration:

extern "C".

4.6.3 Precautions

None.

Copyright © Huawei Technologies Co., Ltd.

4.6.4 Programming Example

Example Description

C++ constructors are initialized during code initialization to make C++ features usable. In this example, scatter loading is to be used, so the LOS_CppSystemInit API needs to be called twice.

Example Code

```
void app_init(void)

{

......

/* C++ constructor initialization during the fast boot phase */

LOS_CppSystemInit((UINT32)&__init_array_start__, (UINT32)&__init_array_end__,
BEFORE_SCATTER);

/* scatter loading */

LOS_ScatterLoad(0x100000, flash_read, NAND_READ_ALIGN_SIZE);

/* C++ constructor initialization during the non-fast boot phase */

LOS_CppSystemInit((UINT32)&__init_array_start__, (UINT32)&__init_array_end__,
AFTER_SCATTER);

......

}
```

4.7 MMU

4.7.1 Overview

Basic Concept

MMU is short for memory management unit.

Operation Mechanism

The table of page table descriptors is created, and the physical address is translated into a virtual address. The physical memory then can be accessed through the virtual address.

A user creates, modifies, and manages a table of page table descriptors with the format of the master chip used by the user. A table of page table descriptors is a reference for memory mapping and permission control. The creation of and modification to it take effect after it is written to the CP15 coprocessor that executes memory management.

Operations on an MMU are performed by modifying a page table descriptor and controlling the CP15 coprocessor. The operation process is as illustrated in **Figure 1** .

Copyright © Huawei Technologies Co., Ltd.

Figure 4-4

An MMU in Huawei LiteOS has the following functions:

- Providing an API to control the cache/nocache attribute of hardware
- Providing an API to control the memory access permission of hardware

4.7.2 Development Guidelines

Usage Scenarios

Some memory does not hope to be modified. Unpredictable results will be caused if the memory is modified. To protect the memory from being modified, an MMU is used to modify the access permission on the memory. The permission of any access to the memory is checked. If the permission is not correct, an exception will be triggered.

Cache and buffer status can be controlled through cache+buffer. For example, you can disable cache, write through, and write back.

Functions

The MMU module of Huawei LiteOS provides the following function:

Table 4-5

Function Category	API	Description
Control the access permission	LOS_MMUParamSet	Changes the status of cache, buffer, and read/write permission of the memory at the specified address.

Parameters:

Copyright © Huawei Technologies Co., Ltd.

```
typedef struct
{
UINT32 startAddr;   // Start memory address
UINT32 endAddr;    // End memory address
UINT32 uwFlag;       // Memory attributes
// BUFFER_ENABLE/BUFFER_DISABLE
// CACHE_ENABLE/CACHE_DISABLE
// ACCESS_PERM_RW_RW/ACCESS_PERM_RO_RO
SENCOND_PAGE *stPage;   // The 2nd-level page table to be operated
}MMU_PARAM;
```

Example:

```
void code_protect(void)
{
MMU_PARAM mPara;
mPara.startAddr = (unsigned long)&__text_start;
mPara.endAddr = (unsigned long)&__rodata1_end;
mPara.uwFlag = BUFFER_ENABLE|CACHE_ENABLE|ACCESS_PERM_RO_RO;
mPara.stPage = (SENCOND_PAGE *)&stOsPage;  // Mark A
LOS_MMUParamSet(&mPara);
}
```

Description:

This API is used to set the memory starting from the address specified by **__text_start** and ending at the address specified by**__rodata1_end** to read-only and enable the cache and buffer of this memory segment. Multiple 2nd-level page tables may be available. Mark A is used to specify the 2nd-level page table to be operated by the API.

Development Process

```
typedef struct
{
UINT32 page_addr;        // Start address of the memory described by a 2nd-level
page table, which must be aligned on the boundary of 1 MB.
UINT32 page_length;     // Length of the memory, which must be aligned on the
boundary of 1 MB.
UINT32 page_descriptor_addr;    // Storage address of the 2nd-level page table
entries, which must be aligned on the boundary of 1 KB.
UINT32 page_type;  // 2nd-level page table type, including small page table (4
KB) and big page table (64 KB).
}SENCOND_PAGE;
```

Refer to the preceding structure and perform the following steps to complete the development:

Step 1 Define a 2nd-level page table structure variable, which should be a global variable.

```
SENCOND_PAGE stOsPage = {0};
```

Step 2 Configure the 2nd-level page table.

```
stOsPage.page_addr = test_page_addr;
stOsPage.page_length = test_length;
stOsPage.page_descriptor_addr = test_addr;
stOsPage.page_type = MMU_SECOND_LEVEL_SMALL_PAGE_TABLE_ID;
```

Step 3 Enable the 2nd-level page table.

```
LOS_SecPageEnable(&stOsPage, BUFFER_ENABLE|CACHE_ENABLE|
ACCESS_PERM_RW_RW);
```

Step 4 Modify the attributes of the 2nd-level page table.

```
void test_func(void)
{
MMU_PARAM mPara;
mPara.startAddr = (unsigned long)&__text_start;
mPara.endAddr = (unsigned long)&__rodata1_end;
mPara.uwFlag = CACHE_ENABLE|ACCESS_PERM_RO_RO;
```


Copyright © Huawei Technologies Co., Ltd.

```
mPara.stPage = (SENCOND_PAGE *)&stOsPage;
LOS_MMUParamSet(&mPara);
}
```

----End

4.7.3 Precautions

- Compared with the 1st-level page table, the efficiency of memory translation configured for the 2nd-level page table is lower, and the 2nd-level page table uses more translation lookaside buffer (TLB) resources. Therefore, you are advised to use the 1st-level page table to process memory that is frequently accessed.

4.7.4 Programming Example

Example Description

You can call the LOS_MMUParamSet API and perform the following steps to check how the API functions:

Step 1 Set the access permission on a specified memory segment to read-only.

Step 2 Write data into the memory segment.

An exception occurs, indicating that the access permission setting succeeds.

Step 3 Delete the write operation in step 2 and call the LOS_MMUAPSet API to set the access permission to read/write.

No exception occurs, indicating that the access permission setting succeeds.

----End

Example Code

```
UINT32 MMU_Sample()
{
    UINT32 *pAlignaddr;
    extern SENCOND_PAGE stOsPage;
    MMU_PARAM mPara;
    mPara.startAddr = (unsigned long)&__text_start;
    mPara.endAddr = (unsigned long)&__text_end;
    mPara.uwFlag = BUFFER_ENABLE|CACHE_ENABLE|ACCESS_PERM_RO_RO;
    mPara.stPage = (SENCOND_PAGE *)&stOsPage;
    PRINTK("---- TEST START ----\n");
    pAlignaddr = (UINT32 *)(&__text_start);
    PRINTK(">>1\n");
    LOS_MMUParamSet(&mPara);
    *pAlignaddr = 0xa; //if done, be exc
    PRINTK(">>2\n");
    mPara.uwFlag = BUFFER_ENABLE|CACHE_ENABLE|ACCESS_PERM_RW_RW;
    LOS_MMUParamSet(&mPara);
    *pAlignaddr = 0xb;
    PRINTK(">>3\n");
    PRINTK("---- TEST END ----\n");
    return LOS_OK;
}
```

Execution Result

The "*pAlignaddr = 0xa" code is not commented out.

Copyright © Huawei Technologies Co., Ltd.

```
---- TEST START ----
>>1
uwExcType = 0x4
puwExcBuffAddr pc = 0x8000adc8
puwExcBuffAddr lr = 0x8000aa78
puwExcBuffAddr sp = 0x80093e30
puwExcBuffAddr fp = 0x80093e44
*******backtrace begin*******
traceback 0 -- lr = 0x8000b1e4
traceback 0 -- fp = 0x80093e84
traceback 1 -- lr = 0x8000b2c4
traceback 1 -- fp = 0x80093e9c
traceback 2 -- lr = 0x8000ae8c
traceback 2 -- fp = 0x80093eac
traceback 3 -- lr = 0x8000ac4c
traceback 3 -- fp = 0x80093ebc
traceback 4 -- lr = 0x800104fc
traceback 4 -- fp = 0x80093ecc
traceback 5 -- lr = 0x80026e1c
traceback 5 -- fp = 0x11111111

Name                        TID    Priority   Status
----                        ---    --------   --------
Swt_Task                    0x0    0          QueuePend
IdleCore000                 0x1    31         Ready
IT_TST_INI                  0x2    25         Running
R0          = 0x800040c0
R1          = 0x800040a8
R2          = 0x800040a4
R3          = 0x0
R4          = 0x8000a000
R5          = 0x0
R6          = 0x8002a000
R7          = 0x800302e4
R8          = 0x80051df4
R9          = 0x8002c020
R10         = 0x80051df4
R11         = 0x80093e44
R12         = 0x2a
SP          = 0x80093e30
LR          = 0x8000aa78
PC          = 0x8000adc8
CPSR        = 0x200f0013
pcTaskName = IT_TST_INI
TaskID = 2
Task StackSize = 86016
system mem addr:0x800561c0

stack name           stack addr     total size   used size
----------           ---------      --------     --------
undef_stack_addr     0x8002c820     0x20         0x0
abt_stack_addr       0x8002c840     0x20         0xb
irq_stack_addr       0x8002c880     0x40         0x10
fiq_stack_addr       0x8002c8c0     0x40         0x0
svc_stack_addr       0x8002d8c0     0x1000       0x14f
startup_stack_addr   0x800563a0     0x200        0x138
```

The "*pAlignaddr = 0xa" code is commented out.

```
---- TEST START ----
>>1
>>2
>>3
---- TEST END ----
```

Complete Example Code

sample_MMU.c

Copyright © Huawei Technologies Co., Ltd.

4.8 Atomic Operation

4.8.1 Overview

Basic Concept

In a multi-tasking operating system (OS), data reading, modification, and writing are three essential procedures of modifying the data in a specified memory. However, the data may be concurrently accessed by multiple tasks. If the data modification is interrupted by other tasks, unexpected modification result will occur.

Multiple tasks can be successfully executed by enabling and disabling interrupts, but this method affects OS performance.

The ARMv6 architecture introduces LDREX and STREX command to support non-blocking synchronization of shared memory, which allows a data modification not to be interrupted and achieves an atomic operation.

Operation Mechanism

Huawei LiteOS provides atomic operation APIs by encapsulating LDREX and STREX in the ARMv6 architecture.

- **LDREX Rx, [Ry]**

  The following is the method to read data in the memory and mark the exclusive access to the memory:

  - Read the 4-byte memory data pointed to by the Ry register and store the read memory data to the Rx register.
  - Add an exclusive access flag to the memory segment pointed by Ry.

- **STREX Rf, Rx, [Ry]**

  The following describes the method to determine whether to update memory data and how STREX functions:

  - If the memory has an exclusive access flag, the memory data will be updated.
    i. Update the memory data pointed to the Ry register to the value in the Rx register.
    ii. Set the Rf flag register to 0.
  - If the memory does not have an exclusive access flag, the memory data should be updated.
    i. The memory data pointed to by Ry will not be updated.
    ii. Set the Rf flag register to 1.

- Flag register
  - If the flag register is 0, the atomic operation ends.
  - If the flag register is 1, the atomic operation cycle proceeds and the atomic operation starts again.

Copyright © Huawei Technologies Co., Ltd.

4.8.2 Development Guidelines

Usage Scenarios

When multiple tasks are performing increasing, decreasing, and exchanging operations on the same memory data, the use of atomic operations will ensure that operation results are predictable.

Functions

The atomic operation module of Huawei LiteOS provides the following functions:

Table 4-6 Functions

Function Category	API	Description
Increasing	LOS_AtomicAdd	Adds a random number to or subtracts a random number from memory data
	LOS_AtomicInc	Adds one to memory data
	LOS_AtomicIncRet	Adds one to memory data and return
Decreasing	LOS_AtomicDec	Subtracts one from memory data
	LOS_AtomicDecRet	Subtracts one from memory data and return
Exchanging	LOS_AtomicXchgByte	Exchanges 8-bit memory data
	LOS_AtomicXchg16bits	Exchanges 16-bit memory data
	LOS_AtomicXchg32bits	Exchanges 32-bit memory data
	LOS_AtomicXchg64bits	Exchanges 64-bit memory data
Exchanging after comparison	LOS_AtomicCmpXchgByte	Compares and exchanges 8-bit memory data
	LOS_AtomicCmpXchg16bits	Compares and exchanges 16-bit memory data
	LOS_AtomicCmpXchg32bits	Compares and exchanges 32-bit memory data
	LOS_AtomicCmpXchg64bits	Compares and exchanges 64-bit memory data

Copyright © Huawei Technologies Co., Ltd.

NOTE

The bits of the input data and the result data cannot exceed the maximum bit allowed by the function.

Platform Differences

IDREX and STREX are used in Cortex-A7 and Cortex-A17 to ensure atomic operations. IDREX and STXR are used in Cortex-A53 to ensure atomic operations. Enabling and disabling interrupts are used to ensure that operations are atomic in ARM926 because ARM926 does not support IDREX and STREX.

4.8.3 Precautions

- Currently, atomic operation APIs can only be used for operations on integer data.

4.8.4 Programming Example

Example Description

Perform the following two steps and view the result:

Step 1 Create two tasks

1. Call the LOS_AtomicInc API to increase the global variable by one for 100 times.
2. Call the LOS_AtomicDec API to decrease the global variable by one for 100 times.

Step 2 After the subtasks are finished, print the global variable value in the major task.

----End

Example Code

```
#include "los_atomic.h"
UINT32 g_TestTaskID01;
UINT32 g_TestTaskID02;
UINT32 g_sum;
UINT32 g_count;
UINT32 It_atomic_001_f01()
{
    int i = 0;
    for(i = 0; i < 100; ++i)
    {
        LOS_AtomicInc(&g_sum);
    }

    ++g_count;
    return LOS_OK;
}
UINT32 It_atomic_001_f02()
{
    int i = 0;
    for(i = 0; i < 100; ++i)
    {
        LOS_AtomicDec(&g_sum);
    }

    ++g_count;
    return LOS_OK;
}
UINT32 it_atomic_test()
{
    UINT32 uwRet,uwCpupUse;
```


Copyright © Huawei Technologies Co., Ltd.

```
    INTPTR uvIntSave;
    TSK_INIT_PARAM_S stTask1={0};
    stTask1.pfnTaskEntry = (TSK_ENTRY_FUNC)It_atomic_001_f01;
    stTask1.pcName       = "TestAtomicTsk1";
    stTask1.uwStackSize  = LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE;
    stTask1.usTaskPrio   = 4;
    stTask1.uwResved     = LOS_TASK_STATUS_DETACHED;

    TSK_INIT_PARAM_S stTask2={0};
    stTask2.pfnTaskEntry = (TSK_ENTRY_FUNC)It_atomic_001_f02;
    stTask2.pcName       = "TestAtomicTsk2";
    stTask2.uwStackSize  = LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE;
    stTask2.usTaskPrio   = 4;
    stTask2.uwResved     = LOS_TASK_STATUS_DETACHED;

    LOS_TaskLock();
    LOS_TaskCreate(&g_TestTaskID01, &stTask1);
    LOS_TaskCreate(&g_TestTaskID02, &stTask2);
    LOS_TaskUnlock();

    while(g_count != 2);
    PRINTK("g_sum = %d\n", g_sum);

    return LOS_OK;
}
```

Verification

```
g_sum = 0
```

Complete Example Code

sample_atomic.c

4.9 Run-Stop

4.9.1 Overview

Basic Concept

Run-Stop is used to store the image of Huawei LiteOS encountering an exception and restore the OS running.

The Run-Stop interface is called to store the CPU threads and memory status snapshot of the running OS to the Flash. After the OS restarts, the OS running status can be restored from the snapshot in the Flash.

Run-Stop can be used to wake up the OS after power-off in WiFi service. When the WiFi service is stable after being initialized, the Run-Stop interface is called to store the snapshot of OS including the CPU threads and memory in Flash. When the OS is idle, the master core is powered off to enter the power-saving mode. When there are services for the WiFi service, the micro control unit (MCU) sends a packet to power on the master core. After the hardware status is restored, the stored snapshot is restored to ensure the continued running of the WiFi service.

Copyright © Huawei Technologies Co., Ltd.

4.9.2 Development Guidelines

Usage Scenarios

You can use the Run-Stop mechanism when you intend to store the snapshot of Huawei LiteOS on a medium after the OS runs for a period of time, run the OS from the snapshot at another point of time, and hope that the OS state at the moment you run the OS the second time is the same as the snapshot.

In an IP camera (IPC) that runs Huawei LiteOS, Run-Stop is used to restore the OS state when a WiFi service will run. The snapshot of Huawei LiteOS at the moment when a WiFi service runs stably is stored. When the OS is idle, the master core is powered off to enter the power-saving mode. When the OS receives a WiFi packet, the memory management unit (MMU) powers on the master core, and the stored snapshot is restored to ensure the running of the WiFi service. In this way, the OS state can be quickly restored to ensure stable WiFi connection in the power-saving mode.

Function

The Run-Stop module of Huawei LiteOS provides the following function:

Table 4-7 Function

API	Description
LOS_MakeImage	Stores the snapshot of Huawei LiteOS on a specified medium.

Copyright © Huawei Technologies Co., Ltd.

Development Process

Figure 4-5 Operation process of Run-Stop

Step 1 Call the LOS_MakeImage API and write the code of the snapshot service.

The entry of the service code is the app_init function contained in the **os_adapt.c** file.

NOTE

The **os_adapt.c** file can be found in the **platform/bsp/board/hi3516a/os_adapt** directory in the Huawei LiteOS cede package.

Following the snapshot service code, call the LOS_MakeImage API to store the OS snapshot on a specified medium. Surround the non-snapshot services following the LOS_MakeImage API with "#ifndef MAKE_WOW_IMAGE" and "#endif". The example code is as follows:

```
void wakeup_callback(void)
{
hal_interrupt_unmask(83);
if(!nand_init())
{
PRINT_ERR("nand init failed\n");
}
}

#define NAND_ERASE_ALIGN_SIZE(128 * 1024)
#define NAND_READ_ALIGN_SIZE(2 * 1024)
#define NAND_WRITE_ALIGN_SIZE(2 * 1024)
```


Copyright © Huawei Technologies Co., Ltd.

```
int flash_read(void *memaddr, unsigned long start, unsigned long size)
{
extern int hinand_read(void *memaddr, unsigned long start, unsigned long size);
return hinand_read(memaddr, start, size);
}

int flash_write(void *memaddr, unsigned long start, unsigned long size)
{
extern int hinand_erase(unsigned long start, unsigned long size);
extern int hinand_write(void *memaddr, unsigned long start, unsigned long size);
(void)hinand_erase(start, size);
return hinand_write(memaddr, start, size);
}
void app_init(void)
{
RUNSTOP_PARAM_S stRunstopParam;

proc_fs_init();

if (hi_uartdev_init() != 0)
{
PRINT_ERR("hi_uartdev_init failed");
}
if (system_console_init(TTY_DEVICE) != 0)
{
PRINT_ERR("system_console_init failed\n");
}

if(nand_init() != 0)
{
PRINT_ERR("nand_init falied\n");
}

memset(&stRunstopParam, 0, sizeof(RUNSTOP_PARAM_S));

/* Parameter configuration */
stRunstopParam.pfFlashReadFunc = flash_read;
stRunstopParam.pfFlashWriteFunc = flash_write;
stRunstopParam.pfImageDoneCallback = NULL;
stRunstopParam.pfIdleWakeupCallback = NULL;
stRunstopParam.pfWakeupCallback = wakeup_callback;
stRunstopParam.uwFlashEraseAlignSize = NAND_ERASE_ALIGN_SIZE;
stRunstopParam.uwFlashWriteAlignSize = NAND_WRITE_ALIGN_SIZE;
stRunstopParam.uwFlashReadAlignSize = NAND_READ_ALIGN_SIZE;
stRunstopParam.uwImageFlashAddr = 0x100000;
stRunstopParam.uwWowFlashAddr = 0x3000000;

LOS_MakeImage(&stRunstopParam);

#ifndef MAKE_WOW_IMAGE
extern UINT32 g_uwWowImgSize;
PRINTK("Image length 0x%x\n", g_uwWowImgSize);

extern unsigned int osShellInit(const char *);
if (osShellInit(TTY_DEVICE) != 0)
{
PRINT_ERR("osShellInit\n");
}

rdk_fs_init();
SDK_init();

vs_server(3, apszArgv);
#endif /* MAKE_WOW_IMAGE */
}
```

Step 2 Configure the **WOW_SRC** variable.

Copyright © Huawei Technologies Co., Ltd.

Run the following command to set the SCATTER_SRC variable in **Makefile** under the root directory to the source file of the services that call the LOS_MakeImage API. **LITEOSTOPDIR** represents the root directory of Huawei LiteOS code.

```
WOW_SRC := $(LITEOSTOPDIR)/platform/bsp/board/$(LITEOS_PLATFORM)/os_adapt/
os_adapt.cWOW_SRC := WOW_SRC := WOW_SRC := WOW_SRC := WOW_SRC := WOW_SRC :=
WOW_SRC := WOW_SRC := WOW_SRC :=
```

Step 3 Run the make command to compile all images.

1. Run the following command under the root directory to compile all service code.
```
Huawei_LiteOS$ make
```
2. Check whether the snapshot image is successfully generated by viewing the image segment allocation. Access the directory where the **vs_server** image file is generated. The directory name is **out/*platform name***. For example, the image of hi3516a is generated in the **out/hi3516a** directory. Run the **readelf -S vs_server** command to open the **vs_server** file. Information similar to the following will be displayed:

Figure 4-6 vs_server.bin image file

```
[12] .wow_rodata       PROGBITS        80008020 0000d8 00b550 00   A  0   0  8
[13] .wow_text         PROGBITS        80013570 00b628 04bb44 00  AX  0   0  8
[14] .wow_data         PROGBITS        8005f0b4 05716c 001c3c 00  WA  0   0  8
[15] .wow_bss          NOBITS          80060cf0 058da8 008e1c 00  WA  0   0  8
```

Information about the segments related to Run-Stop is displayed, including segment name, start address, and offset. In the preceding figure, .wow_rodata and .wow_constdata are read-only data segments. .wow_text, .wow_data, and .wow_bss indicate the code segment, data segment, and bss segment respectively.

3. View the .text segment in the link script of Run-Stop. wow.o(*.text*) is added, as shown in the following figure, indicating that symbols related to the snapshot code are placed in the same area.

Figure 4-7 .text segment in the link script

```
.wow_text ALIGN (0x4): {
wow.O(.text*);
}

.wow_data ALIGN (0x4): {
 . = ALIGN (4); KEEP(wow.O( SORT (.liteos.table.*.wow.*))) ; . = ALIGN (4); wow.O(.data*);
}

__wow_bss_start = ABSOLUTE (.);
```

NOTE

The path to the link script of Run-Stop is **Huawei_LiteOS/tools/scripts/ld/wow.ld**.

Step 4 Run the following command to burn all images into the Flash.

```
tftp 0x82000000 vs_server.bin;nand erase 0x100000 0x700000;nand write 0x82000000
0x100000 0x700000;
```

In the serial port tool interface, enter the following command to burn all images into the Flash at the address of 0x100000.

```
tftp 0x82000000 vs_server.bin;nand erase 0x100000 0x700000;nand write 0x82000000
0x100000 0x700000;
```

vs_server.bin in the command is the name of system image file. Burn this file into memory at the address of 0x82000000. Then burn it into the Flash starting at the address of 0x100000. The size of file to be burnt is 0x700000, indicating that the size of the burnt image file must not exceed 7 MB. Adjust to size of the to-be-burnt file according to the actual image file size.

Copyright © Huawei Technologies Co., Ltd.

Step 5 Run the following command to load all images.

```
nand read 0x80008000 0x100000 0x700000; go 0x80008000;
```

Run the following command to start Huawei LiteOS.

```
nand read 0x80008000 0x100000 0x700000; go 0x80008000;
```

Step 6 Run Huawei LiteOS from the snapshot.

The snapshot image size can be printed, as shown in the following figure:

Figure 4-8 Printed information

```
TFTP from server 192.168.1.2; our IP address is 192.168.1.10
Download Filename 'vs_server.bin'.
Download to address: 0x80008000
Downloading: *#################################################
done
Bytes transferred = 825104 (c9710 hex)
## Starting application at 0x80008000 ...
********hello Huawei LiteOS Cortex-A7********

version : Huawei LiteOS V100R002C00B302
open-version : Huawei LiteOS 1.1.4T6
build data : Aug 11 2016 16:37:58

**********************************
osAppInit
Mount procfs finished.
Nand ID:0x01 0xDA 0x90 0x95 0x44 0x01 0xDA 0x90
Nand:"NAND 256MiB 3,3V 8-bit"
Size:256MB Block:128KB Page:2KB Oob:64B Ecc:8bit/1K
Image length 0xc0000
```

In the sample shown in the foregoing figure, the snapshot size is 0xc0000, and the snapshot is written to the address of 0x3000000 on the medium.

Run the **nand read 0x80008000 0x3000000 0xc0000; go 0x80008000;** command in uboot to start Huawei LiteOS from the snapshot. Information similar to that shown in the following figure will be displayed:

Figure 4-9 Printed information

```
hisilicon # nand read 0x80008000 0x3000000 0xc0000; go 0x80008000;

NAND read: device 0 offset 0x3000000, size 0xc0000
 786432 bytes read: OK
## Starting application at 0x80008000 ...
Nand ID:0x01 0xDA 0x90 0x95 0x44 0x01 0xDA 0x90
Nand:"NAND 256MiB 3,3V 8-bit"
Size:256MB Block:128KB Page:2KB Oob:64B Ecc:8bit/1K
register nand err -17
nand_init(673): Error:nand node register fail!
Image length 0xc0000
```

Huawei LiteOS is successfully started from the snapshot.

----End

4.9.3 Precautions

- When writing Run-Stop services, ensure that all code and data of the snapshot are specified preceding "#ifndef MAKE_WOW_IMAGE". Otherwise, not all Run-Stop services will be contained after the OS snapshot is restored.
- The toolchain of Huawei LiteOS Run-Stop depends on Python 2.7. Therefore, ensure that Python 2.7 is used in the current development environment.

Copyright © Huawei Technologies Co., Ltd.

4.9.4 LOS_MakeImage Parameter Configurations

RUNSTOP_PARAM_S stRunstopParam; // Define a variable for the LOS_MakeImage API:

- Specify the medium read function:

  stRunstopParam.pfFlashReadFunc = flash_read;
- Specify the medium write function:

  stRunstopParam.pfFlashWriteFunc = flash_write;
- Specify the callback function to be executed after the specified snapshot image is generated:

  stRunstopParam.pfImageDoneCallback = NULL;
- Specify the callback function to be executed in the idle task after the specified snapshot is restored:

  stRunstopParam.pfIdleWakeupCallback = NULL;
- Specify the callback function to be executed after the specified snapshot is restored:

  stRunstopParam.pfWakeupCallback = wakeup_callback;
- Erase alignment parameter of the medium

  stRunstopParam.uwFlashEraseAlignSize = NAND_ERASE_ALIGN_SIZE;
- Write alignment parameter of the medium

  stRunstopParam.uwFlashWriteAlignSize = NAND_WRITE_ALIGN_SIZE;
- Read alignment parameter of the medium

  stRunstopParam.uwFlashReadAlignSize = NAND_READ_ALIGN_SIZE;
- Address of all images on the medium (address where all images are burnt on the medium

  stRunstopParam.uwImageFlashAddr = 0x100000;
- Start address of the medium that stores the snapshot

  stRunstopParam.uwWowFlashAddr = 0x3000000;

4.10 Tickless

4.10.1 Overview

Basic Concept

Tickless is a new timing mechanism and replaces periodic timer interrupts with dynamic timer interrupts. Task scheduling is normally triggered by timer interrupts. When periodic timing is implemented, whether task scheduling requirements are met is checked when each timer interrupt occurs. However, a few timer interrupts are able to trigger task scheduling. Software timer tasks must be implemented with the help of timer interrupts. Whether software timers expire is checked when each interrupt occurs. If a timer expires, the callback function associated with this timer is called. Normally, no timers expire when most timer interrupts occur. The tickless mechanism can be used to omit unnecessary timer interrupts and determine the next time when a necessary interrupt occurs. By omitting unnecessary timer interrupts that are scheduled when CPUs are idle, power consumption is vastly reduced.

Copyright © Huawei Technologies Co., Ltd.

Operation Mechanism

In Huawei LiteOS, the tickless mechanism is enabled when CPUs are idle. In the tickless OS, the next time of task scheduling and that of software timer expiration are calculated. The earlier time will be the next time when an interrupt is scheduled. Therefore, unnecessary timer interrupts will not occur in timeless mode.

You can configure this function in the kernel module of menuconfig.

4.10.2 Development Guidelines

Usage Scenarios

The CPU is idle for a long time, and power saving is required.

Functions

Tickless Huawei LiteOS provides the following functions.

Table 4-8 Functions

Function Category	API	Description
Enabling the tickless mode	LOS_TicklessEnable	Enables the tickless mode.
Disabling the tickless mode	LOS_TicklessDisable	Disables the tickless mode.

Development Process

The typical development process of the tickless mode is as follows:

1. When no user services are running in the OS for a long time, call **LOS_TicklessEnable** to enable the tickless mode.
2. When user services are found to be active, call **LOS_TicklessDisable** to disable the tickless mode.

Platform Differences

None.

4.10.3 Precautions

An interrupt timer is used in tickless implementation and the calculation of system time. You are not advised to enable the tickless mode when high resolution of time is required.

Copyright © Huawei Technologies Co., Ltd.

5 File System

About This Chapter

5.1 Functions Overview

5.2 VFS

5.3 NFS

5.4 JFFS2

5.5 FAT

5.6 YAFFS2

5.7 RAMFS

5.8 PROC

5.9 ROMFS

5.1 Functions Overview

Huawei LiteOS supports the following file systems: virtual file system (VFS), network file system (NFS), journal flash file system version 2 (JFFS2), file allocation table (FAT), yet another flash file system version 2 (YAFFS2), RAM file system (RAMFS), and PROC.

Summary of File Systems

Table 5-1 File system functions

File System	Function
VFS	VFS allows reading and writing data into file systems on different physical media by standard Unix system calls, indicating that operations are performed on different file systems in a uniform way.

Copyright © Huawei Technologies Co., Ltd.

File System	Function
NFS	NFS enables various devices and operating systems to share files with each other through networks.
JFFS2	JFFS2 manages journaled file systems, particularly files in NOR flash, on devices. In Huawei LiteOS, JFFS2 supports multiple partitions.
FAT	FAT is classified into four types: FAT12, FAT16, FAT32, exFat. FAT is typically used on removable media storage devices, including USB flash drives, secure digital memory cards (SD cards), and removable hard disks, to maintain good compatibility between devices and desktop systems such as Windows and Linux.
YAFFS2	YAFFS2 is an open-source embedded file system designed for NAND flash. It is used for large-capacity storage devices and keeps NAND flash efficient and robust. Wear leveling and power failure protection ensures that data will not be corrupted when a power outage occurs amid file system modification. In Huawei LiteOS, YAFFS2 supports multiple partitions.
RAMFS	RAMFS is a file system that exports a storage medium as a dynamically re-sizable RAM-based file system. RAMFS places all files in a RAM. Files are read and written from and to the RAM, accelerating the read/write speed and avoiding frequent access to storage. RAMFS is a RAM-based cache mechanism of dynamic file systems.
PROC	PROC is a kind of pseudo file system that exists only in memory and does not use external storage. It provides an interface for accessing the data in Huawei LiteOS Kernel.

5.2 VFS

Copyright © Huawei Technologies Co., Ltd.

5.2.1 Overview

Basic Concept

VFS is a an abstraction layer between lower-layer file systems and upper-layer applications. It provides a uniform Unix file operation interface.

Multiple types of file systems that require different accessing interfaces need to be accessed in different modes and through different non-standard interfaces. The VFS layer provides a uniform file system accessing interface and masks the differences between the lower-layer file systems, enabling applications to access the file systems regardless of the types of lower-layer storage media and file systems and improving the developing efficiency.

In Huawei LiteOS, the VFS framework is realized through the tree structure in memory. Each node of the tree is an inode structure. A node is generated in the tree according to the directory where a device is registered and mounted. VFS has the following two functions:

1. Searching for nodes

2. Unified invoking (standard)

Operation Mechanism

VFS enables access to different file systems on different media by calling standard Unix file operation functions (such as open, read, and write).

Types of inode tree nodes in VFS are as follows:

- Virtual node: A virtual node serves as a virtual file of the VFS framework, which keeps the continuity of the tree. For example, **/bin** or **/bin/vs** is a virtual node.
- Device node: A device node is under the **/dev** directory and corresponds to a device. For example, **/dev/mmc0** is a device node.
- Mounting node: A mounting node, for example, **/bin/vs/sd**, **/ramfs**, and **/yaffs**, is generated after the mount function is called.

The key elements of an inode are the u field that indicates the pointer to the function structure and the i_private field that indicates the pointer to data.

Copyright © Huawei Technologies Co., Ltd.

Figure 5-1 VFS frame structure

5.2.2 Development Guidelines

Development Process

It is recommended that driver developers use the VFS framework to register and uninstall device and use open() and read() operation devices (character devices) to make device files invoke the driver on the application layer.

1. After Huawei LiteOS calls the los_vfs_init() VFS initialization API, **/dev** will serve as the root_inode.
2. The register_driver() and register_blockdriver() APIs are called to generate device nodes, and the mount() API is called to generate mounting nodes where devices are mounted.
3. Structure information is added during node generation. Nodes are then added to appropriate places in the tree according to their names.
4. A device node or mounting node to which the path is specified is searched for in the tree.
5. Corresponding functions can be called by using the pointer to the node that is found.

File Descriptor

In this design, file descriptors are managed by using global arrays.

There are the following two types of file descriptors:

- File descriptor: A File descriptor is a normal file descriptor, in which 0, 1, and 2 are reserved to serve as system stdin (standard input), stdout (standard output), and stderr (standard error). File descriptors that can be allocated are from 3 to CONFIG_NFILE_DESCRIPTORS – 2.

Copyright © Huawei Technologies Co., Ltd.

- Socket descriptor: Socket descriptors that can be allocated are from CONFIG_NFILE_DESCRIPTORS.

The two types of file descriptors correspond to two global arrays respectively and the memory allocated to them are not contiguous.

File Attributes

The following API is only used in FAT to change the attribute of a file:

int chattr(const char *path, mode_t mode)

path specifies the file of which the attribute is to be changed.

mode specifies the attribute after the attribute change. There are four types of attributes (**F_RDO**: read-only; **F_HID**: hidden; **F_SYS**: system file; **F_ARC**: archive file).

NOTE

- Currently, the API is only used to change the file attribute in FAT. The file attributes in other file systems are configured in other ways.
- In Huawei LiteOS, The file attribute can be changed to any one of the four types of attributes.
- In Huawei LiteOS, read-only files and directories must not be deleted.
- In Huawei LiteOS, read-only files and directories can be renamed.
- Read-only files must not be opened in the O_CREAT and O_TRUNC modes or with read permissions.
- In Huawei LiteOS, hidden files are visible. However, hidden files are invisible in Windows when hidden files are set not to be displayed.
- If system files are configured with the hidden property in Huawei LiteOS, it can only be found in Windows by running commands. These files are invisible regardless of whether hidden files are set to be displayed or not.

5.2.3 Precautions

- VFS is a framework of file systems and allows developers of the application layer to call file systems in a uniform way.
- The name length of a directory or a file to be created in a VFS-mounted file system is allowed to be 255 bytes at most. A directory or a file that has a name length exceeding 255 bytes will fail to be created.
- The configuration of file access permission is not enabled for underlying file systems (O_WRONLY|O_CREAT). The file access permission is set to 0666 by default.
- Calling the inode_find() function increases the number of inode connections by 1. Then the inode_release() function needs to be called to decrease the number of inode connections by 1. Therefore, the inode_find() function is used together with the inode_release() function
- Devices are classified into character devices and block devices. Considering the security of file system data on block devices, VFS temporarily does not support raw read and write operations on block devices. File system data should be manipulated using file system interfaces after the file systems are mounted.
- The los_vfs_init() API can be only called once. File systems will be abnormal if this API is called for multiple times.
- A mount point must be an empty directory and cannot be used for repeated mounting or mounted to another mount point.

Copyright © Huawei Technologies Co., Ltd.

- Only hyphens and underscores are allowed to exist in names of files and directories in all file systems in Huawei LiteOS. Results of containing other special characters in file and directory names are unpredictable, and it is recommended that not contain those characters.
- Multilevel directories can be created recursively in VFS.
- Huawei LiteOS does not allow the operation of directly obtaining directory data by calling open() on a directory. Instead, call the opendir API or specify that a directory is to be opened using open(path, flags | O_DIRECTORY).
- Please mount the file system in strict accordance with the manual, a mount may damage the equipment and system.
- The working directory in the Shell is separated from the system directory. Operations will be performed on the working directory in the Shell by running commands such as **cd** and **pwd** through the Shell. And the system directory will be operated by running commands such as **chdir** and **getcwd**. The two directories are irrelevant to each other. Exercise caution when the input parameter of a file system operation command is a relative path.
- Do not mount a file system that does not exist during wakeup or the scatter loading phase.
- In VFS, the length of a full path must be no more than 259 bytes. A file or a directory whose length is more than 259 bytes cannot be created.
- Parameters **O_RDWR**, **O_WRONLY**, and **O_RDONLY** are mutually exclusive. Only one of them can be used to open a file. Do not use more than one of them to open a file. Otherwise, an unexpected error may occur.
- All directories and files must be closed before a file system in Huawei LiteOS is unmounted. Forcible unmount operations will cause problems including but not limited to damages to file systems and devices, for which Huawei assumes no responsibility.
- All directories and files must be closed before an SD card is removed. Focibly removing an SD card will cause problems including but not limited to loss of data on the SD card and SD card damage, for which Huawei assumes no responsibility.

5.2.4 Programming Example

None.

5.3 NFS

5.3.1 Overview

Basic Concept

Network file system (NFS) enables various devices and operating systems to share files with each other through networks. For this reason, it is sometimes regarded as a file system service similar to a shared folder in Windows.

An NFS client can mount the directories shared by a remote NFS host to a local device and run programs and shared files without using resources of the local device, as if directories of the remote host are disks of the NFS client.

NFS reduces the demand of local workstations for disk space because storage devices including floppy drives, CD-ROM, and Zip® (A kind of disc driver and disk with high

Copyright © Huawei Technologies Co., Ltd.

memory density) can be used by other networked devices. This in turn reduces the number of removable media devices.

Operation Mechanism

Figure 1 depicts typical NFS networking.

Figure 5-2 NFS networking

NFS maps the NFS directory (\home\Huawei LiteOS\nfs) on an NFS host to the /nfs directory on an NFS client, and synchronize one directory with another.

5.3.2 Development Guidelines

Development Process

To use NFS functions, perform the following steps: (Every step resolved for detail)

1. **Setting Up the NFS Server**
2. **Setting a Board as an NFS Client**
3. **Using NFS to Share Files**

Setting Up the NFS Server

NOTE

The procedure of setting up an NFS server will be described with Ubuntu OS as an example.

Step 1 Install NFS server software.

Specify the Ubuntu download source and run the following command when network connectivity is good:

```
sudo apt-get install nfs-kernel-server
```

Step 2 Set and start the NFS server.

Add the following line to the /etc/exports NFS configuration file:

```
/home/yourusername *(rw,no_root_squash,async)
```

/home/yourusername is the root directory for NFS sharing.

Run the following command to start the NFS server:

```
sudo /etc/init.d/nfs-kernel-server start
```


Copyright © Huawei Technologies Co., Ltd.

Run the following command to restart the NFS server:

```
sudo /etc/init.d/nfs-kernel-server restart
```

----End

Setting a Board as an NFS Client

The NFS client in this guide refers to a device that runs Huawei LiteOS. Typically, such a device is an IPC board.

Step 1 Configure hardware connectivity.

Connect the NFS client to the network serving the NFS server. Allocate an IP addresses to the NFS client and NFS server. Be sure that the two IP addresses fall within the same address range. If the first three octets in an IP address are the same as those in another IP address, the two IP addresses are considered to fall within the same address range. For example, 10.67.212.178 and 10.67.212.3 are within the same address range.

To query the IP address of the IPC board running Huawei LiteOS, run the ifconfig command.

Step 2 Activate the network to establish good network connectivity between the NFS client and the NFS server.

Activate the Ethernet or another type of network, and run the ping command to check whether the server is pingable.

```
Huawei LiteOS # ping 10.67.212.178

 [0]Reply from 10.67.212.178: time=1ms TTL=63
 [1]Reply from 10.67.212.178: time=0ms TTL=63
 [2]Reply from 10.67.212.178: time=1ms TTL=63
 [3]Reply from 10.67.212.178: time=1ms TTL=63
--- 10.67.212.178 ping statistics ---
4 packets transmitted, 4 received, 0 loss
```

Step 3 Initialize the NFS client.

