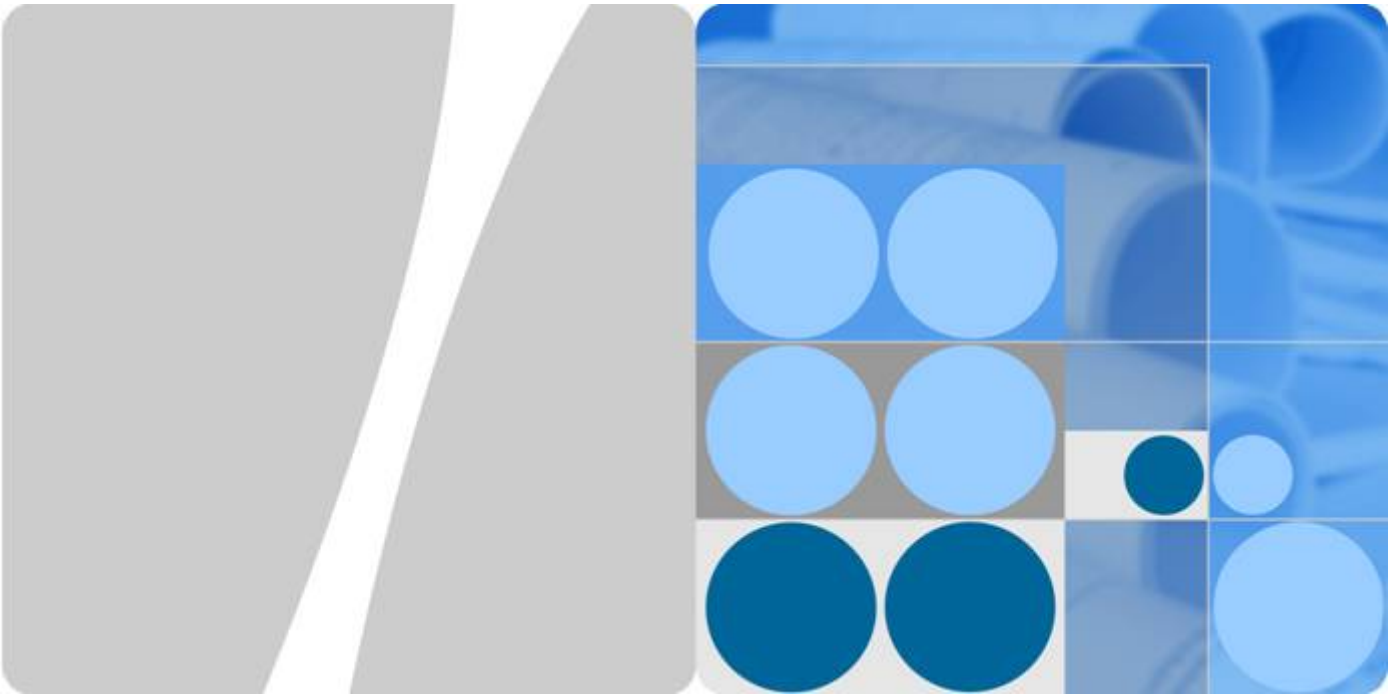


EulerOS V2.0SP2 ARM64 产品文档

文档版本　01

发布日期　2019-08-12

华为技术有限公司

HUAWEI

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 系统规格

1.1 支持的硬件环境

- 支持的服务器类型如下表所示

服务器类别	服务器型号
机架服务器	Taishan 2180
机架服务器	Taishan 5180
机架服务器	Taishan 2280

- 最小硬件规格

  EulerOS支持的最小硬件规格如下表所示

部件名称	EulerOS支持的最小硬件规格	说明
架构	Aarch64	仅支持ARM的64位架构。
CPU	海思Hi1612或者Hi1616芯片	同一集群计算节点物理服务器CPU强烈建议同一系列。
内存	不小于8G	-
硬盘	至少需要一块大于120G的硬盘	支持IDE、SATA、SAS等接口的硬盘。
网口	NIC网口数目>=1，推荐网卡数目为6个，业务功能和网络可靠性都能保证	至少存在一个网口支持PXE功能，否则不支持自动化部署，网卡速率一律要求千兆以上。

部件名称	EulerOS支持的最小硬件规格	说明
RAID卡（可选）	组RAID后映射出一块逻辑盘 推荐使用LSI3008 或LSI3108 raid卡	组RAID后有助于提高硬盘可靠性，RAID建议采用全组方式，不允许采用混搭方式即出现RAID盘和散盘共存的情况。
HBA卡（可选）	无特殊要求	选择FC SAN存储设备时必备。
BMC/IPMI（可选）	兼容标准的IPMI规范	推荐提供。
USB接口（可选）	USB2.0以上标准接口	可用于USB启动/USB光驱启动。
CD-ROM（可选）	标准CD/DVD光驱	可使用光盘自行安装。
其他	非必须	如确需其它特殊硬件，要求能够获得对应驱动。

2 系统安装指南

2.1 安装简介

本章用于指导用户如何通过光盘安装方式或PXE网络安装方式，在华为ARM服务器上部署EulerOS。

- 光盘适合用户在单个服务器上手动安装，自定义安装分区、计算机名等。
- PXE适合于批量自动安装。

2.2 最低硬件要求

表 2-1 最低硬件要求

类型	最小规格和其它要求
内存	8G
硬盘	150G
光驱	标准CD/DVD光驱
网络	以太网或 XGE光纤网

2.3 光盘安装 EulerOS

当准备单机安装EulerOS，并且需要用户自定义安装分区和系统配置时，适合使用光盘安装方式。光盘安装要求服务器具备DVD光驱，并且准备好EulerOS安装光盘。

2.3.1 挂载光盘

使用iBMC web界面中虚拟控制台加载光盘安装ISO镜像文件，如图 1所示。

步骤1 点击工具栏上光驱按钮，选择“镜像文件”。

步骤2 点击 “浏览” 按钮加载本地EulerOS ISO 文件。

步骤3 点击“连接”按钮，完成光盘挂载

图 2-1 iBMC web 界面中虚拟控制台

----结束

2.3.2 设置服务器从光盘启动

- 在iBMC web界面“配置--系统启动项”中选择光驱，指定系统启动项为光驱，如图 1所示。

图 2-2 配置系统启动项

- 在远程控制界面上选择“重启”，如图 2。

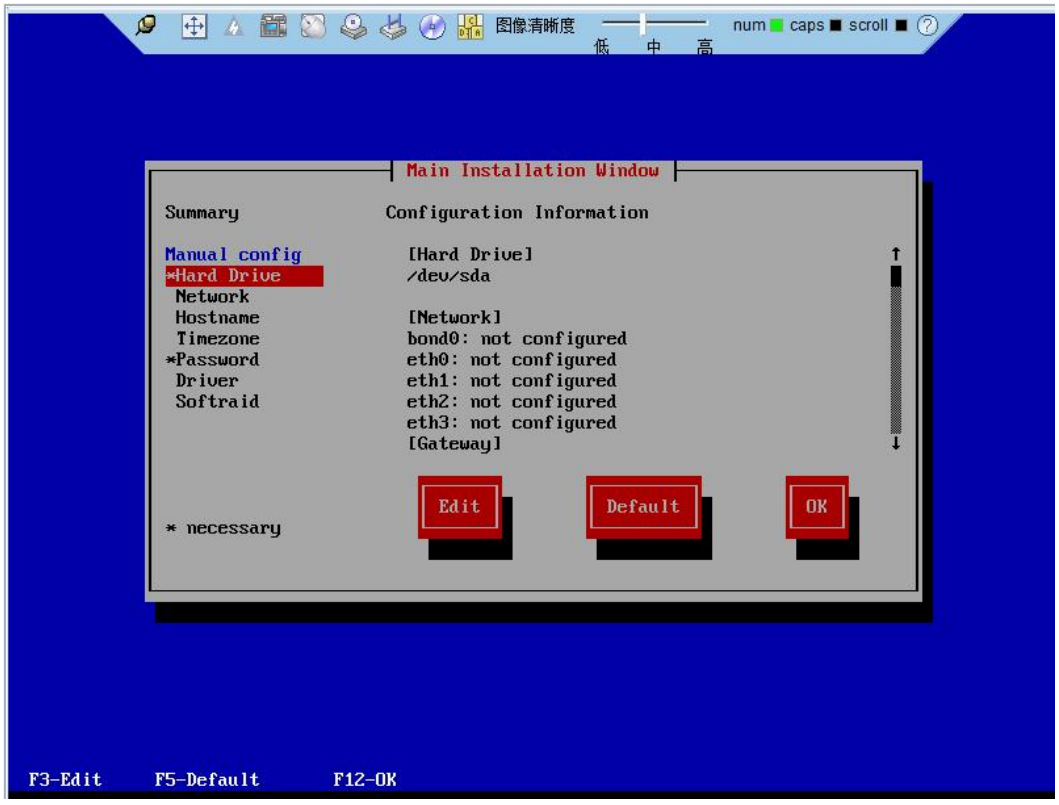
图 2-3 重启

2.3.3 从光盘安装 EulerOS

2.3.3.1 配置安装

- 默认安装在 /dev/sda磁盘设备。在“*Hard Drive”选项中选择将EulerOS安装到其它磁盘。

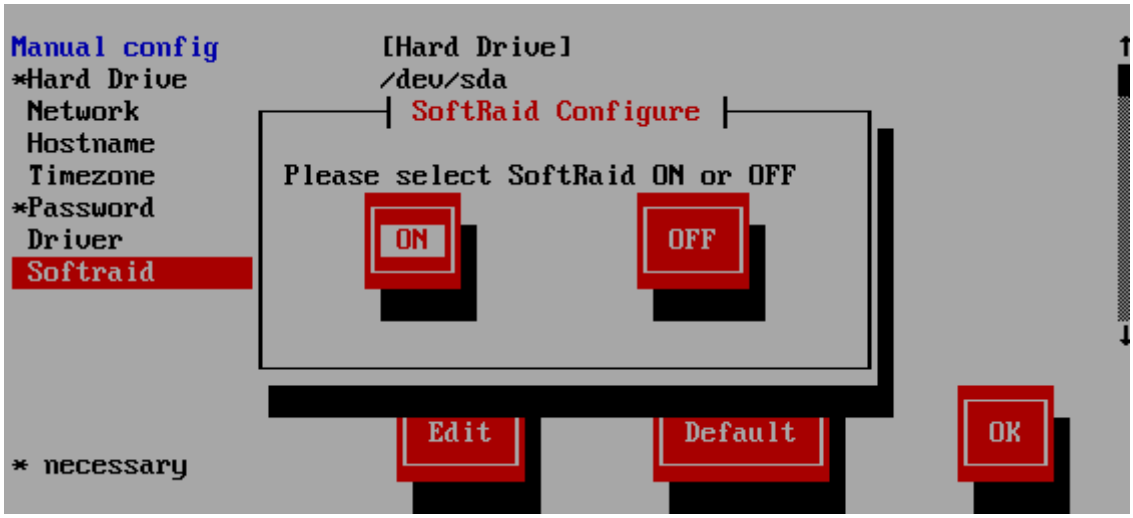
图 2-4 自定义安装到其它磁盘



- 选择打开软raid（可选）

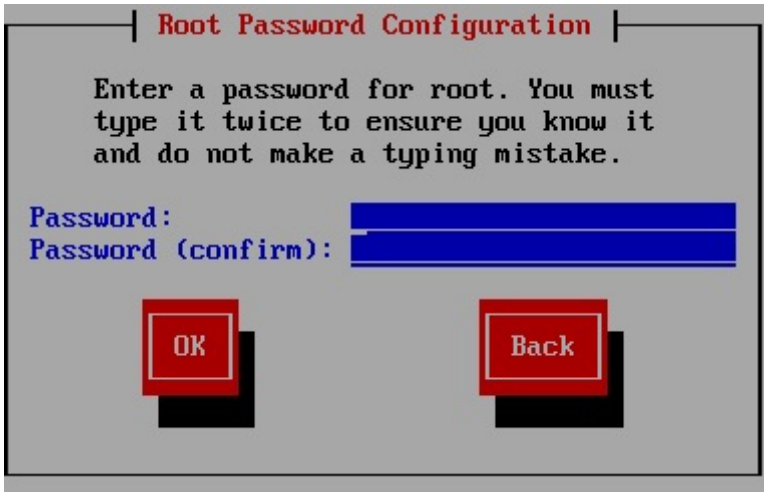
  光盘安装时支持配置软RAID，软RAID配置默认为关闭状态，配可通过如下界面中选择“SoftRaid”，然后使能软RAID（选择on）。

图 2-5 选择 raid



- 在 “*Password”选项中设置Root用户密码。

图 2-6 设置 root 账户密码

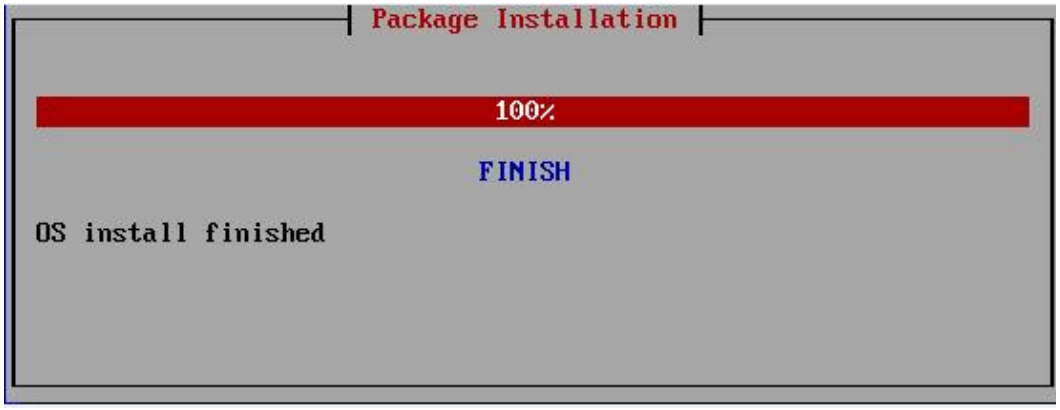


- 带“*”前缀的选项必须有用户设置确认，其它选项建议使用默认配置。
- 点击“OK”按钮完成配置。

2.3.3.2 完成 EulerOS 安装

- 系统安装之后会自动重启，系统默认从硬盘启动。用root用户，以及前面设置过的密码登录到EulerOS。

图 2-7 完成 EulerOS 安装



2.4 PXE 安装 EulerOS

网络安装一般利用PXE方式，用于多台机器并行安装，安装过程自动完成，用户并不需要与安装引导系统进行直接交互，而是通过配置文件对不同机器进行配置。

PXE安装另需要一台计算机作为PXE服务器，并且预装有DHCP服务；另外操作者应当熟悉Linux环境网络配置的相关操作。

2.4.1 部署 PXE 安装服务器

PXE服务器网络地址应当和待安装EulerOS的目标计算机位于同一网段内。

步骤1 配置PXE服务器DHCP服务，详见**2.4.1.1 配置DHCP服务**；

步骤2 配置PXE服务器TFTP服务，详见**2.4.1.2 配置TFTP服务**；

步骤3 部署PXE服务器上引导系统配置文件和安装文件。

----结束

2.4.1.1 配置 DHCP 服务

步骤1 检查DHCP安装。

执行以下命令，检查安装服务器DHCP服务预安装信息。

```
rpm -qa|grep dhcp
```

如果未安装，则可以通过yum（EulerOS/redhat），yast（SUSE）或者rpm安装。

根据需要，修改“etc/sysconfig/”目录下的dhcpd文件。

```
DHCPD_CONF_INCLUDE_FILES="/etc/dhcp/dhcpd.conf"
DHCPD_INTERFACE="eth0"    #eth0是PXE服务器和目标机网络相连的网口。
```

步骤2 修改配置文件。

打开配置文件“/etc/dhcp/dhcpd.conf”，根据实际情况修改带下划线的参数。

```
default-lease-time 6000;
max-lease-time 72000;
ddns-update-style none;
ddns-updates off;
allow booting;
```



```
allow bootp;
subnet  192.168.0.0 netmask 255.255.0.0   #安装服务器地址子网掩码
{
option routers  192.168.0.1;            #路由地址
range 192.168.1.100  192.168.1.200;       #地址分配起至范围
server-name  "bootserver ";           #安装服务器主机名
next-server 192.168.1.1;              #TFTP服务器地址
filename  "grubaa64.efi";        #启动引导程序文件名
}
```

- 下划线的是需要根据实际情况做修改的部分。
- “#”后所跟为注释内容，配置好后需要删除。

步骤3 重启DHCP服务。

```
service dhcpd restart
```

----结束

2.4.1.2 配置 TFTP 服务

步骤1 检查TFTP预安装。

执行以下命令，检查安装服务器TFTP服务预安装信息。

```
rpm-qa|grep tftp
```

如果未安装，则可以通过rpm安装。

步骤2 配置TFTP服务。

打开TFTP配置文件“/etc/xinetd.d/tftp”，将**disable = yes**修改为**disable = no**，在**server_args**中指定用作提供TFTP下载的目录。修改后的文件如下：

```
servicetftp
{
socket_type     = dgram
protocol        = udp
wait            = yes
flags           =IPv6 IPv4
user            = root
server          =/usr/sbin/in.tftpd
server_args    = -s -c /tftpboot     #Download path of tftp
disable         = no                    #Disable TFTP or not
per_source      = 11
cps              = 100 2
flags            = IPv4
}
```