Run the following command:

```
Huawei LiteOS# mount 10.67.212.178:/home/sqbin/nfs /nfs nfs 1011 1000
```

If information similar to the following is displayed, the NFS client is successfully initialized.

```
Huawei LiteOS# mount 10.67.212.178:/home/sqbin/nfs /nfs nfs 1011 1000
 Mount nfs on 10.67.212.178:/home/sqbin/nfs
 Mount nfs finished.
```

The mount command mounts the /home/sqbin/nfs directory on the server with the IP address 10.67.212.178 to the /nfs directory on the NFS client.

Syntax of the mount command:

```
mount [SERVER_IP:SERVER_PATH] [CLIENT_PATH] nfs <uid gid>
```

SERVER_PATH indicates the path to the NFS shared directory on the NFS server. SERVER_IP indicates the IP address of the NFS server. CLIENT_PATH indicates the NFS path on the NFS client and can be set only to /nfs.

uid indicates the Linux user ID, and gid indicates the Linux group user ID. uid and gid are used to obtain the permission to access the directory on the NFS server. To query uid and gid, run the id Linux command. If NFS access control is not required, set the permission on the NFS root directory to 777.

Copyright © Huawei Technologies Co., Ltd.

```
chmod -R 777 /home/sqbin/nfs
```

The setting of the NFS client is completed, and the NFS is successfully mounted to the NFS client.

----End

Using NFS to Share Files

Create a dir directory and save it on the NFS server. Run the ls command in Huawei LiteOS.

```
Huawei LiteOS# ls /nfs
```

Information similar to the following is displayed:

```
Huawei LiteOS#
Huawei LiteOS# ls /nfs
 Directory /nfs:
 dir                <DIR>
```

The command output indicates that the dir directory on the NFS server has been synchronized to the /nfs directory on the client (Huawei LiteOS).

Likewise, files and directories on the NFS client can be accessed from the NFS server.

Platform Differences

Currently, NFS clients only meet parts of NFS v3 specifications and therefore cannot be fully compatible with NFS servers that also meet parts of NFS v3 specifications. During development and tests, you are advised to use the NFS servers for Linux that meet all specifications of NFS v3.

5.3.3 Precautions

- NFS files do not support permission control. When creating NFS directories and files, please set the directory and file permission to 777.
- No tasks can be blocked from reading or writing NFS files.
- NFS files do not support signal functions.
- NFS files support open, close, read, write, seek, dup, opendir, closedir, readdir, rewinddir, statfs, remove, mkdir, rmdir, rename, stat, stat64 and seek64 operations.
- Currently, NFS support UDP and TCP based socket communications. Defaults to TCP.
- The content of a file can be cleared only when you pass in the **O_TRUNC** parameter during the call to the open API and have the write permission on the file.
- After the name of file A is changed to the name of file B using rename() when both files are opened, the content of file A will be returned if you read the file descriptor of file B.
- NFS is a test function, the default configuration is closed, the official product is prohibited to use the function.
- Disclaimer: Huawei is not responsible for any risks brought by using NFS in official Huawei LiteOS.

5.3.4 Programming Example

None.

Copyright © Huawei Technologies Co., Ltd.

5.4 JFFS2

5.4.1 Overview

Basic Concept

Journal flash file system version 2 (JFFS2) manages journaled file systems on devices. JFFS2 is mainly used for NOR flash. It is readable and writable and supports data compression. In addition, it offers protection against panic and power failures, and is capable of write balancing.

Flash memory differs greatly from disks. Running a disk file system on flash memory might compromise system performance and security. With JFFS2, this problem will no longer be seen.

JFFS2 in Huawei LiteOS mainly manages files in NOR flash and supports multiple partitions.

5.4.2 Development Guidelines

Development Process

To use the functions of JFFS2, perform the following operations: (Every step resolve for detail)

1. **Adding JFFS2 Partitions**
2. **Mounting JFFS2**
3. **Unmounting JFFS2**
4. **Deleting JFFS2 Partitions**

Adding JFFS2 Partitions

Call the add_mtd_partition function to add JFFS2 partitions. This function automatically names device nodes. The naming rule for JFFS2 is /dev/spinorblk + partition number.

The add_mtd_partition function expects four parameters. The first parameter indicates the medium type (nand or spinor). nand is used for YAFFS2, and spinor is used for JFFS2 partitions.

The second parameter indicates the start address of a partition. The third parameter indicates the partition size. Both parameters are in hexadecimal format.

The fourth parameter indicates the partition number (0 – 19).

After the partitions are successfully added, you can run the partition jffs command in Shell to query information of the jffs block.

```
if(uwRet = add_mtd_partition("spinor", 0x100000, 0x800000, 0) != 0)
         dprintf("add jffs partition failed, return %d\n", uwRet);
 else
 {
     dprintf("Mount jffs2 on nand.\n");
     uwRet = mount("/dev/spinorblk0", "/jffs0", "jffs", 0, NULL);
     if(uwRet)
```


Copyright © Huawei Technologies Co., Ltd.

```
        dprintf("mount jffs err %d\n",uwRet);
        dprintf("Mount jffs2 on nor finished.\n");
 }
 if(uwRet = add_mtd_partition("spinor", 0x900000, 0x200000, 1) != 0)
        dprintf("add jffs partition failed, return %d\n", uwRet);
Huawei LiteOS# partition jffs
jffs partition num:0, dev name:/dev/spinorblk0, mountpt:/jffs0, startaddr:
0x0100000,length:0x0800000
jffs partition num:1, dev name:/dev/spinorblk1, mountpt:(null), startaddr:
0x0900000,length:0x0200000
```

Mounting JFFS2

Call the mount() function to mount a device node to the mount point.

This function expects five parameters. The first parameter indicates the device node and the second parameter indicates the mount point. Both parameters must be the same as the parameters used in the add_mtd_partition() function.

The third parameter indicates the file type.

The fourth parameter indicates the mount flag (default: 0), and the fifth parameter indicates the mount data (default: NULL). Alternatively, JFFS2 can be mounted by running the mount command in Shell, and you do not need to pass in the last two parameters.

Command for calling the mount() function:

```
Huawei LiteOS# mount /dev/spinorblk1 /jffs1 jffs
```

If information similar to the following is displayed, JFFS2 is successfully mounted:

```
Huawei LiteOS# ls
 Directory /:
 bin                  <DIR>
 dev                  <DIR>
 jffs0                <DIR>
 ramfs                <DIR>
 yaffs0               <DIR>

 Huawei LiteOS# mount /dev/spinorblk1 /jffs1 jffs
 Huawei LiteOS# ls
 Directory /:
 bin                  <DIR>
 dev                  <DIR>
 jffs0                <DIR>
 jffs1                <DIR>
 ramfs                <DIR>
 yaffs0               <DIR>
```

Now you can read from and write to NOR flash.

Unmounting JFFS2

Call the umount() function to unmount JFFS2 partitions. Only the input parameter mount point needs to be specified.

Command for calling the umount() function:

```
Huawei LiteOS# umount /jffs1
```

If information similar to the following is displayed, JFFS2 is successfully unmounted:

```
Huawei LiteOS# ls
 Directory /:
 bin                  <DIR>
```


Copyright © Huawei Technologies Co., Ltd.

```
dev                  <DIR>
jffs0                <DIR>
jffs1                <DIR>
ramfs                <DIR>
yaffs0               <DIR>
Huawei LiteOS# umount /jffs1
umount ok
```

Deleting JFFS2 Partitions

Call the delete_mtd_partition function to delete the unmounted JFFS2 partitions.

This function expects two parameters. The first parameter indicates the partition number, and the second parameter indicates the medium type. Both parameters must be the same as the parameters used in the add_mtd_partition() function.

```
uwRet =delete_mtd_partition(1,"spinor");
 if(uwRet != 0)
     printf("delte jffs error\n");
 else
     printf("delete jffs ok\n");
Huawei LiteOS# partition jffs
jffs partition num:0, dev name:/dev/spinorblk0, mountpt:/jffs0, startaddr:
0x0100000,length:0x0800000
```

Creating a JFFS2 Image

Use mkfs.jffs2 to create a JFFS2 image. The following is the default command for creating a JFFS2 image:

./mkfs.jffs2 -s 0x1000 -e 0x10000 -p 0x100000 -d rootfs/ -o rootfs_64k.jffs2

(The command is added to a script and will be automatically executed during image creation.)

Table 5-2 Command description

Command	Description
-s	Page size
-e	eraseblock size
-p	Image size
-d	Source directory of the file system image to be created
-o	Name of the image to be created

The parameters in the default command are merely illustrative. You can change the parameters when necessary.

5.4.3 Precautions

- JFFS2 manages files in NOR flash and calls the NOR flash drive interface. Before using JFFS2, ensure that NOR flash is present on hardware and the drive is successfully initialized (the value returned by spinor_init() is 0).

Copyright © Huawei Technologies Co., Ltd.

- The start address and the partition size are automatically aligned n the boundary according to the size of block. The valid partition number ranges from 0 to 19.
- JFFS2 images can be created by using mkfs.jffs2 commands in the fsimage/MakeVersion.sh file. Change parameter values in the commands when necessary. To query other commands, run the mkfs.jffs2 command.
- When you open a file by calling the open API and pass in the **O_TRUNC** parameter, the content of the file will be cleared.
- Currently, JFFS2 supports the open, close, read, write, seek, opendir, closedir, readdir, rewinddir, statfs, unlink, mkdir, rmdir, rename, stat, stat64 and seek64 operations.
- JFFS2 supports the read-only mounting attribute. If the input parameter **mountflags** of the mount function is set to **MS_RDONLY**, the read-only mounting attribute is enabled for JFFS2. All APIs used to write data, such as write, mkdir, and unlink, and files opened using the open function with the input parameter **oflags** set to a value other than **O_RDONLY** are rejected and the error code EACCESS is returned.
- Currently, JFFS2 does not support the open function whose **oflags** is set to **O_DIRECTORY**.

5.4.4 Programming Example

None.

5.5 FAT

5.5.1 Overview

Basic Concept

File allocation table (FAT) is classified into three types: FAT12, FAT16, FAT32, and exFAT. FAT divides a disk into five sectors: master boot record (MBR), dos boot record (DBR), FAT, DIR, and DATA.

FAT can be implemented on diverse media, especially on removable media storage devices including USB flash drives, secure digital memory cards (SD cards), and removable hard disks. It maintains good compatibility between embedded devices and desktop systems such as Windows and Linux, which is convenient for developers to manage operation files.

FAT in Huawei LiteOS has small amount of code and use less resources. It is tailorable and supports multiple types of physical media. In addition, it is compatible with operating systems including Windows and Linux and supports multiple devices and the identification of multiple partitions.

FAT in Huawei LiteOS supports disk partitioning. File operations can be performed on the primary partition and logical partitions. Huawei LiteOS is able to identify other types of file systems (such as NTFS) on the hard disk. Currently, only the master boot record (MBR) partition style is supported.

Huawei LiteOS supports the FAT32-based virtual partitioning feature. If the virtual partitioning feature is enabled, you can configure the number of virtual partitions, space percentage, and directory entry name for a specified physical partition by calling the allocation API. In addition, a corresponding folder is created in the root directory as the directory entry of the virtual partition based on the configured directory entry name.

Copyright © Huawei Technologies Co., Ltd.

Performing operations in the directory entry of a virtual partition means performing operations in this virtual partition.

If the space percentage of all virtual partitions does not reach 100%, you can perform write operations outside the directory entry of the virtual partition. In this case, the available space outside the virtual partition directory entry is the total space minus the space of all virtual partitions. If the space percentage of all virtual partitions reaches 100%, the write operation performed outside the directory entry is rejected due to insufficient space.

5.5.2 Development Guidelines

Development Process

The procedure for using the FAT file system is divided into the following steps. Steps 4-9 are optional and required only if the FAT-based virtual partitioning feature is in use.

1. **Identifying Devices**
2. **Mounting FAT**
3. **Unmounting FAT**
4. **(Optional) Checking Prerequisites for Virtual Partitioning**
5. **(Optional) Allocating Virtual Partitions**
6. **(Optional) Mounting and Unmounting Virtual Partitions**
7. **(Optional) Creating and Deleting Virtual Partitions**
8. **(Optional) Compatibility of Virtual Partitions**
9. **(Optional) Virtual Partition Error Codes**
10. **(Optional) API Behavior Change of the Virtual Partition**

Identifying Devices

```
Huawei LiteOS # ls
Directory /dev:
acodec                0
adec                  0
aenc                  0
ai                    0
ao                    0
console               0
fb0                   0
hi_gpio               0
hi_mipi               0
hi_rtc                0
hi_tde                0
i2c-0                 0
i2c-1                 0
i2c-2                 0
isp_dev               0
lcd                   0
logmpp                0
mem                   0
mmcblk0               0
mmcblk0p0             0
mmcblk0p1             0
mmcblk1               0
mmcblk1p0             0
```


Copyright © Huawei Technologies Co., Ltd.

- Configure _MULTI_PARTITION to 1 in the ffconf.h file to enable the identification of multiple partitions.
- Configure _VOLUMES to a value greater than 2 to enable the identification of multiple devices.

Now the OS is able to automatically identify the inserted SD cards. The automatically registered device nodes are listed in the figure above. mmcblk0 and mmcblk1 specify card 0 and card 1 respectively, which are independent master devices. mmcblk0p0, mmcblk0p1 specify two partitions of card 0 and serve as partition devices. When partition devices are available, the first partition device will be automatically invoked when the master device is used.

The information about the identified partitions can be queried by running the **partinfo** command.

```
Huawei LiteOS # partinfo /dev/sdap0
part info :
disk id          : 3
part_id in system: 0
part no in disk  : 0
part no in mbr   : 1
part filesystem  : 0C
part dev name    : sdap0
part sec start   : 2048
part sec count   : 167794688
```

Mounting FAT

Run the following command:

```
Huawei LiteOS# mount /dev/mmcblk0 /bin/vs/sd vfat
```

If information similar to the following is displayed, FAT is successfully mounted:

```
Huawei LiteOS# mount /dev/mmcblk0 /bin/vs/sd vfat
mount ok

Huawei LiteOS# ls
Directory /:
bin                  <DIR>
dev                  <DIR>
ramfs                <DIR>
yaffs0               <DIR>
```

Unmounting FAT

Run the following command:

```
Huawei LiteOS# umount /bin/vs/sd
```

If information similar to the following is displayed, FAT is successfully unmounted:

```
Huawei LiteOS# umount /bin/vs/sd
 umount ok
```

(Optional) Checking Prerequisites for Virtual Partitioning

NOTE

The virtual partitioning feature needs to be manually enabled in menuconfig. After this feature is enabled, the FAT file system of LiteOS preferentially uses the virtual partitioning feature to manage files.

Copyright © Huawei Technologies Co., Ltd.

If the physical partition used by LiteOS belongs to a storage device (such as an SD card, USB disk, and removable hard disk) has never used the virtual partitioning feature, ensure that:

- The physical partition is a FAT32 partition.
- No file exists in the physical partition.

If the physical partition used by LiteOS belongs to a storage device that has already used the virtual partitioning feature, ensure that:

- Directory entries of virtual partitions in the physical partition are not deleted, renamed, or corrupted.
- Data in the reserved sectors of the physical partition is complete.

If any of the preceding conditions is not met, the physical partition should use a regular FAT system instead of the virtual partitioning feature.

In LiteOS, the virtual partitioning feature is enabled for specific physical partitions only. You can specify the physical partition through the allocation API.

(Optional) Allocating Virtual Partitions

The allocation API is used to create virtual partitions that meet the requirements of the number of partitions and partition sizes. A virtual partition can be assumed as a physical partition in a specific scenario. You need to specify a partition device the allocation API and configure related parameters for the virtual partition. For a physical partition of a storage device that does not have virtual partitions in LiteOS, you can configure required parameters through the allocation API before enabling the virtual partitioning feature. In normal cases, after the allocation is successful, you can mount the virtual partition to the physical partition.

If the physical partition already has a virtual partition, and the virtual partition meets your requirements, you can directly manage the virtual partition based on its parameters. Configuring parameters through the allocation API is not necessary.

To enable the virtual partitioning feature after starting LiteOS, you need to call the los_set_virpartparam() API to configure the number of virtual partitions and key global parameters of the virtual partitions. Otherwise, subsequent virtual partition application will fail.

The definition of the allocation structure of a virtual partition is as follows:

```
#define _MAX_ENTRYLENGTH  16
#define _MAX_VIRVOLUMES   5

typedef struct VIRTUAL_PARTITION
{
    char *DevPartPath;
    int  VirPartNum;
    double VirPartPercent[_MAX_VIRVOLUMES];
    char VirPartName[_MAX_VIRVOLUMES][_MAX_ENTRYLENGTH + 1];
}VIR_PART;
```

No.	Definition	Description	Precautions
1	DevPartPath	Indicates the path of a device node.	Set this configuration item to the path of the selected device node. The value must be the device node path under **/dev**. The upper limit of the value is **DISK_NAME**.

Copyright © Huawei Technologies Co., Ltd.

No.	Definition	Description	Precautions
2	VirPartNum	Indicates the number of virtual partitions.	The minimum value is **1** and the maximum value is **_MAX_VIRVOLUMES**, that is, **5**.
3	VirpartPercent	Indicates the space percentage of the virtual partition.	The total value of all virtual partitions must be greater than 0.00 and less than or equal to 1.00. The value of each virtual partition must be greater than 0.00 and less than 1.00.
4	VirPartName	Indicates the directory entry of the virtual partition.	he maximum value of this configuration item is **_MAX_ENTRYLENGTH**, which indicates 16 characters. The value cannot be null. The uploaded directory entry set cannot contain duplicated names. The directory entry name cannot contain invalid characters.

After you configured virtual partition parameters of a specified physical device, the virtual partition configuration of the device is locked and cannot be modified. The virtual partition is mounted based on the configured parameters. After the virtual partition is deleted, the configuration is unlocked and can be modified.

NOTICE

If the specified physical partition has a virtual partition, and the original parameter configuration of the virtual partition is inconsistent with that configured through the allocation API, LiteOS manages the virtual partition based on the original parameter configuration.

(Optional) Mounting and Unmounting Virtual Partitions

The procedure for mounting a virtual partition is divided into two steps. The first step is to mount the FAT file system. After this step is complete, the virtual partitioning feature is then automatically applied to the physical partition of the FAT file system. If virtual partitioning is successful, the physical partition starts its virtual partitions and is managed as virtual partitions. If virtual partitioning fails, the physical partition is managed as a regular FAT file system, and a specified error code is returned to indicate the error that occurred during virtual partitioning.

After virtual partitions are successfully mounted, the directories based on the virtual partition quantity and name will be created in the root directory of the physical partition as directory entries of virtual partitions. Creating a file within the directory entry of a virtual partition means creating a file in this virtual partition. The maximum available space in the directory entry of a virtual partition is configured as a percentage.

The procedure for unmounting virtual partitions is the reverse of the mount procedure and therefore is not described again.

Copyright © Huawei Technologies Co., Ltd.

(Optional) Creating and Deleting Virtual Partitions

In LiteOS, after the virtual partitioning feature is used for a new physical partition, you can create virtual partitions by performing the mount operation and re-divide the physical partition repeatedly. The parameters of the created virtual partition are configured through the allocation API.

To re-divide a physical partition into virtual partitions, format the disk of the physical partition as the FAT32 file system. The physical partition is then re-divided into virtual partitions.

To cancel the virtual partitioning feature for a physical partition, format the physical partition. Virtual partition information is then deleted from the physical partition.

NOTICE

The formatting operation will cause a loss of user data. Exercise caution when performing this operation.

(Optional) Compatibility of Virtual Partitions

The virtual partitioning feature is the FAT-based file management mode specific for LiteOS. When performing operations in physical partitions to which the virtual partitioning feature has been applied on other platforms, ensure that you understand the following compatibility issues to avoid unexpected errors:

- Operations such as re-partitioning the storage medium and adjusting the size of the storage medium may cause the virtual partitioning feature applied to physical partitions to become invalid.
- When creating folders in the physical partitions to which the virtual partitioning feature has been applied on other platforms, pay attention to the following items:
  - If the created folder is not within the directory entry of a virtual partition, and the space percentage of all virtual partitions is 100%, no new files or sub folders can be created under the folder in LiteOS.
- When writing files into the physical partitions to which the virtual partitioning feature has been applied on other platforms, pay attention to the following items:
  - If the files are written outside the directory entry of a virtual partition, and the space percentage of all virtual partitions is 100%, these files can only be read in LiteOS.
  - If the files are written within the directory entry of a virtual partition and the cluster chains of the files are within the management scope of the virtual partition, these files can be normally read and written in LiteOS.
  - If the files are written within the directory entry of a virtual partition but part or all of the cluster chains of the files are not within the management scope of the virtual partition, these files can only be read in LiteOS.
- If you delete the directory entry of a virtual partition on other platforms, the virtual partition will be lost. The system will attempt to rebuild the lost virtual partition when you perform mount operations in LiteOS. When a folder with the same name as the deleted directory entry is created and you perform mount operations in LiteOS:
  - If the entire cluster chain of the folder is within the management scope of the virtual partition, the folder can be normally recognized.

Copyright © Huawei Technologies Co., Ltd.

- If part or all of the cluster chain of the folder is not within the management scope of the virtual partition, verifying the directory entry of the virtual partition will fail and the virtual partitioning feature cannot be applied.

- If you rename the directory entry of the virtual partition on other platforms, the virtual partition will be lost. The system will attempt to rebuild the lost virtual partition when you perform mount operations in LiteOS. The original files in the directory entry are not lost, but the virtual partitioning feature no longer takes effect on these files, and these files can only be read.

(Optional) Virtual Partition Error Codes

An error occurs in the application virtual partition only when the virtual partition is mounted or formatted. After operations on the basic FAT file system are completed, configure and apply the virtual partition. In this case, if a problem occurs or a condition is triggered, the error code of the virtual partition is transmitted to the upper layer, so that the situation in the application virtual partition is known. At the same time, the partition is managed as a basic FAT file system instead of a virtual partition.

NOTE

Returned values upon mount and formatting indicate the success or failure of the basic FAT file system. If the related interface returns value **0**, the corresponding operation of the basic FAT file system is successful. In this case, you can obtain the errno value and check whether an exception occurs during the subsequent application virtual partition. If the related interface returns value **-1**, the operation in the basic FAT file system fails. If the errno value that you obtained is a libc error code, this code indicates the cause of the basic FAT mount failure, not the virtual partition.

No.	Definition	Error Code	Description	Triggered By
1	VIRERR_MODIFIED	0x10000001	The reserved sector information of the virtual partition is damaged.	The partition is repartitioned, the partition size is adjusted, or the SD card is damaged.
2	VIRERR_CHAIN_ERR	0x10000002	The directory entry of the virtual partition has been removed from the virtual partition.	The directory entry of the virtual partition is deleted and then recreated on other platforms. This problem will be triggered in a certain probability.
3	VIRERR_OCCUPIED	0x10000003	The virtual partition entry is occupied by the file with the same name.	A root directory of the physical partition has a file with the same name, which hinders creation of the virtual partition entry.
4	VIRERR_NOTCLEAR	0x10000004	The physical partition is not empty.	A non-null virtual partition is applied to a new physical partition.
5	VIRERR_NOTFIT	0x10000005	The physical partition is not a FAT32 file system.	The physical partition is not a FAT32 file system.

Copyright © Huawei Technologies Co., Ltd.

No.	Definition	Error Code	Description	Triggered By
6	VIRERR_NOTMOUNTY	0x10000006	The device is not mounted.	The device is not mounted during formatting.
7	VIRERR_INTER_ERR	0x10000007	This is an internal error.	An error occurred during virtual partition application.
8	VIRERR_NOPARAM	0x10000008	The parameters are not configured for the virtual partition.	The allocation API is not called before performing the mounting operation.
9	VIRERR_PARMLOCKED	0x10000009	The virtual partition configuration is locked.	You attempt to modify virtual partition parameters after the configuration is locked.
10	VIRERR_PARMNUMERR	0x1000000A	The **VirPartNum** field of the allocation API is incorrect.	The **VirPartNum** field is incorrectly configured.
11	VIRERR_PARMPERCENTERR	0x1000000B	The **VirPartPercent** field of the allocation API is incorrect.	The **VirPartPercent** field is incorrectly configured.
12	VIRERR_PARMNAMEERR	0x1000000C	The **VirPartName** field of the allocation API is incorrect.	The **VirPartName** field is incorrectly configured.
13	VIRERR_PARMDEVERR	0x1000000D	The **DevPartPath** field of the allocation API is incorrect.	The **DevPartPath** field is incorrectly configured.

(Optional) API Behavior Change of the Virtual Partition

After the virtual partition feature is enabled, the behavior of some APIs changes in the virtual partition. The behavior of the following APIs changes only for the physical partition where the virtual partition feature has been enabled. For the physical partitions of FAT and other file systems where the virtual partition feature is disabled, the API behavior does not change.

Only the operation behavior of the following APIs changes in the virtual partition. The function invoking methods and parameters do not change.

Copyright © Huawei Technologies Co., Ltd.

No.	API	Behavior Change	Return Value and Error Code Description
1	mount	After the basic FAT partition is mounted, this API is used to check whether the current physical partition has a virtual partition. If the virtual partition has been enabled for the physical partition, this API will load the virtual partition using the configured information and repair the entry. If the physical partition does not have a virtual partition and meets the requirements for creating a virtual partition, this API is used to automatically create a virtual partition in the physical partition.	If the return value is **-1**, the basic FAT partition fails to be mounted. In this case, the libc error code is triggered to indicate the error. If the return value is **0**, the basic FAT partition is successfully mounted. In this case, If the virtual partition is successfully mounted, the error code is **0**. Otherwise, the error code may be **VIRERR_MODIFIED**, **VIRERR_CHAIN_ERR**, **VIRERR_OCCUPIED**, **VIRERR_NOTCLEAR**, **VIRERR_NOTFIT**, or **VIRERR_NOPARAM**.
2	format	After the FAT partition is mounted, if the physical partition is formatted into the FAT32 file system, this API is used to create a virtual partition in the physical partition after the formatting is completed. If the physical partition is formatted into a system other than FAT32, this API is used to clear the virtual partition in the physical partition after the formatting is completed. If the FAT partition is not mounted, this API is used to clear the virtual partition in the physical partition after the formatting is completed no matter what file systems are used.	If the return value is **-1**, the basic FAT partition fails to be formatted. In this case, the libc error code is triggered to indicate the error. If the return value is **0**, the basic FAT partition is successfully formatted. If the basic FAT partition is not mounted, the error code is **VIRERR_NOTMOUNT**. If the formatted partition is not the physical partition configured through the allocation API or no parameter is configured through the allocation API, the error code is **VIRERR_NOPARAM**. If the basic FAT partition is mounted and the error code can be **0** or **VIRERR_NOTFIT**. Error code **0** indicates that the virtual partition is successfully created.
3	open	See **(Optional) Compatibility of Virtual Partitions**.	None.
4	mkdir	See **(Optional) Compatibility of Virtual Partitions**.	None.

Copyright © Huawei Technologies Co., Ltd.

No.	API	Behavior Change	Return Value and Error Code Description
5	rmdir	After the virtual partition is successfully created in the current physical partition, this API cannot be used to modify the virtual partition directory entry.	None.
6	remove	After the virtual partition is successfully created in the current physical partition, this API cannot be used to delete the virtual partition directory entry.	None.
7	rename	After the virtual partition is successfully created in the current physical partition, this API cannot be used to rename the virtual partition directory entry.	None.

A dedicated API is added to the virtual partition feature, as shown in the following table.

No.	API	Function Description
1	los_set_virpartparam	This API is used to configure parameters for virtual partitions.
2	virstatfs	This API is used to obtain the status information of a specified virtual partition.

5.5.3 Precautions

- The default configurations can be used directly, and you can tailor the configurations based on your needs.
- The configuration items of FAT are in the ffconf.h file.
- If _FS_READONLY is set to 0, the read/write permission on FAT is read/write. If _FS_READONLY is set to 1, the read/write permission on FAT is read-only.
- If _USE_MKFS is set to 1 and _FS_READONLY is set to 1, the formatting function is enabled.
- If _FS_NORTC is set to 1, no real-time clocks exist, and the file creation time will be a fixed time point.
- _FS_LOCK specifies the number of files (folders) that can be opened concurrently.
- If a file is opened in write-only or read/write mode, the file will fail to be opened again before it is closed. To open the same file for multiple times, the file must be opened in

Copyright © Huawei Technologies Co., Ltd.

read-only mode. If a file is opened for a long time and is not closed, data of the file will be lost. The data can be saved only when the file is closed.

- Apart from exFAT-formatted files, the size of a file cannot be greater than 4 GB.
- The total length of file names and path names must not exceed 252 bytes.
- If two SD card slots are available, the SD card that is inserted first is card 0; the one inserted later is card 1.
- When the double-card and multi-partitioning function is enabled and multiple partitions exist, the /dev/mmcblk0 master device node registered by card 0 and the /dev/mmcblk0p0 slave device node registered by card 0 control the same device. Operations are forbidden to be performed on the master device node.
- When the double-card and multi-partitioning functions are enabled, and multiple partitions do not exist, the /dev/mmcblk0 and /dev/mmcblk0p0 device nodes control the same device, and only one of the device nodes can be mounted.
- FAT does not support opening a directory using open() + O_DIRECTORY. Use opendir() to open a directory.
- The read pointer and write pointer are not separated. Therefore, after a file is opened in O_APPEND mode, the read pointer is positioned at the end of the file. Manually put the read pointer to the start of the file before reading a file.
- Calling the stat or lstat functions on FAT will return the modification time of a file. Currently, the file creation time and last access time are not returned. The Microsoft FAT protocol supports only time after A.D. 1980.
- When you open a file by calling the open API and pass in the **O_TRUNC** parameter, the content of the file will be cleared.
- FAT supports the open, close, read, write, seek, sync, opendir, closedir, readdir, rewinddir, statfs, virstatfs, unlink, mkdir, rmdir, rename, stat, stat64, utime, chattr, seek64, getlabel, fallocate, fallocate64, truncate, truncate64 and fscheck operations.
- To prevent SD card exceptions and memory leakages, opened files and directories must be closed and mounted nodes must be unmounted before an SD card is removed.
- FAT supports the read-only mounting attribute. If the input parameter **mountflags** of the mount function is set to **MS_RDONLY**, the read-only mounting attribute is enabled for FAT. All APIs (except format) used to write data, such as write, mkdir, and unlink, and files opened using the open function with the input parameter **oflags** set to a value other than **O_RDONLY** are rejected and the error code EACCESS is returned.

5.5.4 Programming Example

None.

5.6 YAFFS2

5.6.1 Overview

Basic Concept

Yet another flash file system version 2 (YAFFS2) is an open-source embedded file system designed for NAND flash. In YAFFS, the minimum storage unit is page.

Copyright © Huawei Technologies Co., Ltd.

Two versions of YAFFS are available in current YAFFS file system: YAFFS and YAFFS2. The major difference between them is that YAFFS2 supports NAND flash chips with 2 KB pages, far larger than the chips with 512-byte pages supported by YAFFS. Another difference lies in the fact that YAFFS2 has a 64-byte spare area to store bad block information and carry out error checking and correction (ECC).

- YAFFS2 is suitable for large-capacity storage devices. It is specially designed for NAND flash to keep the latter efficient and robust.
- YAFFS2 supports chips with 2 KB pages. Moreover, it uses less memory, and allows faster read/write and junk data reclamation.
- Wear leveling and power failure protection ensures that data will not be corrupted when a power outage occurs amid file system modification.

YAFFS2 in Huawei LiteOS supports multiple partitions.

5.6.2 Development Guidelines

Development Process

In Huawei LiteOS, ulti-partitioning of YAFFS2 is realized using a doubly linked list. To use the functions of YAFFS2, perform the following steps: (Every step resolve for detail)

1. **Calling add_mtd_partition to Create Partitions**
2. **Calling mount to Mount Partitions**
3. **Calling umount to Unmount Partitions**
4. **Calling delete_mtd_partition to Delete Partitions**

Specific examples can be see in part programming examples.

Calling add_mtd_partition to Create Partitions

The add_mtd_partition function automatically names device nodes. The naming rule for YAFFS2 is /dev/nandblk + partition number.

The add_mtd_partition function expects four parameters. The first parameter indicates the media (nand or spinor). nand is used for YAFFS2 partitions, and spinor is used for JFFS2.

The second parameter indicates the start address of a partition. The third parameter indicates the partition size. Both parameters are in hexadecimal format.

The fourth parameter indicates the partition number (0 – 19).

After the partitions are successfully created, you can run the yaffspar command in Shell to query information of the partitions.

Command for creating partitions:

```
if(uwRet = add_mtd_partition("nand", 0x900000, 0x200000, 0) < 0)
         dprintf("add yaffs partition failed, return %d\n", uwRet);
     }

 if(uwRet = add_mtd_partition("nand", 0xb00000, 0x200000, 1) < 0)
         dprintf("add yaffs partition failed, return %d\n", uwRet);
```

Calling mount to Mount Partitions

Call the mount() function to mount a device node to the mount point.

Copyright © Huawei Technologies Co., Ltd.

This function expects five parameters. The first parameter indicates the device node and the second parameter indicates the mount point. Both parameters must be the same as the parameters used in the add_mtd_partition() function.

The third parameter indicates the file type (yaffs or jffs).

The fourth parameter indicates the mount flag (default: 0), and the fifth parameter indicates the private data (default: NULL).

Alternatively, partitions can be mounted by running the mount command in Shell, and you do not need to pass in the last two parameters.

Command for calling the mount() function:

```
Huawei LiteOS# mount /dev/nandblk1 /yaffs1 yaffs
```

If information similar to the following is displayed, partitions are successfully mounted:

```
  Huawei LiteOS# mount /dev/nandblk1 /yaffs1 yaffs

   start-blk:24, end-blk:39
   Huawei LiteOS# partition yaffs
   yaffs partition num:0, dev name:/dev/nandblk0, mountpt:/yaffs0, startaddr:
0x0900000,length:0x0200000
   yaffs partition num:1, dev name:/dev/nandblk1, mountpt:/yaffs1, startaddr:
0x0b00000,length:0x0200000
```

Calling umount to Unmount Partitions

Call the umount() function to unmount YAFFS2 partitions. Only the input parameter mount point needs to be specified. Alternatively, partitions can be unmounted by running the umount command in Shell.

Command for calling the umount() function:

```
Huawei LiteOS# umount /yaffs1
```

If information similar to the following is displayed, partitions are successfully unmounted:

```
Huawei LiteOS# umount /yaffs1
 umount ok

 Huawei LiteOS# umount /yaffs0
 umount ok

 Huawei LiteOS# partition yaffs
 yaffs partition num:0, dev name:/dev/nandblk0, mountpt:(null), startaddr:
0x0900000,length:0x0200000
 yaffs partition num:1, dev name:/dev/nandblk1, mountpt:(null), startaddr:
0x0b00000,length:0x0200000
```

Calling delete_mtd_partition to Delete Partitions

Partitions must be unmounted before they are deleted.

The delete_mtd_partition function expects two parameters. The first parameter indicates the partition number, and the second parameter indicates the medium type. Both parameters must be the same as the parameters used in the add_mtd_partition() function.

```
uwRet = umount("/yaffs1");
 if(uwRet != 0)
     printf("umount error:%d\n", uwRet);
 else
     printf("umount ok\n");
```


Copyright © Huawei Technologies Co., Ltd.

```
uwRet =delete_mtd_partition(1, "nand");
if(uwRet != 0)
    printf("delte yaffs error\n");
else
    printf("delete yaffs ok\n");
```

If information similar to the following is displayed, partitions are successfully deleted:

```
Mount yaffs2 on nand
 start-blk:24,end-blk:39
 umount ok
 delete yaffs ok
```

5.6.3 Precautions

- YAFFS2 in Huawei LiteOS supports multiple partitions. You can allocate the flash memory according to actual needs. The start address of partitions can be flexibly configured, which means that partitions can be created anywhere in the flash memory provided that you know where the start address is. A mechanism is in place to check for partition overlapping. However, it is beyond the scope of Huawei LiteOS to check for address overlapping due to the concurrent use of multi-partitioning and other features.
- The minimum partition for adding is one size of block, but the minimum partition for mounting is 9 block size (decided by characters of YAFFS2). Divide these two concepts.
- YAFFS2 in Huawei LiteOS automatically aligns addresses and partitions on the boundary according to the size of block. The valid partition number ranges from 0 to 19.
- Before using a partition, you are advised to erase data in the partition.
- YAFFS2 in Huawei LiteOS only allows you to continuously open 20 directories.
- The content of a file can be cleared only when you pass in the **O_TRUNC** parameter during the call to the open API and have the write permission on the file.
- YAFFS2 supports the open, close, read, write, seek, sync, dup, opendir, closedir, readdir, rewinddir, statfs, unlink, mkdir, rmdir, rename, stat, stat64 and seek64 operations.
- YAFFS2 supports the read-only mounting attribute. If the input parameter **mountflags** of the mount function is set to **MS_RDONLY**, the read-only mounting attribute is enabled for YAFFS2. All APIs used to write data, such as write, mkdir, and unlink, and files opened using the open function with the input parameter **oflags** set to a value other than **O_RDONLY** are rejected and the error code EACCESS is returned.
- Currently, YAFFS2 does not support the open function whose **oflags** is set to **O_DIRECTORY**.

5.6.4 Programming Example

None.

5.7 RAMFS

5.7.1 Overview

Basic Concept

RAM file system (RAMFS) is a file system as a dynamically re-sizable RAM-based file system. RAMFS does not have backup storage resources. When files are written into

Copyright © Huawei Technologies Co., Ltd.

RAMFS, directory entries and page caches will be allocated, but data is not written to any other storage media. Data will be lost when the power is off.

RAMFS places all files in a RAM. Files are read from and written into the RAM. RAMFS can be used to store temporary data or data that needs to be frequently modified, for example, the /tmp and the /var directories, to accelerate the read/write speed and avoid frequent access to storage.

RAMFS in Huawei LiteOS is a simple file system. It functions as a RAM-based cache mechanism of dynamic file systems.

RAMFS in Huawei LiteOS is based on VFS, can not be formatted.

5.7.2 Development Guidelines

Procedure

To use the functions of RAMFS, perform the following operations:

1. **Initializing RAMFS**
2. **Mounting RAMFS**
3. **Unmounting RAMFS**

Initializing RAMFS

```
void ram_fs_init(void)
 {
     int swRet=0;
     swRet = mount(NULL, RAMFS_DIR, "ramfs", 0, NULL);
     if (swRet) {
         dprintf("mount ramfs err %d\n", swRet);
         return;
     }
     dprintf("Mount ramfs finished.\n");
 }
```

Call the initial function. If information similar to the following is displayed while Huawei LiteOS is being started, RAMFS is successfully initialized:

```
Mount ramfs finished
```

Mounting RAMFS

Run the following command:

```
Huawei LiteOS# mount 0 /ramfs ramfs
```

If information similar to the following is displayed, RAMFS is successfully mounted:

```
Huawei LiteOS# mount 0 /ramfs ramfs
```

Unmounting RAMFS

Run the following command:

```
Huawei LiteOS# umount /ramfs
```

If information similar to the following is displayed, RAMFS is successfully unmounted:

Copyright © Huawei Technologies Co., Ltd.

```
Huawei LiteOS# umount /ramfs
 umount ok
```

5.7.3 Precautions

- Read and write pointers are not separated in RAMFS file system, so while opening the file by using O_APPEND(read added) method, the read pointer is also at the end of the file. Users need to set position manually before reading files.
- Because of the fixed memory space used in RAMFS, RAMFS is able to be mounted only one time for avoid stepping on memory. After successfully mounting one time, followers can not continue being mounted on other directories.
- In RAMFS, filename length and directory name length must not exceed the length specified by RAMFS_NAME_MAX.
- When you open a file by calling the open API and pass in the **O_TRUNC** parameter, the content of the file will be cleared.
- RAMFS supports the open, close, read, write, seek, opendir, closedir, readdir, rewinddir, statfs, unlink, mkdir, rmdir, rename, stat, stat64 and seek64 operations.
- RAMFS is a test function, the default configuration is closed, the official product is prohibited to use the function.
- Disclaimer: Huawei is not responsible for any risks brought by using the Telnet function in official Huawei LiteOS.

5.7.4 Programming Example

None.

5.8 PROC

5.8.1 Overview

Basic Concept

PROC is a pseudo file system that exists only in memory and does not use external storage. It provides an interface for accessing the data in Huawei LiteOS Kernel.

PROC in Huawei LiteOS is a virtual file system and does not support multi-threaded operations.