- 下划线的是需要根据实际情况做修改的部分。
- “#”后所跟为注释内容，配置好后需要删除。

步骤3 复制启动文件

将系统文件及引导启动文件复制到TFTP的下载目录下，并更改所有文件权限为755。

在EulerOS发布的ISO中包含有相应的启动引导文件grubaa64.efi，内核映像文件linux，初始化文件系统initrd以及参考配置文件模板。将这些文件复制到TFTP的下载目录，结构具体如下：

```
# tree tftpboot/
|-- all
|   |-- addonscript
|   |   |-- after_insfail_hook
|   |   |-- after_inssucc_hook
|   |   |   `-- S01service_control.sh
```

```
|   |   |-- before_install_hook
|   |   |-- before_mkfirstboot_hook
|   |   |-- before_mkinitrd_hook
|   |   `-- before_reboot_hook
|   |-- cfg
|   |   |-- UVPOSsetup.conf
|   |   |-- grub.cfg
|   |   `-- isopackage.sdf
|   `-- filelist
|-- grub.cfg
|-- grubaa64.efi
|-- boot
|   |-- initrd                --- Ramdisk file, with install-service
|   `-- linux                 --- kernel Image
`-- tmp
    |-- filelist
    `-- repo
        |-- OS.tar.gz             --- kernel and ramdisk file here
        `-- OS.tar.gz.sha256
```

说明

上面有两个filelist文件，它们描述了所在文件路径下哪些文件需要被下载。tmp/filelist来自 ISO路径下 for-pxe/tmp/fileist。

步骤4 编辑引导系统配置文件grub.cfg。

参照grub.cfg.template 做修改，内容如下：

```
#Sample GRUB configuration file
settimeout=5
Setdefault= euler-c10
# Forbooting GNU/Linux
menuentry"EulerOS2.0-LemonC10" --id euler-c10{
setroot=(tftp, PXE_SERVER_IP)
#echo"Loading kernel..."
linux /boot/linux rdinit=/sbin/init  console=ttyS0,115200earlycon=hisilpcuart,mmio,
0xa01b0000,0,0x2f8 pci=pcie_bus_perf vga=0x317console=tty0  nohz=off crashkernel=1024M\
install_mode=installinstall_media=pxe install_disk=/dev/sda install_repo=tftp://PXE_SERVER_IP/tmp
install_cfg=tftp://PXE_SERVER_IP/all soft_raid=off
#echo"Loading the initrd..."
initrd /boot/initrd
}
```

- 下划线的是需要根据实际情况做修改的部分。 **PXE_SERVER_IP** 安装服务器IP。
- “#”后所跟为注释内容，配置好后需要删除。
- install_disk=/dev/sda 指定OS安装盘，这里可自行指定；如果用默认配置，则会在sda盘创建sda1(512M)、sda2(128G)两个分区。 修改分区大小，在UVPOSsetup.conf中指定：

```
<hd-partitions>
hd0   /boot            1024M      primary    fat32   yes
hd0   /                131072M    primary    ext3    yes
</hd-partitions>
```

步骤5 配置自定义脚本 [可选]

在all/addonscript/路径下，默认有before_install_hook，before_reboot_hook，after_inssucc_hook，after_insfail_hook等几个文件夹，用户可以将自己的脚本放置在响应的路径下，则那些脚本将分别在“开始安装OS之前”，“安装OS后重启系统之前”，“安装OS成功后”，“安装OS失败后”执行。

说明

1. 自定义的脚本需要以”S"开头，表示service；
2. 添加脚本后，要在./all/filelist中添加相应描述。

```
localhost:/tftpboot# cat all/filelist
addonscript/
```



```
addonscript/after_inssucc_hook/
addonscript/after_inssucc_hook/S01service_control.sh
addonscript/before_install_hook/
addonscript/before_mkfirstboot_hook/
addonscript/before_mkinitrd_hook/
addonscript/before_reboot_hook/
cfg/
cfg/UVPOSsetup.conf
cfg/isopackage.sdf
cfg/grub.cfg
```

步骤6 重启TFTP服务。

```
service xinetd restart
```

----结束

2.4.1.3 配置软 RAID（可选）

同时PXE还支持配置软RAID，配置操作步骤如下:

步骤1 使能软raid安装。找到PXE启动的grub配置文件grub.cfg，需要安装的grub选项中将soft_raid=off修改成soft_raid=on（软raid安装默认为关闭状态）。

步骤2 根据实际环境 修改all/cfg/UVPSetup.conf文件配置软raid安装的硬盘，默认配置如下

```
<soft-raid>
clearpart --drivers=sda,sdb
zerogpt
part --size=1024 --asprimary --ondrive=sda --nickname=raid.11 --fstype=vfat
part --size=131072 --asprimary --ondrive=sda --nickname=raid.12
part --size=1024 --asprimary --ondrive=sdb --nickname=raid.21 --fstype=vfat
part --size=131072 --asprimary --ondrive=sdb --nickname=raid.22
raid --fstype=vfat --mountpoint=/boot --level=RAID1 --nickname=raid.11,raid.21 --
options=rw,async,noatime,nodiratime
raid --fstype=ext3 --mountpoint=/ --level=RAID1 --nickname=raid.12,raid.22 --
options=rw,async,noatime,nodiratime
setboot --flag=boot --ondrive=sda
setboot --flag=boot --ondrive=sdb
</soft-raid>
```

配置解读如下：

选择sda和sdb为软raid安装系统盘。

sda1（1024M）和sdb1(1024M)组成raid1作为/boot分区。

sda2（131072M）和sdb2(131072M)组成raid1作为/分区。

其中，raid盘以及分区大小均可根据实际安装环境进行修改。

----结束

2.4.2 通过 PXE 服务器安装 EulerOS

步骤1 BIOS 设置网络启动，详见**2.4.2.1 配置PXE引导安装**；

步骤2 安装EulerOS，详见**2.4.2.2 安装EulerOS**。

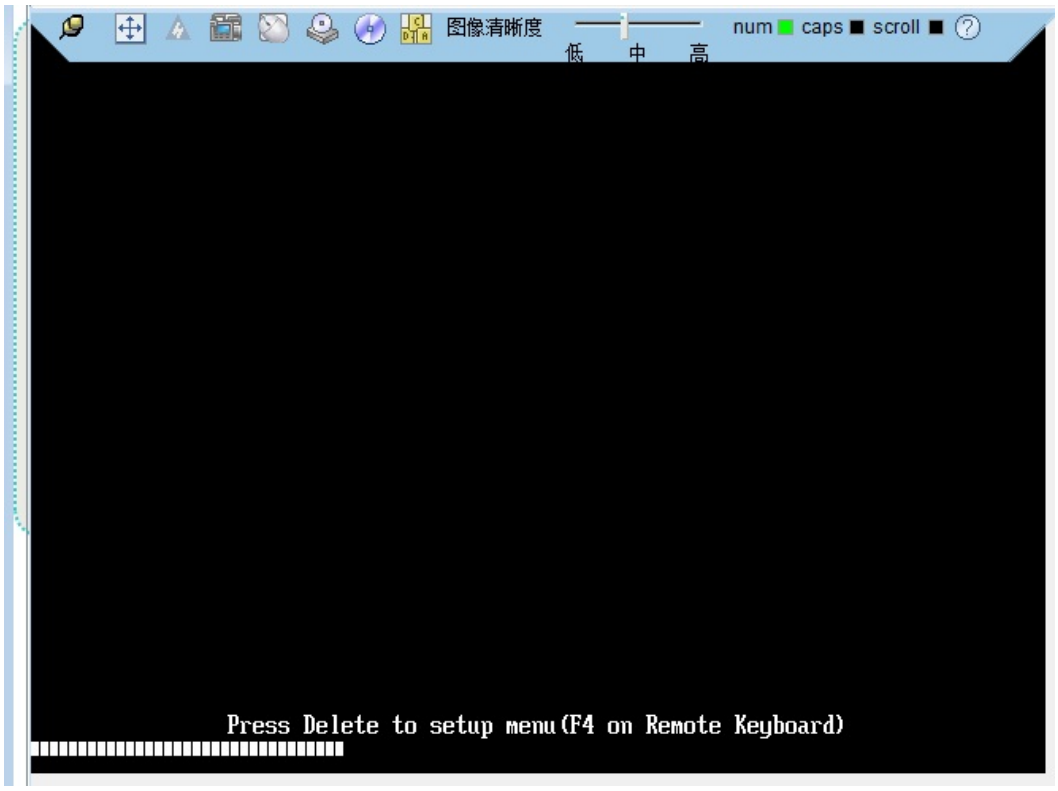
启动方式设置完成后，设备将自动重启。之后系统将自动从网络进行安装直到安装完成，不需要用户操作

----结束

2.4.2.1 配置 PXE 引导安装

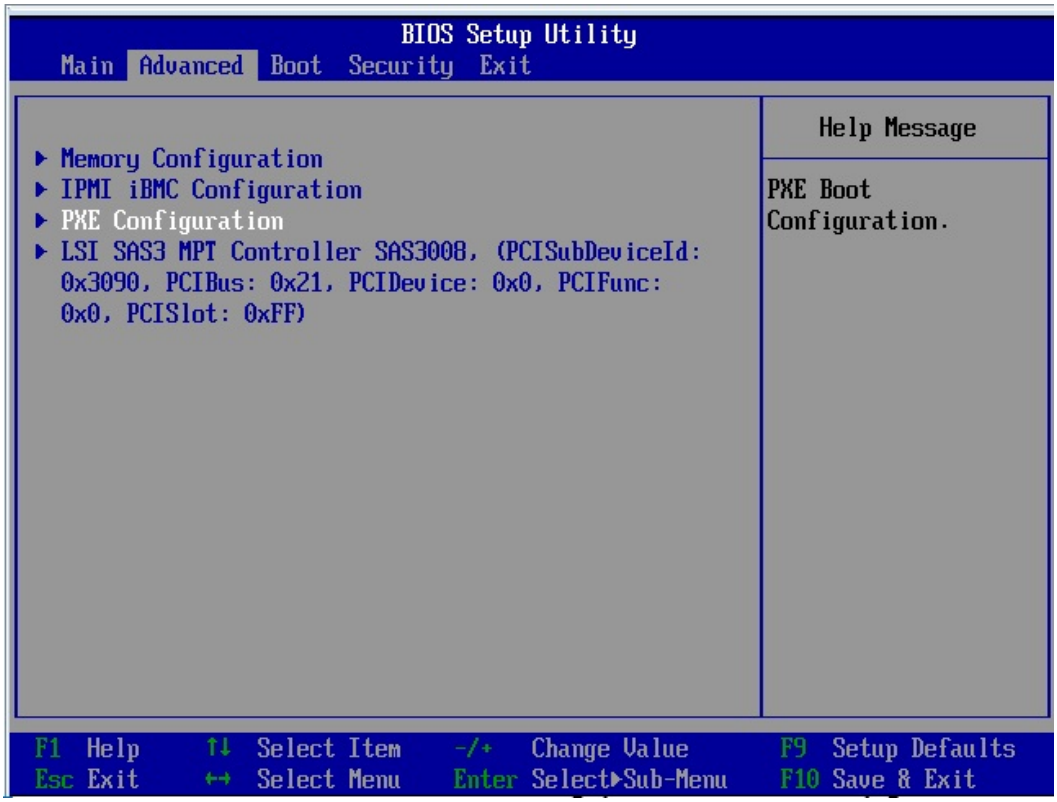
步骤1 进入BIOS，服务器开始启动，当出现图 1 界面时，按“Delete”键，进入Boot启动引导菜单。

图 2-8 启动



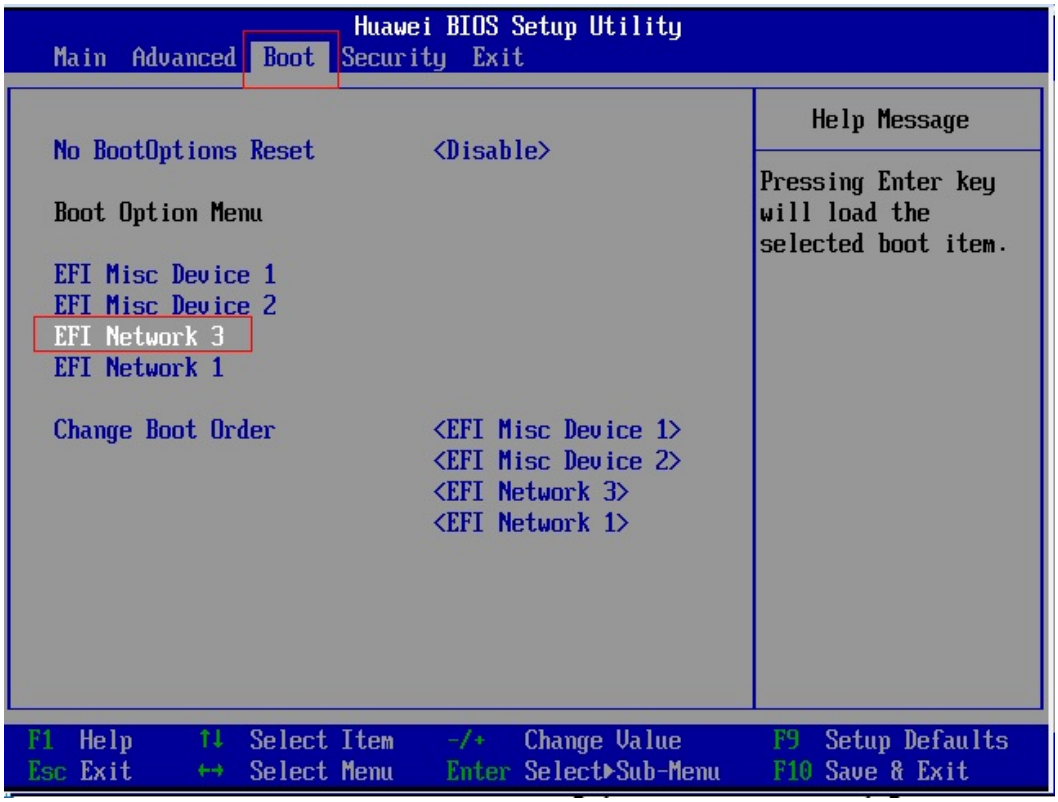
步骤2 使用能与PXE服务器相连的网卡的PXE功能，如图 2。

图 2-9 BIOS Setup Utility



步骤3 选择从PXE启动，如图 3。然后保存设置并退出。

图 2-10 设置从 PXE 启动



----结束

2.4.2.2 安装 EulerOS

配置好PXE启动并重启服务器后，无需任何操作，自动进行EulerOS安装。

说明

pxe安装之后EulerOS的默认密码为Euler12#$，请用户在首次登陆之后进行修改。

2.5 光盘安装 EulerOS 虚拟机

EulerOS支持使用光盘文件安装虚拟机。

2.5.1 准备 EulerOS.xml

步骤1 执行以下命令，创建qcow2文件。

```
qemu-img create -f qcow2 euleros.qcow2 150G
```

步骤2 修改xml文件中“source file”为实际虚拟文件qcow2的路径。如本例为：/file-path/euleros.qcow2。

步骤3 修改xml文件中“source file”为实际ISO文件路径。如本例为：/file-path/EulerOS-V2.0SP2-aarch64-dvd.iso。

----结束

EulerOS.xml 文件如下：

```
<domain type='kvm'>
<name>EulerOS</name>
<memory unit='KiB'>14049280</memory>
<currentMemory unit='KiB'>14049280</currentMemory>
<vcpu placement='static'>4</vcpu>
<resource>
<partition>/machine</partition>
</resource>
<os>
<type arch='aarch64' machine='virt-2.6'>hvm</type>
<loader readonly='yes' type='pflash'>/usr/share/edk2/aarch64/QEMU_EFI-pflash.raw</loader>
<nvram template='/usr/share/edk2/aarch64/vars-template-pflash.raw'>/usr/share/edk2/aarch64/
euler.fd</nvram>
<boot dev='hd'/>
</os>
<features>
<gic version='2'/>
</features>
<cpu mode='host-passthrough'/>
<clock offset='utc'/>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<devices>
<emulator>/usr/bin/qemu-kvm</emulator>
<disk type='file' device='disk'>
<driver name='qemu' type='qcow2' cache='none' io='native'/>
<source file='/file-path/euleros.qcow2'/>
<backingStore/>
<target dev='sda' bus='scsi'/>
</disk>
<disk type='file' device='cdrom'>
<driver name='qemu' type='raw' cache='none' io='native'/>
<source file='/file-path/EulerOS-V2.0SP2-aarch64-dvd.iso'/>
<backingStore/>
<target dev='sdb' bus='scsi'/>
<readonly/>
<alias name='scsi0-0-0-1'/>
<address type='drive' controller='0' bus='0' target='0' unit='1'/>
</disk>
<controller type='usb' index='0' model='ehci'>
<alias name='usb'/>
<address type='pci' domain='0x0000' bus='0x02' slot='0x01' function='0x0'/>
</controller>
<controller type='scsi' index='0' model='virtio-scsi'>
<alias name='scsi0'/>
<address type='pci' domain='0x0000' bus='0x02' slot='0x03' function='0x0'/>
</controller>
<controller type='pci' index='0' model='pcie-root'>
<alias name='pcie.0'/>
</controller>
<controller type='pci' index='1' model='dmi-to-pci-bridge'>
<model name='i82801b11-bridge'/>
<alias name='pci.1'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x0'/>
</controller>
<controller type='pci' index='2' model='pci-bridge'>
<model name='pci-bridge'/>
<target chassisNr='2'/>
<alias name='pci.2'/>
<address type='pci' domain='0x0000' bus='0x01' slot='0x00' function='0x0'/>
</controller>
<interface type='bridge'>
<source bridge='br0'/>
<target dev='vnet1'/>
<model type='virtio'/>
<alias name='net0'/>
<address type='pci' domain='0x0000' bus='0x02' slot='0x02' function='0x0'/>
</interface>
```



```
<serial type='pty'>
<source path='/dev/pts/1'/>
<target port='0'/>
<alias name='serial0'/>
</serial>
<console type='pty' tty='/dev/pts/1'>
<source path='/dev/pts/1'/>
<target type='serial' port='0'/>
<alias name='serial0'/>
</console>
<input type='tablet' bus='usb'>
<alias name='input0'/>
</input>
<input type='keyboard' bus='usb'>
<alias name='input1'/>
</input>
<graphics type='vnc' port='5902' autoport='yes' listen='0.0.0.0'>
<listen type='address' address='0.0.0.0'/>
</graphics>
<video>
<model type='virtio' heads='1' primary='yes'/>
<alias name='video0'/>
<address type='pci' domain='0x0000' bus='0x02' slot='0x04' function='0x0'/>
</video>
</devices>
<seclabel type='none' model='none'/>
<seclabel type='dynamic' model='dac' relabel='yes'>
<label>+0:+0</label>
<imagelabel>+0:+0</imagelabel>
</seclabel>
</domain>
```