5.8.2 Development Guidelines

Development Process

To use the functions of PROC, perform the following operations:

1. **Initializing PROC**

2. **Querying and Modifying PROC Node Information**

Initializing PROC

```
void proc_fs_init(void)
 {
```


Copyright © Huawei Technologies Co., Ltd.

```
    int swRet=0;
    swRet = mount(NULL, PROCFS_DIR, "procfs", 0, NULL);
    if (swRet) {
        dprintf("mount procfs err %d\n", swRet);
        return;
    }
    dprintf("Mount procfs finished.\n");
}
```

Call the initial API. If information similar to the following is displayed while Huawei LiteOS is being started, PROC is successfully initialized:

```
Mount procfs finished
```

Creating a PROC Node

Call the create_proc_entry function to create a file node. The first parameter specifies the name of the node to be created. The second parameter specifies the file mode, including the file type and permission. The third parameter specifies the parent node of the node to be created. If **NULL** is passed in as the value of the third parameter, the parent node is /proc by default. This function will return the created PROC file node.

```
struct proc_dir_entry *pHandle;

pHandle = create_proc_entry("mounts", 0, NULL);
if (pHandle == NULL) {
    dprintf("creat mounts error!\n");
    return;
}
```

Call an operation function to perform a specific operation on the created node.

```
pHandle->proc_fops = &mounts_proc_fops;
```

Call the default operation function proc_file_default_operations if other operation functions are unavailable.

Call the proc_mkdir function to create a directory node. The first parameter specifies the name of the node to be created. The second parameter specifies the parent node of the node to be created. If **NULL** is passed in as the value of the second parameter, the parent node is /proc by default. This function will return the created PROC directory node.

```
struct proc_dir_entry *pHandle;

pHandle = proc_mkdir("test", NULL);
if (pHandle == NULL) {
    dprintf("creat test error!\n");
    return;
}
```

Querying and Modifying PROC Node Information

Run the cat command to query information of a PROC node. For example:

```
Huawei LiteOS # cat /proc/umap/logmpp
```

If information similar to the following is displayed, the query is successful:

```
Huawei LiteOS # cat /proc/umap/logmpp
Huawei LiteOS # -----LOG BUFFER STATE---------------------------------
 MaxLen ReadPos WritePos ButtPos
  64(KB)      0      113 65536

 -----CURRENT LOG LEVEL-----------------------------------------------
 vb : 3
```


Copyright © Huawei Technologies Co., Ltd.

```
 sys : 3
 region : 3
 chnl : 3
 vpss : 3
 venc : 3
 vda : 3
 h264e : 3
 jpege : 3
 vou : 3
 viu : 3
 rc : 3
 aio : 3
 ai : 3
 ao : 3
 aenc : 3
 adec : 3
 isp : 3
 ive : 3
 tde : 3
 vgs : 3
 h265e : 3
```

Run the writeproc command to modify information of a PROC node. For example:

```
Huawei LiteOS # writeproc 'sys=2' >> /proc/umap/logmpp
```

After the sys level is changed, information similar to following is displayed:

```
sys=2 >> /proc/umap/logmpp
```

Run the cat command again to query node information. From the displayed node information, you will see that the sys level has been changed.

```
Huawei LiteOS # cat /proc/umap/logmpp
Huawei LiteOS # -----LOG BUFFER STATE---------------------------------
 MaxLen ReadPos WritePos ButtPos
  64(KB)      0        0 65536

 -----CURRENT LOG LEVEL-----------------------------------------------
 vb : 3
 sys : 2
 region : 3
 chnl : 3
 vpss : 3
 venc : 3
 vda : 3
 h264e : 3
 jpege : 3
 vou : 3
 viu : 3
 rc : 3
 aio : 3
 ai : 3
 ao : 3
 aenc : 3
 adec : 3
 isp : 3
 ive : 3
 tde : 3
 vgs : 3
 h265e : 3
```

5.8.3 Precautions

- In the open command that is used to open files in PROC, only read and write properties take effect.
- In the fseek function, SEEK_END does not take effect, indicating that the offset started from the end of a file is not supported.

Copyright © Huawei Technologies Co., Ltd.

- If the ls command is run to query files in PROC after PROC is initialized, the size of each file is displayed as zero. The file size can be correctly displayed only after the cat command is run to retrieve real-time kernel information.
- PROC in Huawei LiteOS cannot be unmounted and does not support creating and deleting files or directories.
- In PROC, a file name or a directory name contains a maximum of 32 characters.
- PROC in Huawei LiteOS supports the open, close, read, write, seek, opendir, closedir, readdir, rewinddir, statfs, unlink, mkdir, rmdir, rename, stat, stat64 and seek64 operations.
- PROC, as a debugging function, is disabled by default, and do not use it in commercial products.
- Disclaimer: Huawei does not take the liability for the risks caused by using PROC in commercial products.

5.8.4 Programming Example

Example Description

Create the **proc_example** file in the **/proc** directory. The file can be opened, written, read, or sought.

1. Call the **proc_example_init()** function to create the **proc_example** file.

   The **proc_example_init()** function needs to be called after the **proc_fs_init()** function is called.
2. Call the **proc_test()** function to read the **proc_example** file in the **/proc** directory.
3. Obtain specific contents by calling the **proc_test()** function to read the **proc_example** file.

   The obtained contents can be defined by calling the **example_proc_show()** function. When the **proc_example** file is opened, the **example_proc_show()** function has been registered with the **example_proc_open()** function. When reading the **proc_example** file, the **seq_read()** function first calls the **example_proc_show()** function, copies contents in the **seq_file** file to buf, and returns the contents.

Example Code

```
#include "proc_fs.h"
#include "linux/seq_file.h"
#include "stdio.h"
#include "stdlib.h"
#include "fcntl.h"
static int example_proc_show(struct seq_file *m, void *v)
{
    PRINTK("this is example_proc_show\n");
    seq_printf(m,"%s%s\n","my test proc file1!", "f");
    return 0;
}
static int example_proc_open(struct inode *inode_ptr, struct proc_file *pf)
{
    PRINTK("this is example_proc_open\n");
    return single_open(pf, example_proc_show, NULL);
}
static const struct proc_file_operations example_proc_fops = {
    .open       = example_proc_open,
    .read       = seq_read,
    .llseek     = seq_lseek,
    .release    = seq_release,
```


Copyright © Huawei Technologies Co., Ltd.

```
    .write      = NULL,
};
void proc_example_init(void)
{
    struct proc_dir_entry *pH;
    pH = create_proc_entry("proc_example", 0, (struct proc_dir_entry *)NULL);
    if (pH == NULL) {
        PRINT_ERR("creat mounts error!\n");
        return;
    }
    PRINTK("proc_example_init end\n");
    pH->proc_fops = &example_proc_fops;
}
void proc_test(void)
{
    int fd = open("/proc/proc_example", O_NONBLOCK | O_CREAT | O_RDWR, 0777);
    if(fd < 0)
    {
        PRINTK("fd open err\n");
        return;
    }
    char buf[50]= {0};
    int len = read(fd, buf, 50);
    PRINTK("read data:%s\n", buf);
    close(fd);
}
```

Verification

```
proc fs init ...
Mount procfs finished.
proc_example_init end
this is example_proc_open
this is example_proc_show
read data:my test proc file1!f
```

Complete Example Code

proc_test.c

5.9 ROMFS

5.9.1 Overview

ROM filesystem (romfs) is a read-only file system. romfs provides simple functions and occupies less system resources. It is initially designed to provide a more space-saving file system than a common file system in the boot disk scenario. romfs does not support data writing into files but only supports data reading from files.

When creating a romfs, you need to use genromfs, which can be installed on the Linux OS. In the Linux OS, genromfs can be used to make the directory and its included files into a romfs image and burn the romfs image into an SD card or a USB flash drive. In addition, the romfs image can be mounted to the LiteOS with genromfs.

5.9.2 Development Guidelines

Development Procedure

The three steps for the use of a romfs are as follows (for details about specific operations, see the following sections):

Copyright © Huawei Technologies Co., Ltd.

1. Create a romfs.
2. Install the romfs.
3. Uninstall the romfs.

Creating a romfs

Insert an SD card or a USB flash drive into a computer with the Linux OS. A device, for example, the **sdal** device occurs in the **/dev/** directory after the Linux OS identifies the SD card or the USB flash drive. Create a directory on the computer and copy files that need to be made into a romfs to the directory. Use genromfs to make the directory and its included files into a romfs image. The following is an example, assume that the directory name is **romfs_dir** and the device name is **sda1**.

```
sudo genromfs -d romfs_dir -f /dev/sda1
```

Installing the romfs

Run the following command to check whether a device is displayed on the LiteOS, for example, **mmcblk0p0** in the **/dev/** directory:

```
Huawei LiteOS# mount /dev/mmcblk0p0 /romfs romfs
```

If the following command output is displayed, the romfs is successfully installed.

```
mount ok
```

Uninstalling the romfs

Run the following command:

```
Huawei LiteOS# umount /romfs
```

If the following command output is displayed, the romfs is successfully uninstalled.

```
umount ok
```

5.9.3 Precautions

- The maximum size of a romfs is 2 GB.
- The maximum size of a single file in a romfs is 256 MB.
- The length of the directory name or each file name in a romfs cannot exceed 16 bytes.
- A file can be opened only when the third input parameter is set to **O_RDONLY**.
- ROMFS supports the open, close, read, seek, dup, opendir, closedir, readdir, rewinddir, statfs, stat and stat64 operations.
- A romfs supports hard links instead of soft links.

5.9.4 Programming Example

None.

Copyright © Huawei Technologies Co., Ltd.

6 Driver Development

6.1 Driver Framework

6.2 File Adaptation

6.1 Driver Framework

6.1.1 Overview

Basic Concepts

Driver development indicates implement and abstract hardware according to certain specifications, so as to provide a set of unified APIs for developers to develop up-layer services.

1. **Driver Device**

   All external devices are treated as device files, a kind of special files, by the system. External devices can be classified as follows:

   Character device: the sending and receiving of devices are in the form of characters and random access is not supported.

   Block device: a block of system memory is used as the buffer. All data is transmitted through the buffer. Block devices are design for slow devices such as flash, preventing CPU wasting too much time waiting.

   Network devices: all visits to network hardware are through APIs, which are a set of operations used to process the sending and receiving of basic data for all types of network hardware.

2. **Driver Program**

   APIs between kernel and external devices. Driver programs shield details of hardware implementation and enables applications to operation external devices as common files. Hardware devices are regarded as device files to applications. Applications can perform operations such as open (), close (), read (), and write () on hardware devices.

3. **Driver Framework Management**

   Driver framework management is a module used to manage driver device and driver program.

Copyright © Huawei Technologies Co., Ltd.

Figure 6-1 Basic device driver framework

Operation Mechanism

Drive framework involves following key data structures:

1. **bus_type**: describes the bus

   bus_type represents a bus,. Multiple devices and drivers in the system are connected using buses. bus_type contains LOS_DL_LIST device_list and LOS_DL_LIST driver_list, representing driver device linked list and driver program linked list connected to the bus respectively. The system uses a bus descriptor to locate the driver device mounted to the bus, and identify supported driver programs of different devices.

   When the system is started, bus module will create a descriptor for each bus type and link driver devices and driver programs of the bus to the device_list and driver_list of the bus descriptor.

   Platform is an instance of the bus. Platform is a virtual bus, using which users can register and call driver devices and driver programs. For details, see the development process.

2. **device**: describes driver devices

   device represents a driver device. struct device contains struct bus_type *bus and struct device_driver *driver. bus indicates the bus that mounts the driver device. driver indicates the driver program of the driver device. Each driver device can only mount one driver program.

3. **device_driver**: describes driver programs

   device_driver represents a device driver program. struct device_driver contains struct bus_type *bus and struct list_head devices. bus indicates the bus connected to the driver.

Copyright © Huawei Technologies Co., Ltd.

devices linked list records actual driver devices that use the current driver program. A driver program supports multiple driver devices.

Figure 6-2 describes between bus_type, device, and device_driver

Figure 6-2 Relationships between bus_type, device, and device_driver

There may be multiple bus bus_types in the system. Each bus_type may contain multiple devices, and each device has its corresponding device_driver.

bus_type, equivalent to a container, manages devices and device_drivers. bus_type contains a device set and a device_driver set, respectively representing all devices and device drivers mounted to the bus.

device_driver, mounted to a bus_type, contains a device set that consists of all devices operated or controlled by the driver program. device_driver contains a bus_type pointer, indicating the bus to which the driver program is mounted.

Device, mounted to a bus_type, contains a device_driver pointer that indicates the device driver program of the device. device contains a bus_type pointer, indicating the bus to which the device is mounted.

NOTE

An actual bus in the device driver model is represented by bus_type and device. bus_type indicates the bus type, and device indicates the bus device, indicating that the bus is essentially a device.

6.1.2 Development Guidelines

Function

The following table lists functions provided by Huawei LiteOS driver framework

Function Type	API	Description
Bus	bus_register	Registers a bus
	bus_unregister	Deregisters a bus

Copyright © Huawei Technologies Co., Ltd.

Function Type	API	Description
Device	device_register	Registers a device and add the device to the bus device list
	device_unregister	Deregisters a device and removes the device from the device list of the bus
	dev_get_platdata	Obtains platform data
	dev_set_platdata	Configures platform data
	dev_get_drvdata	Obtains the driver count of a device
	dev_set_drvdata	Configures the driver data of a device
Driver	driver_register	Registers a driver. Traverses all devices on the bus and maps them with drivers. After the mapping succeeds, add drivers to driver list, and bind devices and drivers.
	driver_unregister	Deregisters a driver.
Platform	platform_driver_register	Registers platform drivers
	platform_driver_unregister	Deregisters platform drivers
	platform_device_register	Registers a platform device
	platform_device_unregister	Deregisters a platform device
	platform_get_resource	Obtains platform resources
	platform_get_irq	Obtains IRQ
	platform_ioremap_resource	Obtains the io map address

Development Procedure

The I2C driver registration to the platform bus is used as an example.

Step 1 Analyze and extract I2C non-driver attribute.

I2C is a universal driver and has implementation principles similar to other universal drivers. I2C needs to adapt to the system configurations such as register base address, clock frequency, and pin multiplexing. The feature data is configured as the I2C driver device attributes.

Step 2 Register a driver device.

Copyright © Huawei Technologies Co., Ltd.

1. Define the I2C driver device instance.
```
struct platform_device hisilicon_device_i2c0 = {
.name        = I2C_DRIVER_NAME,
.id          = 0,
.resource    = hisilicon_i2c0_resources,
.num_resources  = ARRAY_SIZE(hisilicon_i2c0_resources),
};
```
2. Register the driver device with the platform.
```
platform_device_register(&hisilicon_device_i2c0 );
```
3. Perform preloading operations.

   Invoke the preloading mounting functions in linux/module.h to mount driver device registration function to the system. When the system is started, preloading nodes are traversed according to the priority of mounting functions, in order to register driver devices. Following pre-loading mounting functions with the descending priority are provided by the system:
```
pure_initcall(f)        //0
core_initcall(f)        //1
postcore_initcall(f)    //2
arch_initcall(f)        //3
subsys_initcall(f)      //4
fs_initcall(f)          //5
device_initcall(f)      //6
late_initcall(f)        //7
```
   For details about how to invoke these functions, see **6.1.4编程实例**.

NOTICE

The execution of preloading may affect some system functions. For example, if driver initialization takes a long time, the fast startup phase of scatter loading may be affected. Currently, only pre-loading functions with the priority level less than or equal to 3 are supported.

Step 3 Register a driver program.

1. Define I2C driver program instance.
```
static struct platform_driver i2c_hisilicon_driver = {
.probe      = i2c_hisilicon_probe,
.remove     = i2c_hisilicon_remove,
.driver     = {
.name   = I2C_DRIVER_NAME,
.pm = &i2c_hisilicon_dev_pm_ops,
},
};
```

NOTE

probe is used to initialize the driver program.

remove is used to deinitialize the driver program.

The **name** attribute of driver program must be consistent with the **name** of the corresponding driver device.

2. Register the driver program with the platform.
```
platform_driver_register(&i2c_hisilicon_driver);
```
3. Perform preloading operations. For details about how to invoke these functions, see **参考6.1.4编程实例**.

Step 4 Register driven instances to the file system.

Copyright © Huawei Technologies Co., Ltd.

Driver instances are registered with the file system, so that users are allowed to perform operations to drivers as to files.

For details, see **6.2 File Adaptation**.

----End

6.1.3 Precautions

- If you want to invoke the preloading driver function, add the **–u** to the **LITEOS_TABLES_DRIVER_LDFLAGS** of **build/mk/liteos_tables_ldflags.mk**. Otherwise, linker will optimize driver initialization module, causing unpredictable errors.

```
LITEOS_TABLES_DRIVER_LDFLAGS := \
-ui2c_init \
-ugpio_init \
-uregulator_init \
-umtd_init_list \
```

6.1.4 Programming Example

Example Description

Add I2C driver to platform bus.

Example Code

```
/*-------------------------Driver device-----------------------------*/
static struct resource hisilicon_i2c0_resources[] = {
{
.start  = I2C0_REG_BASE,         //Register base address of I2C0
.end    = I2C0_REG_BASE + 0xA0,  //The length of the I2C0 registers is 0xA0.
.flags  = IORESOURCE_MEM,        //Register base address or memory segment of
IORESOURCE_MEM flag.
}, {
.start  = IRQ_I2C0,              //Interrupt ID of I2C0
.end    = IRQ_I2C0,
.flags  = IORESOURCE_IRQ,        //Interrupt ID of IORESOURCE_IRQ flag.
},
};

struct platform_device hisilicon_device_i2c0 = {
.name       = I2C_DRIVER_NAME,    //The driver program name of I2C is
I2C_DRIVER_NAME.
.id         = 0,
.resource   = hisilicon_i2c0_resources,
.num_resources  = ARRAY_SIZE(hisilicon_i2c0_resources),
};

struct i2c_hisilicon_platform_data hisilicon_pdata_i2c0 = {
.freq = CLK_LIMIT_DEFAULT,         //Clock frequency of I2C.
};

void  hisilicon_register_device(struct platform_device *dev, void *data)
{
int ret;

dev->dev.platform_data = data;      //Assign the hisilicon_pdata_i2c0 as the
platform_data value of driver program.

ret = platform_device_register(dev);//Register I2C driver to platform bus.
if (ret)
dprintf("unable to register hisilicon device %s: %d\n", dev->name, ret);
}
```


Copyright © Huawei Technologies Co., Ltd.

```
void  hisilicon_set_i2c_info(struct platform_device *device, struct
i2c_hisilicon_platform_data *info)
{
hisilicon_register_device(device, info);
}

int machine_init(void)
{
#ifdef LOSCFG_DRIVERS_I2C
#ifdef ENABLE_I2C0
hisilicon_set_i2c_info(&hisilicon_device_i2c0, &hisilicon_pdata_i2c0);
#endif
#endif
return 0;
}

arch_initcall(machine_init);    //Register preloading driver device.
                                //When kernel is being initialized, registered
drivers will be preloaded.
/*----------------------Driver device----------------------------*/

/*----------------------Driver program----------------------------*/
static int i2c_hisilicon_probe(struct platform_device *dev)
{
return i2c_hisilicon_init(dev);  //Initialize I2C driver program using the
function.
}

static int i2c_hisilicon_remove(struct platform_device *dev)
{
return i2c_hisilicon_deinit(dev); //De-initialize I2C using the function.
}

static struct platform_driver i2c_hisilicon_driver = {
.probe      = i2c_hisilicon_probe,
.remove     = i2c_hisilicon_remove,
.driver     = {
.name   = I2C_DRIVER_NAME,
.pm = &i2c_hisilicon_dev_pm_ops,
},
};

int i2c_adap_hisilicon_init(void)
{
return platform_driver_register(&i2c_hisilicon_driver);
                                 //Register I2C driver program to platform bus.
}

void i2c_adap_hisilicon_exit(void)
{
platform_driver_unregister(&i2c_hisilicon_driver);
                                  //Uninstall the I2C driver program from the
platform bus.
}

module_init(i2c_adap_hisilicon_init);    //Register preloading driver program.
module_exit(i2c_adap_hisilicon_exit);    //Register pre-uninstallation driver
program.
/*------------------------Driver program----------------------------*/
```

6.2 File Adaptation

Copyright © Huawei Technologies Co., Ltd.

6.2.1 Overview

Basic Concept

Driver development means implementing and abstracting functions of hardware based on OS specifications and then provides them to application developers to call.

While transplanting system on a new chip, driver development must be carried out based on peripheral equipment supported by the chip specifications.

The driver initialization function for Huawei LiteOS is mainly used to create a driver structure of devices and generate the control node that registers drivers for the upper platform.

6.2.2 Development Guidelines

Development Process

Driver development is related to following two steps:

1. **Driver Initialization**
2. **Driver Node and Using**

Driver Initialization

The first step of driver development is to compile a driver initialization function. In Huawei LiteOS, the driver initialization function is used to initialize a driver structure of devices and generate the driver control node.

The driver initialization function must be used together with the device driver registration function that is used to register and generate a driver node.

```
register_driver(FAR const char *path, FAR const struct file_operations_vfs
*fops,mode_t mode, FAR void *priv)
 register_blockdriver(FAR const char *path,FAR const struct block_operations
*bops,mode_t mode, FAR void *priv)
```

Table 1 describes the parameters expected by the device driver registration function.

Table 6-1 Parameter description

Parameter	Description
*path	Path of the driver node. Application programs will access driver node through this path, and then access operation API supported by device driver.
*fops/*bops	Driver operation structure that provides application programs with an operation function set. fops indicates a character device, and bops indicates a block device.
mode	Application's permission configuration to read from or write into the driver node. This parameter is not supported now.

Copyright © Huawei Technologies Co., Ltd.

Parameter	Description
*priv	Parameter to be passed in during driver node registration.

After the driver program initialization function is compiled, you need to boot this initialization function in a right place.

The codes in sample.c file under the /sample/sample_osdrv directory are used as a simple boot. You can call the compiled device initialization function in the app_init function to boot device initialization.

Driver Node and Using

The device driver node generated after driver initialization provides an interface for applications to access and operate devices. The call relationship between application programs and driver operation functions is described with the i2c device driver as an example.

1. **Operation Function Set**

   The driver operation function set is essential to the call relationship between application programs and driver operation functions. During driver compilation, the operation function set needs to implement various mechanisms of a hardware device and register the mechanisms during device registration. The operation function set meets all requirements on application functions.

   Table 6-2 Operation function set for the i2c device driver

Operation Function Set	Application Layer Interface
i2cdev_open	open
i2cdev_release	close
i2cdev_read	read
i2cdev_write	write
i2cdev_ioctl	ioctl

2. **open Operation**

   When an application program opens a node file, Huawei LiteOS will call the open function in the driver operation function set used when the driver node is initialized.

   The open function instantiates and initializes the driver structure.

3. **read/write Operation**

   After an application program opens a node file, the file descriptor of the driver node will be got. The program can then access the driver that has the file descriptor.

   The read/write operation is a common way for applications to access the driver. The functions of the read/write operation vary depending on the type of devices and drivers. For an i2c device, the read/write operation is used to read from or write into i2c peripherals.

   ```
   i2cdev_read(struct file * filep, char __user *buf, size_t count)
   ```


Copyright © Huawei Technologies Co., Ltd.

```
i2cdev_write(struct file * filep , const char __user *buf, size_t count)
```

Table 3 describes the parameters expected by the read/write function.

Table 6-3 Parameter description

Parameter	Description
*filep	Pointer to the file description structure.
*buf	Buffer that stores the read or written data.
count	Length of the read or written data.

4. **ioctl Operation**

   An ioctl operation provides driver configuration management functions, particularly, defines or accesses device attributes in the device driver by running specific commands.

   For an i2c device, the I2C_16BIT_REG command is used to define the bit width of a transmission register, the I2C_16BIT_DATA command is used to define the bit width of transmitted data, and the I2C_TIMEOUT command is used to define the command expiry time.

```
i2cdev_ioctl(struct file * filep, int cmd, unsigned long arg)
```

   Table 4 describes the parameters expected by the ioctl function.

Table 6-4 Parameter description

Parameter	Description
*filep	Pointer to the file description structure.
cmd	Command for an operation
arg	Additional parameter

5. **close Operation**

   The close operation uses the release function in the operation function set to release resources of a driver.

6.2.3 Precautions

None.

6.2.4 Programming Example

None.

Copyright © Huawei Technologies Co., Ltd.

7 Maintenance and Testing

About This Chapter

7.1 Telnet

7.2 Shell

7.1 Telnet

7.1.1 Overview

Basic Concept

Telnet, as part of TCP/IP protocol, is the standard protocol and major method for remote Internet login. Telnet enables you to operate a remote server on a local computer. Telnet program connects the local computer to the server. Telnet enables you to input commands just like direct input by using the server console.

Huawei LiteOS Telnet

Telnet uses commands to debug a development board. To achieve network connection by using Telnet, users must configure the IP address and gateway (which must be in the same network segment as the development board) for the local computer and run cmd.exe in Windows to execute Telnet IP (IP address of the development board). In other words, Telnet is another debugging mode by using serial ports.

Huawei LiteOS Telnet adopts a simple Telnet protocol, which connects the computers to the development board without authentication on the user name and password. Huawei LiteOS Telnet is used to:

- Read the input characters and transmit them to the development board through TCP.
- Send back the processed data to terminals.

Copyright © Huawei Technologies Co., Ltd.

7.1.2 Development Guidelines

Development Process

1. Run telnet on to start telnet server in Huawei LiteOS Shell.

```
Huawei LiteOS  # telnet on

Huawei LiteOS  # init telnet.
```

2. In a Windows-based OS that has Telnet client installed, run cmd.exe and enter telnet + IP address of the development board to connect the computer to the board, as shown in the following figure.

```
C:\Users\Administrator>telnet 192.168.1.1
```

3. Press Enter. If Huawei LiteOS# (the prompt of Huawei LiteOS Shell) is displayed, it indicates that the computer is connected to the board successfully and you can run the shell commands. For example, you can run the task command to view the status of all tasks.

```
Huawei LiteOS #  task

Name                    PID    Priority   Status                 StackSize    WaterLine    StackPoint
----                    ---    --------   --------               ---------    ---------    ----------
Swt_Task                0x0    0          QueuePend              0x6000       0x320        0x85121038
ui_MainApp              0x1    10         Pend                   0x6000       0x16b0       0x85126f88
IdleCore000             0x2    31         Ready                  0x400        0x154        0x851274a4
RcvAndDispatch          0x3    10         Ready                  0x6000       0x25c        0x8522ed2c
SavePara2Flash          0x4    10         Ready                  0x6000       0x244        0x8529fc94
RecMngMsgRcv            0x5    10         Ready                  0x6000       0x984        0x852ba924
RecMngRecTimer          0x6    10         Ready                  0x6000       0x284        0x852c08ec
RecMsgRcv               0x7    10         Ready                  0x6000       0x23c        0x852e3d40
IspRun                  0x8    10         Ready                  0x6000       0x151c       0x85301424
ProcessEncStreamThread  0x9    2          PendTimeOut            0x6000       0xbac        0x85339cfc
aenc_get                0xa    10         Pend                   0x6000       0x3a4        0x8539d128
SendAFrameToAencThread  0xb    10         Ready                  0x6000       0x4130       0x853a30e8
```

4. Run telnet off to stop telnet server in Huawei LiteOS Shell.

```
Huawei LiteOS # telnet off
telnetd_accept_loop[412] Software caused connection abort
close telnet.

Huawei LiteOS #
```

7.1.3 Precautions

- If **telnet is not recognized as an internal or external command** is returned after the input of telnet + IP in a Windows-based OS, the Telnet is not enabled. Choose **Control Panel > Programs > Programs and Features > Turn Windows features on or off** and select the **Telnet Server** and **Telnet Client**.
- Ensure that the network driver of the board is initialized and opened before starting Telnet.
- Ensure that lwip started normally before starting Telnet. Register safety function and initialize tcpip are needed while using lwip.
- After a client is connected to a development board using Telnet, setting the IP address of the development board using **ifconfig** will break the connection.

Huawei Proprietary and Confidential
Copyright © Huawei Technologies Co., Ltd.

- Currently, only one client can be connected to a development board using telnet and an IP address at one time.
- Telnet only supports the commands supported by Shell.
- Telnet printout is prohibited in the following scenarios: locked tasks, interrupt callback function, and software timer callback function.
- Telnet is a debug function and is disabled by default. It is must not be included in formal Huawei LiteOS.
- Disclaimer: Huawei is not responsible for any risks brought by using the Telnet function in official Huawei LiteOS.

7.1.4 Programming Example

None.

7.2 Shell

7.2.1 Overview

Basic Concept

Shell is the software (command parser) that provides user APIs. It is similar to the command in DOS and the cmd.exe. Shell receives commands and invokes the corresponding programs.

Shell is also a programming language. As a command language, shell interactively interprets and executes commands or automatically interprets and executes the series of commands preset by users. As a programming language, shell defines variables and parameters and provides the control structure, such as the loop and branch, that is available only in high-level languages.

Shell manages the interaction between you and the operating system (OS): waiting for your input, interpreting the input to the OS, and processing diverse outputs of OSs.

Shell builds a bridge for communication between users and OSs. The communication is interactive (you enter through the keyboard and receive instant response) or non-interactive (by shell script). A shell script contains a string of shell commands and OS commands, which can be reused. Essentially, a shell script is a file combining commands.

Shell of Huawei LiteOS helps debug common commands and query system information.

Huawei LiteOS Shell

Shell of Huawei LiteOS provides the basic function used for debugging, including commands for Huawei LiteOS, filesystem, network and scatter loading. In addition, shell of Huawei LiteOS allows command customization.

- Commands for Huawei LiteOS are used for checking system tasks, kernel semaphore, CPU usage, and current interrupts.
- Commands for files include **ls**, **cd**, and **sync**. The **sync** command synchronizes the cache data (data in the file system) to an SD card or flash memory.
- Commands for network are used to view the IP address of the local computer and other devices that connect to the development board, test the network connection, and set the AP and station mode of development board.

Copyright © Huawei Technologies Co., Ltd.

- Commands for dynamic loading are used to obtain the **.obj** file from a specified path and call related functions by searching for the addresses of functions that have loaded the **.obj** file. A user can unload the **.obj** file that has been loaded to a specific path.

For details of adding commands, see Guidelines and Programming Example.

7.2.2 Development Guidelines

Usage Scenarios

Shell commands can be input through serial ports or Telnet. Customized commands can be executed only after their links are recompiled.

Functions

Shell of Huawei LiteOS contains the following commands:

- Commands for Huawei LiteOS, such as task, sem, swtmr, hwi, and cpup.
- Commands for PROC file system, such as writeproc.
- Commands for files, such as ls, cd, uname, cat, touch, rm, and rmdir.
- Commands for network, such as arp, ifconfig, ping, starthapd, and stophapd.

For details of commands, see **7.2.5 Command Reference**. For details of adding commands, see **7.2.4 Programming Example**.

Development Process

1. **Adding Commands to Shell**

   #include "shell.h"

   #include "shcmd.h"

2. Registering the **ls** command.

   The command can be registered statically and dynamically when the OS is running.

   a. Static registration

      SHELLCMD_ENTRY(ls_shellcmd, CMD_TYPE_EX, "ls", XARGS, (CMD_CBK_FUNC)osShellCmdLs);

   b. Dynamic registration

      osCmdReg(CMD_TYPE_EX, "ls", XARGS, (CMD_CBK_FUNC)osShellCmdLs)

   - **ls_shellcmd**: structure variable name that needs to be passed in during static registration. The command to be registered is a field contained in the structure.
   - Command types
     - **CMD_TYPE_EX**: **CMD_TYPE_EX** indicates that the input of standard command parameters is not allowed. When the command type is set to **CMD_TYPE_EX**, command keywords entered by users will be masked. For example, when **ls /ramfs** is entered, only **/ramfs** is passed into the registry function.
     - **CMD_TYPE_STD**: **CMD_TYPE_STD** indicates that the input of standard command parameters is allowed. All entered characters will be passed into the registry function after being parsed by the command.

Copyright © Huawei Technologies Co., Ltd.

- **"ls"**: command keyword that is accessed by the registry function in Shell.
- **XARGS**: the number of input parameters of the execution function that is called.
- **(CMD_CBK_FUNC)osShellCmdLs**: execution function.

This macro encapsulates and registers a command that can be called in Shell.

NOTE

- Normally, static registration is applicable to common system commands, and dynamic registration is applicable user commands.
- During static registration, **-uls_shellcmd** should be added to **LITEOS_TABLES_LDFLAGS** in **build/mk/liteos_tables_ldflags.mk**.
- The command key word must be unique, indicating that multiple commands must not share the same command keyword. Otherwise, only the first command listed in the commands displayed by running the **help** command will be executed.

3. Prototype of added built-in command function.
   - UINT32 cmdHook(UINT32 argc, CHAR **argv)

     Parameters of this function are similar to those of the main prototype of parameter main in C programming language.

     NOTE

     - argc: number of parameters in the shell commands.
     - argv: pointer array. Each element points to a string. You can select the command type to determine whether to transfer the keyword to register function or not.
4. **Entering a Shell Command**

   A Shell command can be entered in the following two ways:
   - Entering the Shell command in the serial port tool
   - Entering the Shell command in the telnet tool

7.2.3 Precautions

- English input is allowed in the default mode. If you enter Chinese characters in UTF8 format, you can delete them only by pressing the **Backspace** key for three times.
- You can press the **Tab** key to let the system automatically complete the shell command, file name, and directory you want to enter. If 2 to 24 matches are found, the system completes the characters common in these matches and displays the matches for you to choose from. If more than 24 matches are found, the system displays "**Display all num possibilities? (y/n)**", prompting for your confirmation. Pressing **N** will exit the type-ahead input function. Pressing **Y** will display the first 24 matches and then the " **--More--**" prompt message. You can press **Enter** to display the rest matches, or **Q** or **Ctrl +C** to exit the type-ahead input function.
- The working directory in the Shell is separated from the system directory. Operations will be performed on the working directory in the Shell by running commands such as **cd** and **pwd** through the Shell. And the system directory will be operated by running commands such as **chdir** and **getcwd**. The two directories are irrelevant to each other. Exercise caution when the input parameter of a file system operation command is a relative path.
- Shell commands take effect after tcpip_init is initialized. Huawei LiteOS does not initialize tcpip_init by default.
- Before shell commands related to dynamic loading are executed, a dynamic loading module must be initialized. For details on the initialization of a dynamic loading module, see **4.1 Dynamic Loading**.

Copyright © Huawei Technologies Co., Ltd.

- Manipulating device files under the **/dev** directory using Shell commands is not recommended because it may cause unpredictable results.
- Shell is a test function, the default configuration is closed, the official product is prohibited including the function.
- Disclaimer: Huawei is not responsible for any risks brought by using the Telnet function in official Huawei LiteOS.

7.2.4 Programming Example

Example Description

The following examples are used to describe how to statically and dynamically register the test command.

Static Registration

1. Define the executive function cmd_test that will be used to add the new command.
2. Call the SHELLCMD_ENTRY function to add the new command.
3. Add the parameter of the new command to the link option liteos_tables_ldflags.mk.
4. Recompile code and run Huawei LiteOS.

Example Code

Define the cmd_test function.

```
#include "shell.h"
#include "shcmd.h"

int cmd_test(void)
{
    printf("hello everybody!\n");
    return 0;
}
```

Add the new command.

```
SHELLCMD_ENTRY(test_shellcmd, CMD_TYPE_STD, "test", 0, (CMD_CBK_FUNC)cmd_test);
```

Add the command parameter to the linker options.

Add **-utest_shellcmd** to **LITEOS_TABLES_LDFLAGS** in **build/mk/liteos_tables_ldflags.mk**.

Recompile the commands.

```
make clean;make
```

Dynamic Registration

1. Call the osCmdReg function to add the new command.
2. Recompile code and run Huawei LiteOS.

Example Code

Call the osCmdReg function in the app_init function to dynamically register the command.

Copyright © Huawei Technologies Co., Ltd.

```
#include "shell.h"
#include "shcmd.h"

int cmd_test(void)
{
    printf("hello everybody!\n");
    return 0;
}

void app_init(void)
{
     ....
     ....
     osCmdReg(CMD_TYPE_EX, "test", 0,(CMD_CBK_FUNC)cmd_test);
     ....
}
```

Recompile the code.

```
make clean;make
```

Run the help command to view all registered commands.

If the test command is displayed, as shown in the following figure, the test command is successfully registered.

```
Huawei LiteOS #
Huawei LiteOS # help
******************shell commands:************************

arp          call         cat          cd           cp           cpup         dns          findsym
format       free         help         hwi          i2c_read     i2c_write    ifconfig     lddrop
ls           mclose       memcheck     mkdir        mopen        mount        netstat      ntpdate
partinfo     partition    ping         pwd          readreg      rm           rmdir        sem
statfs       swtmr        sync         systeminfo   task         telnet       test         tftp
touch        umount       uname        writeproc    writereg

Huawei LiteOS #
```

7.2.5 Command Reference

7.2.5.1 System Commands

7.2.5.1.1 task

Function

The **task** command is used to query the information about tasks in Huawei LiteOS.

Format

task [*ID*]

Parameter Description

Table 7-1 Parameter description

Parameter	Description	Value Range
ID	Task ID	[0, 0xFFFFFFFF]

Copyright © Huawei Technologies Co., Ltd.

User Guide

- If the parameter is left unspecified, all task information will be printed by default.
- An ID is added after task, the task name, task TID, and the call stacks will be displayed. A maximum of 16 call stacks are supported.

Example

For example, enter **task 6**.

Output

Figure 7-1 Information of the task with ID 6

```
Huawei LiteOS # task 6
TaskName = cmdParesTask
TaskID = 0x6
*******backtrace begin*******
traceback 0 -- lr = 0x80057030
traceback 0 -- fp = 0x80536f2c
traceback 1 -- lr = 0x8005346c
traceback 1 -- fp = 0x80536f44
traceback 2 -- lr = 0x8006ee44
traceback 2 -- fp = 0x80536f5c
traceback 3 -- lr = 0x8006e294
traceback 3 -- fp = 0x80536f64
traceback 4 -- lr = 0x8006f5e4
traceback 4 -- fp = 0x80536f84
traceback 5 -- lr = 0x8005a224
traceback 5 -- fp = 0x80536f94
traceback 6 -- lr = 0x8006fa24
traceback 6 -- fp = 0x11111111

Name                    TID   Priority  Status      StackSize   WaterLine   StackPoint  TopOfStack  EventMask  SemID CPUUSE   CPUUSE10s  CPUUSE1s   MEMUSE
----                    ---   --------  --------    ---------   ----------  ----------  ----------  ---------  ----- -------  ---------  ---------  -------
Swt_Task                0x0   0         QueuePend   0x6000      0x2ec       0x801396f8  0x801338a0  0x0        0xffff   0.0       0.0        0.0        0
IdleCore000             0x1   31        Ready       0x400       0x164       0x80139b5c  0x801398c0  0x0        0xffff  99.8      98.2       99.9       0
tcpip_thread            0x3   5         PendTimeOut 0x6000      0x2f4       0x80174608  0x8016e838  0xf        0xffff   0.0       0.0        0.0       76
eth_irq_Task            0x4   3         Pend        0x20000     0x230       0x80532a90  0x80512c30  0xff       0xffff   0.0       0.0        0.0        0
shellTask               0x5   9         Running     0x3000      0x4b4       0x80535a54  0x80532f08  0xfff      0xffff   0.1       1.6        0.0     1656
cmdParesTask            0x6   9         Pend        0x1000      0x278       0x80536db0  0x80535f98  0x1        0xffff   0.0       0.0        0.0      108
```

Figure 7-2 Information of all tasks

```
Huawei LiteOS #
Huawei LiteOS # task

Name                    TID   Priority  Status      StackSize   WaterLine   StackPoint  TopOfStack  EventMask  SemID CPUUSE   CPUUSE10s  CPUUSE1s   MEMUSE
----                    ---   --------  --------    ---------   ----------  ----------  ----------  ---------  ----- -------  ---------  ---------  -------
Swt_Task                0x0   0         QueuePend   0x6000      0x230       0x80107d48  0x80101ee8  0x0        0xffff   0.0       0.0        0.0        0
IdleCore000             0x1   31        Ready       0x400       0x220       0x801081ac  0x80107f08  0x0        0xffff  99.9      99.9       99.9       0
system_wq               0x2   1         Pend        0x6000      0x228       0x8010a240  0x801083d8  0x1        0xffff   0.0       0.0        0.0        0
shellTask               0x4   9         Running     0x3000      0x754       0x8011bd74  0x80119218  0xfff      0xffff   0.0       0.0        0.0     1672
cmdParesTask            0x5   9         Pend        0x1000      0x270       0x8011d0d0  0x8011c2b0  0x1        0xffff   0.0       0.0        0.0      352

Huawei LiteOS #
```

Description of initial system tasks in Huawei LiteOS

Task	Description
Swt_Task	Software timer task, which is used to process the timeout callback function for software timers
IdleCore000	Task that is executed when Huawei LiteOS is idle
system_wq	Default workqueue processing task
cmdParesTask	Reads user input from the lower-layer buf and preliminarily parses a command, such as arrow keys and command completion by pressing Tab
shellTask	Further parses the command sent by cmdParesTask and calls the registration function that matches the command

Copyright © Huawei Technologies Co., Ltd.

Description of task status

Parameter	Description
Ready	Tasks in ready status
Pend	Tasks in pending status
PendTime	Pending tasks in time-out status
QueuePend	Tasks in queue pending status
Running	Tasks running in operating system (OS)
Delay	Tasks in delay waiting status
QueuePendTime	Tasks pending in a queue in time-out status

NOTE

If the task status in the **task** command is invalid, ensure that one of the following operations has been performed when the pthread_create function is creating functions. If none of the following operations are performed, the resources cannot be correctly recycled.

- If you want to choose the block mode, call the pthread_join() function.
- If you want to choose the detach mode, call the pthread_detach() function.
- If you do not want to call either of the preceding functions, set the status of pthread_attr_t to PTHREAD_STATE_DETACHED and transfer the **attr** parameter to the pthread_create function. The result of this operation is the same as that of calling the pthread_detach() function, that is, the detach mode is chosen.
- The **PID** parameter value can be represented either in decimal format or in hexadecimal format.
- When the **PID** parameter value falls in the range [0,64], the status of the task specified by the ID is returned. (A prompt will be displayed if the task specified by the ID is not created.) A prompt indicating a parameter error will be displayed if the parameter value is outside the range [0,64].

Parameter description:

Parameter	Description
Name	Task name
TID	Task ID
Priority	Priority of a task
Status	Status of current task
StackSize	Size of a task stack
WaterLine	Peak usage of a task stack
StackPoint	Start address of a stack
Top0fStack	Address of stack top

Copyright © Huawei Technologies Co., Ltd.

Parameter	Description
EventMask	Event mask of current task, default even mask of task that is not used is 0. (If there are many events used in task, the recently used one will be displayed)
SemId	Semaphore ID that current task owned, default semaphore ID that is not used is 0xFFFF. (If there are many semaphores used in task, the recently used one will be displayed)
CPUUSE	CPU usage since startup
CPUUSE10s	CPU usage in previous 10 seconds
CPUUSE1s	CPU usage in previous 1 second
MEMUSE	Size of memory that is allocated till now, with unit byte.

NOTE

The value of **MEMUSE** can be positive or negative.

If memory is allocated to tasks, the **MEMUSE** value increases. If tasks release memory, the **MEMUSE** value decreases.

If no memory is allocated to tasks, or the allocated memory equals the released memory, the **MEMUSE** value is 0.

If the **MEMUSE** value is positive, some memory is not released by the task.

If the **MEMUSE** value is negative, the released memory is larger than the allocated memory.

The conuting of **MEMUSE** refers to system memory pool. The memory used by operations that take place during an interrupt and any memory processed before task scheduling starts will not be counted.

7.2.5.1.2 sem

Function

The **sem** command is used to query the information about semaphores of Huawei LiteOS Kernel.

Format

sem [*ID*]

Parameter Description

Table 7-2 Parameter description

Parameter	Description	Value Range
ID	Semaphore ID	[0, 0xFFFFFFFF]

Copyright © Huawei Technologies Co., Ltd.

User Guide

- Parameter displays the usage number and total number of semaphore.
- If an ID is added, the usage number of the specified semaphore will be displayed.

Example

Examples, enter **sem 12**.

Output

Figure 7-3 Information of the semaphore

```
Huawei LiteOS # sem 12

   SemID     Count
   -----     -----
   12          0x1
No task is pended on this semphore!
```

Table 7-3 Parameter description

Parameter	Description
SemID	Semaphore ID
Count	Semaphore usage count

NOTE

- The **SemID** parameter value can be represented either in decimal format or in hexadecimal format.
- When the **SemID** parameter value falls in the range [0,1023], the status of the semaphore specified by the ID is returned. (A prompt will be displayed if the semaphore specified by the ID is not in use.) A prompt indicating a parameter error will be displayed if the parameter value is outside the range [0,1023].

7.2.5.1.3 swtmr

Function

The **swtmr** command is used to query the information about system software timers.

Format

swtmr [*ID*]

Copyright © Huawei Technologies Co., Ltd.

Parameter Description

Table 7-4 Parameter description

Parameter	Description	Value Range
ID	ID of a software timer	[0,0xFFFFFFF]

User Guide

- If the parameter is left unspecified, information about all software timers will be displayed.
- If an ID is added after the **swtmr** command, the information about the specified software timer will be displayed.

Example

For example, enter **swtmr** and **swtmr 5**.

Output

Figure 7-4 Information of software timers

```
SwTmrID  State    Mode   Interval   Count    Arg       pfnHandlerAddr
-------  -------  ------- --------- ------- --------  --------
0        Ticking  Period  100        28      0x0       0x800f3a7c
1        Ticking  Period  10         7       0x0       0x8026d270
2        Ticking  Once    300        0       0x80b47cf0  0x8035d620
3        Ticking  Once    1          0       0x80b47cf0  0x8035d4b8
4        Ticking  Once    300        0       0x80b48008  0x8035d620
5        Ticking  Once    1          0       0x80b48008  0x8035d4b8
6        Ticking  Once    300        0       0x80b48320  0x8035d620
7        Ticking  Once    1          0       0x80b48320  0x8035d4b8
8        Ticking  Once    300        133     0x80b479d8  0x8035d620
9        Ticking  Once    1          1       0x80b479d8  0x8035d4b8
```

Figure 7-5 Information of the software timer with specified ID

```
SwTmrID  State    Mode   Interval   Count    Arg       pfnHandlerAddr
-------  -------  ------- --------- ------- --------  --------
5         Ticking  Once     1          0       0x80b48008   0x8035d4b8
```

Table 7-5 Parameter description

Parameter	Description
SwTmrID	ID of a software timer
State	State of a software timer
Mode	Mode of a software timer
Interval	Number of ticks used by a software timer

Copyright © Huawei Technologies Co., Ltd.

Parameter	Description
Count	Times that a software timer works
Arg	Number of input parameters
pfnHandlerAddr	Address of a callback function

NOTE

- The **SwTmrID** parameter value can be represented either in decimal format or in hexadecimal format.
- When the **SwTmrID** parameter value falls in the range [0,current number of software timers –1], the status of the software timer specified by the ID is returned. A prompt indicating a parameter error will be displayed if the parameter value is outside the range [0,current number of software timers –1].

7.2.5.1.4 hwi

Function

The **hwi** command is used to query the information about current interruptions.

Format

hwi

Parameter Description

Table 7-6 Parameter description

Parameter	Description	Value Range
N/A	N/A	N/A

User Guide

- Enter **hwi** to display the interruption number, the number of occurrences, and the name of each current interruption.
- If **LOSCFG_CPUP_INCLUDE_IRQ** is selected, the cpup usage and interruption type of each interruption is displayed.

Example

For example, enter **hwi**.

Output

1. Information of interruptions with **LOSCFG_CPUP_INCLUDE_IRQ** unselected.

```
Huawei LiteOS # hwi
InterruptNo     Count     Name
```


Copyright © Huawei Technologies Co., Ltd.

```
35:             1364:
36:             0:
40:             79:       uart_pl011
```

2. Information of interruptions with **LOSCFG_CPUP_INCLUDE_IRQ** selected.

```
Huawei LiteOS # hwi
 InterruptNo Count  Name     CPUUSE  CPUUSE10s   CPUUSE1s   mode
    2:       768:               0.0    0.0         0.0     normal
    13:      11:    uart_pl011  0.0    0.0         0.0     normal
    18:      180:               0.0    0.0         0.0     normal
```

7.2.5.1.5 cpup

Function

The **cpup** command is used to query the CPU usage of Huawei LiteOS.

Format

cpup [*mode*] [*taskID*]

Parameter Description

Table 7-7 Parameter description

Parameter	Description	Value Range
mode	● Default: display the CPU usage in previous 10 seconds. ● 0: display the CPU usage in previous 10 seconds. ● 1: display the CPU usage in previous 1 second. ● Other value: display the CPU usage since Huawei LiteOS is started.	[0,0xFFFF] or 0xFFFFFFFF
taskID	Task ID	[0,0xFFFF] or 0xFFFFFFFF

User Guide

- If parameter is default, the CPU usage percent of system 10s ago will be displayed.
- If parameter is only one, and the parameter is mode, the CPU usage percent of system corresponding time ago will be displayed.
- If two parameters are passed in, and the first one is mode, the second one is taskID. The CPU usage percent of system with the specific taskID corresponding time ago will be displayed.

Example

For example, enter **cpup 1 1**.

Copyright © Huawei Technologies Co., Ltd.

Output

Figure 7-6 Information of CPU usage

```
Huawei LiteOS # cpup 1 1

TaskId 1 CpuUsage in 1s: 87.3
Huawei LiteOS #
```

7.2.5.1.6 memcheck

Function

The **memcheck** command is used to check whether the dynamically applied memory is complete and whether memory leak occurs causing node destroyed.

Format

memcheck

Parameter Description

Table 7-8 Parameter description

Parameter	Description	Value Range
N/A	N/A	N/A

User Guide

- If memory leak does not occur, the output of memcheck is "memcheck over, all passed! "
- If nodes are not completed memory pool, the output is the information about the memory of the node that destroyed.

Example

For example, enter **memcheck**.

Output

Figure 7-7 Memory leak does not occur

```
Huawei LiteOS # memcheck
memcheck over, all passed!
```


Copyright © Huawei Technologies Co., Ltd.

Figure 7-8 Memory leak occurs

```
[LOS_DLnkCheckMem], 349, memory check error!
stFreeNodeInfo.pstPrev:0x7e0d31f3 is out of legal mem range[0x80ba5f40, 0x83d00000]
cur node: 0x81f2ce0c
pre node: 0x81f28a98
pre node was allocated by task:sofia
uwExcType = 0x2
puwExcBuffAddr pc = 0x803ad7a4
puwExcBuffAddr lr = 0x803ad7a4
puwExcBuffAddr sp = 0x80cb7de0
puwExcBuffAddr fp = 0x80cb7dec
*******backtrace begin********
traceback 0 -- lr = 0x8037cb84
traceback 0 -- fp = 0x80cb7e1c
traceback 1 -- lr = 0x8037033c
traceback 1 -- fp = 0x80cb7e24
traceback 2 -- lr = 0x8000d108
traceback 2 -- fp = 0x80cb7e94
traceback 3 -- lr = 0x8037c7ac
traceback 3 -- fp = 0x80cb7ea4
traceback 4 -- lr = 0x803ad9e8
traceback 4 -- fp = 0x11111111

Name            PID   Priority  Status          StackSize   WaterLine   StackPoint  TopOfStack  EventMask  SemID  CPUUSE  CPUUSE10s  CPUUSE1s  MEMUSE
----            ---   --------  --------        ---------   ---------   ----------  ----------  ---------  -----  ------  ---------  --------  ------
Swt_Task        0x0   0         QueuePend       0x6000      0x360       0x80bce650  0x80bc87f0  0x0        0x6e   1       1          0         0
IdleCore000     0x1   31        Ready           0x400       0x164       0x80bceaac  0x80bce810  0x0        0x6e   17      17         0         1784
system_wq       0x2   10        Pend            0x6000      0x244       0x80bd4b08  0x80bcecb8  0x0        0x2    0       0          0         0
himci_Task      0x3   10        Delay           0x800       0x4c4       0x80bd5348  0x80bd4cd0  0x1        0x6e   17      17         0         5091908
```

7.2.5.1.7 writereg

Function

The **writeReg** command is used to write data to a specified address.

Format

writereg [*address*] [*value*]

Parameter Description

Table 7-9 Parameter description

Parameter	Description	Value Range
address	The address to which data will be written.	[0,0xFFFFFFFF]
value	The data to be written.	[0,0xFFFFFFFF]

NOTE

The values of the **address** and **value** parameters must fall in the valid value range. Otherwise, system exceptions will result.

User Guide

- The **writeReg** command is used to write data to a specified address.
- If data is written successfully, the address and data will be printed on the screen.
- The address is aligned to the largest multiple of 4 that is smaller than the original address. Note that address is hexadecimal.

NOTE

Arbitrary write will cause system crashes.

Example

For example:

Copyright © Huawei Technologies Co., Ltd.

Enter **writereg 0x86412351 0x32**.

Output

Figure 7-9 When data is written to an address, the address is aligned at 0x86412350

```
Huawei LiteOS # writereg 0x86412351 0x32

The align-address:0x86412350,write value:0x00000032
Huawei LiteOS # readreg 0x86412350
86412350

The align-address:0x86412350  value:0x00000032
```

7.2.5.1.8 readreg

Function

The **readreg** command is used to search for data stored in registers.

Format

readreg [*address*] [*length*]

Parameter Description

Table 7-10 Parameter description

Parameter	Description	Value Range
address	Start register address to be checked.	(0x0, 0xFFFFFFFF)
length	Length to be checked.	The address must be within the permitted range.

NOTE

The values of the **address** and **length** parameters must fall in the valid value range.Otherwise, system exceptions will result.

User Guide

- The **readreg** command is used to search for data stored in registers.
- The register address is displayed in hexadecimal format. The address is aligned to 4 bytes that is smaller than the original address. Huawei LiteOS searches for the aligned value. The value to be searched is included in the aligned address. The length will be aligned with the 4 bytes that is larger than the original length and printed in hexadecimal format.

NOTE

Arbitrary address query will cause system crashes.

Copyright © Huawei Technologies Co., Ltd.

Example

For example:

Enter **readreg 0x86412351 0x32**.

Output

Figure 7-10 When data is read from an address that is not aligned, the address is aligned at 0x86412350

```
Huawei LiteOS # readreg 0x86412351 0x32
86412351

The address begin 0x86412350,length:0x34

  0x86412350  :0xd57f7282 0xf5bb69cf 0x6f79b5b4 0x177c9fc3
  0x86412360  :0x5e54f3ac 0xf1fdbb27 0x9b3f6b1f 0x06e27d56
  0x86412370  :0x2bd7f565 0x3f3a3f9a 0xf7277fdb 0xdfe7c91b
  0x86412380  :0xbfb3f5fa
The address end 0x86412384
```

7.2.5.1.9 free

Function

The **free** command displays the usage of memory in Huawei LiteOS and the sizes of the text segment, data segment, rodata segment, and bss segment.

Format

free [*-k* | *-m*]

Parameter Description

Table 7-11 Parameter description

Parameter	Description	Value Range
No parameters	In the unit of byte	N/A
-k	In the unit of KB	N/A
-m	In the unit of MB	N/A

User Guide

- Enter **free** to display the total amount of the dynamic memory pool of Huawei LiteOS. **used** indicates the total amount of used memory, **text** indicates the size of code segment, **data** indicates the size of data segment, **rodata** indicates the size of read-only data segment, and **bss** indicates the size of the memory used by the uninitialized global variables.
- The **free** command can be used to display the memory usage in three units: byte, KB, and MB.

Copyright © Huawei Technologies Co., Ltd.

Example

For example: enter **free**, **free -k**, and **free -m**.

Output

Figure 7-11 Display the memory usage in three units

```
Huawei LiteOS# free

        total        used         free
Mem:    117631744    31826864     85804880

        text         data         rodata       bss
Mem:    4116480      423656       1204224      6659316

Huawei LiteOS#  free -k

        total        used         free
Mem:    114874       31080        83793

        text         data         rodata       bss
Mem:    4020         413          1176         6503

Huawei LiteOS#  free -m

        total        used         free
Mem:    112          30           81

        text         data         rodata       bss
Mem:    3            0            1            6
```

7.2.5.1.10 uname

Function

The **uname** command is used to display the current OS name, time of data creation, name, and version of Huawei LiteOS.

Format

uname[*-a* | *-s* | *-t* |*-v* | *--help*]

Parameter Description

Parameter	Description
-a	Display all information.
-t	Creation time of data.
-s	OS name
-v	Version
--help	Prompt of uname command format

Copyright © Huawei Technologies Co., Ltd.

User Guide

- The **uname** command displays the name of current OS. uname -a / -t/ -s/ -v indicates that the **uname** command is writing the current OS name into standard output. The parameters cannot be used together.

Example

For example: enter **uname -a**.

Output

Figure 7-12 View system information of Huawei LiteOS

```
Huawei LiteOS # uname -a
Huawei LiteOS KernelV100R002C00B111 1.1.3 May 12 2016 16:34:57
```

7.2.5.1.11 systeminfo

Function

The **systeminfo** command is used to view the usage of resources including tasks, semaphores, mutexes, queues, and timers in Huawei LiteOS.

Format

systeminfo

Parameter Description

Parameter	Description	Value Range
N/A	N/A	N/A

User Guide

- The **systeminfo** command is used to view resource usage in Huawei LiteOS.

Example

For example, enter **systeminfo**.

Output

Figure 7-13 Resource usage in Huawei LiteOS

```
Huawei LiteOS # systeminfo

   Module    Used      Total     Enabled
-------------------------------------------
   Task      36        65        YES
   Sem       227       1024      YES
   Mutex     5         1024      YES
   Queue     1         1024      YES
   SwTmr     8         1024      YES
```


Copyright © Huawei Technologies Co., Ltd.

Parameter description

Parameter	Description
Module	Module name
Used	Number of used resources
Total	Maximum number of usable resources
Enabled	Whether to enable a module

7.2.5.1.12 log

Function

The **log** command is used to query & modify log print configuration.

Format

log level [levelNum]

log module [moduleNum]

log path [filepath]

Parameter Description

Parameter	Description	Value
level	config log print level	[0x0，0x5]

NOTE

Configuration of module and path are not supported now.

User Guide

- **log level** command is used to config log print level which includes the following 5 levels:

  TRACE_EMG = 0,

  TRACE_COMMOM = 1,

  TRACE_ERROR = 2,

  TRACE_WARN = 3,

  TRACE_INFO = 4,

  TRACE_DEBUG = 5
- If **log level** has not [**levelNum**] parameter, it will print the current log level by default and display the usage information.

Copyright © Huawei Technologies Co., Ltd.

Example

For example, enter **log level 4**.

Output

```
Huawei LiteOS # log level 4
Set current log level INFO
```

7.2.5.1.13 help

Function

The **help** command is used to view all commands in Huawei LiteOS.

Format

help

Parameter Description

Parameter	Description
N/A	N/A

User Guide

- The **help** command is used to view all commands in Huawei LiteOS.

Example

For example, enter **help**.

Output

Figure 7-14 All commands in Huawei LiteOS

```
Huawei LiteOS # help
******************shell commands:************************

arp          call         cat          cd           cp           cpup         dns          findsym
format       free         help         hwi          ifconfig     lddrop       ldinit       ls
mclose       memcheck     mkdir        mopen        mount        netstat      ntpdate      partition
ping         pwd          readreg      rm           rmdir        sem          statfs       swtmr
sync         systeminfo   task         telnet       tftp         touch        umount       uname
writeproc    writereg
```

7.2.5.1.14 date

Function

The **date** command is used to query and set the system time.

Copyright © Huawei Technologies Co., Ltd.

Format

date

date --help

date +Format

date -s YY/MM/DD

date -s hh:mm:ss

date -r Filename

Parameter Description

Parameter	Description	Value
--help	Get help info.	N/A
+Format	Print the current time in the configured format.	See --help
-s YY/MM/DD	Set the system time. The value is in YY/MM/DD format. The year, month, and date are separated with "/".	>= 1970/01/01
-s hh:mm:ss	Set the system time. The value is in hh:mm:ss format. The hour, minute, and second are separated with ":".	N/A
-r	Query the time of last modification.	N/A
Filename	Name of the file to be queried.	N/A

User Guide

- If the parameter is left unspecified, the current system time will be printed by default.
- The **--help**, **+Format**, **-s**, and **-r** commands cannot be used together.

Example

For example:

Enter **date +%Y--%m--%d**.

Output

Figure 7-15 Print current system time in the following format

```
Huawei LiteOS # date +%Y--%m--%d
2017--07--28
```


Copyright © Huawei Technologies Co., Ltd.

7.2.5.1.15 dmesg

Function

The **dmesg** command is used to control the dmesg buffer in core.

Format

dmesg

dmesg [-c/-C/-D/-E/-L/-U]

dmesg -s size

dmesg -l level

dmesg > fileA

Parameter Description

Parameter	Description	Value
-c	Print logs in the dmesg buffer, and then clean the dmesg buffer.	
-C	Clean the dmesg buffer.	
-D/-E	Disable/enable the console print function.	
-L/-U	Disable/enable the uart print function.	
-s size	Set the size of the dmesg buffer.	
-l level	Set the dmesg level.	0--5
> fileA	Write logs of the dmesg buffer to the file.	

User Guide

- If the parameter is left unspecified, logs in the dmesg buffer will be printed by default.
- The parameters marked with "-" cannot be used together.

NOTE

1. The file system must be mounted when running the **dmesg > fileA** command.
2. Closing the uart print function will affect the running of the **shell** command.

Example

For example:

Enter **dmesg -D**.

Copyright © Huawei Technologies Co., Ltd.

Output

Figure 7-16 Disable the console print function to hide the strings printed through the console

```
Huawei LiteOS # print
1. printf:       printf working...
2. fprintf:      fprintf working...
3. write STDOUT: write STDOUT working...
4. PRINTK: PRINTK working...
5. LOS_LkPrint:  LOS_LkPrint working...
6. lk_dprintf:   lk_dprintf working...
7. dprintf:      dprintf working...
8. printk:       printk working...
[ERR]L2. PRINT_ERR:   PRINT_ERR working...
L1. PRINTK:      PRINTK_COMMON working...
[EMG] L0. PRINT_EMG:   PRINT_EMG working...

Huawei LiteOS # dmesg -D

Huawei LiteOS # print
4. PRINTK: PRINTK working...
5. LOS_LkPrint:  LOS_LkPrint working...
6. lk_dprintf:   lk_dprintf working...
7. dprintf:      dprintf working...
8. printk:       printk working...
[ERR]L2. PRINT_ERR:   PRINT_ERR working...
L1. PRINTK:      PRINTK_COMMON working...
[EMG] L0. PRINT_EMG:   PRINT_EMG working...
```

7.2.5.1.16 watch

Function

The **watch** command is used to periodically monitor the command output.

Format

watch

watch [-c/-n/-t/--count/--interval/-no-title] [command]

Parameter Description

Table 7-12 Parameter description

Parameter	Description	Default Value
-c / --count	Total number of command executions.	0xFFFFFFFF
-n / --interval	Frequency of executing commands.	1s
-t / -no-title	Close the time display on the top.	N/A
command	Command to be listened to.	N/A

Copyright © Huawei Technologies Co., Ltd.

Example

```
watch  -n 2 -c 6 task
```

Output

Figure 7-17 Output

```
Thu Jan  1 00:00:35 1970


Name                        TID    Priority   Status       StackSize    WaterLine    StackPoint  TopOfStack   EventMask  SemID CPUUSE   CPUUSE10s  CPUUSE1s   MEMUSE
----                        ---    --------   --------     ---------    ----------   ----------  ----------   ---------  ----- -------  ---------  ---------  -------
Swt_Task                    0x0    0          QueuePend    0x6000       0x260        0x801c4bd0  0x801bedb0 0x0        0xffff    0.0       0.0        0.0        0
IdleCore000                 0x1    31         Ready        0x800        0x164        0x801c54b4  0x801c4e18 0x0        0xffff   98.8      95.6       99.9       0
devfreq_wq                  0x2    1          Pend         0x6000       0x220        0x801cb4f0  0x801c5690 0x1        0xffff    0.0       0.0        0.0        0
system_wq                   0x3    1          Pend         0x6000       0x220        0x801d15e0  0x801cb780 0x1        0xffff    0.0       0.0        0.0        0
shellcmd_watch              0x4    10         Running      0x3000       0x57c        0x801d421c  0x801d1798 0x1        0xffff    1.1       4.3        0.0        68112
shellTask                   0x5    9          Pend         0x3000       0x35c        0x801e0028  0x801dd1d8 0xfff      0xffff    0.0       0.0        0.0        12620
cmdParesTask                0x6    30         Pend         0x1000       0x2cc        0x801e0fe8  0x801e01f0 0x1        0xffff    0.0       0.0        0.0        52
#
```

7.2.5.2 File

7.2.5.2.1 ls

Function

The **ls** command is used to display the contents of the current directory.

Format

ls [*path*]

Parameter Description

Table 7-13 Parameter description

Parameter	Description	Value Range
path	If the **path** parameter is null, the current contents will be displayed. If the value of the **path** parameter is an invalid file name, no content will be displayed. "No such directory" will be prompted. If the value of the **path** parameter is a valid directory, content under the directory will be displayed.	1. Null 2. Valid directory

User Guide

- The **ls** command displays the contents of the current directory.
- The **ls** command displays the size of files.

Copyright © Huawei Technologies Co., Ltd.

- The **ls** command can not count the size of files in proc, displaying 0.

Example

For example, enter **ls**.

Output

Figure 7-18 Check the contents under the current directory. The displayed content is as follows:

```
Huawei LiteOS# ls
Directory /:
. flash              153
bin                  <DIR>
font                 <DIR>
etc                  <DIR>
lost+found           <DIR>
```

7.2.5.2.2 cd

Function

The **cd** command is used to change the current directory.

Format

cd [*path*]

Parameter Description

Table 7-14 Parameter description

Parameter	Description	Value Range
path	File path	You must have the execution (search) permission of the specified directory.

User Guide

- If the directory parameter is not configured, the **cd** command will jump to the root directory.
- If a complete file path is configured, it will jump to the file path.
- A complete file path starts with a slash (/), which indicates the root directory.
- One point (.) indicates the current directory.
- Two points (..) indicates the parent directory.

Example

For example: **cd**..

Copyright © Huawei Technologies Co., Ltd.

Output

Figure 7-19 Displayed information

```
Huawei LiteOS# cd ..

Huawei LiteOS# ls
Directory /:
                        <DIR>
bin                     <DIR>
dev                     <DIR>
ramfs                   <DIR>
yaffs0                  <DIR>
```

7.2.5.2.3 pwd

Function

The **pwd** command is used to display the current path.

Format

pwd

Parameter Description

Table 7-15 Parameter description

Parameter	Description	Value Range
N/A	N/A	N/A

User Guide

- The **pwd** command writes the full path name (from root directory) of the current directory to the standard output. All directories are separated by slash (/). The first slash indicates the root directory and the last indicates the current directory.

Example

For example, enter **pwd**.

Output

Figure 7-20 View current path

```
Huawei LiteOS# pwd
/bin/vs
```


Copyright © Huawei Technologies Co., Ltd.

7.2.5.2.4 cp

Function

The **cp** command is used to copy files.

Format

cp [*source path*] [*dest path*]

Parameter Description

Table 7-16 Parameter description

Parameter	Description	Value Range
source path	Path to the source file	File name
dest path	Path to the destination file	File name or directory name

User Guide

- The name of the source file cannot be the same as that of the destination file in the same path.
- The source file must exist. Currently, directories cannot be copied.
- If the destination path is a directory, the directory must exist, and the name of the destination file is the same as that of the source file.
- If the destination path is a file, the directory that contains the file must exist, and the name of the destination file is different from that of the source file.
- Currently, multiple files (more than two files) cannot be copied concurrently. If there are more than two source path parameters, files specified by the first two parameters are copied.
- If the destination file does not exist, a destination file will be created. If the destination file already exists, it will be overwritten after the copy operation.

NOTE

Copying important system resources will cause unknown serious problems such as crashes. For example, copying the **/dev/uartdev-0** file by running the **cp** command will cause system crashes.

Example

For example, enter **cp 100HSCAM/FILE0087.MP4**.

Copyright © Huawei Technologies Co., Ltd.

Output

Figure 7-21 Command output

```
Huawei LiteOS# ls
Directory /bin/vs/sd/dcim:
100HSCAM              <DIR>

Huawei LiteOS#  cp 100HSCAM/FILE0087.MP4 .

Huawei LiteOS# ls
Directory /bin/vs/sd/dcim:
100HSCAM              <DIR>
FILE0087.MP4          11332370
```

7.2.5.2.5 cat

Function

The **cat** command is used to display content of text files.

Format

cat [*pathname*]

Parameter Description

Table 7-17 Parameter description

Parameter	Description	Value Range
path name	File path	Existed files

User Guide

- The **cat** command displays content of text files.

Example

For example: **cat w** //w is a file name.

Output

Figure 7-22 View information about the **w** file

```
Huawei LiteOS# cat w
w open return Ux836be3fc
w size is 0
```


Copyright © Huawei Technologies Co., Ltd.

7.2.5.2.6 touch

Function

- The **touch** command is used to create a nonexistent file in the current directory.
- If the touch command is used to create an existing file, no file will be created and the timestamp will not be updated.

Format

touch [*filename*]

Parameter Description

Table 7-18 Parameter description

Parameter	Description	Value Range
filename	Name of the file to be created.	N/A

User Guide

- The **touch** command creates a readable and writable empty file.
- The **touch** command creates only one file each time.

NOTE

Creating a file by running the **touch** command in an important system resource path will cause unknown problems such as crashes. For example, running the **touch uartdev-0** command in the **/dev** path will cause system crashes.

Example

For example, enter **touch file.c**.

Output

Figure 7-23 Create a file named file.c

```
Huawei LiteOS# touch file.c

Huawei LiteOS# ls
Directory /bin/vs:
file.c              0
sd                  <DIR>

Huawei LiteOS#
```


Copyright © Huawei Technologies Co., Ltd.

7.2.5.2.7 rm

Function

The **rm** command is used to delete a file.

Format

rm [*-r*] [*dirname/filename*]

Parameter Description

Table 7-19 Parameter description

Parameter	Description	Value Range
-r	The parameter is optional. The parameter is necessary to delete directories.	N/A
dirname/filename	Name of the file to be deleted, which can be a path name.	N/A

User Guide

- The **rm** command deletes only one file each time.
- The **rm -r** command deletes a non-empty directory.

NOTE

Deleting important system resources such as **/dev** by running the **rm** command will cause unknown problems such as crashes.

Example

For example:

1. Enter **rm 1.c**.
2. Enter **rm -r dir**.

Output

Figure 7-24 The **rm** command deletes the **1.c** file.

```
Huawei LiteOS# ls
Directory /ramfs:
1.c                  0

Huawei LiteOS# rm 1.c

Huawei LiteOS# ls
Directory /ramfs:
```


Copyright © Huawei Technologies Co., Ltd.

Figure 7-25 Delete the **dir** directory by using the **rm -r** command

```
Huawei LiteOS# ls
Directory /ramfs:
dir                    <DIR>

Huawei LiteOS# rm -r dir

Huawei LiteOS# ls
Directory /ramfs:
```

7.2.5.2.8 sync

Function

The **sync** command is used to synchronize the cache data (data in the file system) to an sd card .

Format

sync

Parameter Description

Table 7-20 Parameter description

Parameter	Description	Value Range
N/A	N/A	N/A

User Guide

- The **sync** command refreshes the new cache. When there is no SD card, no operation will be done.
- When there is an SD card, cache data will be synchronized to the SD card memory and no information will be printed.

Example

For example: after **sync** is input, the synchronization will succeed if there is an SD card and no operation will be done if there is not.

Output

None.

7.2.5.2.9 statfs

Function

The **statfs** command is used to print the information about a file system, such as type, total size, and available size.

Copyright © Huawei Technologies Co., Ltd.

Format

statfs [*directory*]

Parameter Description

Parameter	Description	Value Range
directory	Type of the file system.	The existing file system that supports the statfs command

User Guide

The printed information differs with different systems.

Example

Example of printing the information about the YAFFS file system:

statfs yaffs0

Output

Output of the **statfs yaffs0** command

statfs got:

f_type = 1497497427

cluster_size = 2048

total_clusters = 704

free_clusters = 640

avail_clusters = 640

f_namelen = 255

7.2.5.2.10 format

Function

The **format** command is used to format disks.

Format

format [*dev_inodename*] [*sectors*] [*option*] [*label*]

Copyright © Huawei Technologies Co., Ltd.

Parameter Description

Parameter	Description
dev_inodename	Name of a device
sectors	Size of the allocated unit memory or sector. If the value of this parameter is set to **0**, the parameter is null. (The value must be a power of 0 or 2. The maximum value is **128** in the FAT32 file system. When this parameter is set to **0**, the size of the allocated unit memory or sector, which varies with the partition size, is automatically specified. An incorrect size will lead to a formatting failure.)
option	A formatting option used to specify the file system format. For details about the value of **option**, see**User Guide**.
label	(Optional) Volume label name. The value of this parameter is a string. When the value is set to **null**, the volume label name set earlier is cleared.

User Guide

- The **format** command formats disks. The device name can be searched for under the **dev** directory. A storage card must be installed before formatting.
- The **format** command can only be used to format SD cards and MMC cards and is not valid to the NAND flash or NOR flash memory.
- The **sectors** parameter value must be valid. Otherwise, errors may occur.
- The **option** parameter value can be set to one of the following:
  - **0x01**: represents FMT_FAT
  - **0x02**: represents FMT_FAT32
  - **0x04**: represents FMT_EXFAT
  - **0x07**: represents FMT_ANY

  Other values are invalid. If you pass any value other than the preceding values, Huawei LiteOS automatically selects the file system format.

Example

For example, enter **format/dev/mmc**blk0 **128 0x01**.

Output

Figure 7-26 Displayed contents

```
Huawei LiteOS # format /dev/mmcblk0 0
format /dev/mmcblk0 Success
```


Copyright © Huawei Technologies Co., Ltd.

7.2.5.2.11 mount

Function

The **mount** command is used to mount a device to a specified directory.

Format

mount [*device*] [*path*] [*name*]

Parameter Description

Table 7-21 Parameter description

Parameter	Description	Value Range
device	The device to be mounted (the format is the path of the device).	Devices of Huawei LiteOS.
path	Specified directory. You must have the execution (search) permission of the specified directory.	N/A
name	Type of the file system.	vfat, yaffs, jffs, ramfs, nfs

User Guide

- Add device information, the specified directory, and type of the file system to mount the file system to a specified directory.

Example

For example, enter **mount /dev/mmc0 /bin/vs/sd vfat**.

Output

Figure 7-27 Mount **/dev/mmc0** to the **/bin/vs/sd** directory

```
Huawei LiteOS# mount /dev/mmc0 /bin/vs/sd vfat

Huawei LiteOS#
```

7.2.5.2.12 umount

Function

The **umount** command is used to uninstall a specified file system.

Copyright © Huawei Technologies Co., Ltd.

Format

umount [*dir*]

Parameter Description

Table 7-22 Parameter description

Parameter	Description	Value Range
dir	Directory of the file system that needs to be unmounted.	Directory of the mounted file system

User Guide

- Add the directory to be uninstalled (of the specified directory) to the end of the **umount** command.

Example

For example, enter **umount /bin/vs/sd**.

Output

Figure 7-28 Uninstall the file system that is mounted to **/bin/vs/sd**

```
Huawei LiteOS# umount /bin/vs/sd
```

7.2.5.2.13 rmdir

Function

The **rmdir** command is used to delete a directory.

Format

rmdir[*dir*]

Parameter Description

Table 7-23 Parameter description

Parameter	Description	Value Range
dir	Name of the directory to be deleted. The directory must be empty and the name can be a file path.	N/A

Copyright © Huawei Technologies Co., Ltd.

User Guide

- The **rmdir** command only deletes a directory.
- The **rmdir** command deletes only one directory each time.
- The **rmdir** command deletes only empty directory.

Example

For example, enter **rmdir dir**.

Output

Figure 7-29 Delete the **dir** directory

```
Huawei LiteOS# ls
Directory /bin/vs:
dir                 <DIR>
sd                  <DIR>

Huawei LiteOS# rmdir dir

Huawei LiteOS# ls
Directory /bin/vs:
sd                  <DIR>
```

7.2.5.2.14 mkdir

Function

The **mkdir** command is used to create a directory.

Format

mkdir [*directory*]

Parameter Description

Table 7-24 Parameter description

Parameter	Description	Value Range
directory	Directory to be created	N/A

User Guide

- Add a directory name to the end of the **mkdir** command to create a directory.
- Add a path name and a directory name to the end of the **mkdir** command to create a directory under the specified directory.

Example

For example, enter **mkdir share**.

Copyright © Huawei Technologies Co., Ltd.

Output

Figure 7-30 Create the **share** directory

```
Huawei LiteOS# mkdir share

Huawei LiteOS# ls
Directory /bin/vs:
share                <DIR>
sd                   <DIR>
```

7.2.5.2.15 partition

Function

The **partition** command is used to query the information about partitions.

Format

partition [jffs | yaffs]

Parameter Description

Table 7-25 Parameter description

Parameter	Description	Value Range
jffs	Display partition information of jffs file system	N/A
yaffs	Display partition information of yaffs file system	N/A

User Guide

- Enter the **partition** command to display the information about partitions.
- The command only supports yaffs and jffs file systems.

Example

For example, enter **partition yaffs**.

Output

Figure 7-31 Information of partition

```
Huawei LiteOS # partition yaffs
yaffs partition num:0, dev name:/dev/nandblk0,
mountpt:/yaffs0, startaddr:0x00e00000,
length:0x00200000
```


Copyright © Huawei Technologies Co., Ltd.

7.2.5.2.16 writeproc

Function

The **writeproc** command is used to write data to a specified proc file system.

Format

writeproc [*pcval*] [*operational mark*] [*pcPath*]

Parameter Description

Table 7-26 Parameter description

Parameter	Description	Value Range
pcval	Data to be written to the file.	A string.
operational mark	If the operational mark is >>, data is written to an existing file.	Only >> is valid.
pcPath	Path of the file to which data will be written.	Absolute path.

User Guide

- The **writeproc** command writes data to a file.
- If the operational mark is >> and the file to which the data is written exists, the data is written to the file.
- As for several files non user created in file system, **writeproc** is able to make functions such as modifying system information recorded come true.

Example

Enter **writeproc 'sys=2' >> /proc/umap/logmpp**.

Output

Figure 7-32 Modify the level of sys in logmpp

```
Huawei LiteOS # writeproc 'sys=2' >> /proc/umap/logmpp
sys=2 >> /proc/umap/logmpp

Huawei LiteOS # cat /proc/umap/logmpp

Huawei LiteOS # LOG BUFFER STATE-------------------------------------------------------
MaxLen  ReadPos WritePos ButtPos
 64(KB)       0      117   65536

-----CURRENT LOG LEVEL-------------------------------------------------------
vb      :  3
sys     :  2
chnl    :  3
vpss    :  3
venc    :  3
h264e   :  3
jpege   :  3
viu     :  3
rc      :  3
isp     :  3
vgs     :  3
```


Copyright © Huawei Technologies Co., Ltd.

7.2.5.2.17 partinfo

Function

The information about the identified partitions of hard disk and SD card can be queried by running the **partinfo** command.

Format

partinfo <*dev_inodename*>

Parameter Description

Table 7-27 Parameter description

Parameter	Description	Value Range
dev_inodename	The name of the partition to check.	Legal partition name.

User Guide

None.

Example

For example, enter **partinfo /dev/sdap0**.

Output

```
Huawei LiteOS # partinfo /dev/sdap0
part info :
disk id          : 3
part_id in system: 0
part no in disk  : 0
part no in mbr   : 1
part filesystem  : 0C
part dev name    : sdap0
part sec start   : 2048
part sec count   : 167794688
```

7.2.5.2.18 virstatfs

Function

The **virstatfs** command is used to display file system information in a virtual partition, such as the total size and available size of the file system.

Format

virstatfs [*directory*]

Copyright © Huawei Technologies Co., Ltd.

Parameter Description

Parameter	Description	Value Range
directory	Path or subpath of the virtual partition directory entry.	N/A

User Guide

The **virstatfs** command is available only when the virtual partition feature is enabled on the menuconfig page.

The input path of **virstatfs** must be the path or subpath of the virtual partition directory entry of the physical partition. The **virstatfs** command fails when any of the following conditions is met:

- The input path is the root directory of the partition.
- The virtual partition feature is disabled to the physical partition.
- The path is directed to the partition of a file system other than FAT32.

Example

For example, enter **virstatfs /bin/vs/sd/virpart0**.

Output

```
Huawei LiteOS # mount /dev/mmcblk0p0 /bin/vs/sd vfat
mount ok

Huawei LiteOS # ls /bin/vs/sd
Directory /bin/vs/sd:
virpart0             <DIR>
virpart1             <DIR>
virpart2             <DIR>

Huawei LiteOS # virstatfs /bin/vs/sd/virpart0
Virtual partition "/bin/vs/sd/virpart0" Info:
 f_type         = 19780
 cluster_size   = 32768
 total_clusters = 0x23b73f
 free_clusters  = 0x23b73e
 avail_clusters = 0x23b73e
 f_namelen      = 255

/bin/vs/sd/virpart0
 total size: 76699107328 Bytes
 free  size: 76699074560 Bytes

Huawei LiteOS #
```


Huawei Proprietary and Confidential
Copyright © Huawei Technologies Co., Ltd.

7.2.5.2.19 lsfd

Function

The **lsfd** command is used to display the opened file descriptors and their corresponding file names.

Format

lsfd

User Guide

Run the **lsfd** command to view opened file descriptors and their corresponding file names.

Example

Enter **lsfd**.