2.5.2 创建新的网桥 br0

步骤1 创建网桥

```
brctl addbr br0
```

步骤2 使能网桥

```
ifconfig br0 up
```

----结束

2.5.3 定义新的虚拟机 domain

步骤1 定义新的虚拟机domain

```
virsh define EulerOS.xml
```

步骤2 启动虚拟机

```
virsh start EulerOS
```

步骤3 查询虚拟机vnc port

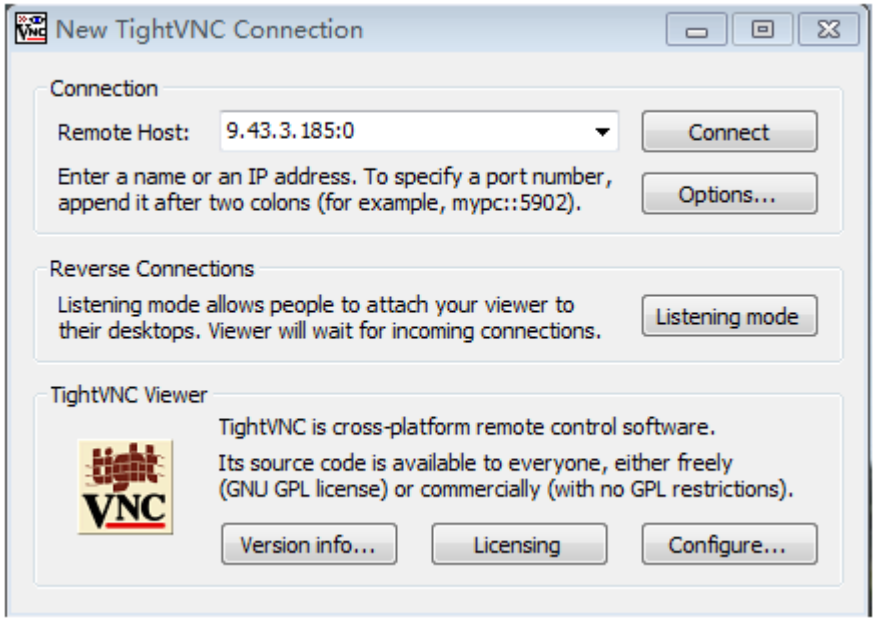
```
EulerOS:~/vm # virsh vncdisplay EulerOS
:0
```

说明

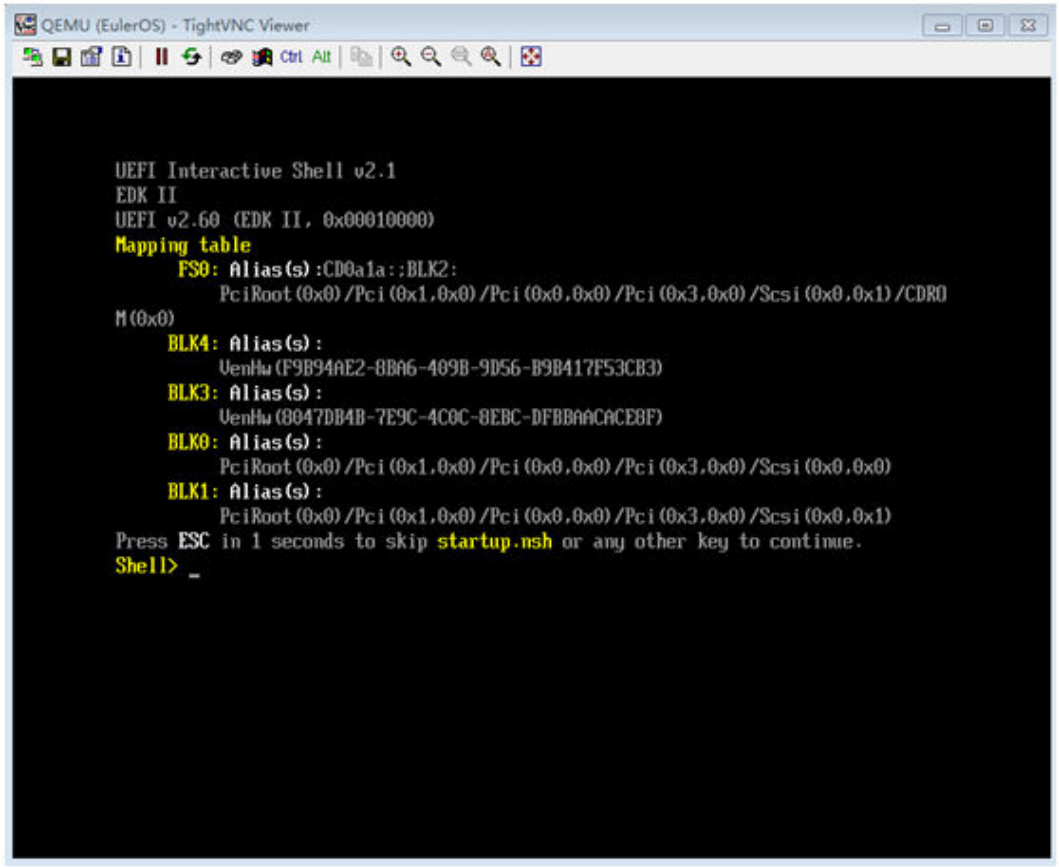
冒号后面就是vnc端口号。如本例中的vnc端口号为0。

步骤4 在PC机使用TightVNC 登陆虚拟机

Remote Host 格式为“IP:vnc-port”。



因为生成qcow2文件作为虚拟机的虚拟硬盘，是一块空白的文件。所以vnc登陆后会停留在UEFI阶段。

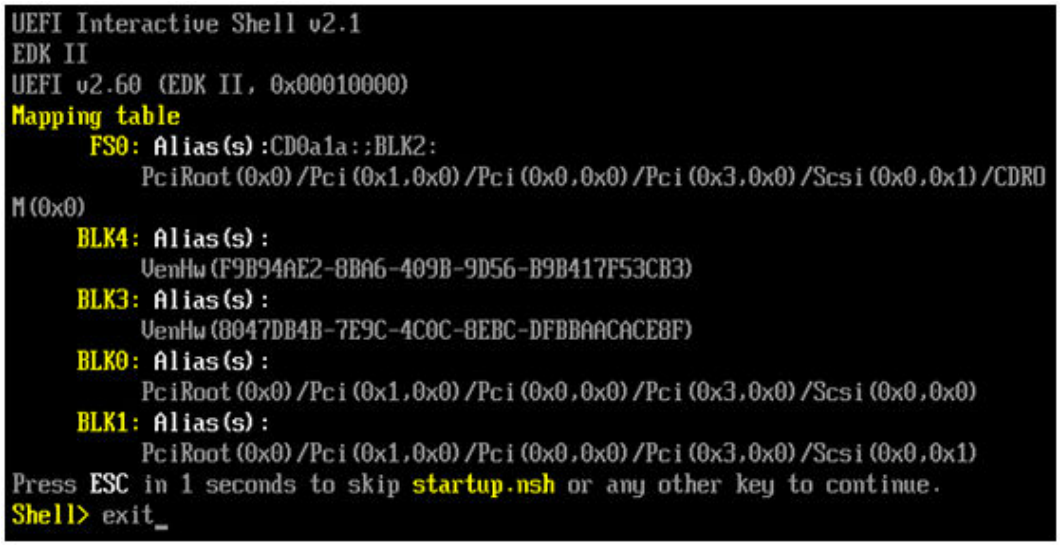


----结束

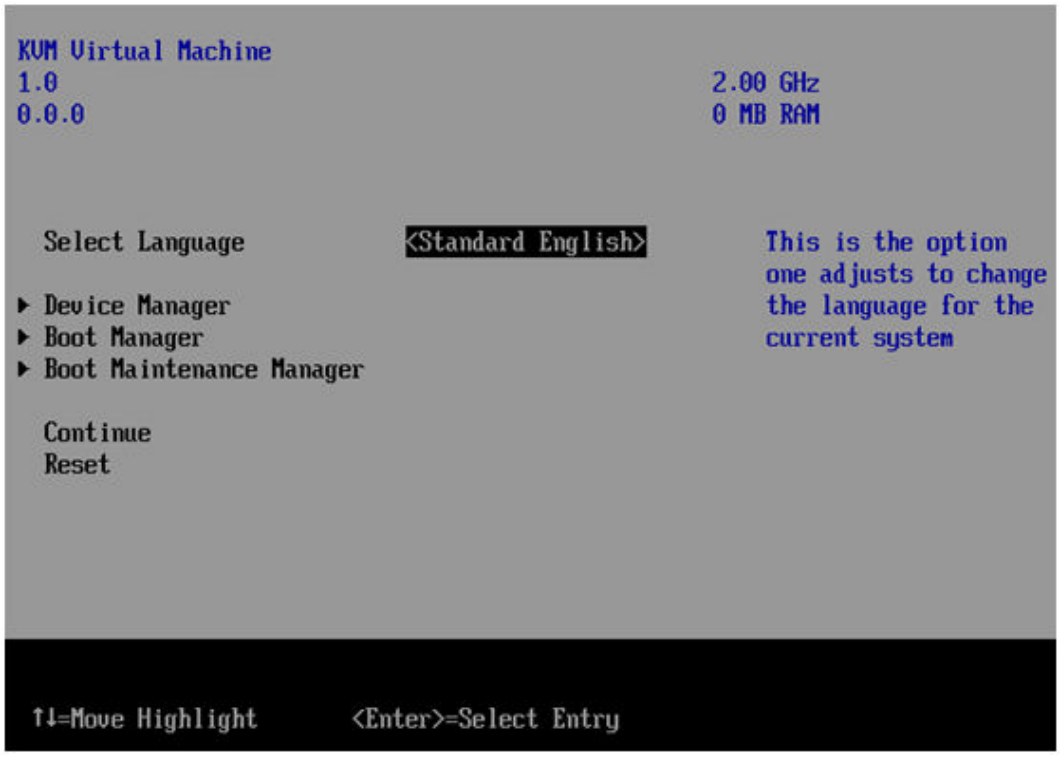
2.5.4 ISO 光盘安装到虚拟机的硬盘中

步骤1 修改启动设备从虚拟光盘启动。

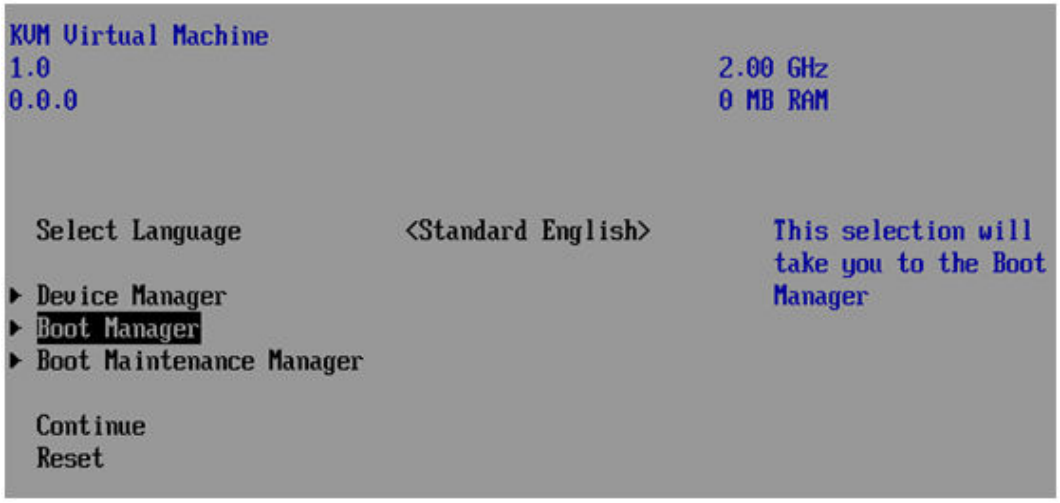
输入exit，退出当前shell：



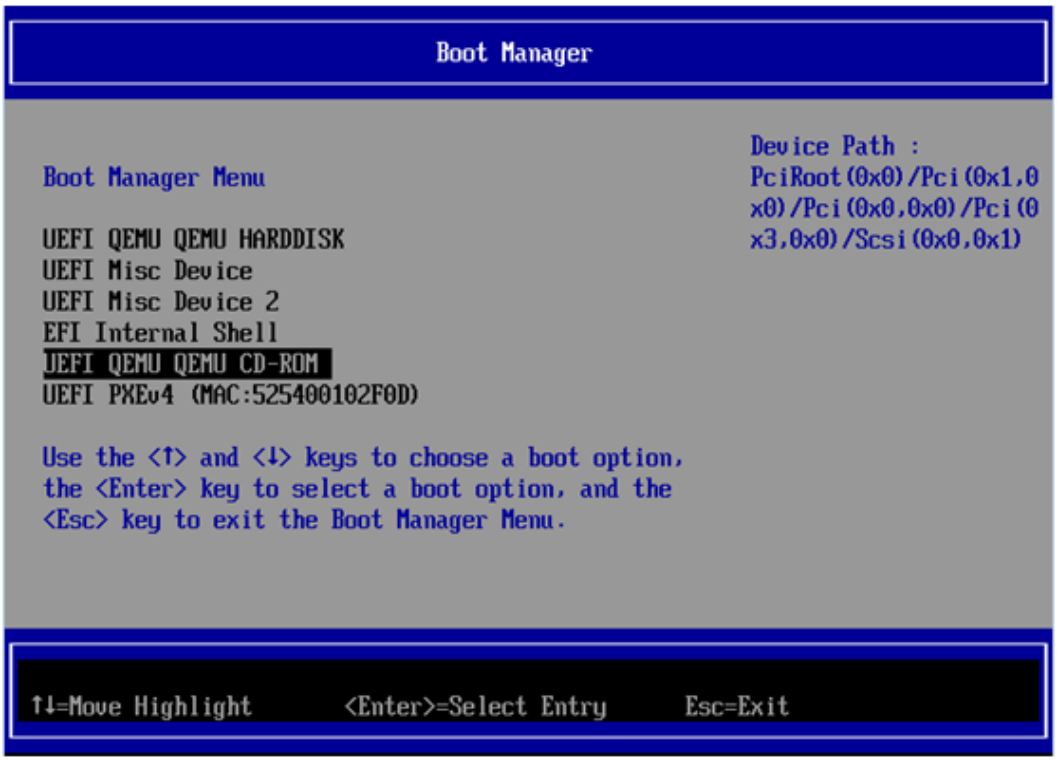
进入如下界面：



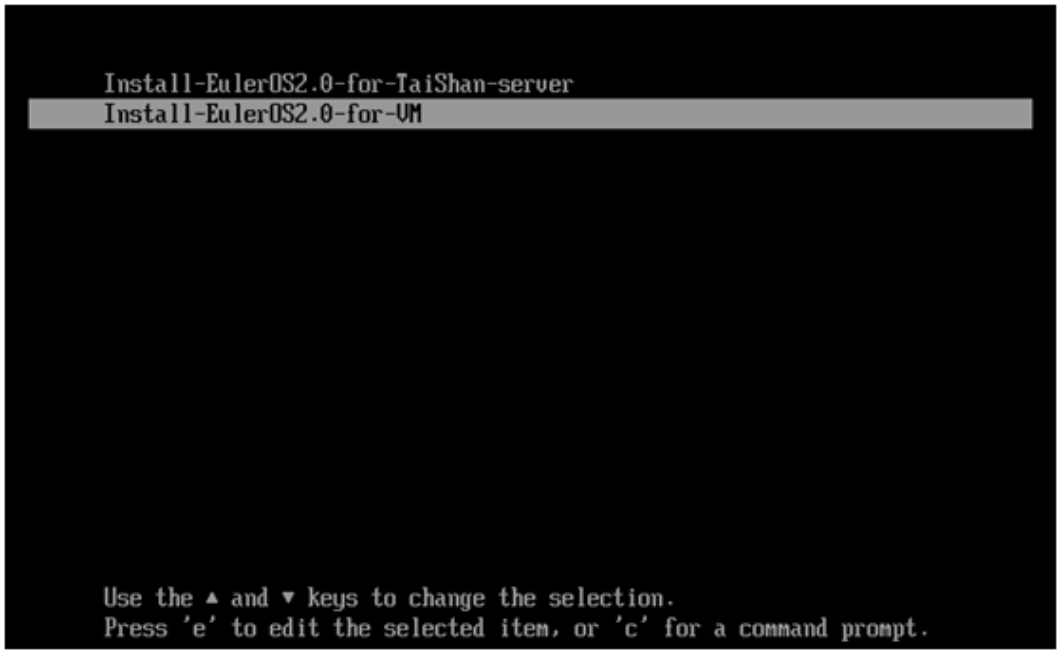
步骤2 按键盘中向下的键↓，移动到Boot Manager， 按回车键。



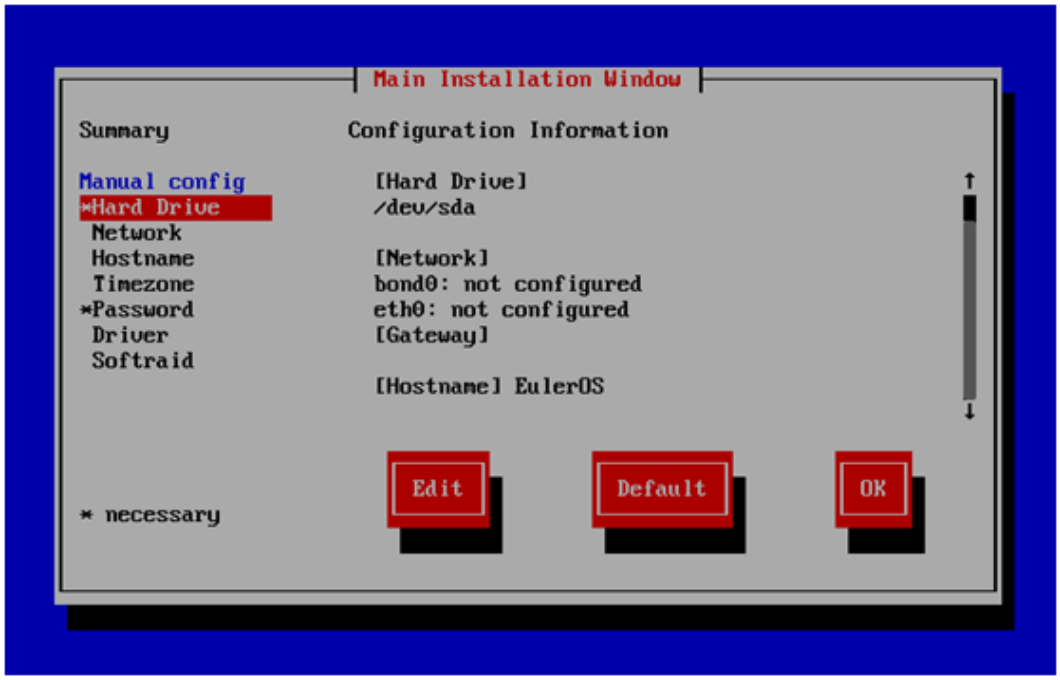
步骤3 会进入如下界面，按键盘中向下的键↓选择 UEFI QEMU QEMU CD-ROM，按回车键。



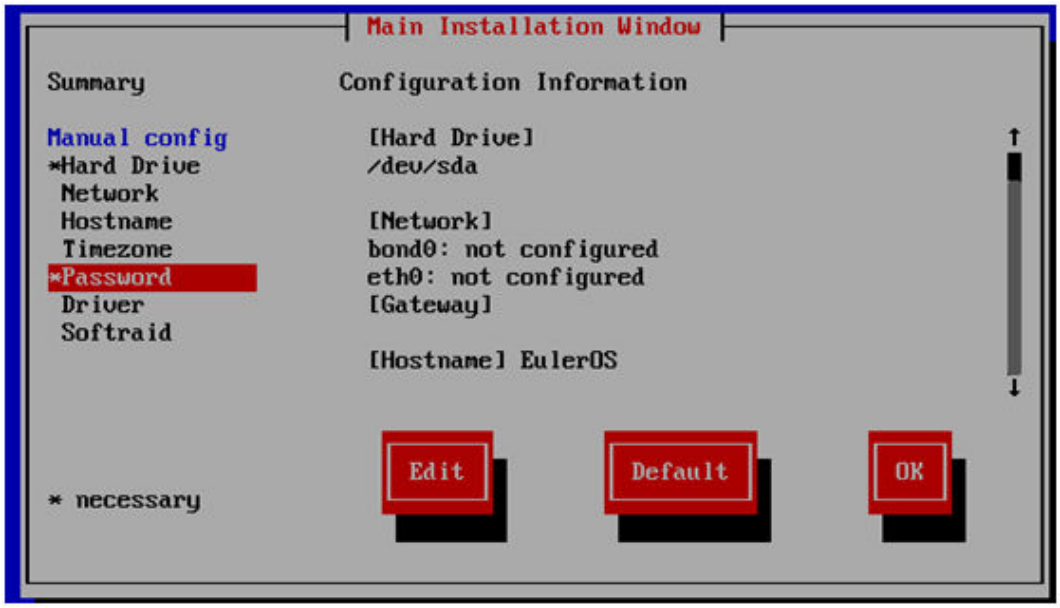
步骤4 进入grub界面，通过↓选择 Install-EulerOS2.0-for-VM，并进入光盘启动。