Output

```
Huawei LiteOS # lsfd
   fd    filename
    3   /dev/shell
```

7.2.5.2.20 fat_fsck

Function

The **fat_fsck** command is used to check and attempt to rectify faults occurred in the **fat32** file system.

Format

fat_fsck [pathname]

Parameter Description

Table 7-28 Parameter description

Parameter	Description	Value Range
pathname	File path.	Block device files stored in the **/dev/** directory.

User Guide

- **pathname** must be the path name of the partition to which the to-be-tested device belongs, for example, **/dev/mmcblk0p0** instead of **/dev/mmcblk**.

Copyright © Huawei Technologies Co., Ltd.

- The cluster size of the to-be-tested **fat32** file system is recommended to be greater than 32 KB. If the cluster size is less than or equal to 32 KB, more memory is required and the test more possibly fails due to insufficient memory during the test.

Example

For example, enter **fat_fsck /dev/mmcblk0p0**.

Output

```
Huawei LiteOS # fat_fsck /dev/mmcblk0p0
[FSTOOL][WARN] File system is clean !
[FSTOOL][WARN] Free space in FSInfo block (-1) not correct (1966079)
***** FILE SYSTEM FSCK SUCCESS *****
```

7.2.5.3 Network

7.2.5.3.1 arp

Function

In Ethernet, hosts communicate with each other by using a MAC address. If a host that uses the IP protocol wants to communicate in LAN (Ethernet), the IP address of the host needs to be transformed into a MAC address. Therefore, the ARP cache, a mapping of IP addresses and MAC addresses, is stored in a host. A host obtains the MAC address from the ARP cache table to send IP packets to a destination IP address in LAN. The ARP cache is maintained by TCP/IP protocol stack. You can view or modify the ARP table by using **ARP** commands.

Format

arp

arp [-i IF] -s IPADDR HWADDR

arp [-i IF] -d IPADDR

Parameter Description

Table 7-29 Parameter description

Parameter	Description	Value Range
None.	Print the contents of the whole ARP cache.	N/A
-i IF	Specify a network API (optional).	N/A
-s IPADDR HWADDR	Add an ARP entry. The parameters next to the command are the IP address and MAC address of another host in LAN.	N/A

Copyright © Huawei Technologies Co., Ltd.

Parameter	Description	Value Range
-d IPADDR	Delete an ARP entry.	N/A

User Guide

- The **arp** command queries and modifies the ARP cache table of TCP/IP protocol stack. It is meaningless to add the ARP entry in non-LAN networks, and protocol stack will return fail.
- Use the command after TCP/IP protocol takes effect.

Example

For example:

1. Enter **arp**.
2. Enter **arp -s 192.168.1.1 00:11:22:33:44:55**.

Output

Figure 7-33 Print the whole ARP cache table

```
Huawei LiteOS # arp
Address                 HWaddress               Iface
192.168.1.2             00:E0:4C:97:83:DB       eth0
```

Table 7-30 Parameter description

Parameter	Description
Address	The IP address of the network device that connects to the board.
HWaddress	The MAC address of the network device that connects to the board.
Iface	Name of the API used by ARP entry.

7.2.5.3.2 ifconfig

Function

The **ifconfig** command is used to query and configure the parameters such as IP address, network mask, gateway, and the MAC address. The command also enables or disables the data processing function of the NIC.

Copyright © Huawei Technologies Co., Ltd.

Format

ifconfig

[-a]

<interface> <address> [netmask <address>] [gateway <address>]

[hw ether <address>]

[up|down]

Parameter Description

Table 7-31 Parameter description

Parameter	Description	Value Range
No parameter	Print the information about all NICs, such as IP address, network mask, gateway, MAC address, MTU, and running status.	N/A
-a	Print the sending and receiving statistics of protocol stack data.	N/A
interface	Name of specified NIC, for example, en0.	N/A
address	Configure the IP address, for example 192.168.1.10. Name of specified NIC card needs to be specified.	N/A
netmask	Configure the mask of subnet. Next to the command is the mask parameter, for example, 255.255.255.0.	N/A
gateway	Configure the gateway. Next to the command is the gateway parameter, for example, 192.168.1.1.	N/A
hw ether	Configure the MAC address. Next to the command is the MAC address, for example, 00:11:22:33:44:55. Only support ether hard type currently.	N/A

Copyright © Huawei Technologies Co., Ltd.

Parameter	Description	Value Range
mtu	Configure the size of mtu. Next to the command is the mtu size, for example, 1000.	[68,1500]
up	Enable the data processing function of NIC. The NIC name needs to be specified.	N/A
down	Disable the data processing function of NIC. The NIC name is needed.	N/A

User Guide

- The **ifconfig** command queries and configures the parameters such as network mode (Wi-Fi or Ethernet), IP address, network mask, gateway, and MAC address.
- Use the command after TCP/IP protocol takes effect.
- Because the IP address collision detection requires response time, IP address configuration by using the **ifconfig** command each time has a delay of about 2 seconds.

Example

1. Enter **ifconfig** *eth0 192.168.100.31 netmask 255.255.255.0 gateway 192.168.100.1 hw ether 00:49:cb:6c:a1:31*.

   Set the IP address of the development board to 192.168.100.31, the mask to 255.255.255.0, the gateway to 192.168.100.1, and the MAC address to 00:49:cb:6c:a1:31.

2. Run the **ifconfig -a** command to obtain protocol stack statistics.

Output

1. Set network parameters.

```
Huawei LiteOS # ifconfig
eth0    ip:192.168.1.2 netmask:255.255.255.0 gateway:192.168.1.1
        HWaddr d2:ba:f4:0d:fb:89 MTU:1500 Runing Default Link UP
lo      ip:127.0.0.1 netmask:255.0.0.0 gateway:127.0.0.1
        HWaddr 00 MTU:0 Runing Link Down

Huawei LiteOS # ifconfig eth0 192.168.100.31 netmask 255.255.255.0 gateway
192.168.100.1 hw ether  00:49:cb:6c:a1:31

Huawei LiteOS # ifconfig
eth0    ip:192.168.100.31 netmask:255.255.255.0 gateway:192.168.100.1
        HWaddr 00:49:cb:6c:a1:31 MTU:1500 Runing Default Link UP
lo      ip:127.0.0.1 netmask:255.0.0.0 gateway:127.0.0.1
        HWaddr 00 MTU:0 Runing Link Down
```

   The following table lists the output parameters.

Copyright © Huawei Technologies Co., Ltd.

Table 7-32 Parameter description

Parameter	Description
ip	IP address of the board
netmask	Network mask
gateway	Gateway
HWaddr	MAC address of the board
MTU	Maximum transmission units of network
Running/Stop	Whether NIC is running
Default	Explain connecting to the default gateway
Link UP/Down	Connect status to NIC

2. Obtain protocol stack statistics.

```
Huawei LiteOS # ifconfig -a
RX packets:23128  error:0  dropped:0  overrun:0  bytes:10390(10.1KB)
TX packets:40921  error:0  dropped:0  overrun:0  bytes:64008(62.5KB)
```

The following table lists the output parameters.

Table 7-33 Parameter description

Parameter	Description
RX packets	The number of normal packets that the IP layer has received.
RX error	The number of packets with errors that the IP layer has received. The error types include length error, verification error, IP option error, and error of the **protocol** field in an IP header.
RX dropped	The number of packets that the IP layer has dropped. The packets are dropped because the packets have errors, the packets cannot be forwarded, or the local NIC that receives the packets is disabled.
RX overrun	The number of packets that the MAC layer fails to deliver to the upper-layer protocol stack. The main failure cause is that the protocol stack resources are insufficient.
RX bytes	The total number of bytes in all normal packets that the IP layer has received, excluding the bytes in the fragments that have not been completely reassembled.
TX packets	The number of packets that the IP layer has successfully sent or forwarded.
TX error	The number of packets that the IP layer fails to send. The packets fail to be sent because the packets cannot be routed, or the packets fail to be processed in the protocol stack.

Copyright © Huawei Technologies Co., Ltd.

Parameter	Description
TX dropped	The number of packets that the MAC layer drops due to failure to send the packets. The packets fail to be sent because the network adapter driver fails to process the packets.
TX overrun	Not in use.
TX bytes	The total number of bytes in the packets that the IP layer has successfully sent or forwarded.

7.2.5.3.3 ping

Function

The **ping** command is used to check the network connectivity.

Format

ping *<IP> [count]*

Parameter Description

Table 7-34 Parameter description

Parameter	Description	Value Range
IP	IP address of the network to be tested.	
count	Times of execution. The default value is 4.	1~65535

User Guide

- The **ping** command tests the connectivity of the target IP network. The parameter is the destination IP address.
- If the destination IP address is not reachable, a message indicating request timeout will be displayed.
- If displaying sends an error, it explains the destination IP route is not reachable.
- Use the command after TCP/IP protocol takes effect.

Example

For example, enter **ping 192.168.0.2**.

Copyright © Huawei Technologies Co., Ltd.

Output

Figure 7-34 Semaphore information about Huawei LiteOS

```
Huawei LiteOS# ping 192.168.0.2
Reply from 192.168.0.2: time=2ms TTL=128
Reply from 192.168.0.2: time=1ms TTL=128
Reply from 192.168.0.2: time=5ms TTL=128
Reply from 192.168.0.2: time=2ms TTL=128
```

7.2.5.3.4 tftp

Function

Trivial File Transfer Protocol (TFTP), one protocol of TCP/IP, provides simple file transmission service between the client and server. The port number is 69.

Format

tftp *<-g/-p> -l [FullPathLocalFile] -r [RemoteFile] [Host]*

Parameter Description

Table 7-35 Parameter description

Parameter	Description	Value Range
-g	Obtain files from the server	Choose one from -g and -p
-p	Upload files to the server	Choose one from -g and -p
-l	Name of a local file (full path should be opened in Huawei LiteOS)	
-r	Name of a file on the server	
Host	Server IP	

User Guide

1. Set up TFTP server. Firstly, you need to ensure that the server has been installed TFTP client, and then configure it according to following figures.

Copyright © Huawei Technologies Co., Ltd.

Click Setting to set the TFTP server.

Copyright © Huawei Technologies Co., Ltd.

Set Base Directory to the TFTP directory and then click OK to exit.

2. Huawei LiteOS board uses the tftp command to upload or download files.
3. The size of a file that is transmitted must not be greater than 32 MB.
4. tftp is a test function, the default configuration is closed, the official product is prohibited to use the function.
5. Disclaimer: Huawei is not responsible for any risks brought by using the Telnet function in official Huawei LiteOS.

Example

For example, download the **vs_server.bin** file from the server.

Output

```
Huawei LiteOS # tftp -g -l /nfs/vs_server.bin -r vs_server.bin 192.168.1.2
 TFTP transfer finish
```

If the transfer succeeds, the message TFTP transfer finish will be displayed. If the transfer fails, other printed information will be displayed to help locate the problem.

Copyright © Huawei Technologies Co., Ltd.

7.2.5.3.5 ntpdate

Function

The **ntpdate** command is used to synchronize the system time from the server.

Format

Obtain the system time from NTP server.

ntpdate [*SERVER_IP1*] [*SERVER_IP2*]...

Parameter Description

Table 7-36 Parameter description

Parameter	Description	Value Range
SERVER_IP	IP of NTP server	

User Guide

Run ntpdate [*SERVER_IP1*] [*SERVER_IP2*]...

The time of the first valid server IP will be obtained and displayed by running the **ntpdate** command.

Example

For example:

Use the **ntpdate** command to update the time of system.

Output

Use the **ntpdate** command to update the time of system.

```
Huawei LiteOS # ntpdate 192.168.1.3
time server 192.168.1.3: Mon Jun 13 09:24:25 2016
```

The displayed time in the board may be different from the server time in several hours due to different time zones.

7.2.5.3.6 dns

Function

The **dns** command is used to configure the DNS server address of the board.

Format

dns <*1-2*> <*IP*>

dns *-a*

Copyright © Huawei Technologies Co., Ltd.

Parameter Description

Table 7-37 Parameter description

Parameter	Description	Value Range
<1-2>	Choose to configure the first or second DNS server.	1~2
<IP>	IP address of the server.	
-a	Display the current configuration state.	

User Guide

Enter the **netstat** command.

Example

For example:

1. View the configuration information about the current DNS.
2. Configure the IP of the second DNS.
3. Check whether the configuration of DNS is successful.

Output

1. View the configuration information about the current DNS.
```
Huawei LiteOS # dns -a
 dns1: 208.67.222.222
 dns2: 0.0.0.0
```
2. Configure the IP of the second DNS.
```
Huawei LiteOS # dns 2 192.168.1.2
```
3. Check whether the configuration of DNS is successful.
```
Huawei LiteOS # dns -a
 dns1: 208.67.222.222
 dns2: 192.168.1.2
```

7.2.5.3.7 netstat

Function

The **netstat**, a console command, is used to view the actual network connectivity and the state of each network API device for TCP/IP network monitoring. The **netstat** command displays the statistics of TCP and UDP protocols to check the connectivity of all APIs of the board.

Format

netstat

Copyright © Huawei Technologies Co., Ltd.

Parameter Description

Table 7-38 Parameter description

Parameter	Description	Value Range
N/A	N/A	N/A

User Guide

Enter the **netstat** command.

Example

For example, enter **netstat**.

Output

Figure 7-35 Information printed using netstat

```
Huawei LiteOS # netstat
========== total sockets 32 ======  unused sockets 22 ==========
Proto   Recv-Q      Send-Q      Local Address           Foreign Address         State
tcp     0           0           0.0.0.0:0               0.0.0.0:43200           LISTEN
tcp     0           0           192.168.1.2:11111       0.0.0.0:0               LISTEN
tcp     0           0           0.0.0.0:11110           0.0.0.0:0               LISTEN

Proto   Recv-Q      Send-Q      Local Address           Foreign Address
udp     0           0           0.0.0.0:63556           192.168.1.3:22500
udp     0           0           192.168.1.2:11110       0.0.0.0:0

Type    Recv-Q      Send-Q      Local Address       Foreign Address     Protocol        HDRINCL
raw     0           0           192.168.1.2         0.0.0.0             17              0
raw     0           0           0.0.0.0             0.0.0.0             6               1

Type        Recv-Q      Send-Q      Protocol        netif
pkt-raw     0           0           806             None
pkt-raw     0           0           800             eth0
```

Table 7-39 Parameter description

Parameter	Description
Proto	Protocol type
Recv-Q	Amount of data that is not read by the user.
Send-Q	When TCP is used, the parameter specifies the amount of data that is sent and not acknowledged. When UDP is used, the parameter specifies the amount of data that is cached because IP address parsing is not complete.
Local Address	Local address and port

Copyright © Huawei Technologies Co., Ltd.

Parameter	Description
Foreign Address	Remote address and port
State	Connectivity of TCP (the parameter is meaningless to UDP)
Type	Raw socket type **raw** indicates the AF_INET type of raw socket. **pkt-raw** indicates the PF_PACKET type of raw socket.
Protocol	For an AF_INET raw socket, **protocol** indicates the type of the protocol in the IP header. For a PF_PACKET raw socket, **protocol** indicates the type of the protocol in the Ethernet header.
HDRINCL	Whether the user has enabled the IP_HDRINCL socket option on the AF_INET raw socket. **1** indicates that the user has enabled this option. **0** indicates that the user has not enabled this option.
netif	NIC that is bound to the PF_PACKET raw socket None indicates that no NIC is bound to the raw socket.

NOTE

"========== total sockets 32 ====== unused sockets 22 ========== "

The preceding printed information indicates that there are 32 sockets in total, and 22 out of them are not in use.

7.2.5.3.8 telnet

Function

telnet is used to access servers through networks from computers of terminal users.

Format

telnet [*on* | *off*]

Copyright © Huawei Technologies Co., Ltd.

Parameter Description

Table 7-40 Parameter description

Parameter	Description	Value Range
on	Turning on a server	N/A
off	Turning off a server	N/A

User Guide

- **telnet** is used to access servers through networks from computers of terminal users.
- To enable **telnet**, the network driver and the lwIP of the development board must be enabled and initialized.
- Currently, only one client can be connected to a development board using telnet and an IP address at one time.

Example

For example, enter **telnet on**.

Output

Figure 7-36 Entering telnet on

```
Huawei LiteOS  # telnet on

Huawei LiteOS  # init telnet.
```

7.2.5.3.9 pppoe

Function

pppoe is used to connect a development board to a pppoe server.

Format

pppoe start <interface> <username> <password>

pppoe stop <interface>

Copyright © Huawei Technologies Co., Ltd.

Parameter Description

Table 7-41 Parameter description

Parameter	Description	Value Range
interface	NIC name	Ethernet cards only
username	Username used to log in to a pppoe server	N/A
password	Password used to log in to a pppoe server	N/A

User Guide

- Connect your development board to a pppoe server.
- Before connecting your board to a pppoe server, the Ethernet driver of your board must be enabled and initialized.
- A board can be connected to only one pppoe server at a time.

Example

Run **pppoe start** *eth0 username 123456* to connect your board to a pppoe server.

Run **pppoe stop** *eth0* to disconnect your board from the pppoe server.

Output

Figure 7-37 Running the pppoe start eth0 username 123456 command

```
Huawei LiteOS # pppoe start eth0 username 123456
Huawei LiteOS # et0: host unique tag found, but it belongs to a connection in
state 3
pppoe: received PADO but could not find request for it
pppoe: connect success.
```

Figure 7-38 Running the pppoe stop eth0 command

```
Huawei LiteOS # pppoe stop eth0
pppoe: disconnecting..

Huawei LiteOS # pppoe: disconnect success.
```

7.2.5.3.10 tcpdump

Function

The **tcpdump** command is used to capture network packets. It does not support protocol analysis on the device. Captured network packets are used to generate a PCAP file which is analyzed using Wireshark.

Format

tcpdump -i ifname -w "path" [-c "package-count"] ["filter expression"]

Copyright © Huawei Technologies Co., Ltd.

tcpdump stop

Parameter Description

Table 7-42 Parameter description

Parameter	Description	Value Range
ifname	NIC name	This parameter can be set to a NIC name only.
path	Absolute path of a PCAP file	This parameter must be set to a valid path.
package-count	Number of network packets to be captured (This parameter is optional. If it is unspecified or set to **0**, the number of network packets to be captured is unlimited.)	Range: 0 to 4294967295
filter expression	Packet filtering expression	See the PCAP filtering rule.

User Guide

- This command can be used only after the network and file system are initialized.
- Only a single NIC can be used to capture packets.
- The direction of the data packets cannot be configured. Currently, the direction is bidirectional.

Example

Enter **tcpdump -i eth0 -w /ramfs/cap.pcap -c 15 "arp or ip"** to start capturing ARP and IP packets. The number of packets to be captured is 15.

Enter **tcpdump -i eth0 -w /ramfs/cap.pcap** to start capturing packets. The number and type of packets are not limited.

Enter **tcpdump stop** to stop capturing packets.

Output

Figure 7-39 Output of **tcpdump -i eth0 -w /ramfs/cap.pcap -c 15 "arp or ip"**

```
Huawei LiteOS # interface: eth0
filename: /ramfs/cap.pcap
count: 15
filter: arp or ip

Huawei LiteOS # tcpdump file saved.
```


Copyright © Huawei Technologies Co., Ltd.

Figure 7-40 Output of **tcpdump stop**

```
Huawei LiteOS # tcpdump stop
Huawei LiteOS # tcpdump stoped.
```

7.2.5.4 Dynamic Loading

7.2.5.4.1 mopen

Function

The **mopen** command is used to load a user module.

Format

mopen *module_path*

Parameter Description

Table 7-43 Parameter description

Parameter	Description
module_path	Path of the user module

User Guide

- **module_path** can be set to a .o file or a .so file.

Example

For example, load **/yaffs/bin/dynload/foo.o**.

Output

```
Huawei LiteOS# mopen /yaffs/bin/dynload/foo.o
 module handle: 0x80391928

 Huawei LiteOS#
```

Module handle will be returned if the loading succeeds. The returned module handle in this example is 0x80391928.

7.2.5.4.2 findsym

Function

The **findsym** command is used to query the symbol address.

Format

findsym *handle symbol_name*

Copyright © Huawei Technologies Co., Ltd.

Parameter Description

Table 7-44 Parameter description

Parameter	Description
handle	Module handle
symbol_name	Name of the symbol to be searched.

User Guide

- If the handle is 0, the command searches the symbol address in the global symbol table. (The global symbol table contains the kernel symbol and other symbols provided by user module).
- If the handle is not 0 and is valid, the command searches the symbol in the module specified by the handle.

Example

For example: search global symbol table for printf symbol and search user module (handle: 0x80391928) opened in mopen for the address of test_0 symbol.

Output

```
Huawei LiteOS# findsym 0 printf
 symbol address:0x8004500c

 Huawei LiteOS#
Huawei LiteOS# findsym 0x80391928 test_0
 symbli address:0x8030f241

 Huawei LiteOS#
```

7.2.5.4.3 call

Function

The **call** command is used to call a function with no parameters.

Format

call *func_address*

Parameter Description

Table 7-45 Parameter description

Parameter	Description
func_address	Memory address of the function

Copyright © Huawei Technologies Co., Ltd.

User Guide

- Call the function after the memory address of a function symbol has been searched in findsym.

NOTE

Arbitrary memory operations by running the call command will cause system crashes.

Example

For example: call test_0 function (address: 0x8030f241) that is searched in findsym.

Output

```
Huawei LiteOS# call 0x8030f241
 test_0

Huawei LiteOS#
```

7.2.5.4.4 mclose

Function

The **mclose** command is used to uninstall a module.

Format

mclose *module_handle*

Parameter Description

Table 7-46 Parameter description

Parameter	Description
module_handle	The handle value returned by the module that is opened in mopen.

User Guide

- If a module with specified handle is uninstalled, it cannot be searched for symbols.

Example

For example: uninstall the user module whose handle is 0x80391928.

Output

```
Huawei LiteOS# mclose 0x80391928

 Huawei LiteOS#
```

If no error information is displayed, the uninstallation is successful.

Copyright © Huawei Technologies Co., Ltd.

7.2.5.4.5 lddrop

Function

The **lddrop** command is used to uninstall the dynamic loading module.

Format

lddrop

Parameter Description

Table 7-47 Parameter description

Parameter	Description	Value Range
N/A	N/A	N/A

User Guide

- The dynamic loading function is not available if the dynamic loading module is uninstalled. To use this function, reinitialize the dynamic loading module.

Example

For example: uninstall the dynamic loading module.

Output

```
Huawei LiteOS# lddrop

Huawei LiteOS#
```

If no error information is displayed, the uninstallation is successful.

Copyright © Huawei Technologies Co., Ltd.

8 Debug Guidelines

8.1 Common Exception Locating Methods

8.2 Common Solutions

8.3 Debug Tools

8.1 Common Exception Locating Methods

8.1.1 Methods for Locating Illegal Memory Write

8.1.1.1 Locating the Exception Based on the Exception Information

Information about some key registers can be viewed in the serial port after the exception of illegal memory write occurs.

The information will help you locate the function where the exception occurs and call stack relations of the function.

Figure 8-1

```
uwExcType = 0x4
puwExcBuffAddr pc = 0x8034d3cc
puwExcBuffAddr lr = 0x8034d3cc
puwExcBuffAddr sp = 0x809ca358
puwExcBuffAddr fp = 0x809ca36c
*******backtrace begin*******
traceback 0 -- lr = 0x803482fc
traceback 0 -- fp = 0x809ca38c
traceback 1 -- lr = 0x80393e34
traceback 1 -- fp = 0x809ca3a4
traceback 2 -- lr = 0x8039e0d0
traceback 2 -- fp = 0x809ca3b4
traceback 3 -- lr = 0x80386bec
traceback 3 -- fp = 0x809ca424
traceback 4 -- lr = 0x800a6210
traceback 4 -- fp = 0x805da164
```


Copyright © Huawei Technologies Co., Ltd.

1. uwExcType (exception type)

   The value is 4, which refers to Data Abort exception. For the values that represent other exception types, see **Table 1** in **4.3.2 Development Guidelines** or the **los_exc.h** file.

2. pc (program counter)

   The pc address points to the instruction that the program is executing when the exception occurs. The debug engineer needs to open the .asm file that is generated together with the binary file that is being executed and locate the pc in the .asm file to view the instruction that is being executed. For example, as shown in **Figure 1**, search the .asm file for 8034d3cc. The result shown in **Figure 2** is obtained.

Figure 8-2

```
8034d3bc <osSlabMemFree>:
8034d3bc:   e92d48f0    push    {r4, r5, r6, r7, fp, lr}
8034d3c0:   e1a04001    mov r4, r1
8034d3c4:   e28db014    add fp, sp, #20
8034d3c8:   ebfff21e    bl  80349c48 <osSlabCtrlHdrGet>
8034d3cc:   e15420b4    ldrh    r2, [r4, #-4]
8034d3d0:   e59f308c    ldr r3, [pc, #140]  ; 8034d464 <osSlabMemFree+0xa8>
8034d3d4:   e1520003    cmp r2, r3
```

You can obtain the following information from the figure above:

a. The CPU is executing the **ldrh r2, [r4, #-4]** instruction when the exception occurs.

b. The exception occurs in the **osSlabMemFree** function.

The **ldrh r2, [r4, #-4]** instruction is used to read value from the (r4-4) memory address and load the value to register r2. **Figure 3** shows the register information printed upon exception. As shown in **Figure 3**, the value in r4 is 0xffffffff when the exception occurs.

Figure 8-3

```
R0          = 0x807243c8
R1          = 0xffffffff
R2          = 0x80000013
R3          = 0x80623de0
R4          = 0xffffffff
R5          = 0x80000013
R6          = 0x807243c0
R7          = 0x1
R8          = 0x809ca490
R9          = 0x4e4
R10         = 0x11
R11         = 0x809ca36c
R12         = 0x800000d3
SP          = 0x809ca358
LR          = 0x8034d3cc
PC          = 0x8034d3cc
CPSR        = 0xa00000d3
```

The r4 value exceeds the memory range, and therefore the Data Abort exception occurs while the CPU is executing this instruction.

According to the .asm file, the abnormal r4 value is moved from r1, which is an input parameter of the **osSlabMemFree** function. Therefore, the incorrect input parameter value 0xffffffff (or －1) is passed in when the **osSlabMemFree** function is called.

Copyright © Huawei Technologies Co., Ltd.

Next, you need to find out the caller of the **osSlabMemFree** function.

3. lr (link register)

   The call stack information starts from **backtrace begin** in the printed exception information shown in **Figure 1**. In the .asm file, search for the lr value (803482fc) corresponding to traceback 0. The result shown in **Figure 4** is obtained.

Figure 8-4

```
803482c4 <LOS_MemFree>:
803482c4:   e92d4bf0    push    {r4, r5, r6, r7, r8, r9, fp, lr}
803482c8:   e28db01c    add fp, sp, #28
803482cc:   e10f5000    mrs r5, CPSR
803482d0:   e385c0c0    orr ip, r5, #192    ; 0xc0
803482d4:   e121f00c    msr CPSR_c, ip
803482d8:   e3510000    cmp r1, #0
803482dc:   13500000    cmpne   r0, #0
803482e0:   1a000002    bne 803482f0 <LOS_MemFree+0x2c>
803482e4:   e3a00001    mov r0, #1
803482e8:   e121f005    msr CPSR_c, r5
803482ec:   e8bd8bf0    pop {r4, r5, r6, r7, r8, r9, fp, pc}
803482f0:   e1a04001    mov r4, r1
803482f4:   e1a06000    mov r6, r0
803482f8:   eb00142f    bl  8034d3bc <osSlabMemFree>
803482fc:   e3500001    cmp r0, #1
80348300:   1affffe8    bne 803482e8 <LOS_MemFree+0x24>
```

As shown in **Figure 4**, the **cmp r0, #1** instruction is executed, and therefore the **LOS_MemFree** function calls the **osSlabMemFree** function. By using this method, you can find that the calling relationships between functions are **MNT_buf_send** (service function) -> **free** -> **LOS_MemFree** -> **osSlabMemFree**.

By checking the implementation of **MNT_buf_send**, the problem of incorrectly using a pointer is found. This problem causes an incorrect address to be freed. As a result, the exception occurs.

8.1.1.2 Memory Integrity Check

The causes of illegal memory access sometimes cannot be determined by referring to the exception information as described in the previous topic. In addition, register information is often incorrect, which causes locating failure. If you suspect that memory overwriting causes the exception, whether memory overwriting causes the exception can be checked by calling the osShellCmdMemcheck memory integrity check function.

After the osShellCmdMemcheck function is called, all nodes in the dynamic memory pool are checked. When all nodes are normal, a log containing "memcheck over, all passed!" is printed. If not all nodes are normal, error information is printed.

A memory integrity check example is described as follows:

```
VOID sampleFunc(VOID *p)
{
memset(p, 0, 0x110);//memset that involves length excess; setting the scenario of
illegal memory write
}

#include "los_dlinkmem.h"
UINT32 test(UINT32 argc, CHAR **args)
{
void *p1,*p2;
```


Copyright © Huawei Technologies Co., Ltd.

```
p1 = LOS_MemAlloc((void*)OS_SYS_MEM_ADDR,0x100);
p2 = LOS_MemAlloc((void*)OS_SYS_MEM_ADDR,0x100);
dprintf("p1 = %p, p2 = %p \n", p1, p2);

osShellCmdMemcheck(0,NULL); //memory integrity check
sampleFunc(p1); // assuming that the memory is illegally written here
osShellCmdMemcheck(0,NULL); //memory integrity check

LOS_MemFree(OS_SYS_MEM_ADDR, (void *)p1);
LOS_MemFree(OS_SYS_MEM_ADDR, (void *)p2);

return 0;
}
```

Check procedure:

Step 1 Run the task command to print the task status.

Step 2 Run the test command and execute the preceding example program.

----End

Information contained in the log that is printed after the check is described as follows:

- Information of "memcheck over, all passed!" is printed after the first time when osShellCmdMemcheck function is called, indicating that no memory is illegally accessed.
- After the second time when osShellCmdMemcheck function is called, error information is printed, indicating that operations performed between two function calling cause illegal data write. Figure 1 shows the log information. "ur node: 0x81ff0078" highlighted by mark 3 indicates that memory of the current node is illegally written. As shown in figure 1, "p2= 0x81ff0188". After subtracting 0x10 which is the size of control head. That is "p2-0x10=cur node". (The prerequisite is that p1 and p2 must be connected, which is able to verify by comparing with the address printed by p1 and p2. p2-p1=0x110. The 0x110 express size of it, 0x10 is the size of the control head.)
- Information of "pre node was allocated by task:shellTask" highlighted by mark 4 indicates that illegal memory write occurs in shellTask.

Figure 8-5

```
Huawei LiteOS # task

Name                      PID   Priority  Status     StackSize  WaterLine  StackPoint  TopOfStack
----                      ---   --------  ------     ---------  ---------  ----------  ----------
Swt_Task                  0x0   0         QueuePend  0x6000     0x38c      0x80caf880  0x80ca9a28
IdleCore000               0x1   31        Ready      0x400      0x230      0x80cafce4  0x80cafa48
system_wq                 0x2   1         Pend       0x6000     0x240      0x80cb5d40  0x80cafef0
app_Task                  0x3   10        Delay      0x6000     0x34a4     0x80cbbce8  0x80cb5f08
shellTask                 0x4   9         Running    0x3000     0x6a4      0x80cc3d3c  0x80cc1420
cmdParesTask              0x5   9         Pend       0x1000     0x268      0x80cc5248  0x80cc44b0
yaffs_bg_gc               0x6   10        Delay      0x6000     0x238      0x80ccb328  0x80cc5440
RcvAnDispatch             0x7   10        Ready      0x6000     0x260      0x80da5d28  0x80d9fef8
SavePara2Flash            0x8   10        Ready      0x6000     0x238      0x80dabe70  0x80da6018
ProcessEncStreamThread    0x9   10        Ready      0x10000    0xc38      0x80dbe430  0x80dae640
UpdateTime0xdThread       0xa   10        Ready      0x6000     0x868      0x80dc44e8  0x80dbe708
IspRun                    0xb   10        Ready      0x6000     0x168c     0x80dcd510  0x80dc7a08
aenc_get                  0xc   10        Pend       0x12000    0x400      0x80e4cee0  0x80e3b250
SendAFrameToAencThread    0xd   10        Ready      0x6000     0x5a64     0x80e52f58  0x80e4d268
WORKSTATE_CHECK           0xe   10        Ready      0x6000     0x238      0x81c9efd8  0x81c99180
STORAGE_Listen_Thread_sd0 0xf   10        Ready      0x6000     0x5a8      0x81cafd80  0x81caa078
himci_Task                0x10  10        Ready      0x800      0x248      0x81ca2a28  0x81ca23b8
TIMER_CHECK               0x11  10        Ready      0x6000     0x248      0x81ca8a20  0x81ca2b88
STORAGE_Process_Thread_sd0 0x12 10        Pend       0x6000     0x544      0x81cb5c68  0x81cb0098
Index_Adjust_Thread       0x13  10        Pend       0x6000     0x934      0x81cbc150  0x81cb6990
pth14                     0x14  10        Pend       0x6000     0x270      0x81cc27d0  0x81cbc9b0
pth15                     0x15  10        Ready      0x6000     0x238      0x81cc8820  0x81cc29c8
hi_Timer                  0x16  10        Ready      0x6000     0x260      0x81cce818  0x81cc89e8
rec_tsk_proc              0x17  10        Pend       0x6000     0x270      0x81cd6020  0x81cd0200
rec_tsk_preproc           0x18  10        Pend       0x6000     0x280      0x81cdc030  0x81cd6220
rec_tsk_proc              0x19  10        Pend       0x6000     0x270      0x81ce3858  0x81cdda38
rec_tsk_preproc           0x1a  10        Pend       0x6000     0x280      0x81ce9868  0x81ce3a58
hi_Timer                  0x1c  10        Pend       0x6000     0x278      0x81cf58a8  0x81cefa90
gui_getimer               0x1d  10        Ready      0x6000     0x2e8      0x8200c390  0x820065e8

Huawei LiteOS # test
p1 = 0x81ff0078, p2 = 0x81ff0188
memcheck over, all passed!
[ERR] [LOS_MemIntegrityCheck], 437, memory check error!
stFreeNodeInfo.pstPrev:0x0 is out of legal mem range[0x80c87280, 0x87d00000]
cur node: 0x81ff0178
pre node: 0x81ff0068
pre node was allocated by task:shellTask
```


Copyright © Huawei Technologies Co., Ltd.

8.1.1.3 Check of Usable memset and memcpy Length

To accelerate the exception locating, a check of memset and memcpy length is added because memset and memcpy are most likely to cause memory overwriting. If the length to be operated exceeds the length that can be used by the operated node, a log will be printed to prompt the length excess, and the memset and memcpy operations will be canceled (a complete log cannot be viewed if the operations are not canceled). The following describes an example of a memory integrity check in which a memset and memcpy length check is enabled:

```
VOID sampleFunc(VOID *p)
{
memset(p, 0, 0x110);//memset that involves length excess; set the scenario of
illegal memory write
}

#include "los_dlinkmem.h"
UINT32 test(UINT32 argc, CHAR **args)
{
void *p1,*p2;

p1 = LOS_MemAlloc((void*)OS_SYS_MEM_ADDR,0x100);
p2 = LOS_MemAlloc((void*)OS_SYS_MEM_ADDR,0x100);
dprintf("p1 = %p, p2 = %p \n", p1, p2);

LOS_MemCheckLevelSet(LOS_MEM_CHECK_LEVEL_HIGH); //enabling the memset and memcpy
length check

osShellCmdMemcheck(0,NULL); //memory integrity check
sampleFunc(p1); // assuming that the memory is illegally written here
osShellCmdMemcheck(0,NULL); //memory integrity check

LOS_MemCheckLevelSet(LOS_MEM_CHECK_LEVEL_DISABLE);//disabling the memset and
memcpy length check

LOS_MemFree(OS_SYS_MEM_ADDR, (void *)p1);
LOS_MemFree(OS_SYS_MEM_ADDR, (void *)p2);

return 0;
}
```

Figure 1 shows the result of running the preceding code. The complete log is printed because the memset and memcpy length check is enabled and illegal memset operations can be canceled. Mark 1 indicates that memset and memcpy length excess occurs. Mark 2 indicates that the illegal operation occurs in shellTask. Mark 3 highlights lr and fp information. You can open the **vs_server** file under the **Huawei_LiteOS\out\<platform>** directory and check the recursive function calling by referring to lr values.

Copyright © Huawei Technologies Co., Ltd.

Figure 8-6

```
Huawei LiteOS # test
p1 = 0x81feff38, p2 = 0x81ff0048
LOS_MemCheckLevelSet: LOS_MEM_CHECK_LEVEL_HIGH
memcheck over, all passed!
[ERR] ---------------------------------------------
[ERR] memset: dst inode uwAvailSize is not enough uwAvailSize = 0x100 uwLength = 0x110
osBackTrace fp = 0x80cc40e0
g_stLosTask.pstRunTask->pcTaskName = shellTask
g_stLosTask.pstRunTask->uwTaskID = 4
*******backtrace begin*******
traceback 0 -- lr = 0x80061b98
traceback 0 -- fp = 0x80cc4100
traceback 1 -- lr = 0x80017898
traceback 1 -- fp = 0x80cc412c
traceback 2 -- lr = 0x800f8eec
traceback 2 -- fp = 0x80cc43fc
traceback 3 -- lr = 0x800f8f54
traceback 3 -- fp = 0x80cc4404
traceback 4 -- lr = 0x8004b1f8
traceback 4 -- fp = 0x80cc441c
traceback 5 -- lr = 0x8003b6d4
traceback 5 -- fp = 0x11111111

Name                        PID    Priority   Status     StackSize   WaterLine   StackPoint  TopOfStack
----                        ---    --------   -------    ---------   ---------   ----------  ----------
Swt_Task                    0x0    0          QueuePend  0x6000      0x38c       0x80caf880  0x80ca9a28
IdleCore000                 0x1    31         Ready      0x400       0x230       0x80cafce4  0x80cafa48
system_wq                   0x2    1          Pend       0x6000      0x240       0x80cb5d40  0x80cafef0
app_Task                    0x3    10         Ready      0x6000      0x34a4      0x80cbbce8  0x80cb5f08
shellTask                   0x4    9          Running    0x3000      0x730       0x80cc3d00  0x80cc1420
cmdParesTask                0x5    9          Pend       0x1000      0x268       0x80cc52d8  0x80cc44b0
yaffs_bg_gc                 0x6    10         Ready      0x6000      0x238       0x80ccb328  0x80cc54d0
RcvAndDispatch              0x7    10         Ready      0x6000      0x260       0x80da5d28  0x80d9fef8
SavePara2Flash              0x8    10         Ready      0x6000      0x238       0x80dabd70  0x80da5f18
ProcessEncStreamThread      0x9    10         Ready      0x10000     0xc64       0x80dbe430  0x80dae6d0
UpdateTimeOsdThread         0xa    10         Ready      0x6000      0x868       0x80dc44e8  0x80dbe708
IspRun                      0xb    10         Ready      0x6000      0x165c      0x80dcd700  0x80dc7a10
aenc_get                    0xc    10         Pend       0x12000     0x400       0x80e4cee8  0x80e3b258
SendAFrameToAencThread      0xd    10         Ready      0x6000      0x5a64      0x80e52f60  0x80e4d270
WORKSTATE_CHECK             0xe    10         Ready      0x6000      0x238       0x81c9efe0  0x81c99188
STORAGE_Listen_Thread_sd0   0xf    10         Ready      0x6000      0x5a8       0x81cafd88  0x81caa080
himci_Task                  0x10   10         Ready      0x800       0x248       0x81ca2a08  0x81ca23c0
TIMER_CHECK                 0x11   10         Ready      0x6000      0x248       0x81ca8a28  0x81ca2be0
STORAGE_Process_Thread_sd0  0x12   10         Pend       0x6000      0x544       0x81cb5c90  0x81cb00a0
Index_Adjust_Thread         0x13   10         Pend       0x6000      0x934       0x81cbc158  0x81cb6998
pth14                       0x14   10         Pend       0x6000      0x270       0x81cc27d8  0x81cbc9b8
pth15                       0x15   10         Ready      0x6000      0x238       0x81cc8828  0x81cc29d0
hi_Timer                    0x16   10         Ready      0x6000      0x298       0x81cce820  0x81cc89f0
rec_tsk_proc                0x17   10         Pend       0x6000      0x270       0x81cd6028  0x81cd0208
rec_tsk_preproc             0x18   10         Pend       0x6000      0x280       0x81cdc038  0x81cd6228
rec_tsk_proc                0x19   10         Pend       0x6000      0x270       0x81ce3860  0x81cdda40
rec_tsk_preproc             0x1a   10         Pend       0x6000      0x280       0x81ce9870  0x81ce3a60
hi_Timer                    0x1c   10         Pend       0x6000      0x278       0x81cf58b0  0x81cefa98
gui_petimer                 0x1d   10         Ready      0x6000      0x2e8       0x8200c398  0x820065f0
[ERR] ---------------------------------------------
memcheck over, all passed!
LOS_MemCheckLevelSet: LOS_MEM_CHECK_LEVEL_DISABLE
```

8.1.1.4 Global Variable Check

If the memory of a global variable is illegally accessed, you can find the address of the global variable in the **vs_server** file under the Huawei_LiteOS\build\<platform> directory and pay attention to the nearest variable before the address because it is possible that memory overwriting occurs when this variable is being used and memory allocated to the global variable is illegally accessed. An example of global variable check is described as follows:

Two global variables are defined in the **erase** file and are initialized.

```
/*------------------------------------------------------------------*/

UINT32 g_uwEraseMap[16] = {0};

UINT32 g_uwEraseCount = 0;

/*------------------------------------------------------------------*/
```

Figure 8-7

```
.bss.g_uwEraseCount
                0x800ce12c        0x4 /usr1/wangyun/liuxuwei2/temp_test/HuaweiLite_OS/out/hi3516a/lib/obj/
                wow.O
                0x800ce12c                g_uwEraseCount
.bss.g_uwEraseMap
                0x800ce130       0x40 /usr1/wangyun/liuxuwei2/temp_test/HuaweiLite_OS/out/hi3516a/lib/obj/
                wow.O
                0x800ce130                g_uwEraseMap
```


Copyright © Huawei Technologies Co., Ltd.