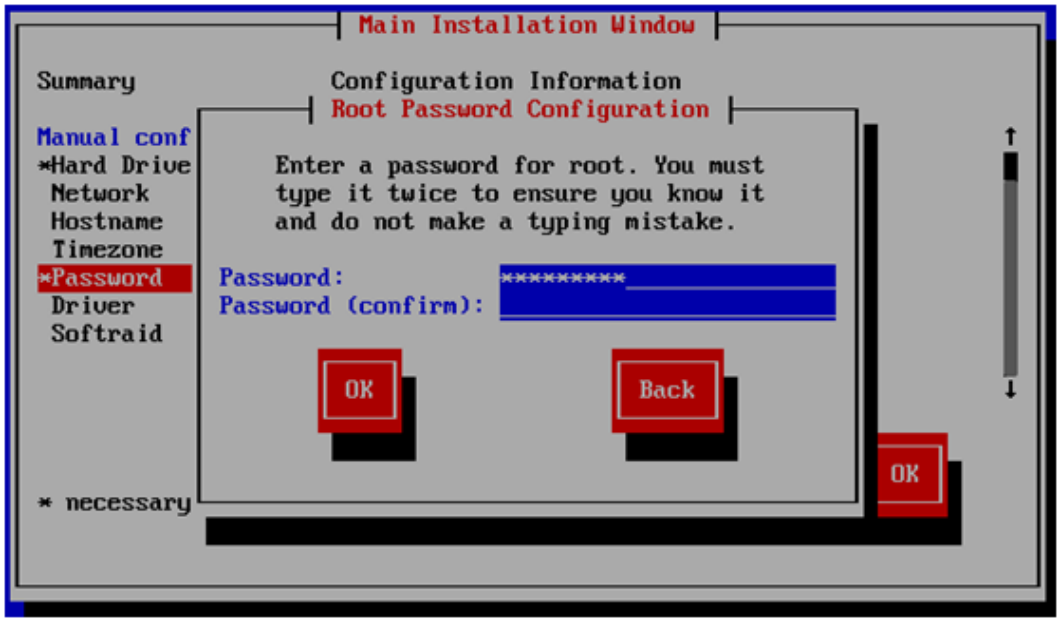


步骤5 稍等片刻，虚拟机会启动到如下界面。

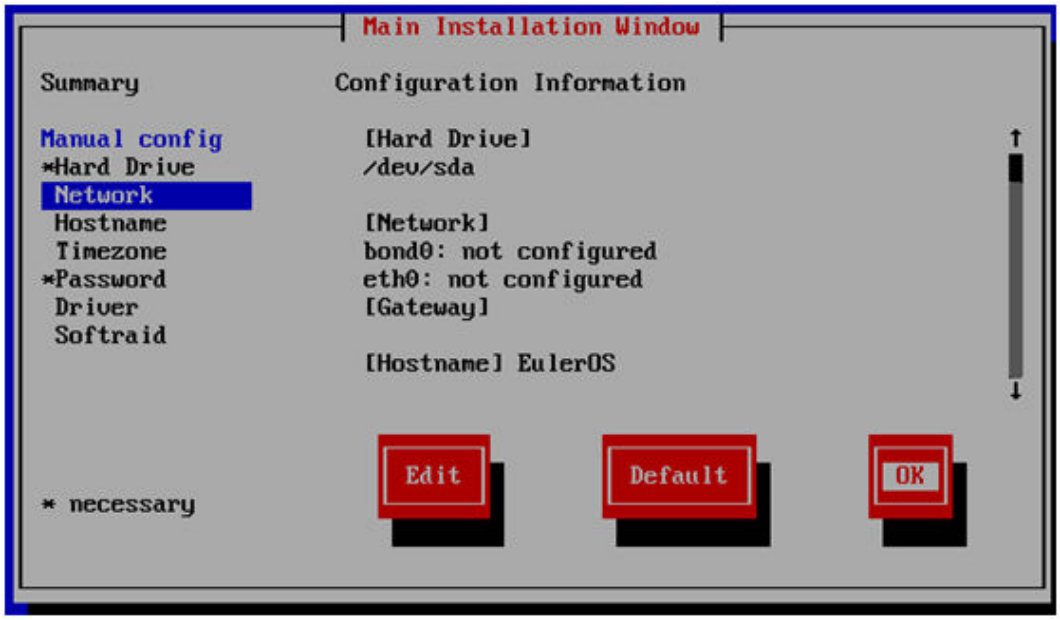


步骤6 通过↓，选择Password，并为root用户设置密码。

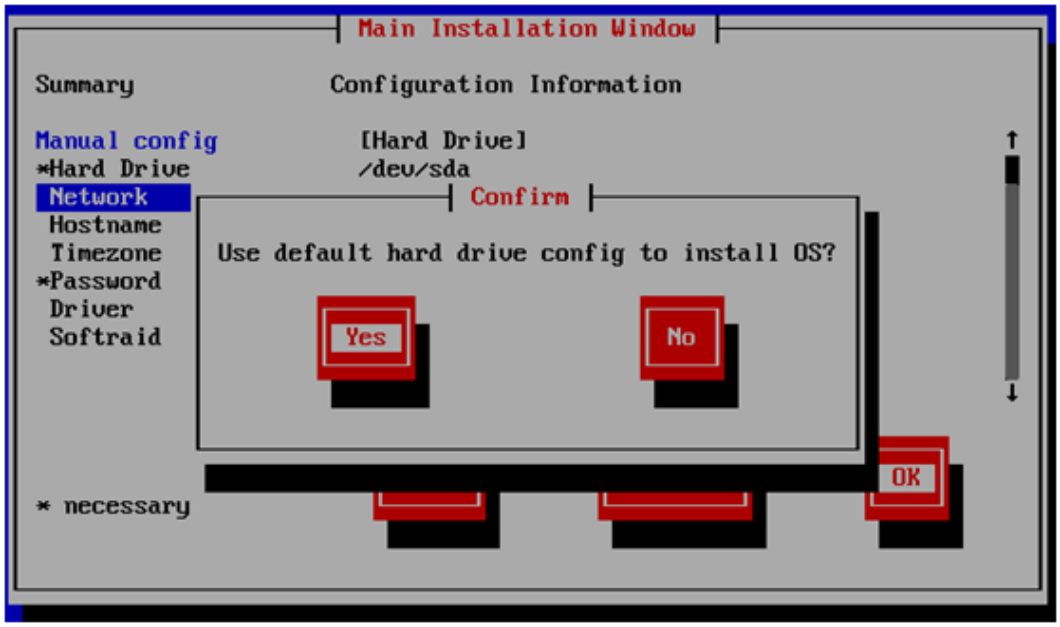


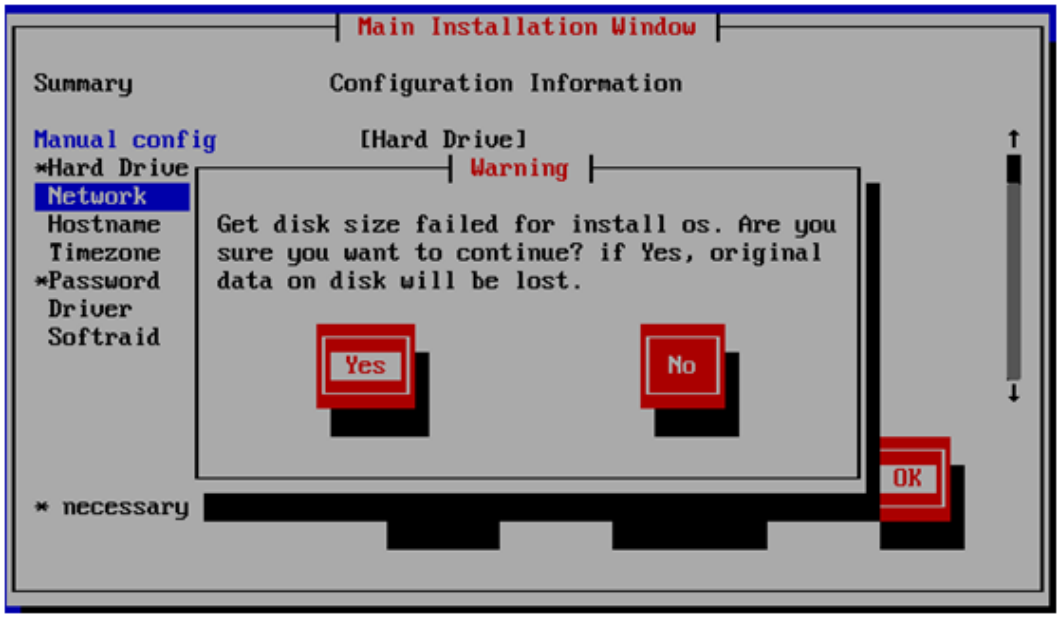


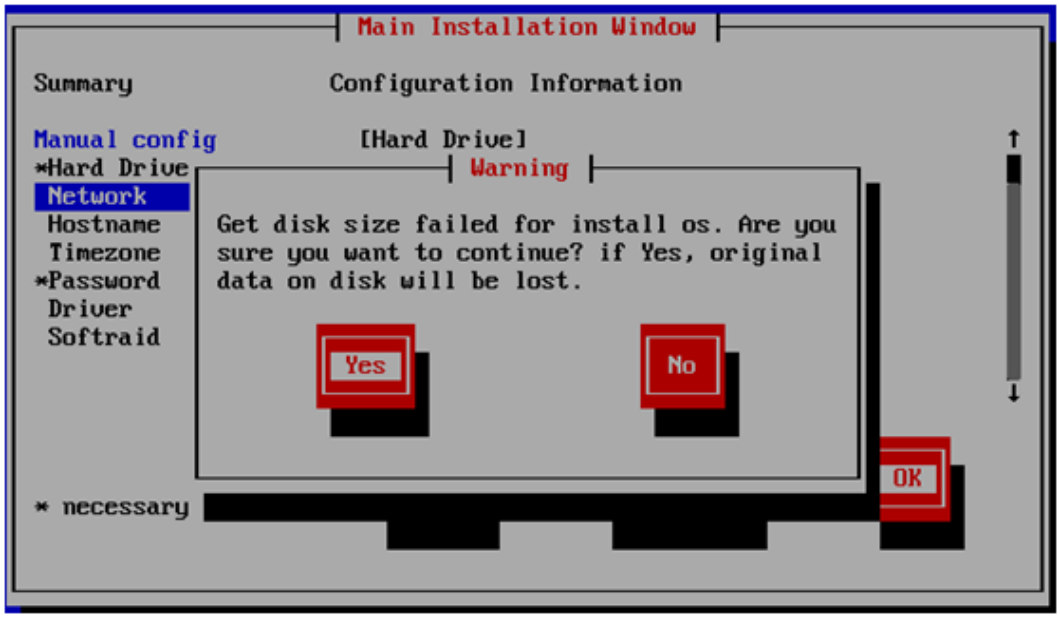
步骤7 设置完密码后，通过tab键，把焦点移动到OK按钮，并按回车键。



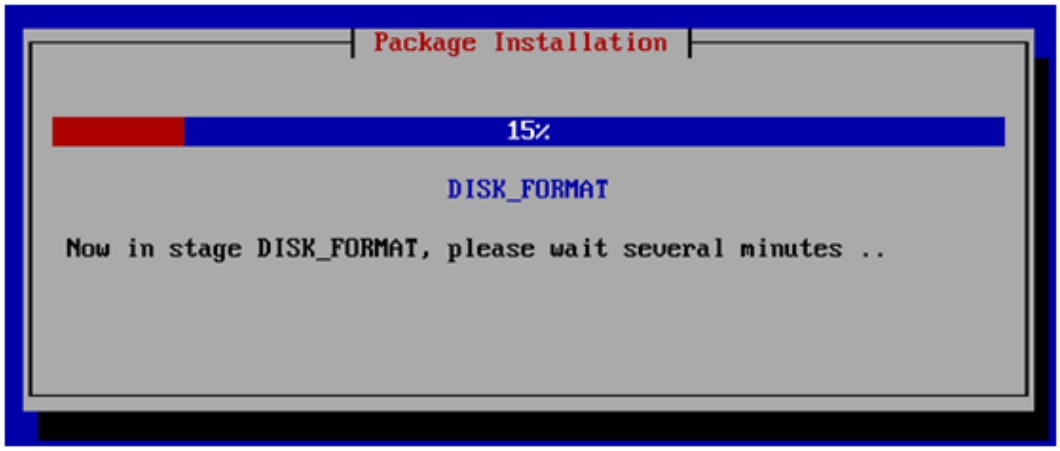
会弹出相继弹出3个对话框，选择yes按钮，继续按回车：



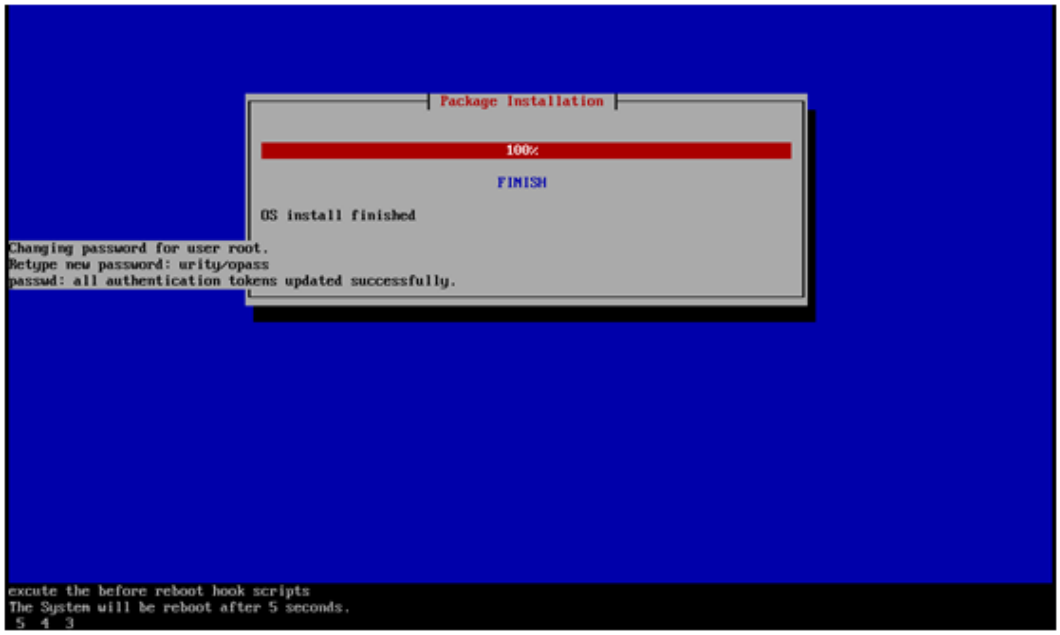




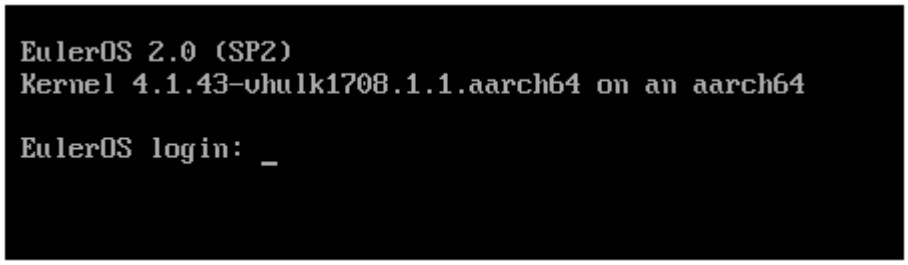
步骤8 进入安装进度条，耐心等待它安装完成。



步骤9 安装完成后，系统会自动重启。



步骤10 重启后虚拟机会自动从硬盘启动到login，用户可以登录使用该虚拟机。



----结束

2.6 配置软件包安装源

EulerOS支持yum方式安装和卸载软件，方便用户开发和拓展业务。使用yum安装软件需要配置软件源。

2.6.1 本地软件源

可以用EulerOS发布包，创建本地repo源：

```
mount  EulerOS2.0SP2-aarch64.iso  /media
cp /usr/Euler/conf/hyperstackos.repo   /etc/yum.repos.d/

cat /etc/yum.repos.d/hyperstackos.repo
[HyperstackOS]
name=hyperstackos
baseurl=file:///media/repos
enabled=1
gpgcheck=0
```

2.6.2 网络软件源

使用EulerOS发布包中的rpm包搭建网络repo源：

1. 选择一台网络中的服务器部署repo源，确认createrepo工具是否存在，如果未安装则使用如下命令安装：
   ```
   rpm -ivh createrepo-x.x.x.noarch.rpm
   ```
2. 创建并启动http服务，切换到web路径如htdocs，新建文件夹myrepo。
3. 复制OS安装镜像EulerOS2.0SP2-aarch64.iso中repos目录下所有rpm包到myrepo中，执行如下命令，查看myreo下会生成一个repodata的文件夹，存放此yum源的数据信息。
   ```
   createrepo myrepo
   ```

至此网络repo源已搭建完毕，使用方式：

1. 在/etc/yum.repos.d/下创建一个repo文件，例如euleros.repo（注意一定要以.repo结尾），在文件中添加以下内容：
   ```
   cat /etc/yum.repos.d/euleros.repo
   [EulerOS]
   name=euleros
   baseurl=http://192.168.1.254/myrepo/   #仓库url，根据repo源所在服务器ip配置
   enabled=1   #开启仓库
   gpgcheck=0    #不检查gpkey
   ```
2. 查看yum源状态，使用如下命令：
   ```
   yum repolist
   ```
   如果有错误，需要检查下，可尝试使用如下命令清理然后重置：
   ```
   yum clean all
   ```


3 系统使用指南

本章主要描述EulerOS V2.0SP2 aarch64版本新增调测功能kbox和kdump的使用，更多的使用命令可以参考《EulerOS V2.0SP2 管理员指南》

3.1 kbox

3.1.1 概述以及约束限制

- 概述

  linux内核比较复杂，且各个模块之间联系紧密，缺少有效的维护工具，进一步增加了维护难度。虽然内核有klogd和syslogd等日志记录系统，但在一些异常情况下，如系统突然重启、内核崩溃、oom（内存溢出）等，无法（或者来不及）记录这些信息，进而无法对这些问题的根因进行定位。

  针对上述情况，为了挽救丢失的内核日志，EulerOS提供了内核黑匣子（Kernel black box）特性。类似在飞行系统中常用的“黑匣子”，在系统异常触发的时候记录重要信息，并通过某种特殊渠道（非易失性存储）把这些信息保存下来，供维护人员用来分析发生异常时的系统状态。

- 约束限制

  本节介绍Kbox使用时会遇到的一些限制约束。

  说明

  禁止使用Kbox特性导出敏感信息。

- 当异常事件并发触发时，Kbox只记录最早发生的异常事件，并转储到存储设备。例如：当系统发生panic、oops或者oom等事件后，如果又有同类panic、oops等事件发生，Kbox不做记录，只打印提示信息到系统日志中 。
- Kbox在记录过程中，如果当前CPU上有中断到来，Kbox执行进程被打断，且在中断中又触发异常事件，此时Kbox只打印提示信息到系统日志中，不记录日志信息（包括之前和中断又触发的异常事件）到存储设备。
- 在使用命令行os_kbox_config导出Kbox记录的转储信息时，系统发生panic、oops或者oom等异常事件，由于并发对锁的获取，Kbox可能会概率性出现转储信息不完整的现象。

- 栈向下溢出场景下，Stack获取到信息全为0。栈刚好等于8k时，Stack获取到信息则为空。
- kbox导出日志功能，只能在第一次启动kbox服务的时候导出，重启kbox服务后，由于kbox记录的标记消失，再次导出日志会提示无日志。
- 虚拟机中kbox使用的普通内存保存日志，系统重启后，可能存在内存重新布局划分，kbox日志丢失。

3.1.2 kbox 使用

使用流程

Kbox的使用流程如图3-1所示。

图 3-1 Kbox 使用简单流程

说明

1. 在EulerOS系统中，kbox默认随内核一起发布，系统启动时kbox服务默认启动，但没有配置存储介质无法存储日志。
2. 在ARM64版本上需要配合bios内存预留一起使用，当bios提供了内存预留功能后，需要修改kbox的配置，kbox才能正常捕获日志。
3. 当hmem被设置成"on"，并且修改了hmem_param参数，必须重启kdump服务，kbox日志才能正常转储。详情请见表3-5

表 3-1 Kbox 步骤说明

编号	流程	说明
1	启动系统	在EulerOS系统中，Kbox默认随内核一起发布，系统启动时Kbox服务会自动开启。
2	修改配置文件	配置产品信息、抓取的异常事件、以及可用的存储方式。Kbox配置文件路径为/etc/kbox/config。

编号	流程	说明
3	重启Kbox服务	修改配置文件后，通过重启Kbox服务，使配置文件生效。
4	记录异常信息	当异常事件发生后，Kbox会根据配置文件，将指定的异常事件信息记录到相应的存储设备中。
5	查看异常信息	具有root权限的用户通过Kbox提供的日志导出命令，将存储设备中的异常信息导出到指定目录的文件中，进行查看。

修改配置文件

本节介绍如何正确配置Kbox工具。

- 前提条件

  在EulerOS系统中，bios已经为kbox预留内存，并且可以正常工作。
- 注意事项

  用户修改完配置文件，重启Kbox后才生效。
- 配置步骤

  以root权限，在shell命令行环境下：

步骤1 打开配置文件。在任一路径下，使用如下命令：

```
vi  /etc/kbox/config
```

配置文件内容如下：

```
## This is kbox's configuration file.
# Depending on the circumstances of each product to complete the configuration.
# * IMPORTANT *: configure parameters correctly according to the prompt message, the parameters must contain quotes, no spaces inside and outside the quotation marks.


##############  Basic information of product  ###############
# Product name, eg: "EulerOS" "CloudEdge" "Pangea" "MSP" "N9000" "SVP" "CGP".
product="EulerOS"

# Product version number: e.g. V100R001C00B251.
version="EulerOS_V200R002C20"

# the frame number: e.g., frame = 1 ~ 9 (generally used for cascade)
frame="1"

# Slot number of the board: e.g., slot = 0 ~ 12
slot="0"

# Location board: e.g., after the front board or card (0-1)
locate="0"

# Board hardware information: e.g., pangea-v2
hard_version=""


#############  Abnormal event scenarios, if the option is empty or invalid characters, the default is no  ###############
# Panic events: mainly explicitly call the kernel panic function or abnormal triggering cause kernel oops.
panic_event="yes"
```

```
# Normal reboot events: the event mainly triggered by user space command reboot, shutdown, halt or
init.
reboot_event="yes"

# Abnormal reboot event: the main exception is triggered by the kernel mode machine restart.
emerge_event="yes"

# Die event: the event mainly triggered by kernel mode illegal pointer accesses, etc.
die_event="yes"

# If call panic after die event process handled.
die_call_panic="no"

# Out of memory event.
oom_event="yes"

# If call panic after oom event process handled.
oom_call_panic="yes"

# R deadlock event,unsupport
rlock_event="no"

################  Storage device type and parameters  ################
# Note: start_pddr must be a physical address, and 4K aligned with hexadecimal.
#       mem_size is a minimum of 8M bytes with hexadecimal.
# Separated by a space between the parameters.

# bios device. Used for pangea product.
bios="off"
# Pangea product' kbox stored in nvram, you can not configure this.
bios_param=""

# pcie device. Used for MSP, SVP guest etc..
pcie="off"
# Parameter for pcie device.
pcie_param="pci_bus_id=00 pci_device_id=00"

# high memory device. Used for EulerOS, CloudEdge, N9000 etc..
hmem="off"
# Parameter for high memory device.
hmem_param="start_paddr=0x0 mem_size=0x0"

# sigma device.
sigma="off"
sigma_param=""

############### Fault reporting interface  ###############
die_notify_func=""


################## Auto export kbox log ##################
# This option can only enable in nonvolatile storage scenes.
# If this option enable, kbox will export log to $export_kbox_log.
# Absolute path are expected and only support for numbers, letters, and underscore.
export_log_path=""
```

步骤2 修改配置文件。

按照当前系统的硬件环境，配置好系统的基本信息、抓取异常的事件以及存储设备。配置主要包含三方面产品信息、异常事件和存储设备三方面。

注意

1. 参数值请严格按照说明进行配置，配置参数必须使用英文引号，引号内不能有空格。如果字符中间需要空格，请以"_"字符代替。配置为yes/no或YES/NO、on/off或ON/OFF的配置项，大小写要统一，且不能包含空格，否则默认处理为no。
2. 异常事件必须至少配置1项，否则Kbox不能加载。必须配置存储设备，否则异常信息不能转储，Kbox重启后信息丢失。
3. 修改配置文件后，请重启Kbox，否则配置不生效。
4. 当前arm版本默认hmem为off，当bios为kbox预留了内存之后，需要用户将改配置项修改为on

具体参数说明和配置指导如 **表 2 产品信息**、**表 3 异常事件**、**表3-4**和 **表3-5**所示：

----结束

表 3-2 产品信息

列序号	参数名称	说明	参数值	是否必配
1	product	当前设备名称（或产品名称）。	用户自定义字符串。	否
2	version	当前系统发布的版本号。	用户自定义字符串。	否
3	frame	当前机架号。	大于0的整数。	否
4	slot	当前单板槽位号。	大于0的整数。	否
5	locate	单板位置。	0或1。	否
6	hard_version	硬件版本。	用户自定义字符串。	否

表 3-3 异常事件

列序号	参数名称	说明	参数值	是否支持	是否必配
1	panic_event	panic事件。	yes/no	是	是，至少配置一种异常事件。
2	reboot_event	安全重启事件。	yes/no	是	
3	emerge_event	异常重启事件。	yes/no	是	
4	die_event	die事件。	yes/no	是	

列序号	参数名称	说明	参数值	是否支持	是否必配
5	oom_event	oom事件。	yes/no	是	
6	rlock_event	r死锁事件	yes/no	否	否
7	oom_call_panic	oom发生后是否调用panic。	yes/no	是	否
8	die_call_panic	die事件后是否调用panic	yes/no	是	否

说明

只有在oom_event项设置为yes时，oom_call_panic配置项才能够生效。oom_call_panic默认设置为yes，如果系统需要在oom时，不让kbox调用panic复位，请将oom_call_panic设置为no，这种情况下kbox不再抓取oom事件日志，只打印内存占用高的top10进程信息。

表 3-4 存储设备配置

列序号	参数名称	说明	参数值	是否必配
1	bios	bios nvram存储。	on/off	是，至少配置其中一种存储方式。
2	pcie	PCIE存储。	on/off	
3	hmem	普通内存存储或者高端内存存储。	on/off	
4	sigma	sigma存储	on/off	

说明

说明：

1. 表3和表4必须结合使用。
2. EulerOS-V2.0SP2 aarch64产品预留非遗失存储后，需要配置hmem="on"。修改配置后，必须重启kbox服务生效。

表 3-5 存储配置参数

列序号	参数名称	说明	参数值	是否必配
1	bios_param	biosnvram驱动参数	驱动自带默认参数,不可以配置	否
2	pcie_param	pcie驱动参数	pci_bus_id=00 pci_device_id=00	是

列序号	参数名称	说明	参数值	是否必配
3	hmem_param	普通内存存储或者高端内存存储参数	该参数有两种取值方式： ● sym_start_paddr_name=reserve_kbox_mem_start sym_mem_size_name=reserve_kbox_mem_len ● start_paddr=0xxxxxx mem_size=0xxxxx。 start_paddr和mem_size为16进制整数，start_paddr为起始物理地址，mem_size为内存大小。 **说明** 当hmem被设置成"on"，并且修改了hmem_param参数时，必须**重启Kdump服务...**，kbox日志才能正常转储。因为hmem类型的存储设备是利用kdump机制转储kbox日志的。	是
4	sigma	sigma存储	暂不支持	否

说明

如果没有非易失性存储设备，可以通过启动参数保留内存，用普通内存代替非易失性存储介质，然后通过kdump将这段内存保存为内存镜像，该内存镜像可通过export_kbox_img_to_txt命令读取。

表 3-6 导出 kbox 日志配置参数

序列号	参数名称	说明	参数值	是否必配
1	export_log_path	在有非遗失存储的服务器上导出上一次复位日志到指定的路径。仅支持数字/字母/下划线组成的绝对路径。如果配置为空，表示不导出。	日志绝对路径	否

3.1.3 kbox 命令参考

命令参考

说明

/usr/sbin/kbox命令为kbox内部使用命令，用来启动kbox，请用户不要使用。

启动 Kbox 服务

```
systemctl  start  kbox
```

重启 Kbox 服务

```
systemctl  restart  kbox
```

查看 Kbox 状态

```
systemctl  status  kbox
```

查看 Kbox 有效配置信息、临时存储区名称及已注册的存储设备

```
kboxstatus
```

说明

如果kbox服务未启动，则不会显示当前临时存储区(region)以及注册的存储设备

导出日志信息

os_kbox_config命令用来导出非易失性存储里面的日志。

说明

如果系统中没有非易失性存储设备，则执行此命令无效。

os_kbox_config -o dev=[存储设备名称]，type=[导出日志类型]，index=[第几条] | -h | -v

详细的参数说明请参见表3-7。

表 3-7 参数说明

参数名称	描述
dev=[存储设备名称]	当前系统中注册的存储设备名称（biosnvram、hmem、pcie），保存异常信息。
type=[导出日志类型]	导出日志类型，与region名称相同，可取oom/console/message/die/panic/rlock。
index=[第几条]	查看第几条信息，取值[1,32]。取1时为最近的一条记录，最多记录32条信息。
-h	查看帮助。
-v	查看kbox版本信息。

解析日志文件

由于ARM通用服务器默认情况下，BIOS没有为没有非易失性存储系统，因此kbox只能配置为在普通内存中存储日志。当系统crash时，kdump会保存这段普通内存为内存镜像文件，通过export_kbox_img_to_txt来解析这个镜像文件，输出文本格式的日志文件。

export_kbox_img_to_txt -i kbox文件

说明

- kbox文件默认使用gzip压缩，解析时请先解压，参考命令gunzip xxx-xxx.img.gz。
- 该命令只适用于通用服务器上kbox日志解析。

3.2 kdump

默认情况下EulerOS aarch64版本的kdump功能和开源一样，本章节主要描述使能了kbox功能之后，对kdump的配置要求以及影响。

当程序或操作人员通过魔术键c或其他原因导致标准内核（即正常运行的内核）崩溃时，Kdump会切换到crash内核，该内核用于捕获标准内核崩溃时的内存信息。

注意

使用默认配置，Kdump在系统崩溃时是不会保存用户数据的。如果用户修改相关配置，可能会保存部分用户数据，请谨慎使用。如需关闭kdump功能，请参考管理kdump服务

使用约束

- 当前仅支持EulerOS操作系统。
- kdump过程受到磁盘写速度、内存使用率、逻辑狗叫时间等因素限制，kdump过程产生的vmcore可能存在不完整的情况。
- kdump时间与内存规格强相关，内存越大，dump时间越长。
- 踩内存、页表破坏的场景，kdump有可能生成的vmcore不完整，crash工具无法解析。