Then find the locations of the two global variables in the bss segment in the **vs_server** file. If memory allocated to g_uwEraseMap is illegally accessed, analyze the usage of g_uwEraseCount to check whether memory overwriting occurs.

8.1.1.5 Task Status Check

After Huawei LiteOS runs normally, run the task command to view status of all tasks. Values of **stackSize**, **WaterLine**, **StackPoint**, and **Top0fStack** can be used to determine whether a task stack causes illegal memory access.

Figure 8-8

```
HuaweiLite OS # task
Name                    PID   Priority  Status     StackSize  WaterLine  StackPoint  Top0fStack  EventMask  SemID  CPUUSE   CPUUSE10s  CPUUSE1s  MEMUSE
----                    ---   --------  --------   ---------  ---------  ----------  ----------  ---------  -----  ------   ---------  --------  ------
Swt_Task                0x0   0         QueueFend  0x6000     0x360      0x80cfbad0  0x80cf5c70  0x0        0xffff 1        1          1         -120080
IdleCore000             0x1   31        Ready      0x400      0x22c      0x80cfbf2c  0x80cfbc90  0x0        0xffff 77       78         78        -7566584
system_wq               0x2   10        Pend       0x6000     0x23c      0x80d01f90  0x80cfc138  0x0        0x2    0        0          0         0
app_Task                0x3   10        Delay      0x6000     0x3450     0x80d07f38  0x80d02150  0x1        0xffff 0        0          0         21010728
shellTask               0x4   9         Running    0x3000     0x2810     0x80d10084  0x80d0d768  0xffff     0xffff 0        0          0         -12148
cmdFaresTask            0x5   9         Pend       0x1000     0x26c      0x80d11840  0x80d10818  0x1        0xffff 0        0          0         216
pth01                   0x6   10        Delay      0x6000     0x26b0     0x80d176e8  0x80d11868  0x0        0xffff 0        0          0         -340
RcvAndDispatch          0x7   10        Ready      0x6000     0x25c      0x80d8d18   0x80d2ee0   0x0        0xffff 0        0          0         -2704
SavePara2Flash          0x8   10        Ready      0x6000     0x234      0x80dfee60  0x80df9000  0x0        0xffff 0        0          0         -340
ProcessEncStreamThread  0x9   10        Ready      0x10000    0xc40      0x80e11440  0x80e016b8  0x1        0xffff 3        4          4         8593536
UpdateTimeOsdThread     0xa   10        Delay      0x6000     0x860      0x80e174d8  0x80e116f0  0x0        0xffff 0        0          0         3008
IspRun                  0xb   10        Ready      0x6000     0x1604     0x80e206d8  0x80e1a9f0  0x1        0xffff 11       9          10        -485952
aenc_get                0xc   10        Pend       0x12000    0x3f4      0x80e9fe8   0x80e8e238  0x1        0xffff 0        0          0         0
SendAFrameToAencThread  0xd   10        Ready      0x6000     0x5a3c     0x80ea5f70  0x80ea0250  0x0        0xffff 4        4          4         -345056
WORKSTATE_CHECK         0xe   10        Ready      0x6000     0x234      0x81cf23b0  0x81cec550  0x0        0xffff 0        0          0         -3136
himci_Task              0xf   10        Ready      0x800      0x4d4      0x81cf4268  0x81cf3c20  0x1        0xffff 0        0          0         653876
himci_Task              0x10  10        Ready      0x800      0x460      0x81cf5a8   0x81cf5490  0x0        0xffff 0        0          0         -40
TIMER_CHECK             0x11  10        Delay      0x6000     0x7ec      0x81cfba60  0x81cf5cb0  0x1        0xffff 0        0          0         -1060
```

The status of a task is described as follows by taking the task name shellTask as an example:

StackSize = 0x3000 (size of the stack allocated to the task when the task is created)

WaterLine = 0x2810 (size of the used memory of the stack)

StackPoint = 0x80d10084 (stack pointer that points to the address of the task)

Top0fStack = 0x80d0d768 (top of the stack)

MaxStackPoint = Top0fStack + StackSize = 0x80d10768 (maximum range of accessible stack)

Compare the **WaterLine** value with the **StackSize** value. If the **WaterLine** value is greater than the **StackSize** value, the task causes illegal memory access.

Check the **StackPoint** value, which should range from the **TopOfStack** value to the **MaxStackPoint** value. If the **StackPoint** value is not in this range, the task stack causes illegal memory access.

8.1.2 Method for Locating a Deadlock

Basic Information

Table 8-1

Applicable Scope	Description
Operating system	Huawei LiteOS

Symptom

Two tasks are deadlocked but the shell function is available.

Copyright © Huawei Technologies Co., Ltd.

Cause Analysis

A deadlock occurs when both the following conditions are met:

- Task A holds mutex X and waits forever for mutex Y.
- Task B holds mutex Y and waits forever for mutex X.

In this case, task A and task B are deadlocked.

Solution

Step 1 Enter **task** in the shell CLI of LiteOS.

The system displays the status and information of all running tasks.

Figure 8-9 Status and information of running tasks in the system

```
Huawei LiteOS # task

Name            TID  Priority  Status     StackSize  WaterLine  StackPoint  TopOfStack  EventMask  SemID CPUUSE  CPUUSE10s  CPUUSE1s  MEMUSE
----            ---  --------  --------   ---------  ---------  ----------  ----------  ---------  ----- -------  ---------  --------  -------
Swt_Task        0x0  0         QueuePend  0x6000     0x180      0x802cc630  0x802c6720  0x0        0xffff   0.0       0.0        0.0     0
IdleCore000     0x1  31        Ready      0x800      0x64       0x801c4774  0x801c3fd8  0x0        0xffff  99.0      98.7       99.9     0
devfreq_wq      0x2  1         Pend       0x6000     0x140      0x802d2688  0x802cc738  0x1        0xffff   0.0       0.0        0.0     0
system_wq       0x3  1         Pend       0x6000     0x140      0x802d86b8  0x802d2768  0x1        0xffff   0.0       0.0        0.0     0
Tsk001A         0x4  24        Pend       0x600      0x120      0x802d8cf0  0x802d8780  0x0        0xffff   0.0       0.0        0.0     35888
shellTask       0x5  9         Running    0x3000     0x454      0x801cbaf4  0x801c8f48  0xffff     0xffff   0.9       1.2        0.0     5512
cmdParesTask    0x6  9         Pend       0x1000     0x288      0x801cce50  0x801cbf58  0x1        0xffff   0.0       0.0        0.0     80
Tsk001B         0x7  24        Pend       0x600      0x120      0x802d9300  0x802d8d90  0x0        0xffff   0.0       0.0        0.0     0
```

Step 2 Filter the task that may be deadlocked, record its task ID, and enter **task** plus the task ID in the shell CLI to view the call stack information of the task.

Figure 8-10 Call stack information of a specified task

```
Huawei LiteOS # task 4
TaskName = Tsk001A
TaskID = 0x4
*******backtrace begin*******
traceback 0 -- lr = 0x80070b3c    fp = 0x802d8d64
traceback 1 -- lr = 0x80076f00    fp = 0x802d8d7c
traceback 2 -- lr = 0x801128ac    fp = 0x11111111

Name            TID  Priority  Status     StackSize  WaterLine  StackPoint  TopOfStack  EventMask  SemID CPUUSE  CPUUSE10s  CPUUSE1s  MEMUSE
----            ---  --------  --------   ---------  ---------  ----------  ----------  ---------  ----- -------  ---------  --------  -------
Swt_Task        0x0  0         QueuePend  0x6000     0x180      0x802cc630  0x802c6720  0x0        0xffff   0.0       0.0        0.0     0
IdleCore000     0x1  31        Ready      0x800      0x64       0x801c4774  0x801c3fd8  0x0        0xffff  99.2      99.8       99.9     0
devfreq_wq      0x2  1         Pend       0x6000     0x140      0x802d2688  0x802cc738  0x1        0xffff   0.0       0.0        0.0     0
system_wq       0x3  1         Pend       0x6000     0x140      0x802d86b8  0x802d2768  0x1        0xffff   0.0       0.0        0.0     0
Tsk001A         0x4  24        Pend       0x600      0x120      0x802d8cf0  0x802d8780  0x0        0xffff   0.0       0.0        0.0     35888
shellTask       0x5  9         Running    0x3000     0x4cc      0x801cba7c  0x801c8f48  0xffff     0xffff   0.7       0.1        0.0     5532
cmdParesTask    0x6  9         Pend       0x1000     0x288      0x801cce50  0x801cbf58  0x1        0xffff   0.0       0.0        0.0     116
Tsk001B         0x7  24        Pend       0x600      0x120      0x802d9300  0x802d8d90  0x0        0xffff   0.0       0.0        0.0     0
```

Step 3 Record the **lr** value (for example, **0x80070b3c**) of traceback 0. Locate the **lr** value in the **vs_server.asm** disassembly file, as shown in the following figure. Locate the pending location (for example, **task_f01**) and called API of the mutex.

Figure 8-11 Pending location in the disassembly file

```
80070b04 <task_f01>:
80070b04:   e92d4830    push    {r4, r5, fp, lr}
80070b08:   e59f407c    ldr r4, [pc, #124]  ; 80070b8c <task_f01+0x88>
80070b0c:   e28db00c    add fp, sp, #12
80070b10:   e3e01000    mvn r1, #0
80070b14:   e5940000    ldr r0, [r4]
80070b18:   eb000d27    bl  80073fbc <LOS_MuxPend>
80070b1c:   e3500000    cmp r0, #0
80070b20:   1a000015    bne 80070b7c <task_f01+0x78>
80070b24:   e3a0000a    mov r0, #10
80070b28:   eb001836    bl  80076c08 <LOS_TaskDelay>
80070b2c:   e59f305c    ldr r3, [pc, #92]   ; 80070b90 <task_f01+0x8c>
80070b30:   e3e01000    mvn r1, #0
80070b34:   e5930000    ldr r0, [r3]
80070b38:   eb000d1f    bl  80073fbc <LOS_MuxPend>
80070b3c:   e3500000    cmp r0, #0
80070b40:   1a000009    bne 80070b6c <task_f01+0x68>
80070b44:   e3a0000a    mov r0, #10
80070b48:   eb00182e    bl  80076c08 <LOS_TaskDelay>
```


Copyright © Huawei Technologies Co., Ltd.

Step 4 Locate the call locations of traceback 1 and traceback 2, and check whether a deadlock occurs based on the context.

----End

Suggestion and Summary

None.

8.2 Common Solutions

8.2.1 Solutions to Illegal Memory Access

8.2.1.1 Illegal Memory Access Caused by the Audio Library

Basic Information

Table 8-2

Applicable for	Description
Operating system	Huawei LiteOS

Symptom

In a service version of Huawei LiteOS that runs audio libraries, Huawei LiteOS sometimes breaks down.

Cause Analysis

Processing of the AEC algorithm needs a buffer that is aligned on the boundary of 8 bytes. However, the alignment of the buffer is incorrectly processed ((state->pAEC_buffer) & 0xFFFFFFF8)) when Huawei LiteOS was encapsulated by the upper-layer AEC service. If the buffer that is allocated is not aligned on the boundary of 8 bytes, the buffer address will be rolled back by 8 bytes after the processing of the AEC algorithm, resulting in memory overwriting. This problem was not found in Linux because the memory allocated by the malloc function is aligned, or because the overwritten memory is not used.

Solution and Summary

Call the malloc function to allocate a buffer that is aligned on the boundary of 8 bytes.

```
state->pAEC_buffer = malloc(s32AecSize + 8);

if(HI_NULL == state->pAEC_buffer)

{

…;
```


Copyright © Huawei Technologies Co., Ltd.

```
}
/*pAEC_buffer should be 8 byte alignment*/
if((state->pAEC_buffer - (HI_VOID*)HI_NULL) & 0x7)
{
s32AlignNum = 8 - ((state->pAEC_buffer - (HI_VOID*)HI_NULL) & 0x7);
}
else
{
s32AlignNum = 0;
}
...
(void *)((HI_U8 *)(state->pAEC_buffer) + s32AlignNum)
```

Suggestion and Summary

Be careful to avoid memory overwriting in operations on memory, especially on pointers.

8.2.1.2 Unreadable Audio Task Name

Basic Information

Table 8-3

Applicable for	Description
Operating system	Huawei LiteOS

Symptom

An unreadable audio task name appears after the task command is typed in the shell of Huawei LiteOS.

Cause Analysis

The possible causes are as follows:

1. Memory overwriting
2. Use of a wild pointer

If memory is overwritten, and the task name overflows, it is likely that large-scale memory overwriting causes the task name being unreadable. However, after the task command is run, the printed information apart from the task name is normal, and the task name is passed in by a pointer, indicating that the use of a wild pointer is more likely to cause the problem. In addition, the memory check performed by typing the memcheck command in shell is passed

Copyright © Huawei Technologies Co., Ltd.

whether the task name is readable or not. Therefore, memory overwriting does not cause the problem.

```
HI_CHAR aszThreadName[16] = "adec_sendao";
```

The task name stored in the task control block is a character string pointer, indicating that the pointer only stores the address of the character string that contains the task name. A local variable of a function can be found in the startup thread of the task. After the function is executed, the local variable exits, and the function stack is released, which means the memory of the released stack can be accessed and used by other operations. If the memory of the released stack is accessed and used, the address of the character string containing the task name may be overwritten when the task status is checked by running the task command (if the address happens to be not overwritten, the displayed task name is normal, but the released function stack is still operated, which is illegal).

Solution

Pass in the character string that contains the task name rather than passing in a pointer that points to the character string.

Suggestion and Summary

None.

8.2.1.3 Illegal Memory Access Caused by a Global Variable

Basic Information

Table 8-4

Applicable for	Description
Operating system	Huawei LiteOS

Symptom

During the debug of Huawei LiteOS, a global variable should be set to 0. However, the value changes into a nonzero value when the global variable is used.

Cause Analysis

That the global variable is illegally accessed is likely to be the cause. Memory overwriting occurs when the memcpy and memset operations are performed on the variable before the global variable, and the global variable is illegally written.

Solution

View the **map** file and check whether the variable before the global variable is incorrectly operated, which causes illegal memory write.

Copyright © Huawei Technologies Co., Ltd.

Suggestion and Summary

This solution can effectively locate the variable that causes the illegal memory write.

8.3 Debug Tools

8.3.1 LiteDebug

Function Description

LiteDebug is developed by Huawei LiteOS for debugging and locating faults in Windows. Huawei-developed LiteDebug integrates Coredump and GDB, simplifying user operations. Based on LiteDebug, GDB implements code debugging and fault locating through a single serial port, which resolves the inconvenience caused by two serial ports. LiteDebug also enables exception file receiving by simply clicking a button for Coredump and locates faults efficiently.

The LiteDebug installation and usage guidelines are as follows:

1. **Environment Setup**
2. Use one of the following debugging tools based on situation requirements:
   - **Coredump**
   - **GDB**

Environment Setup

Step 1 Install Python 2.7.3 or later on Windows.

Step 2 Copy the LiteDebug package in the **Huawei_LiteOS/tools/win** directory to any directory on Windows, and click the LiteDebug application in the **LiteDebug** directory. In the displayed **LiteDebug** window, perform the following operations:

1. Set **Serial line to connect to** and **Speed (baud)**.
2. Set **Current Directory** to the directory where **vs_server** is stored, for example, hi3518ev200.
3. Select **Tftp Server**.
4. Click **Start**.

Copyright © Huawei Technologies Co., Ltd.

Step 3 On the LiteShell CLI, burn the image to the memory.

Copyright © Huawei Technologies Co., Ltd.

----End

Coredump

Basic Concept

Coredump saves core files when system exceptions occur, used to locate faults on Linux. Generally, you can locate faults in Huawei LiteOS by adding print information or viewing memory information in **vs_server.asm** based on exception addresses generated by program exceptions. Coredump saves and analyzes core files, which allows you to view information about the memory, register status, stack pointers, memory management, and stack calling of functions during program running. By using Coredump, you can locate faults accurately and efficiently.

Copyright © Huawei Technologies Co., Ltd.

Usage Guidelines

Step 1 If you select **Enable GCC -g Option** and **Enable Coredump**, after Coredump startup information is displayed, in the LiteDebug window, set **File Save** to the directory where **vs_server** resides, and select **Coredumper** to receive data. When a message containing **100.00%** is displayed, data has been completely received. See the following figures.

Copyright © Huawei Technologies Co., Ltd.

Step 2 Enter the following command in **vs_server**:

arm-hisiv300-linux-gdb vs_server core_liteos.lcore

Enter the GDB command to view exception information, as shown in the following figure. For details about the GDB command, see **GDB**.

```
jpin@HGH1000030636:~/code/work/coredump/Huawei_LiteOS/out/hi3518ev200$
jpin@HGH1000030636:~/code/work/coredump/Huawei_LiteOS/out/hi3518ev200$ arm-hisiv300-linux-gdb vs_server core_liteos.lcore
GNU gdb (Hisilicon_v300) 7.4.50.20120716-cvs
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i386-redhat-linux --target=arm-hisiv300-linux-uclibcgnueabi".
For bug reporting instructions, please see:
<http://www.hisilicon.com/cn/service/claim.html>...
Reading symbols from /user1/jpin/code/work/coredump/Huawei_LiteOS/out/hi3518ev200/vs_server...done.
[New process 1]
Core was generated by `vs_server'.
#0  LOS_Panic (fmt=0x800d2f2c "") at src/los_exc.c:206
206     }/*lint !e438*/
(gdb) bt
#0  LOS_Panic (fmt=0x800d2f2c "") at src/los_exc.c:206
#1  0x800968d0 in osShMsgParse (pMsg=pMsg@entry=0x800f5010 <g_acShellbuf>, TrasID=0) at src/base/shMsg.c:450
#2  0x80096968 in osShMsgParse (pMsg=<optimized out>, TrasID=0) at src/base/shMsg.c:506
#3  osShellTask (uwParam1=<optimized out>, uwParam2=<optimized out>, uwParam3=<optimized out>, uwParam4=<optimized out>) at
#4  0x800118f4 in osTaskEntry ()
#5  0x8007b044 in osExcHandleEntry (uwExcType=2148347692, puwExcBufAddr=0x8013d198) at src/los_exc.c:310
Backtrace stopped: frame did not save the PC
(gdb)
#0  LOS_Panic (fmt=0x800d2f2c "") at src/los_exc.c:206
#1  0x800968d0 in osShMsgParse (pMsg=pMsg@entry=0x800f5010 <g_acShellbuf>, TrasID=0) at src/base/shMsg.c:450
#2  0x80096968 in osShMsgParse (pMsg=<optimized out>, TrasID=0) at src/base/shMsg.c:506
#3  osShellTask (uwParam1=<optimized out>, uwParam2=<optimized out>, uwParam3=<optimized out>, uwParam4=<optimized out>) at
#4  0x800118f4 in osTaskEntry ()
#5  0x8007b044 in osExcHandleEntry (uwExcType=2148347692, puwExcBufAddr=0x8013d198) at src/los_exc.c:310
Backtrace stopped: frame did not save the PC
(gdb)
```

----End

Precautions

- The commands used to burn the compiled Huawei LiteOS image to the board is those used to burn the image into memory.
- If the GCC -g option is enabled, Coredump performance may be affected. However, if the option is disabled, stack information may be lost.
- Coredump is a debug tool and disabled by default. The use of Coredump is not allowed in formal products.
- Disclaimer: Huawei shares no responsibilities for any risks caused during the use of Coredump in commercialized products.

GDB

Basic Concept

GDB is a debug tool. GDB is connected to a platform through a serial port. GDB provides different versions for Windows and Linux. The platform is used as a GDB stub that can debug in single steps, set breakpoints, and print global variables. GDB enables you to debug a program and stop the program at your desired location, during which time you can view variables, registers, memory, and stacks. You can also modify variables and memory value. GDB debugs C and C++ programs. In Huawei LiteOS, the -g debugging mode is enabled by default.

Copyright © Huawei Technologies Co., Ltd.

Usage Guidelines

Step 1 If you select **Enable GCC -g Option** and **Enable GDB** in menuconfig, after GDB startup information is displayed, select **Liteos GDB Server** in the LiteDebug window. Then, open GDB. See the following figures.

You can select GDB running on Linux or on Windows.

Copyright © Huawei Technologies Co., Ltd.

Step 2 On Linux: Load **vs_server** on GDB, and connect GDB to a board using UDP.

arm-hisiv300-linux-gdb ./out/hi3518ev200/vs_server

target remote udp:10.173.211.122:8001

```
jpin@HGH1000030636:~/code/work/coredump/Huawei_LiteOS$ arm-hisiv300-linux-gdb ./out/hi3518ev200/vs_server
GNU gdb (Hisilicon_v300) 7.4.50.20120716-cvs
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i386-redhat-linux --target=arm-hisiv300-linux-uclibcgnueabi".
For bug reporting instructions, please see:
<http://www.hisilicon.com/cn/service/claim.html>...
Reading symbols from /user1/jpin/code/work/coredump/Huawei_LiteOS/out/hi3518ev200/vs_server...done.
(gdb) target remote udp:10.173.211.122:8001
warning: The remote protocol may be unreliable over UDP.
Some events may be lost, rendering further debugging impossible.
Remote debugging using udp:10.173.211.122:8001
0x8007c568 in main () at los_config.c:246
246         PRINTK("-----------GDB start,please open GDB tools-----------\n");
(gdb)
```

Step 3 On Windows: Click **Open GDB** in the following LiteDebug window. In the displayed CLI, connect GDB to a board using UDP or TCP. The following uses UDP as an example:

target remote udp::8001

Copyright © Huawei Technologies Co., Ltd.

----End

Precautions

- Currently, only the GDB for the hi3519 platform is supported.
- The commands used to burn the compiled Huawei LiteOS image to the board is those used to burn the image into memory.
- By default, the GCC -g option is enabled for GDB. This may affect performance. However, if the option is disabled, stack information may be lost.
- GDB only supports basic commands. Commands such as **run** and **watch** are not supported due to hardware problems. If commands are incorrectly used, Huawei LiteOS may be abnormal.
- GDB is a debug tool and is disabled by default. The use of GDB is not allowed in commercialized products.
- Disclaimer: Huawei is not responsible for any risks brought by using GDB in official Huawei LiteOS.

Copyright © Huawei Technologies Co., Ltd.

Command Reference

NOTE

The abbreviation of each command in the table is used in the same way as each command. For example, the function of **directory** *filepath* is the same as that of **dir** *filepath*. The abbreviation of each command is used in **Table 1** to introduce how to use each command:

Table 8-5 GDB commands

Command	Description	Usage	Parameter Description
directory	**directory** is abbreviated as **dir**. This command is used to set the directory that stores the makefile compiled by source code.	dir *filepath*	filepath: directory that stores the makefile compiled by source code
break	**break** is abbreviated as **b**. This command is used to set a breakpoint.	b *line*	line: The breakpoint is set at the line specified by **line**.
		b *func*	func: function name. The breakpoint is set at entry to the function.
tbreak	**tbreak** is abbreviated as **tb**. This command is used to set a temporary breakpoint and is used in the similar way to **break**. The breakpoint is effective for once only.	tb *func*	func: function name. The breakpoint is set at the function entry point. (The breakpoint is effective for once only.)
disable	**disable** is abbreviated as **dis**. This command is used to disable a breakpoint.	dis *n*	Disable the breakpoint number *n*.
enable	**enable** is abbreviated as **ena**. This command is used to enable a breakpoint.	ena *n*	Enable the breakpoint number *n*.
clear	**clear** is abbreviated as **cl**. This command is used to clear the breakpoint at a line of the current file.	clear *line*	Clear the breakpoint at the line specified by **line**.
info	**info** is abbreviated as **i**. This command is used to display the execution result of a command.	i b	Display the information about all breakpoints.
		i reg	Display the information about all registers.
delete	**delete** is abbreviated as **d**. This command is used to delete breakpoints.	d	Delete all breakpoints.
		d *n*	Delete the breakpoint number *n*.

Copyright © Huawei Technologies Co., Ltd.

Command	Description	Usage	Parameter Description
where	**where** is abbreviated as **whe**. This command is used to view the list of running stacks.	whe	View the information about the current stacks.
up	Go up *n* levels in the stack.	up *n*	Go up *n* levels in the stack. (*n* must not be greater than the number of levels in the stack.)
down	Go down *n* levels in the stack.	down *n*	Go down *n* levels in the stack. (*n* must not be greater than the number of levels in the stack.)
continue	**continue** is abbreviated as **c**. This command is used to continue program execution until the next breakpoint (or end).	c	Continue program execution until the next breakpoint (or end).
next	**next** is abbreviated as **n**. Run the next line of the program. If the next line is a function call, this command does not step into the function. The main difference between this command and **step** is that **step** steps into user-defined functions when encountering them. However, **next** directly call the functions without entering the functions.	n	Run the next line of the program. If the next line is a function call, this command does not step into the function.
step	**step** is abbreviated as **s**. Run the next line of the program. If the next line is a function call, this command steps into the function.	s	Run the next line of the program. If the next line is a function call, this command steps into the function.
until	When you single step through a loop, **until** makes your program continue execution until it exits the loop.	until *line*	Jump out of the loop and reach the line specified by **line**.
finish	Run the program until the current function is finished and returns and print information including stack address, return value of the function, and the passed parameter values.	finish	Run the program until the current function is finished and returns and print information including stack address, return value of the function, and the passed parameter values.

Copyright © Huawei Technologies Co., Ltd.

9 Standard Libraries

About This Chapter

9.1 POSIX APIs

9.2 Libc/Libm APIs

9.3 C++ Compatibility Specifications

9.1 POSIX APIs

9.1.1 POSIX Adaption APIs

Huawei LiteOS provides a set of POSIX adaption APIs. The following table lists specifications of POSIX adaption APIs.

Header File	API	Type	Description	Remark
sys/socket.h	accept	Function	Accept a connection on a socket.	For details on this API, see the lwip_accept API in *Huawei LiteOS LwIP API Reference*. The implementation of these two APIs is the same.

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
sys/socket.h	bind	Function	Locate a socket.	For details on this API, see the lwip_bind API in *Huawei LiteOS LwIP API Reference*. The implementation of these two APIs is the same.
time.h	clock	Function	Return the time that the processor spends on calling a process or a function.	
time.h	clock_getres	Function	Get the resolution of a specified clock.	
time.h	clock_gettime	Function	Retrieve the time of a specified clock.	
time.h	clock_settime	Function	Set the time of a specified clock.	
sys/socket.h	connect	Function	Make a connection on a socket.	For details on this API, see the lwip_connect API in *Huawei LiteOS LwIP API Reference*. The implementation of these two APIs is the same.
time.h	difftime	Function	Calculate the time elapsed between two times.	

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
dlfcn.h	dlclose	Macro	Unload an opened dynamic link library.	The implementation of this API is the same as that of the LOS_ModuleUnload API.

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
dlfcn.h	dlopen	Macro	Open a dynamic link library file in a specified mode.	Standard: The second parameter of dlopen specifies the symbol resolution mode. RTLD_LAZY: Undefined symbols in the dynamic link library are not resolved before dlopen returns. RTLD_NOW: All undefined symbols in the dynamic link library should be resolved before dlopen returns. If they are not resolved, dlopen will return **NULL**. Huawei LiteOS: The symbol resolution mode parameter is not supported when the dynamic link library is opened. The symbol resolution behavior is the same as that of in standard RTLD_NOW mode. The implementation of this API is the same as that of the LOS_SoLoad API.

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
dlfcn.h	dlsym	Macro	Take a handle of a dynamic link library and a symbol name, and return an address of a function or a variable.	The implementation of this API is the same as that of the LOS_FindSymByName API.
sys/socket.h	getpeername	Function	Get the peer address of a socket	For details on this API, see the lwip_getpeername API in *Huawei LiteOS LwIP API Reference*. The implementation of these two APIs is the same.
unistd.h	getpid	Function	Get the ID of a process.	The return type differs from that in Huawei LiteOS.
sys/socket.h	getsockname	Function	Get the name of a socket.	For details on this API, see the lwip_getsockname API in *Huawei LiteOS LwIP API Reference*. The implementation of these two APIs is the same.
sys/socket.h	getsockopt	Function	Get the socket options.	For details on this API, see the lwip_getsockopt API in *Huawei LiteOS LwIP API Reference*. The implementation of these two APIs is the same.
sys/time.h	gettimeofday	Function	Get the current time.	

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
sys/time.h	settimeofday	Function	Set the current time.	
time.h	gettimeofday64	Function	Get the current time.	This API is a custom API of Huawei LiteOS. You can use the API to obtain the system time of a 32-bit system since year 2038.
time.h	settimeofday64	Function	Set the current time.	This API is a custom API of Huawei LiteOS. You can use the API to set the system time of a 32-bit system since year 2038.
in.h	htonl	Function	Convert an unsigned long integer from host byte order to network byte order.	The implementation of this API is the same as that of the lwip_htonl API.
in.h	htons	Function	Convert an unsigned short integer from host byte order to network byte order.	The implementation of this API is the same as that of the lwip_htons API.
arpa/inet.h	inet_aton	Function	Convert a string in the Internet standard dot notation to a network address.	For details on this API, see the inet_aton API in *Huawei LiteOS LwIP API Reference*.
arpa/inet.h	inet_addr	Function	Convert an address expressed in the standard dotted-decimal notation to in_addr.	For details on this API, see the in_addr API in *Huawei LiteOS LwIP API Reference*.
mqueue.h	mq_open	Function	Open a message queue.	

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
mqueue.h	mq_receive	Function	Receive a message from a message queue.	
mqueue.h	mq_send	Function	Send a message to a message queue.	
mqueue.h	mq_setattr	Function	Set the attribute of a message queue.	
mqueue.h	mq_timedreceive	Function	Receive messages at a scheduled time.	
mqueue.h	mq_timedsend	Function	Send messages at a scheduled time.	Standard: A deadline for the send time must be specified. You are allowed to enter a deadline earlier than the current time, but the operating system considers the deadline invalid. Huawei LiteOS: The time interval between consecutive receipts must be specified. It is not allowed to enter a negative value while specifying the length of time interval.
mqueue.h	mq_unlink	Function	Remove a message queue.	

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
time.h	nanosleep	Function	High-resolution (nanosecond precision) sleep.	Standard: nanosleep achieves sleep for nanoseconds. If the call is interrupted by a signal, the remaining sleep time is written into the second parameter. Huawei LiteOS: Currently, nanosleep achieves sleep for the time interval of tick (10ms) precision, and the second parameter is not supported. The passed-in number of seconds must not be greater than 4292 seconds.
in.h	ntohl	Function	Convert an unsigned long integer from network byte order to host byte order.	The implementation of this API is the same as that of the lwip_ntohl API.
in.h	ntohs	Function	Convert an unsigned short integer from network byte order to host byte order.	The implementation of this API is the same as that of the lwip_ntohs API.
pthread.h	pthread_attr_getinheritsched	Function	Get the scheduling mode of a task.	
pthread.h	pthread_attr_getschedparam	Function	Get task scheduling priority.	

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
pthread.h	pthread_attr_getschedpolicy	Function	Get the task scheduling policy attribute.	Standard: The scheduling policy can be SCHED_OTHER, SCHED_FIFO, or SCHED_RR. Huawei LiteOS: The scheduling policy must be SCHED_RR.
pthread.h	pthread_attr_getscope	Function	Get the task scope attribute.	Standard: The task scope can be either PTHREAD_SCOPE_SYSTEM or PTHREAD_SCOPE_PROCESS. Huawei LiteOS: The task scope must be PTHREAD_SCOPE_SYSTEM.
pthread.h	pthread_attr_getstacksize	Function	Get the size of task attribute stack.	
pthread.h	pthread_attr_init	Function	Initialize task attributes.	
pthread.h	pthread_attr_setdetachstate	Function	Set the detach state of task attributes.	
pthread.h	pthread_attr_setinheritsched	Function	Set the scheduling mode of a task.	

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
pthread.h	pthread_attr_setschedparam	Function	Set task scheduling priority.	Standard: A larger value indicates a higher priority. Huawei LiteOS: A larger value indicates a lower priority. Note: The **inheritsched** field of the **pthread_attr_t task** attribute needs to be set to **PTHREAD_EXPLICIT_SCHED**, otherwise the task scheduling priority configuration will not take effect. The default value is **PTHREAD_INHERIT_SCHED**.
pthread.h	pthread_attr_setschedpolicy	Function	Set the task scheduling policy attribute.	Standard: The scheduling policy can be SCHED_OTHER, SCHED_FIFO, or SCHED_RR. Huawei LiteOS: The scheduling policy must be SCHED_RR.

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
pthread.h	pthread_attr_setscope	Function	Set the task scope.	Standard: The task scope can be either PTHREAD_SCOPE_SYSTEM or PTHREAD_SCOPE_PROCESS. Huawei LiteOS: The task scope must be PTHREAD_SCOPE_SYSTEM.
pthread.h	pthread_attr_setstacksize	Function	Set the size of task attribute stack.	
pthread.h	pthread_cancel	Function	Cancel a task.	A task can be canceled at a blocking point. Huawei LiteOS: The PTHREAD_CANCEL_ASNCHRONOUS status must be set before calling pthread_cancel to cancel a task.
pthread_cond.h	pthread_cond_broadcast	Function	Unblock all threads blocked on a condition variable.	
pthread_cond.h	pthread_cond_destroy	Function	Destroy a condition variable.	
pthread_cond.h	pthread_cond_init	Function	Initialize a condition variable.	
pthread_cond.h	pthread_cond_signal	Function	Unblock threads blocked on a condition variable.	

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
pthread_cond.h	pthread_cond_timedwait	Function	Wait on a condition variable within a timeout interval.	Standard: A deadline for the wait period must be specified. You are allowed to enter a deadline earlier than the current time, but the operating system considers the deadline invalid. Huawei LiteOS: The time interval between consecutive receipts must be specified. It is not allowed to enter a negative value while specifying the length of time interval.
pthread_cond.h	pthread_cond_wait	Function	Wait on a condition variable.	
pthread_cond.h	pthread_condattr_getpshared	Function	Get the attributes of a condition variable.	Standard: The attribute can be either PTHREAD_PROCESS_PRIVATE or PTHREAD_PROCESS_SHARED. Huawei LiteOS: The attribute must be PTHREAD_PROCESS_PRIVATE.

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
pthread_cond.h	pthread_condattr_setpshared	Function	Set the attributes of a condition variable.	Standard: The attribute can be either PTHREAD_PROCESS_PRIVATE or PTHREAD_PROCESS_SHARED. Huawei LiteOS: The attribute must be PTHREAD_PROCESS_PRIVATE.
pthread.h	pthread_create	Function	Create a task.	
pthread.h	pthread_detach	Function	Detach a task.	
pthread.h	pthread_equal	Function	Determine whether the threads are the same.	
pthread.h	pthread_exit	Function	Terminate a task.	
pthread.h	pthread_getschedparam	Function	Get task priority and task scheduling policy.	Standard: The scheduling policy can be SCHED_OTHER, SCHED_FIFO, or SCHED_RR. Huawei LiteOS: The scheduling policy must be SCHED_RR.
pthread.h	pthread_join	Function	Block a task.	
mutex.h	pthread_mutex_destroy	Function	Destroy a mutex.	
mutex.h	pthread_mutex_getprioceiling	Function	Get the upper limit of the mutex priority.	
mutex.h	pthread_mutex_init	Function	Initialize a mutex.	

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
mutex.h	pthread_mutex_lock	Function	Lock a mutex.	
mutex.h	pthread_mutex_setprioceiling	Function	Set the upper limit of the mutex priority.	
mutex.h	pthread_mutex_trylock	Function	Attempt to lock a mutex.	
mutex.h	pthread_mutex_unlock	Function	Unlock a mutex.	
mutex.h	pthread_mutexattr_destroy	Function	Destroy the mutex attributes.	
mutex.h	pthread_mutexattr_getprioceiling	Function	Get the priority ceiling attribute of a mutex.	
mutex.h	pthread_mutexattr_getprotocol	Function	Get the protocol in the mutex attribute.	
mutex.h	pthread_mutexattr_gettype	Function	Get the type in the mutex attribute.	
mutex.h	pthread_mutexattr_init	Function	Initialize the mutex attributes.	
mutex.h	pthread_mutexattr_setprioceiling	Function	Set the priority ceiling attribute of a mutex.	
mutex.h	pthread_mutexattr_setprotocol	Function	Set the protocol in the mutex attribute.	
mutex.h	pthread_mutexattr_settype	Function	Set the type in the mutex attribute.	
pthread.h	pthread_once	Function	Operate the task once.	
pthread.h	pthread_self	Function	Get the task ID.	
pthread.h	pthread_setcancelstate	Function	Set the cancelability state.	

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
pthread.h	pthread_setcanceltype	Function	Set the cancelability type.	
pthread.h	pthread_setschedparam	Function	Set the priority and the scheduling policy of a task.	Standard: The scheduling policy can be SCHED_OTHER, SCHED_FIFO, or SCHED_RR. Huawei LiteOS: The scheduling policy must be SCHED_RR.
pthread.h	pthread_testcancel	Function	Cancel a task.	
sys/socket.h	recv	Function	Receive data from a socket.	For details on this API, see the lwip_recv API in *Huawei LiteOS LwIP API Reference*. The implementation of these two APIs is the same.
sys/socket.h	recvfrom	Function	Receive data from a socket.	For details on this API, see the lwip_recvfrom API in *Huawei LiteOS LwIP API Reference*. The implementation of these two APIs is the same.
sched.h	sched_get_priority_max	Function	Get the supported maximum priority value.	
sched.h	sched_get_priority_min	Function	Get the supported minimum priority value.	

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
sched.h	sched_yield	Function	Cause the running thread to relinquish the processor.	
semaphore.h	sem_destroy	Function	Destroy an unnamed semaphore.	
semaphore.h	sem_getvalue	Function	Get the value of a specified semaphore.	
semaphore.h	sem_init	Function	Initialize an unnamed semaphore.	
semaphore.h	sem_post	Function	Release a specified unnamed semaphore.	
semaphore.h	sem_timedwait	Function	Wait for an unnamed semaphore within a timeout interval.	Standard: The timeout time is absolute time and the previous timeout semaphores can be processed. Huawei LiteOS: The time interval between consecutive receipts must be specified. It is not allowed to enter a negative value while specifying the length of time interval.
semaphore.h	sem_trywait	Function	Attempt to wait for an unnamed semaphore.	
semaphore.h	sem_wait	Function	Wait for an unnamed semaphore.	

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
sys/socket.h	send	Function	Send data through a socket.	For details on this API, see the lwip_send API in *Huawei LiteOS LwIP API Reference*. The implementation of these two APIs is the same.
sys/socket.h	sendto	Function	Send data through a socket.	For details on this API, see the lwip_sendto API in *Huawei LiteOS LwIP API Reference*. The implementation of these two APIs is the same.
sys/socket.h	setsockopt	Function	Set the status of a socket.	For details on this API, see the lwip_setscokopt API in *Huawei LiteOS LwIP API Reference*. The implementation of these two APIs is the same.
sys/socket.h	socket	Function	Create socket communication .	For details on this API, see the lwip_socket API in *Huawei LiteOS LwIP API Reference*. The implementation of these two APIs is the same.

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
time.h	timer_create	Function	Create a timer and specify the timer expiry notification mechanism.	
time.h	timer_delete	Function	Delete a timer.	Standard: A one-off software timer will not be automatically deleted after it is run. Huawei LiteOS: A one-off software timer will be automatically deleted after it is run. This API can be used to create a periodic timer.
time.h	timer_getoverrun	Function	Get the number of lost timer notifications.	Standard: This API can be used to get the timer expiration overrun count. Huawei LiteOS: The return value shall be the number of execution periods of a periodic timer.
time.h	timer_gettime	Function	Get the time remaining on a POSIX.1b interval timer.	
time.h	timer_settime	Function	Start or stop a timer.	
sys/utsname.h	uname	Function	Get the name and other information of the current kernel.	In Huawei LiteOS, the header file is **utsname.h**.
stdarg.h	va_arg	Macro	Return variadic arguments.	

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Remark
stdarg.h	va_copy	Macro	Copy the initialized va_list to the target argument list.	
stdarg.h	va_end	Macro	Put the acquisition of variadic arguments to an end.	
stdarg.h	va_start	Macro	Initialize the variable **arg_ptr**.	

9.1.2 POSIX APIs Not Supported

Some POSIX APIs are not supported in Huawei LiteOS. The following table lists the detailed specifications:

File	API	Type	Description	Supported/Not Supported
dirent.h	fdopendir	Function	Convert a file descriptor to a pointer to the directory structure.	Not supported
mqueue.h	mq_notify	Function	Notify the calling process that a message is available in a message queue.	Not supported
mqueue.h	mq_unlink	Function	Remove a message queue.	Not supported
pthread.h	pthread_attr_destroy	Function	Destroy a thread attributes object.	Not supported
pthread_cond.h	pthread_condattr_destroy	Function	Destroy a condition variable attributes object.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Type	Description	Supported/Not Supported
pthread_cond.h	pthread_condattr_init	Function	Initialize a condition variable attributes object.	Not supported
pthread.h	pthread_getspecific	Function	Return the value currently bound to the specified key on behalf of the calling thread.	Not supported
pthread.h	pthread_key_create	Function	Create a thread-specific data key.	Not supported
pthread.h	pthread_key_delete	Function	Delete a thread-specific data key.	Not supported
pthread.h	pthread_mutex_timedlock	Function	Lock a mutex before a specified timeout expires.	Not supported
pthread.h	pthread_setspecific	Function	Associate a thread-specific value with a key.	Not supported
semaphore.h	sem_close	Function	Close a named semaphore.	Not supported
semaphore.h	sem_open	Function	Open a named semaphore.	Not supported
semaphore.h	sem_unlink	Function	Remove a named semaphore.	Not supported
signal.h	kill	Function	Send a signal to a process.	Not supported
signal.h	pthread_kill	Function	Send a signal to a thread.	Not supported
signal.h	pthread_sigmask	Function	Mask a thread's responses to some signals.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Type	Description	Supported/Not Supported
signal.h	raise	Function	Send a signal to the calling process.	Not supported
signal.h	sigaction	Function	Examine or specify the action to be associated with a specific signal.	Not supported
signal.h	sigaddset	Function	Add a signal to a signal set.	Not supported
signal.h	sigdelset	Function	Delete a signal from a signal set.	Not supported
signal.h	sigemptyset	Function	Initialize a signal set.	Not supported
signal.h	sigfillset	Function	Add all signals to a signal set.	Not supported
signal.h	sigismember	Function	Test whether a signal is a member of a signal set.	Not supported
signal.h	signal	Function	Set the disposition of a signal.	Not supported
signal.h	sigpending	Function	Query pending signals.	Not supported
signal.h	sigprocmask	Function	Query or set the signal mask of the calling thread.	Not supported
signal.h	sigqueue	Function	Queue a signal to a process.	Not supported
signal.h	sigsuspend	Function	Suspend a process until it catches a signal.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Type	Description	Supported/Not Supported
signal.h	sigtimedwait	Function	Suspend the execution of the calling thread until a signal in a signal set is delivered before a specified timeout expires.	Not supported
signal.h	sigwait	Function	Suspend the execution of the calling thread until a signal in a signal set is delivered.	Not supported
signal.h	sigwaitinfo	Function	Suspend the execution of the calling thread until a signal in a signal set is delivered.	Not supported
stdio.h	getdelim	Function	Set the position where file reading ends.	Not supported
stdio.h	getline	Function	Read a line from a stream.	Not supported
stdio.h	tempnam	Function	Return a unique filename that contains a directory name.	Not supported
stdio.h	tmpfile	Function	Create a temporary binary file.	Not supported
stdio.h	vsscanf	Function	Format string input.	Not supported
stdlib.h	abort	Function	Abort the current process.	Not supported
stdlib.h	atexit	Function	Register a function to be called at normal process termination.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Type	Description	Supported/Not Supported
stdlib.h	div	Function	Return the quotient and remainder of an integer division.	Not supported
stdlib.h	exit	Function	Cause normal process termination.	Not supported
stdlib.h	getenv	Function	Get the value of an environment variable.	Not supported
stdlib.h	ldiv	Function	Return the quotient and remainder of an integer division.	Not supported
stdlib.h	mblen	Function	Return the size of a multibyte character.	Not supported
stdlib.h	mbstowcs	Function	Convert a multibyte sequence to a wide character.	Not supported
stdlib.h	mbtowc	Function	Convert a multibyte sequence to a wide character.	Not supported
stdlib.h	system	Function	Execute a shell command.	Not supported
stdlib.h	wctomb	Function	Check the coding of a multibyte character.	Not supported
sys/wait.h	waitpid	Function	Wait for a child process to stop or terminate.	Not supported
syslog.h	closelog	Function	Close the descriptor being used to write to the system logger.	Not supported
syslog.h	setlogmask	Function	Set the syslog log priority mask.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Type	Description	Supported/Not Supported
syslog.h	syslog	Function	Send log messages to the system logger syslogd.	Not supported
unistd.h	alarm	Function	Set a signal transmission alarm.	Not supported
unistd.h	_exit	Function	Terminate the calling process.	Not supported
unistd.h	execve	Function	Execute a file.	Not supported
unistd.h	fchown	Function	Change the owner of a file	Not supported
unistd.h	fork	Function	Create a child process.	Not supported
unistd.h	gethostname	Function	Get the hostname.	Not supported
unistd.h	isatty	Function	Test whether a specified file descriptor is a tty.	Not supported
unistd.h	nice	Function	Change the priority of a process.	Not supported
unistd.h	pipe	Function	Create a pipe.	Not supported
unistd.h	readlink	Function	Read the file pointed to by a symbolic link.	Not supported
wchar.h	fgetws	Function	Read a wide-character string from a stream.	Not supported
wchar.h	fputws	Function	Write a wide-character string to a stream.	Not supported
wchar.h	fwide	Function	Set a stream to be byte-/wide character-oriented.	Not supported
wchar.h	fwprintf	Function	Format wide-character output to a file.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Type	Description	Supported/Not Supported
wchar.h	fwscanf	Function	Format wide-character string input.	Not supported
wchar.h	getwchar	Function	Read a wide character from the standard input.	Not supported
wchar.h	mbrlen	Function	Return the size of a multibyte character.	Not supported
wchar.h	mbsrtowcs	Function	Convert a multibyte sequence to a wide character.	Not supported
wchar.h	putwchar	Function	Write a specified wide character to the standard output.	Not supported
wchar.h	swprintf	Function	Copy a formatted wide-character string.	Not supported
wchar.h	swscanf	Function	Format wide-character string input.	Not supported
wchar.h	vfwprintf	Function	Format wide-character output to a file.	Not supported
wchar.h	vfwscanf	Function	Format wide-character string input.	Not supported
wchar.h	vswprintf	Function	Copy a formatted wide-character string.	Not supported
wchar.h	vswscanf	Function	Format wide-character string input.	Not supported
wchar.h	vwprintf	Function	Format wide-character output.	Not supported
wchar.h	vwscanf	Function	Format wide-character input.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Type	Description	Supported/Not Supported
wchar.h	wcscat	Function	Concatenate two wide-character strings.	Not supported
wchar.h	wcschr	Function	Locate the first occurrence of a specified wide character in a wide-character string.	Not supported
wchar.h	wcscpy	Function	Copy a wide-character string.	Not supported
wchar.h	wcscspn	Function	Return the number of continuous wide characters in a wide-character string that do not contain the specified wide-character string.	Not supported
wchar.h	wcsncat	Function	Concatenate two wide-character strings.	Not supported
wchar.h	wcsncmp	Function	Compare two wide-character strings.	Not supported
wchar.h	wcsncpy	Function	Copy characters from a wide-character string.	Not supported
wchar.h	wcspbrk	Function	Return the position the first wide characters that two wide-character strings have in common.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Type	Description	Supported/Not Supported
wchar.h	wcsrchr	Function	Locate the last occurrence of a specified wide character in a wide-character string.	Not supported
wchar.h	wcsrtombs	Function	Convert a wide-character string to a multibyte string.	Not supported
wchar.h	wcsspn	Function	Return the number of continuous wide characters in a wide-character string that do not contain the specified wide-character string.	Not supported
wchar.h	wcsstr	Function	Locate a substring in a wide-character string.	Not supported
wchar.h	wcstod	Function	Convert a wide-character string to a double-precision floating number.	Not supported
wchar.h	wcstof	Function	Convert a wide-character string to a single-precision floating number.	Not supported
wchar.h	wcstok	Function	Split a wide-character string.	Not supported
wchar.h	wcstol	Function	Convert a wide-character string to a long integer.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Type	Description	Supported/Not Supported
wchar.h	wcstoul	Function	Convert a wide-character string to an unsigned long integer.	Not supported
wchar.h	wprintf	Function	Format wide-character output.	Not supported
wchar.h	wscanf	Function	Format wide-character string input.	Not supported
wctype.h	towctrans	Function	Wide character conversion.	Not supported
wctype.h	wctrans	Function	Wide-character translation mapping.	Not supported

9.2 Libc/Libm APIs

9.2.1 Libc Adaption APIs

Huawei LiteOS provides Libcadaption APIs. The following table lists detailed specifications.

Header File	API	Type	Description
stdlib.h	arc4random_uniform	Random number function	Generate a random number in the range of 0 to (x–1).
time.h	asctime_r	Time function	Display time and date in the format of string. You need to check whether the passed-in tm structure is correct. The week value must be in the range of [0,6]. The month value must be in the range of [0,11].
time64.h	asctime64	Time function	Convert tm structure time to corresponding string

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description
time64.h	asctime64_r	Time function	Convert tm structure time to corresponding string
assert.h	assert	Assertion macro	Terminate the program if an expression is false. This API is used for debugging.
stdlib.h	calloc	Memory configuring function	Allocate dynamic memory.
checksum.h	csum_fold	Data check function	Convert a 32-bit cumulative sum to a 16-bit checksum.
time.h	ctime	Time function	Display time and date in the format of string.
time.h	ctime_r	Time function	Display time and date in the format of string.
time64.h	ctime64	Time function	Convert calendar time to corresponding string
time64.h	ctime64_r	Time function	Convert calendar time to corresponding string
stdio.h	fgets	Standard I/O function	Read a string from a file.
stdio.h	fopen64	Standard I/O function	Open a file.
stdlib.h	free	Memory configuring function	Free the previously allocated dynamic memory.
stdio.h	freopen	Standard I/O function	Redirect a stream. The **fd** parameter in the returned value of this API may be different from that of the standard, because part of file systems (RAMFS) do not support the dup2() function called by this API.

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description
stdio.h	fseeko64	Standard I/O function	Move the read and write position of a stream.
stdio.h	ftello64	Standard I/O function	Get the file position indicator for a stream.
stdio.h	getc_unlocked	Standard I/O function	Non-locking stdio getc operation
stdio.h	getchar	Standard I/O function	Read a character from the standard input.
stdio.h	getchar_unlocked	Standard I/O function	Non-locking stdio getchar operation
unistd.h	getcwd	String processing function	Copy the working directory to a string that meets the length requirement. If the string is shorter than required, a null value and error code are returned.
stdio.h	gets	Standard I/O function	Read a character from the standard input. You are advised to use fgets().
time64.h	gmtime64	Time function	Convert calendar time to Greenwich Mean Time(GMT)
time64.h	gmtime64_r	Time function	Convert calendar time to Greenwich Mean Time(GMT)
inet.h	inet_ntop	Data conversion function	Convert the dotted decimal notation to binary integer.
inet.h	inet_pton	Data conversion function	Convert the binary integer to dotted deciaml notation
if.h	if_indextoname	Data conversion function	Convert the network card number to name

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description
if.h	if_nametoindex	Data conversion function	Convert the netword name to number
time.h	localtime	Time function	Get the current local time and date.
time64.h	localtime64	Time function	Convert calendar time to tm structure
time64.h	localtime64_r	Time function	Convert calendar time to tm structure
stdlib.h	malloc	Memory configuring function	Allocate dynamic memory.
stdlib.h	memalign	Memory allocation function	Allocate memory. The passed-in alignment must be a positive integer power of two.
libcmini.h	memchr	Memory processing function	Scan a memory area for a character.
libcmini.h	memcpy	Memory processing function	Copy memory content.
libcmini.h	memmove	Memory processing function	Copy **count** bytes from memory area **src** to memory area **dest**.
libcmini.h	memset	Memory processing function	Fill n bytes of a memory block with a given value.
time64.h	mktime64	Time function	Convert tm structure time to calendar time
stdio.h	perror	Error processing function	Print error information.
stdlib.h	posix_memalign	Memory allocation function	Apply for a byte-aligned memory.
unistd.h	pread	File operation function	Read data from a file with an offset(32-bit)
unistd.h	pread64	File operation function	Read data from a file with an offset(64-bit)

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description
unistd.h	pwrite	File operation function	Write data to file with an offset(32-bit)
unistd.h	pwrite64	File operation function	Write data to file with an offset(64-bit)
unistd.h	read	File operation function	File read operation
stdlib.h	realloc	Memory configuring function	Reallocate memory.
locale.h	setLocaleInit	Locale function	Initialize the set locale.
string.h	strcat	String processing function	Concatenate two strings.
string.h	strcoll	String processing function	Compare two strings, both interpreted as appropriate to the LC_COLLATE category of the current locale.
string.h	strerror	Error processing function	Return a string that describes the error code.
string.h	strncat	String processing function	Concatenate two strings.
string.h	strnchr	String processing function	Locate the first occurrence of a specified length character in a string.
string.h	strnlen	String processing function	String length calculation
string.h	strnstr	String processing function	Locate a substring in a specified length string.
string.h	strxfrm	String processing function	Transform a string.
time64.h	timegm64	Time function	Convert Greenwich Mean Time(GMT) to calendar time

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description
time64.h	timelocal64	Time function	Convert tm structure time to calendar time
ctype.h	tolower	Data conversion function	Convert an uppercase letter to its lowercase equivalent.
ctype.h	toupper	Data conversion function	Convert a lowercase letter to its uppercase equivalent.
time.h	tzset	Time function	Initialize time conversion information.
unistd.h	write	File operation function	File write operation
stdlib.h	zalloc	Memory configuring function	Allocate dynamic memory and initialize this memory to zero.

9.2.2 Libc Open Source APIs

Huawei LiteOS provides a set of Libc open source APIs. The following table lists the detailed specifications.

Header File	API	Type	Description	Source
libcmini.h	__fpclassify d	Floating point number calculation function	Classify floating point values.	bionic 5.0
local.h	__vfprintf	File operation function	Format the output data to a file.	bionic 5.0
floatio.h	__hdtoa	Floating point number calculation function	Convert an IEEE double precision value to a hexadecimal string.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
floatio.h	__hldtoa	Floating point number calculation function	Convert an IEEE floating point value to a hexadecimal string.	bionic 5.0
libcmini.h	__isnan	Floating point number calculation function	Determine whether a floating point number is Not a Number (NaN).	bionic 5.0
floatio.h	__ldtoa	Floating point number calculation function	A wrapper for gdtoa() that makes its function similar with dtoa().	bionic 5.0
stdlib.h	abs	Mathematical calculation function	Return the absolute value of an integer.	nuttx 7.8
stdlib.h	arc4random	Random number function	Generate a random number.	bionic 5.0
time.h	asctime	Time function	Display time and date in the format of string.	bionic 5.0
stdio.h	asprintf	Format string function	Copy the format string.	bionic 7.1
stdlib.h	atoi	Data conversion function	Convert a string to an integer.	bionic 5.0
stdlib.h	atol	Data conversion function	Convert a string to a long integer.	bionic 5.0
stdlib.h	atoll	Data conversion function	Convert a string to a long long integer.	bionic 5.0
stdlib.h	bsearch	Data structure function	Binary search	bionic 5.0
wchar.h	btowc	Wide character processing function	Convert a single byte to a wide character.	bionic 5.0
string.h	bzero	String processing function	Set the first n bytes of a memory area to zero.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
stdio.h	clearerr	Standard I/O function	Delete the error flag of a stream.	bionic 5.0
fcntl.h	creat	File operation function	Create a file.	bionic 5.0
stdio.h	fclose	Standard I/O function	Close a file.	bionic 5.0
stdio.h	fdopen	Standard I/O function	Associate a standard I/O stream with an existing file descriptor.	bionic 5.0
stdio.h	feof	Standard I/O function	Check whether a stream has read the file tail.	bionic 5.0
stdio.h	ferror	Error processing function	Check whether any error occurs in a stream.	bionic 5.0
stdio.h	fflush	Standard I/O function	Update a buffer.	bionic 5.0
stdio.h	fgetc	Standard I/O function	Read a character from a file.	bionic 5.0
stdio.h	fgetpos	Standard I/O function	Get the file position indicator for a stream.	bionic 5.0
wchar.h	fgetwc	Wide character processing function	Convert a single byte to a wide character.	bionic 5.0
stdio.h	fileno	Standard I/O function	Return the file descriptor of the stream specified by **stream**.	bionic 5.0
libcmini.h	finite	Floating point number calculation function	Determine whether a floating point number is finite.	bionic 5.0
stdio.h	flockfile	Standard I/O function	Lock a file.	bionic 5.0
stdio.h	fopen	Standard I/O function	Open a file.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
stdio.h	fprintf	Format input/output function	Format the output data to a file.	bionic 5.0
stdio.h	fputc	Standard I/O function	Write a specified character to a stream.	bionic 5.0
stdio.h	fputs	Standard I/O function	Write a specified string to a stream.	bionic 5.0
wchar.h	fputwc	Wide character processing function	Write a wide character to a stream.	bionic 5.0
stdio.h	fread	Standard I/O function	Read data from a stream.	bionic 5.0
stdio.h	fscanf	Format input/output function	Read formatted input.	bionic 5.0
stdio.h	fseek	Standard I/O function	Move the file position indicator for a stream.	bionic 5.0
stdio.h	fseeko	Standard I/O function	Move the read and write position of a stream.	bionic 5.0
stdio.h	fsetpos	Standard I/O function	Move the file position indicator for a stream.	bionic 5.0
stdio.h	ftell	Standard I/O function	Get the file position indicator for a stream.	bionic 5.0
stdio.h	ftello	Standard I/O function	Get the file position indicator for a stream.	bionic 5.0
stdio.h	ftrylockfile	Standard I/O function	Lock a file for stdio.	bionic 5.0
stdio.h	funlockfile	Standard I/O function	Unlock a file for stdio.	bionic 5.0
stdio.h	fwrite	Standard I/O function	Write data to a stream.	bionic 5.0
stdio.h	getc	Standard I/O function	Read a character from a file.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
wchar.h	getwc	Wide character processing function	Read a wide character from a file.	bionic 5.0
time.h	gmtime	Time function	Get the current time and date.	nuttx 7.8
time.h	gmtime_r	Time function	Get the current time and date.	nuttx 7.8
netdb.h	hstrerror	Error function	Description of the cause of the error.	bionic 7.1
ctype.h	isalnum	Character type check function	Check whether a character is alphanumeric.	bionic 5.0
ctype.h	isalpha	Character type check function	Check whether a character is alphabetic.	bionic 5.0
ctype.h	isascii	Character type check macro	Check whether a character is an ASCII code.	bionic 5.0
ctype.h	isblank	Character type check function	Check whether a character is a blank character; that is, a space or a tab.	bionic 5.0
ctype.h	iscntrl	Character type check function	Check whether a character is a control character.	bionic 5.0
ctype.h	isdigit	Character type check function	Check whether a character is a decimal digit.	bionic 5.0
ctype.h	isgraph	Character type check function	Check whether a character has a graphical representation.	bionic 5.0
ctype.h	islower	Character type check function	Check whether a character is a lowercase letter.	bionic 5.0
libcmini.h	isnan	Floating point number calculation function	Determine whether a floating point number is Not a Number (NaN).	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
ctype.h	isprint	Character type check function	Check whether a character is printable.	bionic 5.0
ctype.h	ispunct	Character type check function	Check whether a character is a punctuation character.	bionic 5.0
ctype.h	isspace	Character type check function	Check whether a character is a white-space character.	bionic 5.0
ctype.h	isupper	Character type check function	Check whether a character is an uppercase letter.	bionic 5.0
wctype.h	iswalnum	Wide character processing function	Check whether a wide character is alphanumeric.	bionic 5.0
wctype.h	iswalpha	Wide character processing function	Check whether a wide character is alphabetic.	bionic 5.0
wctype.h	iswblank	Wide character processing function	Check whether a wide character is a blank character; that is, a space or a tab.	bionic 5.0
wctype.h	iswcntrl	Wide character processing function	Check whether a wide character is a control character.	bionic 5.0
wctype.h	iswctype	Wide character processing function	Check whether a wide character belongs to a specified character class.	bionic 5.0
wctype.h	iswdigit	Wide character processing function	Check whether a wide character is a decimal digit (0 through 9).	bionic 5.0
wctype.h	iswgraph	Wide character processing function	Check whether a wide character is a printable character except a space.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
wctype.h	iswlower	Wide character processing function	Check whether a wide character is a lowercase letter.	bionic 5.0
wctype.h	iswprint	Wide character processing function	Check whether a wide character is printable.	bionic 5.0
wctype.h	iswpunct	Wide character processing function	Check whether a wide character is a punctuation character or a special character.	bionic 5.0
wctype.h	iswspace	Wide character processing function	Check whether a wide character is a white-space character.	bionic 5.0
wctype.h	iswupper	Wide character processing function	Check whether a wide character is an uppercase letter.	bionic 5.0
wctype.h	iswxdigit	Wide character processing function	Check whether a wide character is a hexadecimal digit.	bionic 5.0
ctype.h	isxdigit	Character type check function	Check whether a character is a hexadecimal digit.	bionic 5.0
stdlib.h	labs	Mathematical calculation function	Return the absolute value of a long integer.	nuttx 7.8
stdlib.h	llabs	Mathematical calculation function	Return the absolute value of a long integer.	nuttx 7.8
time.h	localtime_r	Time function	Get the current local time and date.	nuttex 7.8
stdlib.h	lrand48	Random number function	Get a positive long integer random number.	bionic 7.1
wchar.h	mbrtowc	Wide character processing function	Convert a multibyte sequence to a wide character.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
wchar.h	mbsinit	Wide character processing function	Test for initial shift state.	bionic 5.0
bionic_mbstate.h	mbstate_bytes_so_far	Other function	Return bytes whose stream status is nonzero.	bionic 5.0
bionic_mbstate.h	mbstate_get_byte	Other function	Return the stream status.	bionic 5.0
bionic_mbstate.h	mbstate_set_byte	Other function	Set the stream status.	bionic 5.0
string.h	memchr	String processing function	Scan a memory area for a character.	nuttx 7.8
string.h	memcmp	String processing function	Compare memory areas.	bionic 5.0
string.h	memcpy	String processing function	Copy memory content.	bionic 5.0
string.h	memmove	String processing function	Copy memory content.	bionic 5.0
string.h	memset	String processing function	Fill memory with a value.	bionic 5.0
stdlib.h	mkstemp	Memory processing function	Create a unique temporary file.	bionic 5.0
stdlib.h	mkstemps	Memory handler function	Create a temporary file with a limited file name length.	bionic 5.0
time.h	mktime	Time function	Convert the time structure data into the number of elapsed seconds.	nuttx 7.8
reentrant.h	mutex_lock	Mutex operation function	Lock a mutex.	bionic 5.0
reentrant.h	mutex_unlock	Mutex operation function	Unlock a mutex.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
stddef.h	offsetof	Obtaining offset	Return the offset of a structure member from the start of the structure.	nuttx 7.8
stdio.h	printf	Format input/output function	Format output data.	bionic 5.0
stdio.h	putc	Standard I/O function	Write a specified character to a file.	bionic 5.0
stdio.h	putc_unlocked	Standard I/O function	Non-locking stdio putc operation	bionic 5.0
stdio.h	putchar	Standard I/O function	Write a specified character to the standard output. (This interface is compatible with the linux part. If output is needed, it must be full of buffer or print '\n' to force output.)	bionic 5.0
stdio.h	putchar_unlocked	Standard I/O function	Non-locking stdio putchar operation	bionic 5.0
stdio.h	puts	Standard I/O function	Write a specified string to the standard output.	bionic 5.0
wchar.h	putwc	Wide character processing function	Write a specified wide character to a file.	bionic 5.0
stdlib.h	qsort	Data structure function	Sort an array by using a quick sorting method.	bionic 5.0
stdlib.h	rand	Random number function	Generate a random number.	bionic 5.0
stdio.h	remove	File and directory function	Delete a file.	bionic 5.0
bionic_mbstate.h	reset_and_return	Other function	Initialize the stream status.	bionic 5.0
bionic_mbstate.h	reset_and_return_illegal	Other function	Initialize the stream status and return －1.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
stdio.h	rewind	Standard I/O function	Set the file position indicator for a stream to the beginning of the file.	bionic 5.0
stdio.h	scanf	Format input/output function	Format string input.	bionic 5.0
stdio.h	setbuf	Standard I/O function	Set the buffer of a stream.	bionic 5.0
locale.h	setlocale	Locale function	Set or retrieve locale information.	bionic 5.0
stdio.h	setvbuf	Standard I/O function	Set the buffer of a stream.	bionic 5.0
stdio.h	snprintf	Format input/output function	Copy a formatted string.	bionic 5.0
string.h	snprintf	Format input/output function	Copy a formatted string.	bionic 5.0
stdio.h	sprintf	Format input/output function	Copy a formatted string.	bionic 5.0
stdlib.h	srand48	Random number function	Set a 48-bit random seed.	bionic 7.1
stdlib.h	srand	Random number function	Set a random seed.	bionic 5.0
stdlib.h	srandom	Random number function	Generate a random seed.	bionic 5.0
stdio.h	sscanf	Format input/output function	Read formatted input from a string.	bionic 5.0
string.h	strcasecmp	String processing function	Compare two strings.	bionic 5.0
string.h	strcasestr	String processing function	Determine whether string **str2** is the substring of string **str1**, and ignore the case of both strings.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
string.h	strchr	String processing function	Locate the first occurrence of a specified character in a string.	nuttx7.8
string.h	strcmp	String processing function	Compare two strings.	bionic 5.0
string.h	strcpy	String processing function	Copy a string.	bionic 5.0
string.h	strcspn	String processing function	Return the number of continuous characters that do not contain the specified string.	bionic 5.0
string.h	strdup	String processing function	Duplicate a string.	bionic 5.0
string.h	strerror_r	Format input/output function	Return a string that describes the error code.	bionic 5.0
time.h	strftime	Time function	Format date and time.	nuttx7.8
string.h	strlcpy	String processing function	Copy characters from a string.	bionic 5.0
string.h	strlen	String processing function	Return the length of a string.	bionic 5.0
libcmini.h	strlen	String processing function	Calculate the length of a string.	bionic 5.0
string.h	strncasecmp	String processing function	Compare two strings.	bionic 5.0
string.h	strncmp	String processing function	Compare two strings.	bionic 5.0
string.h	strncpy	String processing function	Copy characters from a string.	bionic 5.0
libcmini.h	strncpy	String processing function	Copy characters from a string.	bionic 5.0
string.h	strpbrk	String processing function	Locate the first occurrence of a specified character in a string.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
time.h	strptime	Time function	Convert a string to a time type in a specific time format	bionic 7.1
string.h	strrchr	String processing function	Locate the last occurrence of a specified character in a string.	nuttx7.8
string.h	strsep	String processing function	Break a string into a set of strings.	bionic 5.0
string.h	strspn	String processing function	Return the number of continuous characters that do not contain the specified string.	bionic 5.0
string.h	strstr	String processing function	Locate a substring in a string.	bionic 5.0
stdlib.h	strtod	Data conversion function	Convert a string to a floating point number.	bionic 5.0
stdlib.h	strtof	String processing function	Convert string to single precision floating point number	bionic 7.1
string.h	strtok	String processing function	Split a string.	nuttx7.8
string.h	strtok_r	String processing function	Split a string.	nuttx7.8
stdlib.h	strtol	Data conversion function	Convert a string to a long integer.	bionic 5.0
stdlib.h	strtold	String processing function	Convert string to long double precision floating point number	bionic 7.1
stdlib.h	strtoll	String processing function	Convert string to double long integer	bionic 7.1
stdlib.h	strtoul	Data conversion function	Convert a string to an unsigned long integer.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
stdlib.h	strtoull	String processing function	Convert string to unsigned double long integer	bionic 7.1
time.h	time	Time function	Get the current time.	nuttx7.8
time.h	timer_create	Time function	Create a timer.	nuttx7.8
time.h	timer_delete	Time function	Delete a timer.	nuttx7.8
time.h	timer_gettime	Time function	Return the amount of time until a timer expires.	nuttx7.8
time.h	timer_settime	Time function	Initialize or disarm a timer.	nuttx7.8
time.h	times	Time function	Fill the tms structure pointed to by buffer with time-accounting information.	nuttx7.8
stdio.h	tmpnam	Standard I/O function	Return a pointer to a unique filename.	bionic 5.0
ctype.h	toascii	Character type check macro	Convert a character to its corresponding ASCII code.	bionic 5.0
wctype.h	towlower	Wide character processing function	Convert a wide character to lowercase.	bionic 5.0
wctype.h	towupper	Wide character processing function	Convert a wide character to upper case.	bionic 5.0
stdio.h	ungetc	Standard I/O function	Put a character back to a stream.	bionic 5.0
wchar.h	ungetwc	Wide character processing function	Put a wide character back to a stream.	bionic 5.0
stdarg.h	va_arg	Argument retrieving macro	Retrieve the next argument in an argument list.	bionic 5.0
stdarg.h	va_copy	Argument retrieving macro	Copying macro.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
stdarg.h	va_end	Argument retrieving macro	Retrieve the return of va_start.	bionic 5.0
stdarg.h	va_start	Argument retrieving macro	Initialize a variable.	bionic 5.0
stdio.h	vasprintf	Format string copy function	Copy format string	bionic 7.1
stdio.h	vfprintf	Format input/ output function	Format the output data to a file.	bionic 5.0
stdio.h	vfscanf	Format input/ output function	Read a string from a stream, convert the string format to that specified by **format,** and format the data.	bionic 5.0
stdio.h	vprintf	Format input/ output function	Format output.	bionic 5.0
stdio.h	vscanf	Format input/ output function	Format string input.	bionic 5.0
stdio.h	vsnprintf	Format input/ output function	Copy a formatted string.	bionic 5.0
stdio.h	vsprintf	Format input/ output function	Copy a formatted string.	bionic 5.0
wchar.h	wcrtomb	Wide character processing function	Check the coding of a multibyte character.	bionic 5.0
wchar.h	wcscmp	Wide character processing function	Compare two wide-character strings.	bionic 5.0
wchar.h	wcscoll	Wide character processing function	Compare two wide-character strings, both interpreted as appropriate to the LC_COLLATE category of the current locale.	bionic 5.0
wchar.h	wcsftime	Wide character processing function	Format time.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
wchar.h	wcslcpy	Wide character processing function	Copy characters from a wide-character string.	bionic 5.0
wchar.h	wcslen	Wide character processing function	Return the length of a wide-character string.	bionic 5.0
wctype.h	wcslen	Wide character processing function	Return the length of a wide-character string.	bionic 5.0
stdlib.h	wcstombs	Conversion function	Convert a wide-character string to a multibyte string.	bionic 5.0
wctype.h	wcstombs	Wide character processing function	Convert a wide-character string to a multibyte string.	bionic 5.0
wchar.h	wcsxfrm	Wide character processing function	Convert the first n characters according to the LC_COLLATE category of the current locale.	bionic 5.0
wchar.h	wctob	Wide character processing function	Convert a wide character to a single-byte character.	bionic 5.0
wctype.h	wctype	Wide character processing function	Check whether a wide character belongs to a specified character class.	bionic 5.0
wchar.h	wmemchr	Wide character processing function	Search a wide-character array for a wide character.	bionic 5.0
wctype.h	wmemchr	Wide character processing function	Search a wide-character array for a wide character.	bionic 5.0
wchar.h	wmemcmp	Wide character processing function	Compare two wide-character arrays.	bionic 5.0
wctype.h	wmemcmp	Wide character processing function	Compare two wide-character arrays.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
wchar.h	wmemcpy	Wide character processing function	Copy wide characters from a wide-character array.	bionic 5.0
wctype.h	wmemcpy	Wide character processing function	Copy wide characters from a wide-character array.	bionic 5.0
wchar.h	wmemmove	Wide character processing function	Copy wide characters from a wide-character array.	bionic 5.0
wctype.h	wmemmove	Wide character processing function	Copy wide characters from a wide-character array.	bionic 5.0
wchar.h	wmemset	Wide character processing function	Fill a wide-character array.	bionic 5.0
wctype.h	wmemset	Wide character processing function	Fill a wide-character array.	bionic 5.0

9.2.3 Libm Open Source APIs

Huawei LiteOS provides a set of Libm open source APIs. The following table lists the detailed specifications.

NOTE

Returned error code cannot be configured for Libm APIs.

Header File	API	Type	Description	Source
float.h	__ieee754_exp	Floating point number calculation function	Exponential calculation (double precision)	bionic 5.0
float.h	__ieee754_expf	Floating point number calculation function	Exponential calculation (floating type)	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
float.h	__ieee754_log	Floating point number calculation function	Logarithm calculation (double precision)	bionic 5.0
float.h	__ieee754_logf	Floating point number calculation function	Logarithm calculation (floating type)	bionic 5.0
float.h	__ieee754_rem_pio2	Floating point number calculation function	Return the remainder of x rem pi/2 in y[0]+y[1]	bionic 5.0
float.h	__ieee754_rem_pio2f	Floating point number calculation function	Return the remainder of x rem pi/2 in y[0]+y[1]	bionic 5.0
float.h	__ieee754_sqrt	Floating point number calculation function	Square root calculation (double precision)	bionic 5.0
float.h	__ieee754_sqrtf	Floating point number calculation function	Square root calculation (floating type)	bionic 5.0
float.h	__kernel_cos	Floating point number calculation function	Cosine calculation	bionic 5.0
float.h	__kernel_rem_pio2	Floating point number calculation function	__kernel_rem_pio2 return the last three digits of N with y = x - N*pi/2	bionic 5.0
float.h	__kernel_sin	Floating point number calculation function	Sine calculation	bionic 5.0
float.h	__kernel_tan	Floating point number calculation function	Tangent calculation	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
float.h	__kernel_tandf	Floating point number calculation function	Tangent calculation (floating type)	bionic 5.0
math.h	acos	Mathematical calculation function	Arc cosine function	bionic 5.0
math.h	acosf	Mathematical calculation function	Arc cosine function	bionic 5.0
math.h	acosh	Mathematical calculation function	Inverse hyperbolic cosine function	bionic 5.0
math.h	acoshf	Mathematical calculation function	Inverse hyperbolic cosine function	bionic 5.0
math.h	acoshl	Mathematical calculation function	Inverse hyperbolic cosine function	nuttx7.8
math.h	acosl	Mathematical calculation function	Arc cosine function	nuttx7.8
math.h	asin	Mathematical calculation function	Arc sine function	bionic 5.0
math.h	asinf	Mathematical calculation function	Arc sine function	bionic 5.0
math.h	asinh	Mathematical calculation function	Inverse hyperbolic sine function	bionic 5.0
math.h	asinhf	Mathematical calculation function	Inverse hyperbolic sine function	bionic 5.0
math.h	asinhl	Mathematical calculation function	Inverse hyperbolic sine function	bionic 5.0
math.h	asinl	Mathematical calculation function	Arc sine function	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
math.h	atan	Mathematical calculation function	Arc tangent function	bionic 5.0
math.h	atan2	Mathematical calculation function	Arc tangent function	bionic 5.0
math.h	atan2f	Mathematical calculation function	Arc tangent function (The value of the arc tangent is returned in radians.)	bionic 5.0
math.h	atan2l	Mathematical calculation function	Arc tangent function (The value of the arc tangent is returned in radians.)	bionic 5.0
math.h	atanf	Mathematical calculation function	Arc tangent function	bionic 5.0
math.h	atanh	Mathematical calculation function	Inverse hyperbolic tangent function	bionic 5.0
math.h	atanhf	Mathematical calculation function	Inverse hyperbolic tangent function	bionic 5.0
math.h	atanhl	Mathematical calculation function	Inverse hyperbolic tangent function	bionic 5.0
math.h	atanl	Mathematical calculation function	Arc tangent function	bionic 5.0
math.h	cbrt	Mathematical calculation function	Cube root function	bionic 5.0
math.h	cbrtf	Mathematical calculation function	Cube root function	bionic 5.0
math.h	cbrtl	Mathematical calculation function	Cube root function	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
math.h	ceil	Mathematical calculation function	Return the smallest integral value that is not less than **x**.	bionic 5.0
math.h	ceilf	Mathematical calculation function	Return the smallest integral value that is not less than **x**.	bionic 5.0
math.h	ceill	Mathematical calculation function	Return the smallest integral value that is not less than **x**.	bionic 5.0
math.h	copysign	Mathematical calculation function	Return a value whose absolute value matches that of **x**, but whose sign bit matches that of **y**.	bionic 5.0
math.h	copysignf	Mathematical calculation function	Return x after the symbl of y	bionic 7.1
math.h	copysignl	Mathematical calculation function	Return a value whose absolute value matches that of **x**, but whose sign bit matches that of **y**.	bionic 5.0
math.h	cos	Mathematical calculation function	Cosine function	bionic 5.0
math.h	cosf	Mathematical calculation function	Cosine function	bionic 5.0
math.h	cosh	Mathematical calculation function	Hyperbolic cosine function	nuttx7.8
math.h	coshf	Mathematical calculation function	Hyperbolic cosine function	nuttx7.8

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
math.h	coshl	Mathematical calculation function	Hyperbolic cosine function	nuttx7.8
math.h	cosl	Mathematical calculation function	Cosine function	nuttx7.8
math.h	erf	Mathematical calculation function	Error function	bionic 5.0
math.h	erfc	Mathematical calculation function	Complementary error function	bionic 5.0
math.h	erfcf	Mathematical calculation function	Complementary error function	bionic 5.0
math.h	erfcl	Mathematical calculation function	Complementary error function	bionic 5.0
math.h	erff	Mathematical calculation function	Error function	bionic 5.0
math.h	erfl	Mathematical calculation function	Error function	bionic 5.0
math.h	exp	Mathematical calculation function	Return the value of e raised to the power of **x**.	bionic 5.0
math.h	exp2	Mathematical calculation function	Return the value of 2 raised to the power of **x**.	bionic 5.0
math.h	expf	Mathematical calculation function	Return the value of e raised to the power of **x**.	bionic 5.0
math.h	expl	Mathematical calculation function	Return the value of e raised to the power of **x**.	nuttx7.8