获取 Kdump 信息

说明

系统启动后，Kdump服务默认处于工作状态。

每发生一次内核崩溃，Kdump便会在/crash目录(默认)下生成一个以127.0.0.1-时间戳命名的文件夹。/crash目录下默认只能存放1个文件夹，即上一次内核崩溃的Kdump信息。当再次发生内核crash时，则会先删除之前保存的目录，再保存本次crash的信息。

默认情况下，kdump保存了以下文件：

- kdump_status.log

  记录kdump开始dump时的关键信息日志。
- vmcore-dmesg.txt

  记录从最后一次启动到异常发生时的dmesg日志。
- vmcore

  异常内核的内存镜像。

如果用户配置了转储kbox文件，该目录下会产生kbox日志镜像。

- xxx-xxx.img.gz

  该文件为kbox存储日志文件。

  ```
  gunzip xxx-xxx.img.gz  //解压
  export_kbox_img_to_txt -i xxx-xxx.img  //解析
  ```

  解析后的文件默认存放在/tmp/kbox_log.txt，查看该文件即可。

配置 kdump 参数

注意

- Kdump配置文件用户使用默认配置即可，随意修改可能导致异常。
- 系统启动后的cmdline参数中必须包含保留内核参数crashkernel=1024M，不要修改(用户可通过命令“cat /proc/cmdline”进行查看)。

Kdump各参数在“/etc/kdump.conf ”文件中配置，这里介绍几个重要的参数配置，如表1。

表 3-8 Kdump 参数说明

参数	含义	取值
default	kdump失败后的默认操作	● shell：kdump失败后启动bash。 ● reboot | halt | poweroff：重启或者关机。 当前默认设置为reboot。
path	保存kdump日志绝对路径 **说明** 该路径必须在根（/）分区	默认保存在/var/crash路径下，可设置为/分区下任意绝对路径。
kbox_mem	kbox存储日志的内存段	当前默认没开启，如果需要保存kbox日志，请配置kbox_mem 十六进制起始地址-十六进制长度。
core_collector	保存kdump日志工具	默认为makedumpfile，其中-d项指定过滤内存级别为31，用户无需修改。
kdump_obj	设置kdump保存kbox日志或者vmcore	● vmcore：保存vmcre。 ● kbox：保存kbox。 ● all：默认，保存vmcore和kbox。
keep_old_dumps	历史kdump日志保存个数	● 0和-1：只保存当前日志，默认设置-1。 ● 1 | 2 | 3 | ...：支持设置>0个历史日志。 ● 其他配置不支持。
watchdog_time	设置kdump过程自动复位时间（秒）	默认为3600秒，如果kdump_obj配置项设置为vmcore或者all，该项设置为>600秒。

参数	含义	取值
kdump_post	kdump流程中当日志转储完成后执行的脚本	默认配置请不要修改。
kdump_pre	kdump流程中当日志转储开始前执行的脚本	默认配置请不要修改。
vmcore_size_limit	dump目录使用的磁盘大小限制（MB）	默认是0，表示不限制。 **说明** ● 单位是MB。 ● 设置的最大值不能超过kdump的日志转储目录所在分区的磁盘大小。 例如此参数值设置为1024，则表示日志转储目录的磁盘使用量如果大于1G，那么kdump会删除vmcore文件，避免转储目录所在的分区因为磁盘占用过大而导致系统异常。
ipmi_watchdog_timeout	带外硬件狗超时时间设置	默认为0，表示不设置。 **说明** ● 单位是秒。 ● 最大值为65535。 ● 参数设置的生效条件是必须要硬件支持，即硬件必须支持标准IPMI协议，同时有狗叫超时时间设置功能。 ● 若要打开此功能需要在dracut_args参数中增加--add "ipmi-watchdog"参数。 例如此参数设置为300，表示kdump会在crash kernel启动流程中通过ipmi协议接口设置带外硬件狗的超时时间为300秒。目的是避免因为在dump流程中系统卡死，而造成业务不可恢复。

注意

修改kdump配置文件后，一定要先删除原来boot目录下kdump生成的initrd，

```
# rm  /boot/initramfs-*aarch64kdump.img
```

然后重启kdump服务。

```
# systemctl restart  kdump
```


管理 kdump 服务

以下命令用于启动、停止或查询Kdump服务：

- 停止Kdump服务

```
#systemctl  stop  kdump
```

- 启动Kdump服务

```
#systemctl  start  kdump
```

- 重启Kdump服务

```
#systemctl  restart  kdump
```

- 查询Kdump服务

```
#systemctl  status  kdump
```

- 永久关闭kdump服务

打开启动参数配置文件/boot/EFI/grub2/grub.cfg，删除系统启动参数crashkernel=xxxx，重启操作系统。

vmcore 文件解析

用户可使用EulerOS提供的crash工具，对Kdump保存的vmcore文件进行分析，进而定位问题。工具使用流程如下：

1. 在VMP发布包的EulerOS-V2.0SP2-aarch64-dvd.iso中获取crash-7.*.aarch64.rpm和kernel-debuginfo-4.*.aarch64.rpm。
2. 将rpm解开后，提取crash命令和vmlinux。
3. 使用crash命令开始解析。

   crash vmlinux vmcore

   说明

   - 获取iso的版本必须和产生vmcore的系统版本完全一致。
   - 请在产生vmcore的环境或者相同的环境上解析vmcore。

4 日志防爆特性

4.1 功能简介

日志防爆特性，是指系统会通过日志切割工具logrotate ，每隔一小时对/etc/logrotate.d目录下的日志配置文件中配置的日志文件进行切割，以防止磁盘空间被占满。

4.2 使用场景

系统使用过程中，日志持续增长会使存储日志的磁盘空间被占满，从而导致日志无法继续打印，或者影响到系统部分业务功能（进程异常或一些操作无法进行）；另外如果系统出现了问题，因为缺少日志也会影响问题定位效率，甚至导致问题无法定位。

日志防爆特性，主要是针对大小会持续增长的日志文件，对其添加切割配置，确保存储日志文件磁盘空间不被占满，从而增强系统的稳定性与可靠性。

4.3 使用方法

在/etc/logrotate.d目录下，对需要切割日志文件的进行配置。

文件配置方法

在/etc/logrotate.d目录下，添加配置文件（新增配置文件的权限建议不大于640），一个配置文件中可同时配置多个需要切割的日志。

如：/etc/logrotate.d/example文件中配置。

```
/var/log/logexample
/var/log/logexample1
{
maxage 365
rotate 30
notifempty
compress
```



```
copytruncate
missingok
size +4096k
}
```

该配置会对/var/log/logexample、/var/log/logexample1进行切割。

- maxage 365：只存储最近365天的切割出来的日志文件，超过365天则删除。
- rotate 30：指定日志文件删除之前切割的次数，此处保留30个备份。
- notifempty：表示日志为空则不处理。
- compress：通过gzip压缩转储以后的日志。
- copytruncate：用于还在打开中的日志文件，把当前日志备份并截断。
- missingok：如果日志文件丢失，不报错继续执行下一个。
- size +4096k：表示日志超过4096k大小才分割，size默认单位是KB，可使用k、M和G来指定KB、MB和GB。

配置项说明

配置项说明如**表1**。

表 4-1 配置项说明

配置项	功能
compress	通过gzip压缩转储以后的日志。
missingok	找不到日志时，跳过。
nomissingok	找不到日志时，报错。
nocompress	不需要压缩时，用这个参数。
copytruncate	用于还在打开中的日志文件，把当前日志备份并截断。
nocopytruncate	备份日志文件但是不截断。
create mode owner group	转储文件，使用指定的文件模式创建新的日志文件。
nocreate	不建立新的日志文件。
prerotate/endscript	在转储以前需要执行的命令可以放入这个对，这两个关键字必须单独成行。
postrotate/endscript	在转储以后需要执行的命令可以放入这个对，这两个关键字必须单独成行。
daily	指定转储周期为每天。
weekly	指定转储周期为每周。
monthly	指定转储周期为每月。
rotate count	指定日志文件删除之前转储的次数，0指没有备份，5指保留5个备份。

配置项	功能
size	当日志文件到达指定的大小时才转储，size可以指定bytes（缺省）以及KB、MB或者GB。

配置项中“转储”的含义：用来把旧的日志文件删除，并创建新的日志文件。

说明

1. nocreate与配置文件中的copytruncate是互斥的，不能同时配置，否则nocreate不生效。
2. create mode owner group（例如：create 0600 root root）与配置文件中的copytruncate是互斥的，否则create配置不生效。
3. 时间频度（daily，weekly，monthly，yearly）和日志大小（size）这两项参数同时配置的时候，会以日志大小为条件，达到一定大小就会切割。

5 热补丁

内核热补丁是一种在不重启操作系统或者插拔内核模块的前提下，修复内核和内核模块中缺陷的一种工具，可以在不中断业务的情况下解决问题。本章介绍内核热补丁的原理、主要功能、使用场景、使用方法和约束限制。

5.1 概述

本节介绍内核热补丁的一些基础概念，让您了解内核热补丁的作用。

背景描述

内核热补丁是用发布的补丁文件修改内核或者内核模块中的缺陷，它可以不影响业务的情况下在线解决大部分内核或者内核模块的问题，增强公司产品竞争力，从而有效提升公司形象及客户满意度。

补丁管理服务具备以下优点：

- 缩短版本发布的时间，提高市场的响应速度。
- 不影响现网的业务，提高客户的满意度。

- 从版本验证改为补丁验证，缩短测试时间。

内核热补丁简介

内核热补丁对编译生成的二进制文件进行操作生成补丁文件，提取缺陷函数的二进制代码，以内核模块形式插入到系统中，检测系统中所有进程是否在调用所修改的函数，通过修改函数的代码段实现函数跳转，进而实现修改内核和模块中函数缺陷。

通过在缺陷函数中插入的钩子，让缺陷函数在被调用时跳转到新函数的地址中，使热补丁生效。

基本概念

- 内核热补丁：linux内核的热补丁，主要是Euler部门用，产品部门可以不关注。
- 模块热补丁：产品模块驱动的热补丁。
- 补丁： 一般一个修改点，做成一个补丁。

工具说明

make_hotpatch：热补丁制作工具。

livepatch：加载补丁、激活补丁、补丁查询、删除补丁、回退补丁工具。

5.2 约束限制

用户在使用内核热补丁功能时，请注意以下约束限制：

- 不支持
  - 不支持对初始化函数打补丁（初始化函数只执行一次，补丁函数执行不到）。
  - 不支持汇编文件打补丁。
  - 不支持对死循环、不退出函数打补丁（旧函数不退出调用栈，没有机会调用新函数）。
  - 不支持对有前缀notrace修饰的函数打补丁（产品一般不涉及）。
  - 不支持修改数据结构成员（热补丁原理是做函数替换）。
  - 不允许删除函数内部静态局部变量。
  - 不支持新增同名静态局部变量。
  - 不支持对头文件进行修改。
  - 不支持对非C语言编写的代码程序打热补丁。
  - 不允许对NMI中断的处理函数打补丁（stop machine无法stop住NMI中断处理流程，补丁无法保证对该类函数打补丁的一致性和安全性）。
  - 不支持修改全局变量初始值。
  - 不支持删除函数。
  - 不支持对修改前后内敛情况发生变化的函数打补丁。
  - 不支持对包含以下弱符号的函数打补丁。

    "kallsyms_addresses"

    "kallsyms_num_syms"

    "kallsyms_names"

"kallsyms_markers"

"kallsyms_token_table"

"kallsyms_token_index"

- 支持
  - 支持新增全局数据结构。
  - 支持对内联函数打补丁。
  - 支持对静态函数打补丁。
  - 支持修改多个文件的多个函数。
  - 支持新增全局变量。

  制作热补丁时，用户必须保证编译环境包和基线代码所编译出的二进制与运行环境中一致。
- **补丁ID长度限制**

  由于内核模块名称长度（包含\0）不得超过56个字节，否则会出现补丁加载上后无法卸载的问题，而热补丁生成的ko最终的命名格式为klp_补丁ID.ko，因此制作热补丁时的补丁ID要保证，最终的补丁名“klp_补丁ID”要小于56个字节。
- **Makefile的约束限制**

  xxx-y目前不支持./xxx.o

```
#ifeq ($(stub),1)^M
ifeq ($(suse11sp1),1)^M
EXTRA_CFLAGS += -DDRV_STUB^M
endif^M
^M
drvmml-y := ./drv_mml.o^M
^M
drvmml-y += ./drv_mmlcommon.o ^M
^M
drvmml-y += ./drv_mmlapi.o ^M
#drvmml-y += ./drv_mmlautotest.o^M
#drvmml-y += ./drv_mmlft.o^M
#ifeq ($(fc),1)^M
drvmml-y += ./drv_mmlfc.o^M
#endif^M
^M
^M
```

  Makefile中不可显示定义CROSS_COMPILE和CC这两个变量，如果有需要在制作热补丁时注释掉。

```
export ARCH=x86_64
PREFIX=
KERNELDIR=/lib/modules/3.10.0-327.22.2.26.x86_64/build
KERNELSOURCEDIR=/lib/modules/3.10.0-327.22.2.26.x86_64/source
KERNELVERSION=3.10.0-327.22.2.26.x86_64
#export CROSS_COMPILE=
#export CC=gcc
export LD=ld
export OBJCOPY=objcopy
export AR=ar
KVM_VERSION=kvm-devel
```

  编译过程中，不可删除中间生成的二进制文件，如果有也请注释掉。


```
default:
        @rm -f sal_version.o
        $(MAKE) -C $(KDIR) M=$(PWD)
        #cat Module.symvers >> $(WORK_DIR)/source/modules/Module.symvers
clean:
        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) clean
        @rm -f ?odule*.*
```

部分模块在编译时会动态生成头文件，请在制作补丁时保持头文件不变。

```
default:
        @echo \#define DMI_DATE \"`date +%Y/%m/%d" "%H:%M:%S`\" > $(WORK_DIR)/dmi/include/dmi_compile.h
```

部分模块一次编译会生成多个ko，制作热补丁，需要修改Makefile，保证每次只编译出一个ko

-obj-m := kvm.o kvm-intel.o kvm-amd.o

+obj-m := kvm.o

-obj-m := kvm.o kvm-intel.o kvm-amd.o

+obj-m := kvm-intel.o

5.3 使用场景

内核热补丁最大的特点是在不重启系统和不中断业务前提下修改内核中的函数，达到动态替换内核函数的目的，最主要的应用场景有两个：

1. 修复内核和模块的缺陷函数。

   内核热补丁能够动态的修复内核和模块的缺陷函数。在开发人员发现问题，或者操作系统发现安全漏洞需要修复时，可以通过将缺陷函数或者安全补丁制作成内核热补丁打入系统中的方法，在不需要重启系统或者插拔模块、不中断业务的前提下修复缺陷。

2. 开发过程中的调试和测试手段。

   内核热补丁也适用于在开发过程中进行调试和测试。比如在模块或者内核的开发过程中，如果需要通过在某一个函数中添加打印信息，或者为函数中某一个变量赋予特定的值，可以通过内核热补丁的形式实现，而不需要重新编译内核、安装然后重启的操作。

5.4 准备软硬件环境

本章介绍内核热补丁使用的软硬件环境。

硬件要求

- 编译环境：即热补丁制作环境，ARM64热补丁通过交叉编译环境制作，需要能够正常运行X86_64位的Linux操作系统，制作补丁需要编译内核和模块，如果有多CPU支持，速度更快。
- 运行环境：即要安装热补丁的环境，必须是ARM64架构。

软件要求

- 编译环境：需要安装Linux环境，并搭建下载与运行环境相匹配的编译环境压缩包Euler_compile_env_cross.tar.gz。

- 在编译环境中做补丁的基线源代码、配置文件、Makefile编译出的二进制要和运行环境中的二进制完全一致。
- 运行环境的EulerOS系统中已经带有livepatch命令。

5.5 管理补丁(运行环境)

本章介绍如何对补丁进行加载、激活、查询、回退等操作。

5.5.1 加载补丁

本章介绍热补丁的加载过程。

使用场景

补丁已从编译环境上传到运行环境，在运行环境使用livapatch工具将补丁文件加载到内核中。

注意事项

- 若加载的补丁P2依赖另一个补丁P1，在加载补丁P2之前需先激活补丁P1。
- 若对模块打补丁，在加载此补丁前需先加载此模块。

操作过程

步骤1 假设已将补丁文件（如klp_test.tar.gz）上传到运行环境上，存放在/root/目录下。

步骤2 可在任意目录下执行livepatch命令来加载补丁。可使用相对路径。

```
livepatch -l /root/klp_test.tar.gz
```

显示如下：

```
install patch /root/klp_test.tar.gz success
```

----结束

验证结果

加载补丁成功后，返回补丁加载成功信息。通过“查询补丁信息”查询补丁当前的状态：

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Deactive
Changes:
        cmdline_proc_show
Denpendency: vmlinux
------------------------------------------------------------
[root@EulerOS ~]#
```

5.5.2 激活补丁

本章介绍热补丁的激活。

使用场景

对已经加载完成的补丁进行操作，设置补丁的状态，使之为激活。

注意事项

激活补丁命令中使用的补丁名必须是通过“查询补丁信息”查询出的补丁名。

操作过程

可在任意目录下执行livepatch命令激活补丁。

步骤1 查询所有补丁文件信息：

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Deactive
Changes:
        cmdline_proc_show
Denpendency: vmlinux
------------------------------------------------------------
[root@EulerOS ~]#
```

步骤2 激活补丁：

```
[root@EulerOS ~]# livepatch -a test
active patch klp_test success
```