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
math.h	expm1	Mathematical calculation function	exp(x) - 1.0	bionic 5.0
math.h	expm1f	Mathematical calculation function	exp(x) － 1	bionic 5.0
math.h	fabs	Mathematical calculation function	Return the absolute value of a floating point number.	bionic 5.0
math.h	fabsf	Mathematical calculation function	Return the absolute value of a floating point number.	nuttx7.8
math.h	fabsl	Mathematical calculation function	Return the absolute value of a floating point number.	nuttx7.8
math.h	finite	Mathematical calculation function	Return a nonzero value.	bionic 5.0
math.h	finitef	Mathematical calculation function	Get a non-zero value	bionic 7.1
math.h	floor	Mathematical calculation function	Return the largest integral value that is not greater than **x**.	bionic 5.0
math.h	floorf	Mathematical calculation function	Return the largest integral value that is not greater than **x**.	bionic 5.0
math.h	floorl	Mathematical calculation function	Return the largest integral value that is not greater than **x**.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
float.h	FLT_DIG DBL_DIG LDBL_DIG	Floating point constant	Number of decimal digits that can be rounded into a floating-point and back without change in the number of decimal digits.	bionic 5.0
float.h	FLT_EPSILON DBL_EPSILON LDBL_EPSILON	Floating point constant	Difference between 1 and the least value greater than 1 that is representable in the given floating-point type.	bionic 5.0
float.h	FLT_MANT_DIG DBL_MANT_DIG LDBL_MANT_DIG	Floating point constant	Number of base-FLT_RADIX digits in the floating-point significand.	bionic 5.0
float.h	FLT_MAX 1E DBL_MAX 1E LDBL_MAX 1E	Floating point constant	Maximum representable finite floating-point number.	bionic 5.0
float.h	FLT_MAX_10_EXP DBL_MAX_10_EXP LDBL_MAX_10_EXP	Floating point constant	Maximum integer value for the exponent of a floating point value expressed in base 10.	bionic 5.0
float.h	FLT_MAX_EXP DBL_MAX_EXP LDBL_MAX_EXP	Floating point constant	Maximum integer value for the exponent of a floating point value expressed in base FLT_RADIX.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
float.h	FLT_MIN DBL_MIN LDBL_MIN	Floating point constant	Minimum representable floating-point number.	bionic 5.0
float.h	FLT_MIN_10_EXP DBL_MIN_10_EXP LDBL_MIN_10_EXP	Floating point constant	Minimum negative integer value for the exponent of a floating point value expressed in base 10.	bionic 5.0
float.h	FLT_MIN_EXP DBL_MIN_EXP LDBL_MIN_EXP	Floating point constant	Minimum negative integer value for the exponent of a floating point value expressed in base FLT_RADIX.	bionic 5.0
float.h	FLT_RADIX	Floating point constant	Base used for representing the exponent.	bionic 5.0
math.h	fmod	Mathematical calculation function	Floating-point remainder function	bionic 5.0
math.h	fmodf	Mathematical calculation function	Floating-point remainder function	bionic 5.0
math.h	fmodl	Mathematical calculation function	Floating-point remainder function	bionic 5.0
math.h	fpclassify	Mathematical calculation macro	Return the type of a parameter (floating-point expression).	bionic 5.0
math.h	frexp	Mathematical calculation function	Split a floating point number into a normalized fraction and an exponent.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
math.h	frexpf	Mathematical calculation function	Split a floating point number into a normalized fraction and an exponent.	bionic 5.0
math.h	frexpl	Mathematical calculation function	Split a floating point number into a normalized fraction and an exponent.	bionic 5.0
math.h	HUGE_VAL	Variable	The result is too large in magnitude to be representable.	bionic 5.0
math.h	hypot	Mathematical calculation function	Return the length of the hypotenuse of a right-angled triangle.	bionic 5.0
math.h	isfinite	Mathematical calculation macro	Return a nonzero value if (fpclassify(x)!=FP_NAN&&fpclassify(x)!=FP_INFINITE).	bionic 5.0
math.h	isinf	Mathematical calculation macro	Determine whether a parameter value is an infinite.	bionic 5.0
math.h	isnan	Mathematical calculation function	Determine whether a parameter is Not a Number (NaN).	bionic 6.0
math.h	isnormal	Mathematical calculation macro	Return a nonzero value if (fpclassify(x)==FP_NORMAL).	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
math.h	ldexpf	Mathematical calculation function	Return the result of multiplying **x** by 2 raised to the power **exp**.	nuttx7.8
math.h	ldexpl	Mathematical calculation function	Return the result of multiplying **x** by 2 raised to the power **exp**.	nuttx7.8
math.h	llrint	Mathematical calculation function	Return the rounded integer value.	bionic 5.0
math.h	log	Mathematical calculation function	Return the base e logarithm of **x**.	bionic 5.0
math.h	log10	Mathematical calculation function	Return the base 10 logarithm of **x**.	bionic 5.0
math.h	log10f	Mathematical calculation function	Return the base 10 logarithm of **x**.	bionic 5.0
math.h	log10l	Mathematical calculation function	Return the base 10 logarithm of **x**.	bionic 5.0
float.h	log1p	Floating point number calculation function	Natural logarithm calculation (double precision)	bionic 5.0
math.h	log1p	Mathematical calculation function	log(1+x)	bionic 5.0
float.h	log1pf	Floating point number calculation function	Natural logarithm calculation (floating type)	bionic 5.0
math.h	log1pf	Mathematical calculation function	log(1+x)	bionic 5.0
math.h	log2	Mathematical calculation function	Return the base 2 logarithm of **x**.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
math.h	log2f	Mathematical calculation function	Return the base 2 logarithm of **x**.	bionic 5.0
math.h	log2l	Mathematical calculation function	Return the base 2 logarithm of **x**.	bionic 5.0
math.h	logf	Mathematical calculation function	Return the base e logarithm of **x**.	bionic 5.0
math.h	logl	Mathematical calculation function	Return the base e logarithm of **x**.	bionic 5.0
math.h	lround	Mathematical calculation function	The double type number rounds off the decimal part, and returns the integer part of the long integer.	bionic 7.1
math.h	lroundf	Mathematical calculation function	The floating-point number rounds off the decimal part, and returns the integer part of the long integer.	bionic 7.1
math.h	modf	Mathematical calculation function	Breaks a floating point number into an integral part and a fractional part, and return the fractional part.	bionic 5.0
math.h	modff	Mathematical calculation function	Breaks a floating point number into an integral part and a fractional part.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
math.h	modfl	Mathematical calculation function	Breaks a floating point number into an integral part and a fractional part.	bionic 5.0
math.h	nextafter	Mathematical calculation function	Takes two floating-point numbers that are adjacent to the first number between double-precision floating-point numbers. If the two numbers are equal, the second number is returned.	bionic 7.1
math.h	nextafterf	Mathematical calculation function	Takes a floating-point number that is adjacent to the first number between two floating-point numbers. If the two numbers are equal, the second number is returned.	bionic 7.1
math.h	pow	Mathematical calculation function	Return the value of **x** raised to the power of **y**.	bionic 5.0
math.h	powf	Mathematical calculation function	Return the value of **x** raised to the power of **y**.	bionic 5.0
math.h	powl	Mathematical calculation function	Return the value of **x** raised to the power of **y**.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
math.h	rint	Mathematical calculation function	Round a floating point number to the nearest integer.	bionic 5.0
math.h	rintf	Mathematical calculation function	Round a floating point number to the nearest integer.	bionic 5.0
math.h	rintl	Mathematical calculation function	Round a floating point number to the nearest integer.	nuttx7.8
math.h	round	Mathematical calculation function	Round x to the nearest integer.	nuttx7.8
math.h	roundf	Mathematical calculation function	Round **x** to the nearest integer.	nuttx7.8
math.h	roundl	Mathematical calculation function	Round **x** to the nearest integer.	nuttx7.8
math.h	scalbn	Mathematical calculation function	Return the result of multiplying **x** by **FLT_RADIX** raised to the power n.	bionic 5.0
float.h	scalbnf	Floating point number calculation function	Return the result of multiplying **x** by **FLT_RADIX** raised to the power n.	bionic 5.0
math.h	scalbnf	Mathematical calculation function	Return the result of multiplying **x** by **FLT_RADIX** raised to the power n.	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
math.h	sin	Mathematical calculation function	Sine function	bionic 5.0
math.h	sincos	Mathematical calculation function	Sine cosine	bionic 6.0
math.h	sincosf	Mathematical calculation function	Sine cosine	bionic 6.0
math.h	sincosl	Mathematical calculation function	Sine cosine	bionic 6.0
math.h	sinhf	Mathematical calculation function	Hyperbolic sine function	nuttx7.8
math.h	sinhl	Mathematical calculation function	Hyperbolic sine function	nuttx7.8
math.h	sinl	Mathematical calculation function	Sine function	nuttx7.8
math.h	sqrt	Mathematical calculation function	Square root function	bionic 5.0
math.h	sqrtf	Mathematical calculation function	Square root function	bionic 5.0
math.h	sqrtl	Mathematical calculation function	Square root function	bionic 5.0
math.h	tan	Mathematical calculation function	Tangent function	bionic 5.0
math.h	tanf	Mathematical calculation function	Tangent function	bionic 5.0
math.h	tanh	Mathematical calculation function	Hyperbolic tangent function	bionic 5.0

Copyright © Huawei Technologies Co., Ltd.

Header File	API	Type	Description	Source
math.h	tanhf	Mathematical calculation function	Hyperbolic tangent function	bionic 5.0
math.h	tanhl	Mathematical calculation function	Hyperbolic tangent function	bionic 5.0
math.h	tanl	Mathematical calculation function	Tangent function	bionic 5.0
math.h	trunc	Mathematical calculation function	Truncate a data or number, and return the truncated value.	bionic 5.0
math.h	truncf	Mathematical calculation function	Truncate a data or number, and return the truncated value.	bionic 5.0
math.h	truncl	Mathematical calculation function	Truncate a data or number, and return the truncated value.	bionic 5.0

9.2.4 Libc/Libm APIs Not Supported

Some Libc/Libm APIs are not supported in Huawei LiteOS. The following table lists the detailed specifications:

File	API	Type	Description	Supported/Not Supported
locale.h	localeconv	Locale function	Set or retrieve locale information.	Not supported
statfs.h	fstatfs	File operation function	Return information about a mounted file system.	Not supported
statfs.h	statfs64	File operation function	Return information about a file system.	Not supported

Copyright © Huawei Technologies Co., Ltd.

File	API	Type	Description	Supported/Not Supported
time.h	posix2time	Time function	Convert posix time_t to local time_t.	Not supported
time.h	time2posix	Time function	Convert local time_t to posix time_t.	Not supported
time.h	timegm	Time function	Convert the struct tm structure to the time_t structure.	Not supported
time.h	timelocal	Time function	Get the current time and date, and convert them to the local time.	Not supported
stdio.h	tmpfile	Standard I/O function	Create a temporary binary file.	Not supported
unistd.h	isatty	File operation function	Test whether a specified file descriptor is a tty.	Not supported

9.3 C++ Compatibility Specifications

The following tables list the compatibility specifications of the C++ standard library and standard template library (STL).

NOTE

The C++ standard library supports the exception processing and the run-time type information (RTTI) features. Other features are supported by the compiler. The following tables describe the features supported by the STL. Other features are currently not supported.

- Language support

Header File	Description
<limits>	Provides definitions related to basic data types. For example, defines the maximum and minimum values and the number of binary digits for each numeric data type in this file.
<new>	Supports dynamic memory allocation.

Copyright © Huawei Technologies Co., Ltd.

- Tool functions

Header File	Description
<utility>	Defines the overloaded rational operator, which simplifies the write of the rational operator; defines the pair type, which is a template type and can be used to store value pairs.
<functional>	Defines the types of function objects and supports the utilities of function objects. Function objects are any objects that support the function call operator.
<memory>	Defines the standard memory allocator for container functions, memory management functions, and the auto_ptr template class.

- String processing

Header File	Description
<string>	Provides supports and definitions for character string types, including single-character strings (consisting of char types) and multi-character strings (consisting of wchar_t types).

- Templates for container classes

Header File	Description
<vector>	Defines the vector sequence template, which is resizable array type and is safer and more flexible than plain arrays.
<list>	Defines the list sequence template, which is a linked list for sequences that often have elements inserted or deleted from arbitrary positions.
<deque>	Defines the deque sequence template, which supports efficient insertion and deletion at each beginning and end.
<queue>	Defines sequence adapters "queue" and "priority_queue" for queue (first in, first out) data structures.
<stack>	Define sequence adapter "stack" for stack (last in, first out) data structures.

Copyright © Huawei Technologies Co., Ltd.

Header File	Description
<map>	An associative container type that allows values to be searched by a key value. The key values are unique and are stored in ascending order.
<set>	An associative container type that stores unique values in ascending order.
<bitset>	Defines the bitset template for fixed-length bit sequences. A bitset template can be considered as a fixed-length packed bool array.

- Iterators

Header File	Description
<iterator>	Provides definitions and support for iterators.

- Algorithms

Header File	Description
<algorithm>	Provides a set of algorithm-based functions, including substitution, sequencing, merge, and search functions.

- Numerical operations

Header File	Description
<complex>	Support complex numerical definitions and operations.
<valarray>	Supports numerical vector operations.
<numeric>	Defines a group of common mathematics operations, such as "accumulate" and "inner_product" for a numerical sequence.

NOTE

The memory and uninitialized_fill functions provided by Huawei LiteOS possibly cause memory leaks. Therefore, exercise caution when using them.

Copyright © Huawei Technologies Co., Ltd.

10 Configuration Reference

About This Chapter

10.1 Configuration Tool Instructions

10.2 Time Management Configuration Parameters

10.3 Memory Management Configuration Parameters

10.4 Memory Maintenance & Testing Configuration Parameters

10.5 Task Configuration Parameters

10.6 Software Timer Configuration Parameters

10.7 Semaphore Configuration Parameters

10.8 Mutex Configuration Parameters

10.9 Hardware Interrupt Configuration Parameters

10.10 Queue Configuration Parameters

10.11 Module Compaction Configuration Parameters

10.1 Configuration Tool Instructions

Tool Introduction

Menuconfig provides configurations based on menus. Kconfig that is used by Menuconfig is a menu configuration language. Config.in and Kconfig are compiled by using this language.

Use Steps

Execute **make menuconfig** under the **Huawei_LiteOS** directory.

Instruction

Methods of using menuconfig are as follows:

Up and down arrow keys: to select different rows (options)

Copyright © Huawei Technologies Co., Ltd.

Space bar: to select an option and exclude an option

1. After an option (which in the row) is selected: an asterisk appears in the square brackets next to the option.
2. After the option is excluded: the asterisk disappears from the square brackets next to the option.

Left and right arrow keys: to switch between Select/Exit/Help

Enter: to perform the Select/Exit/Help operations.

1. Select: to enter the submenu of an option followed by three hyphens and a greater than bracket (--->).
2. Exit: to exit the current configuration

When you change some configurations without saving the changes, you are asked whether to save the changed configurations and then exit.

3. Help: to view help information of an option

Figure 1 shows the menuconfig page.

Figure 10-1

```
                    Huawei LiteOS Configuration
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> selectes a feature, while
<N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for Help.
Legend: [*] feature is selected  [ ] feature is excluded

        Compiler  --->
        Platform  --->
        Kernel  --->
        Lib  --->
        Compat  --->
        FileSystem  --->
        Net  --->
        Debug  --->
        Driver  --->
        Stack Smashing Protector (SSP) Compiler Feature  --->
    ---
    Load an Alternate Configuration File
    Save Configuration to an Alternate File

                <Select>    < Exit >    < Help >
```

Precautions

1. Ensure that cross compilation toolchains, arm-liteos-eabi-, aarch64-liteos-ebai-, arm-none-eabi-,arm-liteos-linux-uclibcgnueabi- series, arm-hisiv500-linux-uclibcgnueabi- series, arm-hisiv300-linux-uclibcgnueabi- series, or arm-hisiv600-linux-gnueabi- series, are installed before using menuconfig.

Copyright © Huawei Technologies Co., Ltd.

2. If you copy a piece of Huawei LiteOS source code and run make menuconfig, and the menu fails to be displayed, delete all binary files in the tools/menuconfig/extra/config directory and run make menuconfig in the top directory.

Configuration Instructions

1. Run make menuconfig to enter the Huawei LiteOS Configuration page that currently contains the **Compiler**, **Platform**, **Kernel**, **Lib**, **Compat**, **FileSystem**, **Net**, **Debug**, **Driver**, and **SSP** options.

Figure 10-2

2. Select the **Compiler** option that indicates the types of cross compilation toolchains. Enter the submenu of the **Compiler** option and configure the LiteOS_Compiler_Type. Seven types of cross compilers are available. arm-liteos-eabi is selected by default.

Copyright © Huawei Technologies Co., Ltd.

Figure 10-3

3. Enter the submenu of the Platform option. Select IPCAMERA Chip, and then select a chip. Platforms hi3516a, hi3518ev200, hi3519, hi3519v101, hi3559, hi3911, him5v100, hi3516cv300, hi3731 and HiFone are available. hi3516a is selected by default.

– **NOTE**

CPU options are available for hi3519v101, hi3559, and HiFone.

Copyright © Huawei Technologies Co., Ltd.

Figure 10-4

4. Select the **Kernel** option and enter its submenu. In the submenu, Lite Kernel is the basic kernel and must be selected. mem tlsf is used to select the dynamic storage allocation (DSA) algorithms to be compiled. The selection of mem tlsf is optional. The extended kernel includes features of C++ support, CPU usage, dynamic loading, run-stop (wifi wakeup), scatter loading, and tickless feature, which can be enabled based on your needs.

Copyright © Huawei Technologies Co., Ltd.

Figure 10-5

```
                                Kernel
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> selectes a feature, while
  <N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for Help.
  Legend: [*] feature is selected  [ ] feature is excluded

                  [ ] Enable Lite Kernel
                  [ ]   Enable mem tlsf
                  [*]   Enable Extend Kernel
                  [*]     C++ Support
                  [ ]     Enable MPU Support
                  [*]     Enable Cpup
                  [ ]     Enable Dynamic Load Feature
                  [ ]     Enable Run stop Feature
                  [ ]     Enable Scatter Feature
                  [ ]     Enable Tickless Feature

                    <Select>    < Exit >    < Help >
```

5. Select the **Lib** option and enter its submenu. In the submenu, Libc, Libm, and Zlib are available. Generally, Lib must be enabled.

Figure 10-6

```
                                Lib
navigate the menu.  <Enter> selects submenus --
 letters are hotkeys.  Pressing <Y> selectes a
clude a feature.  Press <Esc><Esc> to exit, <?>
 feature is selected  [ ] feature is excluded

    Enable Libc (Enable Full Libc)  --->
  [*] Enable Libm
  [*]   Enable Zlib
```

Enable Full Libc when compiling all code, and enable Mini Libc when compiling the Kernel.

Figure 10-7

```
    Enable Libc
 navigate this window c
select followed by the
formation about this op

(X) Enable Full Libc
( ) Enable Mini Libc
```


Copyright © Huawei Technologies Co., Ltd.

6. Select the **Compat** option and enter its submenu. In the submenu, Cmsis, Posix, and Linux are available. Posix must be enabled.

Figure 10-8

```
                Compat
lighted letters are hotkeys.  Pr
ected  [ ] feature is excluded


        [*] Enable Cmsis
        [*] Enable Posix
        [*] Enable Linux
```

7. Select the **FileSystem** option and enter its submenu. In the submenu, FAT, RAMFS, NFS, PROC, YAFFS2, and JFFS2 are available. Under the FAT option, FAT cache and Chinese supported can be enabled. VFS must be enabled. RAMFS and NFS are usable only usable after a Debug version is enabled.

Figure 10-9

```
                FileSystem
.  Highlighted letters are hotk
*] feature is selected  [ ] fea


    [*] Enable VFS
    [*]   Enable FAT
    [*]     Enable FAT Cache
    [*]     Enable Chinese
    [*]   Enable RAMFS
    [*]   Enable YAFFS2
    [*]   Enable NFS
    [*]   Enable PROC
    [*]   Enable JFFS2
```

8. Select the **Net** option and enter its submenu. In the submenu, network-related LWIP and WiFi security–related WPA are available. WPA is useable only after a Debug version is enabled.

Figure 10-10

```
                Net
ighted letters are hotkeys.  Pres
cted  [ ] feature is excluded


      [*] Enable Lwipsack
      [*] Enable Wpa
```

9. Select the **Debug** option and enter its submenu. In the submenu, you can configure whether to use the functions shown in **Figure 12** and .**Figure 13**

Copyright © Huawei Technologies Co., Ltd.

```
                                Debug
eys navigate the menu.  <Enter> selects submenus --->.
hted letters are hotkeys.  Pressing <Y> selectes a featur
l exclude a feature.  Press <Esc><Esc> to exit, <?> for H
 [*] feature is selected  [ ] feature is excluded
---------------------------------------------------------
 [ ] Enable GCC -g Option
 [*] Enable Os_adapt
 [ ] Enable Coredump
 [ ] Enable GDB
 [ ] Enable 3rdParty
 [ ] Enable Thumb
 [ ] Enable Dvfs
 [ ] Enable a Debug Version
     Enable Uart (General Uart)  --->
     Enable TestSuit or AppInit (Enable Appinit)  --->
 [ ] Enable LLT Report
```

Figure 10-11

```
[*] Enable a Debug Version
[*]   Enable Shell
[*]     Enable Shell lk
[*]     Enable Tftp
[*]     Enable Telnet
[*]     Enable Iperf-2.0.5
[*]   Enable Memory module statistics
[*]   Enable Memory multi-pool control
[ ]   Enable Node Head Backup
[ ]   Enable integrity check or not
[*]   Enable size check or not
```

10. Select the **Driver** option and enter its submenu. In the submenu, many types of hardware drivers are available. Two nand flash chips can be selected. For WiFi chips, QRD or BCM can be selected based on your needs. Other available drivers that are closely related to chips are automatically selected by menuconfig. WiFi drivers of Qualcomm depend on WPA.

    WiFi drivers of Qualcomm are usable after a debug version is enabled. WiFi drivers of Broadcom are usable in a release version.

Huawei Proprietary and Confidential
Copyright © Huawei Technologies Co., Ltd.

Figure 10-12

```
                                    Driver
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> selectes a feature, while
  <N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for Help.
  Legend: [*] feature is selected  [ ] feature is excluded

  [*] Enable Driver
  [*]   Enable Driver Base
  [ ]   Enable CpuFreq
  [*]   Enable DevFreq
  [*]   Enable Regulator
  [*]   Enable GPIO
  [*]   Enable HIDMAC
  [*]   Enable Net Device
          MAC (Enable higmac)  --->
  [*]   Enable I2C
  [*]   Enable MEM
  [*]   Enable MMC
  [ ]     Enable MMC support speedup
  [*]   Enable MTD
  [*]     Enable MTD nandflash
            NandFlash Chips (Enable hifmc100 for spi nand)  --->
  [*]     Enable MTD spi_nor flash
            SpiNorFlash Chips (Enable hifmc100)  --->
  [*]   Enable random
  [*]   Enable RTC
  [*]   Enable SPI
  [*]   Enable USB
  [*]     Enable USB HCD
            USB HCD (Enable EHCI HCD (USB 2.0))  --->
  [*]       Enable USB Device Class Drivers
              USB Device Class Drivers  --->
  [*]     Enable USB DWC Controller
```

Figure 10-13

```
  [*]     Enable USB DWC Controller
            DWC Controller (Enable USB2.0 Device Controller)  --->
  [*]       Enable USB Gadget Support
              USB Gadget Drivers (Enable USB Mass Storage Gadget)  --->
  [*] Enable Video
  [*] Enable WIFI
        WIFI Chips (Enable Broadcom WIFI)  --->
  [ ] Enable Watch Dog

                    <Select>    < Exit >    < Help >
```

11. Select the **Stack Smashing Protector (SSP) Compiler Feature** option and enter its submenu. This option is configured to enable or disable the stack protection function.
    - **-fno-stack-protector**: to disable stack protection. **-fno-stack-protector** is selected by default.
    - **-fstack-protector**: to enable stack protection. Protection code can be only inserted in a C function in which the local variables contain character arrays. Each character array must be greater than or equal to 4 bytes.
    - **-fstack-protector-all**: to insert protection code in all C functions. Compared with selecting **-fstack-protector**, selecting **-fstack-protector-all** will greatly increase performance costs.

    You are advised to select **-fstack-protector** to ensure performance and improve security.

Copyright © Huawei Technologies Co., Ltd.

10.2 Time Management Configuration Parameters

Configuration Item Description

The time management module works only after the OS_SYS_CLOCK of the sys module is enabled and the LOSCFG_BASE_CORE_TICK_PER_SECOND of the Tick module is specified. The default values of the following configuration items depend on the configurations of hi3516a chips.

In Huawei LiteOS, the **msecs_to_jiffies** function that is adapted from Linux can be successfully executed when the value of **LOSCFG_BASE_CORE_TICK_PER_SECOND** is less than or equal to the number of milliseconds (1000 ms) in a second and this value can divide the number of milliseconds in a second exactly. If the value of **LOSCFG_BASE_CORE_TICK_PER_SECOND** cannot meet the aforementioned conditions, an error is reported and ENOSUPP is returned after the execution of the **msecs_to_jiffies** function.

Configuration Item	Description	Value Range	Default Value	Dependency	Whether Can Be Dynamically Updated
OS_SYS_CLOCK	System clock speed	(0, n)	50000000	None	No

Copyright © Huawei Technologies Co., Ltd.

Configuration Item	Description	Value Range	Default Value	Dependency	Whether Can Be Dynamically Updated
LOSCFG_BASE_CORE_TICK_PER_SECOND	Number of ticks per second	(0, 1000]	100	None	No

10.3 Memory Management Configuration Parameters

Configuration Item Description

Configuration Item	Description	Value Range	Default Value	Dependency	Whether Can Be Dynamically Updated
OS_SYS_MEM_ADDR	Start address of the dynamic memory	[0, n)	&m_aucSysMem0[0]	None	No
OS_SYS_MEM_SIZE	Size of the dynamic memory (the default DDR allocated to the dynamic memory is automatically configured)	[0, n)	From the end of the bss segment to the end of DDR	None	Yes

Copyright © Huawei Technologies Co., Ltd.

10.4 Memory Maintenance & Testing Configuration Parameters

Configuration Item Description

Configuration Item	Description	Value Range	Default Value	Dependency	Whether Can Be Dynamically Updated
LOSCFG_BASE_MEM_NODE_INTEGRITY_CHECK	Whether to enable the memory node integrity detection.	YES/NO	NO	None	No
LOSCFG_BASE_MEM_NODE_SIZE_CHECK	Whether to open system memory node size detection.	YES/NO	YES	None	No
LOSCFG_MEM_MUL_MODULE	Whether to support the calculation of the memory allocated to each module.	YES/NO	YES	None	No
LOSCFG_MEM_MUL_POOL	Whether to support the management of multiple memory pools.	YES/NO	YES	None	No
LOSCFG_MEM_HEAD_BACKUP	Whether to support backing up control headers of memory nodes.	YES/NO	YES	None	No

You can configure these two items using menuconfig without modifying header files.

Copyright © Huawei Technologies Co., Ltd.

10.5 Task Configuration Parameters

Configuration Item Description

Configuration Item	Description	Value Range	Default Value	Dependency
LOSCFG_BASE_CORE_TSK_CONFIG	Maximum number of tasks	[0, n)	64	None
LOSCFG_BASE_CORE_TSK_IDLE_STACK_SIZE	Idle task stack size	[0, n)	0x800	None
LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE	Default task stack size	[0, n)	0x6000	None
LOSCFG_BASE_CORE_TIMESLICE	Whether to enable time slice	YES/NO	YES	None
LOSCFG_BASE_CORE_TIMESLICE_TIMEOUT	Maximum number of ticks for which tasks with the same priority can be executed	[1, n)	2	None
LOSCFG_BASE_CORE_TSK_MONITOR	Whether to enable the task stack overflow check and stack pointer exception check	YES/NO	YES	None
LOS_TASK_MIN_STACK_SIZE	Size of the smallest task stack	[0,n)	(ALIGN(0x600, 8))	None
LOSCFG_BASE_CORE_TSK_DEFAULT_PRIO	Default task priority	[0,31]	10	The value depends on the specific service. In most cases, the default value is retained.

Copyright © Huawei Technologies Co., Ltd.

10.6 Software Timer Configuration Parameters

Configuration Item Description

Configuration Item	Description	Value Range	Default Value	Dependency	Whether Can Be Dynamically Updated
LOSCFG_BASE_CORE_SWTMR	Whether to enable the software timer module	YES/NO	YES	LOSCFG_BASE_IPC_QUEUE	No
LOSCFG_BASE_CORE_SWTMR_LIMIT	Maximum number of supported software timers	[0, n)	1024	LOSCFG_BASE_IPC_QUEUE	No
OS_SWTMR_HANDLE_QUEUE_SIZE	Size of a software timer queue	[0, n)	1024	LOSCFG_BASE_CORE_SWTMR_LIMIT	No

10.7 Semaphore Configuration Parameters

Configuration Item Description

Configuration Item	Description	Value Range	Default Value	Dependency
LOSCFG_BASE_IPC_SEM	Whether to enable the semaphore module	YES/NO	YES	None
LOSCFG_BASE_IPC_SEM_LIMIT	Maximum number of semaphores	[0, n)	1024	None

Copyright © Huawei Technologies Co., Ltd.

10.8 Mutex Configuration Parameters

Configuration Item Description

Configuration Item	Description	Value Range	Default Value	Dependency
LOSCFG_BASE_IPC_MUX	Whether to enable the mutex module	YES/NO	YES	None
LOSCFG_BASE_IPC_MUX_LIMIT	Maximum number of mutexes	[0, n)	1024	None

10.9 Hardware Interrupt Configuration Parameters

Configuration Item Description

Configuration Item	Description	Value Range	Default Value	Dependency	Whether Can Be Dynamically Updated
LOSCFG_PLATFORM_HWI	Whether to enable the hardware interrupt module	YES/NO	YES	None	No
LOSCFG_PLATFORM_HWI_LIMIT	Maximum number of hardware interrupts	Configured according to the chip manual	Configured according to the chip manual	None	No

Copyright © Huawei Technologies Co., Ltd.

10.10 Queue Configuration Parameters

Configuration Item Description

Configuration Item	Description	Value Range	Default Value	Dependency	Whether Can Be Dynamically Updated
LOSCFG_BASE_IPC_QUEUE	Whether to enable the queue module	YES/NO	YES	None	No
LOSCFG_BASE_IPC_QUEUE_LIMIT	Maximum number of supported queues (including the queue occupied by the Huawei LiteOS software timer module)	[0, n)	1024	None	No

10.11 Module Compaction Configuration Parameters

You can enable or disable modules according to your needs.

Dynamic Loading

Switch to enable or disable dynamic loading:

LOSCFG_KERNEL_DYNLOAD

Procedure:

In the **.config** file in the root directory, set the value of LOSCFG_KERNEL_DYNLOAD to n. Alternatively, use make menuconfig to disable dynamic loading in the submenu of the Kernel option.

Dependency: none.

Precautions: none.

Copyright © Huawei Technologies Co., Ltd.

Scatter Loading

Switch to enable or disable scatter loading:

LOSCFG_KERNEL_SCATTER

Procedure:

In the **.config** file in the root directory, set the value of LOSCFG_KERNEL_SCATTER to n. Alternatively, use make menuconfig to disable scatter loading in the submenu of the Kernel option.

Dependency: none.

Precaution: Turning off the LOSCFG_KERNEL_SCATTER will affect the startup performance

File Systems

JFFS2

Switch to enable or disable file systems:

LOSCFG_FS_JFFS

Procedure:

In the **.config** file in the root directory, set the value of LOSCFG_FS_JFFS to n. Alternatively, use make menuconfig to disable JFFS in the submenu of the FileSystem option.

Dependency: none.

Precautions: none.

FAT

Switch to enable or disable JFFS2:

LOSCFG_FS_FAT

Procedure:

In the **.config** file in the root directory, set the value of LOSCFG_FS_FAT to n. Alternatively, use make menuconfig to disable FAT in the submenu of the FileSystem option.

Dependency: none.

Precaution: none.

YAFFS2

Switch to enable or disable YAFFS2:

LOSCFG_FS_YAFFS

Procedure:

In the **.config** file in the root directory, set the value of LOSCFG_FS_YAFFS to n. Alternatively, use make menuconfig to disable YAFFS in the submenu of the FileSystem option.

Copyright © Huawei Technologies Co., Ltd.

Dependency: none.

Precaution: none.

RAMFS

Switch to enable or disable RAMFS:

LOSCFG_FS_RAMFS

Procedure:

In the **.config** file in the root directory, set the value of LOSCFG_FS_RAMFS to n. Alternatively, use make menuconfig to disable RAMFS in the submenu of the FileSystem option.

Dependency: none.

Precautions: none.

PROCFS

Switch to enable or disable PROCFS:

LOSCFG_FS_PROC

Procedure:

In the **.config** file in the root directory, set the value of LOSCFG_FS_PROC to n. Alternatively, use make menuconfig to disable PROC in the submenu of the FileSystem option.

Dependency: none.

Precautions: none.

Copyright © Huawei Technologies Co., Ltd.

11 Appendix

About This Chapter

11.1 OS Memory Usage

11.2 Kernel Boot Process Introduction

11.3 Compatibility Modifications in 32-Bit or 64-Bit Environments

11.1 OS Memory Usage

A tool is embedded into Huawei LiteOS to measure CPU usage of the OS in real time.

To use this tool, perform the following steps:

Step 1 On the menuconfig page, select the platform to be compiled. Then select the chip.

Copyright © Huawei Technologies Co., Ltd.

Step 2 Run the following command to compile the platform:

```
make
```

Step 3 Run the following script:

```
./tools/scripts/mem_statistic/mem_statistic.py hi3516a | more
```

hi3516a must be the platform that has been chosen and compiled in step 1.

```
liuxuwei@ubuntu:~/code/main_up/Huawei_LiteOS$ ./tools/scripts/mem_statistic/mem_statistic.py hi3516a | more
readelf: Error: './out/hi3516a/lib/liblitekernel.a': No such file
readelf: Error: './out/hi3516a/lib/libm2mcomm.a': No such file
readelf: Error: './out/hi3516a/lib/liblinuxadp.a': No such file
readelf: Error: './out/hi3516a/lib/libwpa.a': No such file

hi3516a
===================
runstop:
total:      1.7k
.text:      1.4k        .data:      0.0k        .bss:       0.1k        .rodata:    0.2k

kernel:
total:      0.0k
.text:      0.0k        .data:      0.0k        .bss:       0.0k        .rodata:    0.0k

shell:
total:      8.6k
.text:      7.3k        .data:      0.0k        .bss:       0.3k        .rodata:    1.0k

lwip:
total:    557.0k
.text:    171.1k        .data:      0.3k        .bss:     378.5k        .rodata:    7.1k

usb:
total:    146.6k
.text:    132.2k        .data:      1.9k        .bss:       0.2k        .rodata:   12.3k

dynload:
total:     23.7k
.text:     21.2k        .data:      0.0k        .bss:       0.0k        .rodata:    2.5k

libc:
total:    112.3k
--More--
```

----End

Copyright © Huawei Technologies Co., Ltd.

11.2 Kernel Boot Process Introduction

Huawei LiteOS Kernel Boot Process

Copyright © Huawei Technologies Co., Ltd.

Copyright © Huawei Technologies Co., Ltd.

11.3 Compatibility Modifications in 32-Bit or 64-Bit Environments

Data Type Size Comparison

The sizes of some data types in a 32-bit environment are different from those in a 64-bit environment. Table 1 compares data models and data type sizes of the 32-bit and 64-bit environments. During development, you need to pay attention to the changes in data type sizes.

Table 11-1 Data type size differences

Programming Type	Size in A32	Size in A64-LP64
char	8-bit	8-bit
short	16-bit	16-bit
int	32-bit	32-bit
int *(pointer)	32-bit	64-bit
long	32-bit	64-bit
long long	64-bit	64-bit
float	32-bit	32-bit
double	64-bit	64-bit
size_t	32-bit	64-bit

The 64-bit compiler of LiteOS uses the LP64 data model. As shown above, the int type stays the same size as in the 32-bit data model. The pointer, long, and size_t type sizes grow to 64 bits.

Due to the size changes of these data types, in a 64-bit environment, data types are used differently from how they are used in a 32-bit environment. Pay attention to the following points when using the data types in the 64-bit environment to ensure that you use the data types correctly.

Key Points of Data Type Usage

1. The processing of pointers

   Pointers must not be forcibly converted to non-pointer types, such as the long and int data types. Pointers can be forcibly converted to other types of pointers and void*. If a pointer is forcibly converted to a non-pointer type, the 64-bit compiler will report an alarm, which needs to be cleared.

Copyright © Huawei Technologies Co., Ltd.

NOTICE

When a pointer is used for mathematical calculation, it must not be forcibly and temporarily converted to the unsigned long or int type. You can convert the pointer to the char* type and then increment or decrement it by constants.

2. Change longs to ints.

   You need to change longs to ints, unless you use longs in public kernel modules. Do not use longs or forcibly convert other types to longs.

NOTICE

In some 64-bit programming standards, the use of the long type is not allowed. If the long type is used, errors are very likely to occur.

3. Check all assignments and comparison statements that are related to ints, longs, and pointers.

   Though the use of longs is not allowed, the referenced data type may be the long type that is defined inside the Kernel. In this case where you cannot change the data type definition, remember to check whether longs are used correctly.

   In new compilers, data types are automatically converted according to the following rules:

   int + long = long;

   unsigned + signed = unsigned

   Here is an example:

```
long long a;
int b; unsigned int c;
b = -2;
c = 1;
a = b + c;
```

   The value of a is 0x00000000FFFFFFFF rather than －1. The cause of this result is that b+c indicates signed+unsigned, of which the result is an unsigned integer, and to convert the unsigned integer to a long long, 0 must be inserted from the left of FFFFFFFF. Here is one more example:

```
long long a; int b; unsigned int c;
b = -2;
c = 1;
a = (long long) b + c;
```

   The value of a is 0xffffffffffffffff. The value of b becomes 0xfffffffffffffffe after b is converted to a long. The result of 0xfffffffffffffffe plus the unsigned int c is the value of a.

4. Specify data types in constant expressions.

   If you want to use a constant as 64 bits, append **ll** or **LL** to the end of the constant. If the constant is not followed by **ll** or **LL**, the compiler may process it as an int. For example:

```
long long SetBitN(long long value, unsigned bitNum) {
long long mask;
mask = 1 << bitNum;
return value | mask;
}
```


Copyright © Huawei Technologies Co., Ltd.

In this example code, 1 has been processed as an int. If bitNum is a number exceeding 32, the result of **1 << bitNum** is invalid because the part that is higher than the first 32 bits is truncated. Then the result of **1 << bitNum** becomes 0. You need to write the code in the following way:

```
long long SetBitN(long long value, unsigned bitNum) {
long long mask;
mask = (long long)1 << bitNum;
return value | mask;
}
```

In the following code, the two operations will clear the upper 32 bits of the pointers.

```
(void*) ((long long)p & ~(16-1))
(void*) ((long long)p & ~(uwAlain-1))/* uwAlain is u32 */
```

You can modify the code as follows:

```
(void*) ((long long)p & ~((long long)16-1))
(void*) ((long long)p & ~((long long)uwAlain-1))/* uwAlain is u32 */
```

Alternatively, you can use the following definitions to avoid the problem:

```
1LL // (long long)
1ULL // (unsigned long long)
```

5. Formatted input/output problems and solutions

   Compared with a 32-bit system, the following problems exist in a 64-bit system:

   - In a 32-bit system, %x can be used to print pointers. However in a 64-bit system, truncation occurs when %x is used to print pointers.

     Solution: Use %p to input and output pointers.
   - In a 32-bit system, %d can be used to print a value of type int or long. However in a 64-bit system, this usage of %d results in truncation.

     Solution: Use %ld to input and output longs.

   On a 32-bit platform, a size_t is actually an unsigned int. On a 64-bit platform, a size_t is actually an unsigned long. As a result, when **printf** and other printing functions are used to print a size_t, a warning is generated regardless of whether %u or %llu is used. To avoid forcible data type conversion caused by formatted print of data types, you can use %lu to print a size_t.

   The buffer required in a 64-bit is larger than that required in a 32-bit system. During input, ensure that the buffer size is sufficient. Here is an example code:

6.
```
char * pbuf[9];
int a = 0;
Sprint(pbuf, “%p”, &a);
```

   In a 32-bit system, this code is correct because **&a** has 4 bytes in a 32-bit system. However in a 64-bit system, this code is incorrect because **&a** has 8 bytes in length.

7. Macro definitions of maximum numbers and invalid numbers

   The macro definitions of maximum numbers and invalid numbers may change in a 64-bit system. For example:

8.
```
#define BYTES_IN_WORD 4
#define INVALID_DMA_ADDRESS 0xffffffff
```

   You need to check whether the numbers are still the maximum number and invalid in the 64-bit system. Here is another example:

```
size_t Count = BIG_NUMBER;
for (unsigned int index = 0; index != Count; index++)
……
```

   The size_t is 64-bit, but the index is 32-bit. Therefore, the **for** statement will be repeated.

Copyright © Huawei Technologies Co., Ltd.

9. Check the sizes of member variables in a union.

   All member variables in a union must have the same size. In a 64-bit environment, some variables may become larger in size than they are in a 32-bit environment. This breaks the balance between member variables. Ensure that the all member variables in a union have the same size.

   The following example is a common open source code package, which can run in an ILP32 environment but cannot run in an LP64 environment. In this example code, the length of the unsigned short array is 2, and the unsigned short array shares the same space with the long. This code is incorrect in an LP64 environment.

10.
```
union{
unsigned long bytes;
unsigned short len[2];
} size;
```

   Check whether there are special applications of unions. If there are special applications, check unions in all code to ensure that all members of each union have the same size in an LP64 environment

Optimization Suggestions

1. Data types
   - Define the data type as long long when the data is used for 64-bit computing and storage. If 32 bits are sufficient, you are advised to define the data type as int or unsigned int so that memory use can be reduced. Do not define the data type as long.
   - Use architecture-independent type definitions, such as u32, u16, u64, s16, s32, and s64.
2. Structure definition

   Place commonly used data fields at adjacent positions so that they can be fetched to the cache at once. This improves operating performance and efficiency.
3. In a 64-bit system, the 8-byte alignment is used. Note that you need to place structures according to the new alignment mode.

```
struct MyStruct
{
bool m_bool;
char *m_pointer;
int m_int;
};
```

   In the preceding example, 4 bytes will be inserted between bool and char* to align them on an 8-byte boundary. To reduce memory use, you need to arrange the structures as follows:

```
struct MyStructOpt
{
char *m_pointer;
int m_int;
bool m_bool;
};
```


Copyright © Huawei Technologies Co., Ltd.