----结束

验证结果

激活补丁成功后，返回补丁激活成功信息，通过查询命令可以查询到补丁状态信息为Active。

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Active
Changes:
        cmdline_proc_show
Denpendency: vmlinux
------------------------------------------------------------
[root@EulerOS ~]#
```

5.5.3 查询补丁

本章介绍如何查询补丁状态。

使用场景

对加载到系统中的补丁进行状态查询。

注意事项

- 如果是查询单个补丁，需要将补丁的id直接跟着-q参数后面。
- 如果是对内联函数打补丁，在补丁查询的时候只会显示调用该内联函数的缺陷函数名称，而不是内联函数本身。

操作过程

步骤1 查询所有补丁文件信息：

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Active
Changes:
        cmdline_proc_show
Denpendency: vmlinux
-------------------------------------------------------------
[root@EulerOS ~]#
```

步骤2 查询单个补丁文件信息：

```
[root@EulerOS ~]# livepatch -q test
Patch Name: test
Patch State: Active
Changes:
        cmdline_proc_show
Denpendency: vmlinux
-------------------------------------------------------------
```

步骤3 回退补丁：

```
[root@EulerOS ~]# livepatch -d test
deactive patch klp_test success
[root@EulerOS ~]#
```

----结束

验证结果

回退补丁成功后，返回补丁回退成功信息，通过查询命令可以查询到补丁状态信息发生改变。

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Deactive
Changes:
        cmdline_proc_show
Denpendency: vmlinux
-------------------------------------------------------------
[root@EulerOS ~]#
```

调用被打补丁的函数可以进一步验证内核热补丁是否成功去激活，即回退补丁后被打补丁函数是否还原。具体步骤略。

5.5.4 回退补丁

本章介绍补丁的回退。

使用场景

补丁处于激活状态，要使其失效。

注意事项

补丁已加载并处于激活状态。

补丁如果被其他补丁依赖则无法回退，必须先回退依赖补丁，再回退该补丁。

操作过程

可在任意目录下执行livepatch命令回退补丁。

步骤1 查询所有补丁文件信息：

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Deactive
Changes:
        cmdline_proc_show
Denpendency: vmlinux
-------------------------------------------------------------
[root@EulerOS ~]#
```

步骤2 回退补丁：

```
[root@EulerOS ~]# livepatch -d test
deactive patch klp_test success
```

----结束

验证结果

回退补丁成功后，返回补丁回退成功信息，通过查询命令可以查询到补丁状态信息发生改变。

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Deactive
Changes:
?cmdline_proc_show
Denpendency: vmlinux
-------------------------------------------------------------
[root@EulerOS ~]#
```

5.5.5 卸载补丁

本章介绍卸载热补丁。

使用场景

在运行环境使用livepatch工具将补丁文件从内核补丁区移除。

注意事项

- 卸载补丁命令中的补丁名必须是通过“查询补丁信息”查询出的补丁名。
- 若有其他已加载的或处于激活状态的补丁依赖于补丁P1，不允许卸载补丁P1。

操作过程

步骤1 查询所有补丁信息：

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Deactive
Changes:
?cmdline_proc_show
Denpendency: vmlinux
-------------------------------------------------------------
[root@EulerOS ~]#
```


步骤2 回退补丁：

```
[root@EulerOS ~]# livepatch -d test
deactive patch klp_test success
```

步骤3 卸载补丁：

```
[root@EulerOS ~]# livepatch -r test
remove patch klp_test success
```

----结束

验证结果

卸载补丁成功后，返回补丁卸载成功信息，通过查询命令查询不到补丁的信息（补丁已不在内核补丁区）。

5.6 热补丁制作（编译环境）

5.6.1 制作模块热补丁

内核模块指用户开发的驱动。由于热补丁使用是有一定限制和约束的，所以制作前，请先阅读 **5.2 约束限制** 一节。

前提条件

- EulerOS编译环境中已经安装补丁制作工具和补丁制作目录。
- 修改前的模块需要能够编译通过。
- 修改后的模块需要能够编译通过。

注意事项

- 补丁制作的源码、源码在编译环境中的路径、编译环境等需要和运行环境运行的内核、模块完全一致。
- 如果之前已经做过补丁，现在需要对同一个文件的相同或其他函数再做补丁，需要再将之前的改动代码和本次将要修改的代码都包含在唯一后缀的文件中。

制作步骤

步骤1 部署编译环境（这一步通常由CI工程师搭建一次即可）。

EulerOS提供了对应的编译环境包（ARM64架构交叉编译环境Euler_compile_env_cross.tar.gz）。用户可从VMP或CI工程师处获取对应的编译环境包等工具，本章节以ARM64编译环境包为例。

```
tar xf Euler_compile_env_cross.tar.gz
```

步骤2 拷贝或挂载产品代码到编译环境目录下，例如Euler_compile_env_cross/code/。

步骤3 chroot到编译环境中进行工作，开始制作热补丁（以下步骤以testmod模块制作热补丁作为示例，其他模块参照）。

```
chroot Euler_compile_env_cross/ /bin/bash --login
```

步骤4 进入源码目录，拷贝源文件为唯一后缀的文件（这里的后缀即为**步骤5**中的-d参数的值），并修改文件中需要打补丁的函数。


```
cd /code/testmod/
cp testmod_drv.c testmod_drv.c.new
vi testmod_drv.c.new
```

步骤5 进入补丁制作目录，制作热补丁。

```
cd /opt/patch_workspace/
./make_hotpatch -d .new -i test -j 64 -m /code/testmod
```

- -d 后面跟上前面的唯一后缀名。
- -i 后跟补丁ID，可包括字母和数字。
- -j 后跟make编译时的线程数。
- -m 后跟模块源码绝对路径。

步骤6 补丁制作成功后在编译环境/opt/patch_workspace/hotpatch/目录下。

步骤7 将补丁klp_test.tar.gz上传到运行环境上，用livepatch命令加载并激活热补丁。

```
livepatch -l klp_test.tar.gz
livepatch -a test
```

以上两步执行成功后即可验证业务逻辑是否正常，bug是否成功修复。

----结束

特殊处理

- 热补丁编译模块时仅执行make操作，当模块带参数编译时，例如make DEBUG=1。用户需要将DEBUG=1写入一个flag文本，并在制作热补丁时增加--extra_flags参数指定flag文本路径。示例如下：
  ```
  ./make_hotpatch -d .new -i test -j 64 -m /code/testmod --extra_flags /opt/patch_workspace/flags
  ```
- 当模块的Makefile不在源码目录下时，比如Makefile在testmod/build目录，模块源码在testmod/src目录，制作补丁时需要增加-f参数指定Makefile路径。示例如下：
  ```
  ./make_hotpatch -d .new -i test -j 64 -m /code/testmod/src -f /code/testmod/build/Makefile
  ```

5.6.2 制作内核热补丁

由于热补丁使用是有一定限制和约束的，所以制作前，请先阅读“**约束限制**”一节。

前提条件

- EulerOS编译环境中已经安装补丁制作工具和补丁制作目录。
- 修改前的模块需要能够编译通过。
- 修改后的模块需要能够编译通过。

注意事项

- 补丁制作的源码、源码在编译环境中的路径、编译环境等需要和运行环境运行的内核、模块完全一致。
- 如果之前已经做过补丁，现在需要对同一个文件的相同或其他函数再做补丁，需要再将之前的改动代码和本次将要修改的代码都包含在唯一后缀的文件中。

制作步骤

步骤1 部署编译环境（这一步通常由CI工程师搭建一次即可）。

EulerOS提供了对应的编译环境包（ARM64架构交叉编译环境Euler_compile_env_cross.tar.gz）。用户可从VMP或CI工程师处获取对应的编译环境包等工具，本章节以ARM64编译环境包为例。

```
tar xf Euler_compile_env_cross.tar.gz
```

步骤2 chroot到编译环境中进行工作，开始制作热补丁（以下步骤以修改内核源码fs/proc/cmdline.c为示例）。

```
chroot Euler_compile_env_cross/ /bin/bash --login
cd /opt/patch_workspace/
./make_hotpatch
```

步骤3 上面这一步执行完后，会在/opt/patch_workspace/目录下生成一个kernel-source的软链接，指向内核源码目录。进入内核源码目录，拷贝源文件为唯一后缀的文件（这里的后缀即为步骤4中的-d参数的值），并修改文件中需要打补丁的函数。

```
cd /opt/patch_workspace/kenrel-source
cd fs/proc/
cp cmdline.c cmdline.c.new
vim cmdline.c.new
```

步骤4 进入补丁制作目录，制作热补丁。

```
cd /opt/patch_workspace/
./make_hotpatch -d .new -i test -j 64
```

- -d 后面跟上前面的唯一后缀名。
- -i 后跟补丁ID，可包括字母和数字。
- -j 后跟make编译时的线程数。

步骤5 补丁制作成功后在编译环境/opt/patch_workspace/hotpatch/目录下。

步骤6 将补丁klp_test.tar.gz上传到运行环境上，用livepatch命令加载并激活热补丁。

```
livepatch -l klp_test.tar.gz
livepatch -a test
```

以上两步执行成功后即可验证业务逻辑是否正常，bug是否成功修复。

----结束

5.7 命令参考

make_hotpatch

- **功能说明**

  通过此命令创建一个热补丁，新创建的补丁将会存放在补丁工作目录的hopatch下。

  说明

  该命令不支持并发，即不能同时执行多个该命令。

- **命令原型**

  ```
  make_hotpatch -d patch_diffext -i patch_id  -j CPU_number -m module_src -f makefile_path --- extra_flags [-h]
  ```

- **参数说明**

表 5-1 参数说明

参数名称	描述	可选or必选
-d,--diffext	用来指定修改文件的后缀名，比如“.patch”。	必选
-j,--jobs	指定补丁制作过程中使用的CPU个数，CPU越多速度越快。	可选
-i,--id	指定补丁的id，只能用数字或者字符表示id，并不能超过20个字符。	必选
-m,--modulesrc	用来指定用户模块源码所在的路径，请使用绝对路径，且保证该路径与之前编译模块（编译对应系统运行中的模块）的路径相同。如果是内核或者内核模块补丁，可以省略这个参数。	● 当制作用户模块补丁时，该参数必选。 ● 当制作内核补丁时，该参数不需要。
-f,--makefile	用来指定模块编译时Makefile的路径和文件名，使用绝对路径，且这个Makefile中需要指定obj-m编译的目标 **说明** 该参数只适用于制作模块补丁，并且只有在模块的源码和Makefile文件不在同一目录时必配，通过该参数指定makefile的路径。	● 当制作用户模块补丁，且模块的源码和Makefile文件不在同一目录时，该参数必选。 ● 当制作内核补丁时，该参数不需要。
--extra_flags	用来指定模块编译Makefile中的全局变量，配置文件flags定义了用户模块Makefile引用的全局变量，此参数为可选参数。 **说明** 如果用户模块的Makefile引用了非本Makefile定义的全局变量，在制作用户模块补丁时，可能导致制作补丁失败。使用“--extra_flags”可以将Makefile中非本Makefile定义的全局变量，定义到指定配置文件中，避免上述情况的发生。	● 当制作用户模块补丁时，该参数可选。 ● 当制作内核补丁时，该参数不需要。
-h,--help	获取帮助信息。	可选。

livepatch

- **功能说明**

  通过此命令管理一个补丁。

- **命令原型**

```
livepatch -l/--load -r/--remove -a/--activate -d/--deactivate <patch>
                -q[patch]/--query[=patch]
                -h/--help -v/--version
```

- **参数说明**

表 5-2 参数说明

参数名称	描述
-l/--load	加载补丁。
-r/--remove	卸载一个补丁。
-a/--activate	激活一个补丁。
-d/--deactivate	回退一个补丁。
-q[patch]/ --query[=patch]	查询所有补丁或者查询指定补丁状态。
-h/--help	帮助信息。
-v/--version	查询livepatch的版本号。

说明

具体的使用实例可以参考**管理补丁**模块。

5.8 常见错误处理

本章介绍在进行内核热补丁操作时遇到的一些常见问题及处理方法。

5.8.1 补丁制作失败，no changed objects found

问题描述

补丁制作完成，没有报错，但没有生成补丁文件，打印出信息no changed objects found.。

问题原因

模块Makefile中显示定义了CC或CROSS_COMPILE。

处理方法

请检查你的Makefile中（包括其include的其他Makefile）是不是显示指定了CC或CROSS_COMPILE这两个变量，如果有，请将其注释后再重新制作热补丁。

图 5-1 某产品公共 Makefile：plat_pub.mak

```
# for arm
ifeq ($(ARM), 1)
#       CC=aarch64-linux-gnu-gcc
        RM=@rm
        CP=@cp
#       GCC=aarch64-linux-gnu-gcc
        AR=aarch64-linux-gnu-ar
        USR_INCLUDE=/arm/cross_compile/install/sysroot/usr/i
        ROOT_PATH=/arm/cross_compile/install/sysroot
else
# for X86
        GCC=@gcc
```

5.8.2 补丁制作失败，can't find parent xxx for xxx

问题描述

模块热补丁制作失败。

问题原因

模块Makefile问题，编译末尾存在mv或rm操作。

处理方法

请检查你的Makefile，在编译完成后是否有mv或rm操作将生成的二进制移走或删除，如果有，请将其注释后再重新制作热补丁。

图 5-2 某产品模块 Makefile

```
default:
        $(MAKE) -j $(CPUNUM) -C $(KERNELDIR) M=$(PWD) modules
#       -mkdir objs misc
#       -mv ../../../cache/ramcache/*/*.o ../../../cache/ramcache/*/.*.cmd */*.o *.ko objs
#       -mv .tmp_versions .*.cmd *.o *.symvers *.order *.mod.c */.*.cmd  misc
clean:
```

5.8.3 补丁制作失败，reference to static local variable xxx in xxx was removed

问题描述

补丁制作失败，提示reference to static local variable xxx in xxx was removed。

问题原因

热补丁不支持删除静态局部变量。

处理方法

热补丁不支持删除静态局部变量，请检查你的源码改动，找到被删除的静态局部变量，通过在修改处恢复该静态局部变量进行规避，然后再重新制作热补丁。

示例：找到根据错误提示的函数，这里是handleWriteChunkWriteSuccessEvent，开发在该函数中删了一个内联函数的引用，引用了其他函数，原来的内联函数中存在日志限频打印宏（可能层层内联，需要一层一层往下找）PRINT_LIMINT_PERIOD，这个宏中存在静态局部变量，即报错中提示的ulLast。

图 5-3 示例

```
#define PRINT_LIMIT_PERIOD( level, logid, interval, burst, can) \
do {                                                            \
    /*使用静态变量保存最大配额，减少运算*/                      \
    static OSP_U64 ulMaxToks = (burst) * (interval);            \
    static OSP_U64 ulToks = (burst) * (interval);               \
    static OSP_U32 uiMissed = 0;                                \
    static OSP_U64 ulLast = 0;                                  \
    OSP_U64 ulNow = jiffies;                                    \
                                                                \
```

规避方法

在改动处，增加类似以下语句，注意静态局部变量的初始值要和删除之前的保持一致，如果存在多个静态局部变量，也作相同处理。

```
do{
static u64 ulLast = 0;
if(!jiffies)
printk("%lx\n",ulLast++);
}while(0);
```

5.8.4 补丁制作失败，invalid ancestor xxx for xxx

问题描述

补丁制作失败，提示类似invalid ancestor xxx for xxx。

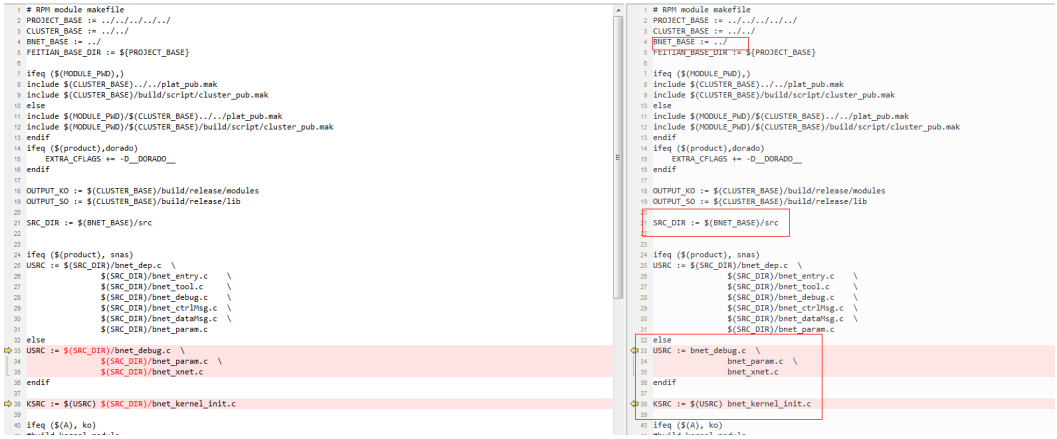
问题原因

模块Makefile问题，使用了相对路径。

处理方法

请修改你的Makefile，在编译中尽量使用绝对路径，避免使用相对路径。示例如图 1：

图 5-4 某产品模块 Makefile



5.8.5 补丁加载失败，Invalid parameters

问题描述

补丁加载失败，提示Invalid parameters。

问题原因

基线代码正确，运行环境是DEBUG包，但编译时flag文件中缺少DEBUG=1之类的宏，导致编译出的补丁是release版本的热补丁。

处理方法

请确保release版本的热补丁到release包的运行环境验证，debug版本的热补丁到debug包的运行环境验证。

5.8.6 补丁加载失败，Invalid module format

问题描述

补丁加载失败，messages日志“exports duplicate symbol xxx”。

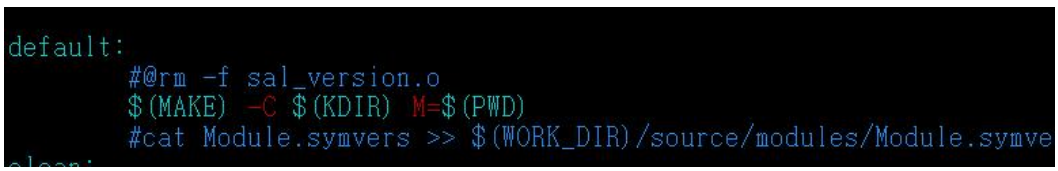
问题原因

模块Makefile问题，每次编译前会删除二进制导致部分符号被热补丁工具识别为新增符号而重复导出。

处理方法

请检查你的Makefile，在MAKE之前是否有删除二进制的操作，有的话请将其注释掉后重新制作热补丁。

图 5-5 示例



5.8.7 补丁激活失败，编译器优化导致未改动函数被做到补丁中

问题描述

热补丁激活失败，日志显示函数正在被调用，但日志显示的函数并没有修改。

问题原因

编译器优化行为会导致出现没有发生修改的函数在前后两次编译中汇编发生变化，被补丁工具做到热补丁中。

处理方法

在修改的文件中对应的函数xxx后增加以下两句：

```
#include "/usr/share/kpatch/patch/kpatch-macros.h"
KPATCH_IGNORE_FUNCTION(xxx)
```

“xxx”为函数名称。

说明

风险知会：这种方法会导致xxx函数即使发生改动也不做到补丁中，因此必须在确保是由于编译器优化行为导致的函数改动（只是寄存器号发生变化，函数整体汇编逻辑没变）的场景才可使用，否则会导致模块逻辑功能异常。使用时请评估清楚，风险由使用者自行承担。

5.8.8 补丁激活失败，改动的函数频繁调用导致激活失败

问题描述

补丁激活失败，日志显示改动函数正在被调用。

问题原因

改动函数属于频繁调用函数，容易出现激活失败。

处理方法

在修改的文件中对应的函数xxx后增加以下两句：

```
#include "/usr/share/kpatch/patch/kpatch-macros.h"
KPATCH_FORCE_UNSAFE(xxx)
```

“xxx”为函数名称。

说明

风险知会：这种方法会导致xxx函数在补丁激活时不会进行调用栈检查，存在前后一致性风险，即补丁生效后旧函数和新函数可能存在同时运行的场景，请用户充分评估清楚这种场景对模块逻辑的风险，慎重使用，风险由使用者自行承担。另外，由于livepatch不支持强制去激活热补丁，频繁调用函数的热补丁还存在补丁回退失败的风险。

6 系统版本升级

目前repo源制作和配置由用户来设置，是否要更新也由用户决策。repo源建议搭建在LINUX PXE服务区上。

提示有RPM更新，用户可以用以下命令更新所有软件包：

```
yum update -y
```

或者

```
yum upgrade -y
```


7 FAQ

7.1 查询 EulerOS 版本标识

EulerOS V2.0SP2 版本标识的配置文件是 /etc/euleros-release，此文件中包含了EulerOS版本信息，使用如下命令：

```
[root@localhost ~]# cat /etc/euleros-release
EulerOS release 2.0 (SP2)              #EulerOS版本
```

查询内核版本信息，使用如下命令：

```
[root@localhost ~]# uname -a
Linux localhost.localdomain 3.10.0-327.44.58.19.x86_64 #1 SMP Wed Feb 22 17:25:10 UTC 2017 x86_64
x86_64 x86_64 GNU/Linux
```

说明

EulerOS V2.0SP2系统中存在一个/etc/euleros-lastest文件，该文件是开发人员内部使用的文件，用户不能依赖该文件来获取EulerOS的版本以及内核版本信息。

7.2 网络配置约束限制

EulerOS通过 NetworkManager 服务进行网络管理。如果开启了该服务，则必须使用 nmcli 命令或通过修改配置文件来配置网络（如ip、路由等），而不能使用 ip/ifconfig/route 等命令来配置。

说明

在开启NetworkManager 服务的场景下，使用 ip/ifconfig/route 等命令配置网络，则一段时间后配置会被NetworkManager覆盖，导致 ip/ifconfig/route 配置不生效。

查看NetworkManager 服务是否开启：

```
systemctl status NetworkManager
```

说明

nmcli命令使用参考“nmcli --help”或者“man nmcli”。

如果要使用 ip/ifconfig/route 等命令来管理网络，请先关闭 NetworkManager 服务，使用如下命令：

```
systemctl stop NetworkManager
```

说明

如果只是关闭NetworkManager，重启后还是会被其他依赖服务拉起来。所以针对不同业务使用场景，若需要完全屏蔽NetworkManager，仅建议通过卸载该软件包方式解决，具体卸载命令如下：

```
rpm -e --nodeps NetworkManager
```

7.3 如何设置防火墙提高容器网络的性能

EulerOS 的防火墙基于 CentOS 的 firewalld 进行构建，并跟随大多数操作系统，如 Windows/Redhat/CentOS/Ubuntu等的策略，默认打开防火墙。

firewalld 打开时，因为会加载一系列iptable/ebtable 的内核模块及默认规则，导致容器网络性能下降 20% 左右（测试环境不同对下降幅度有影响）。相同测试条件下，关闭防火墙性能略优于 SuSE12，打开后劣于 SuSE12。

说明

关闭防火墙时，所有iptable/ebtable 中规则及模块本身都会被删除卸载，即默认的和用户定义的防护规则都将失效。

防护规则及功能

当前 firewalld 支持的防护规则及功能主要有：

1. Zone

   定义安全级别，不同安全级别的接口/连接/地址放入不同的 zone 进行防护，预定义有 drop/block/public/external/dmz/work/home/internal/trusted 9 类。如家庭网络接口可以放入home 甚至 trusted zone，而公共 wifi 则适合 public 级别。

2. Service

   虽然 Service 最终对应的是端口和地址，但 firewalld 将其抽象出来，使得用户可以便捷地使能/关闭对一些服务的防护。例如：

   ```
   firewall-cmd --permanent --zone=public --add-service=http
   ```


3. IPSet

   即 IP 和 MAC 的绑定设置。

4. ICMP Type

   鉴于 ICMP 在 IP 网络里的特殊性，firewalld 将其提出来进行防护。

5. Direct Interface

   用户可以在 firewalld 里直接定义 iptable/ebtable 的规则。

重要命令

1. 查看防火墙状态

```
[root@localhost firewalld]# systemctl status firewalld
firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled)
   Active: active (running) since Wed 2017-03-22 06:17:32 EDT; 1h 25min ago
 Main PID: 28597 (firewalld)
   CGroup: /system.slice/firewalld.service
           └─28597 /usr/bin/python -Es /usr/sbin/firewalld --nofork --nopid
```

2. 开启防火墙

```
[root@localhost firewalld]# systemctl start firewalld
```

   开启后查看防火墙状态

```
[root@localhost firewalld]# systemctl status firewalld
firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled)
   Active: active (running) since Wed 2017-03-22 10:36:38 EDT; 2s ago
 Main PID: 1500 (firewalld)
   CGroup: /system.slice/firewalld.service
           └─1500 /usr/bin/python -Es /usr/sbin/firewalld --nofork --nopid
```

3. 关闭防火墙

```
[root@localhost firewalld]# systemctl stop firewalld
```

   关闭后查看防火墙状态

```
[root@localhost firewalld]# systemctl status firewalld
firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled)
   Active: inactive (dead) since Wed 2017-03-22 07:43:05 EDT; 4min 15s ago
  Process: 28597 ExecStart=/usr/sbin/firewalld --nofork --nopid $FIREWALLD_ARGS
(code=exited, status=0/SUCCESS)
 Main PID: 28597 (code=exited, status=0/SUCC
```

7.4 老版本 kernel 删除方法

现象描述

```
rpm  -Uvh kernel
```

或者

```
yum  update  kernel
```

使用上述命令对 kernel 包升级之后，老版本 kernel 会被保留，以防止新内核有问题无法启动时，可以切换回老版本内核， 是 rpm/yum 对 kernel 包的特殊处理，别的rpm则直接会把老版本删除掉。

解决方法

要删除老版本的kernel，可以在升级kernel成功之后，使用

```
rpm -e kernel-xxx
```


或者

```
yum remove kernel-xxx
```

xxx指定具体的版本号来删除。

7.5 配置虚拟机串口输出文件

背景描述

虚拟机通过串口输出的内容定位到文件中，方便串口日志分析、传递。

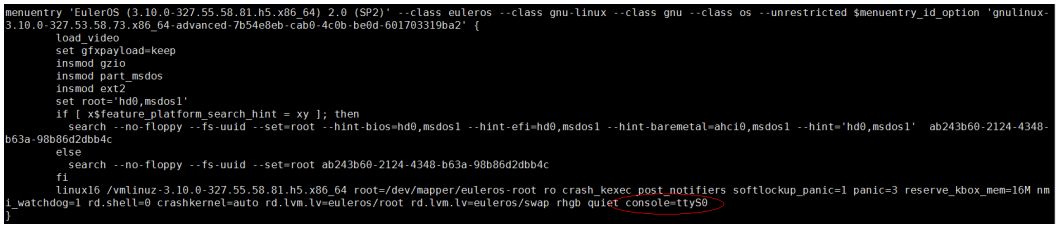
配置方法

要实现该功能，需要配置以下两个地方。

1. **修改配置文件grub.cfg中内核启动参数**

   在/etc/grub2/grub.cfg 文件中找到要修改的内核，在该内核的启动参数后增加console=ttyS0，如图7-1。

   图 7-1 增加内核启动参数

2. **配置Host端**

   Host侧需要在定义虚拟机时配置串口输出文件路径，修改示例如下，需要根据实际情况配置“source path”。

   ```
   <serial type='file'>
   <source path='/sdb/test/Euler2.2_vm/serial.log'/>
   <target port='0'/>
   </serial>
   <console type='file'>
   <source path='/sdb/test/Euler2.2_vm/serial.log'/>
   <target type='serial' port='0'/>
   </console>
   ```

7.6 rootsh 日志防爆特性的约束限制

rootsh日志防爆特性利用logrotate日志来实现日志防爆功能，用日志文件的时间戳来判断是否删除该日志文件以满足日志的空间大小要求。当系统时间发生向前改变时，则存在将新产生的rootsh日志删除，从而导致日志丢失的风险。

说明

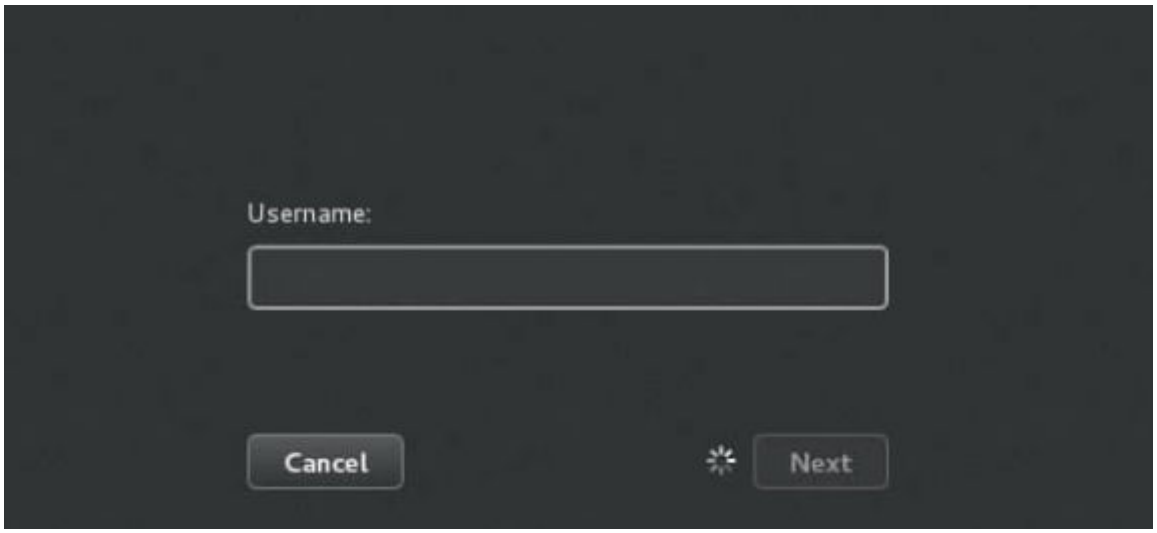
处理该场景会引入其他风险，故对此场景不做处理。

7.7 图形界面用户登录卡住现象说明

问题现象

用户使用图形界面时，如果输错密码三次，触发锁定300s，解锁后输入用户名，点击Next按钮，会一直卡住，无法登录成功。或者处于图形登录界面长时间不操作时，同样会出现该现象。如图7-2所示：

图 7-2 用户无法登录



原因分析

该现象是由于GNOME图形登录通信依赖GNOME shell与GDM之间直连的dbus连接，当用户处于登录界面长时间无响应，连接超时会自动断开，断开后继续通信则会一直卡住。点击Cancel回到登录主界面重新建立连接则恢复正常。

解决方法

点击cancel回到主界面，重新输入用户名，则可正常登录。

7.8 NetworkManager 内存占用说明

问题现象

使用ip命令批量创建大量虚拟网卡设备并删除，NetworkManager的内存占用不回落，再次创建同等数量的虚拟网络设备不上涨。

原因分析

NetworkManager大量使用g_ptr_array，g_array以及ghash等glib2动态库的数据结构来管理设备信息。当大量地并发执行创建、删除、修改设备时会产生内存碎片，这种内存占用不是内存泄露，是NetworkManager设计上的缺陷。

对于不同的虚拟网卡，以1000个数量为例，通过批量添加，配置ip，路由，设置网卡信息等操作，内存占用情况如下：

虚拟网卡类型	数量	内存占用
veth	1000	192M
vlan	1000	128M
dummy	1000	192M
macvlan	1000	128M
bridge	1000	192M
bond	1000	192M

解决方法

可以根据系统实际网卡情况来监控NetworkManager的内存占用。如果所有虚拟网卡被删除而NetworkManager内存不回落，可通过重启NetworkManager服务释放内存占用。

重启服务命令：

```
systemctl restart NetworkManager
```

7.9 极端情况下 nslcd 服务 OOM 导致 host 解析失败的现象说明

nslcd在高并发（1000个并发）时内存使用率会持续增加，当系统内存使用率达到一定负荷时，nslcd服务有可能发生OOM进而导致进程终止（panic_on_oom=0时），此后nslcd服务进入失败状态。经过一段时间（默认一小时）后，nscd缓存中的host记录开始过期，此时ping命令在解析域名对应的IP地址时由于缓存已过期且nslcd服务已经失败，导致无法从本地缓存和openLDAP服务端获取任何信息，最终提示“unknown host”。建议根据系统内存大小合理限制并发数量，避免内存耗尽。

7.10 逻辑卷创建后被自动挂载且挂载点不可用现象说明

问题现象

当使用lvcreate命令创建一个新的逻辑卷后，使用"df -h"查看挂载情况：

```
[root@localhost lvtest]# df -h
Filesystem                Size  Used Avail Use% Mounted on
/dev/mapper/euleros-root   13G  2.5G  9.7G  21% /
devtmpfs                  1.9G     0  1.9G   0% /dev
tmpfs                     1.9G     0  1.9G   0% /dev/shm
tmpfs                     1.9G   41M  1.9G   3% /run
tmpfs                     1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/sda1                 477M   98M  351M  22% /boot
tmpfs                     378M     0  378M   0% /run/user/0
/dev/mapper/vgtest-lvtest  64Z   64Z  958M 100% /home/lvtest
[root@localhost lvtest]#
```

发现新建的逻辑卷被自动挂载了，但逻辑卷的size变得无限大，并且挂载点无法写入。

原因分析

引起该问题需要以下四个要素：

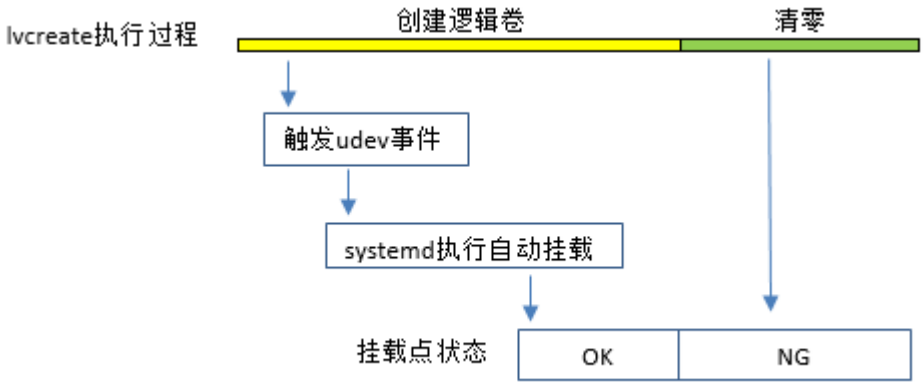
1. 在/etc/fstab中配置了逻辑卷自动挂载项，且已生效。
2. 逻辑卷的文件系统为ext文件系统（fat文件系统已验证不存在此问题，其他文件系统未验证）。
3. udev相关的服务开启：

   systemd-udevd

   systemd-udevd-control.socket

   systemd-udevd-kernel.socket
4. 在同一个卷组中，对名为xxx的逻辑卷执行“创建-删除-再创建”操作（再创建的逻辑卷的大小不能小于前者）。

当删除逻辑卷时，原逻辑卷中文件系统相关的信息仍保留在磁盘上。在磁盘的相同位置新建一个同名同大小的逻辑卷时，该文件系统的信息会被新的逻辑卷复用。lvcreate执行过程如下：



由于lvcreate执行清零的操作在文件系统挂载操作之后，已经挂载的文件系统被清零操作破坏，从而导致了"df -h"显示异常，挂载点不可用。

解决方法

在删除逻辑卷之前，先将逻辑卷中的文件系统系统信息清除。以删除/dev/vgtest/lvtest逻辑卷为例：

```
#dd if=/dev/zero of=/dev/vgtest/lvtest bs=1M count=32
#lvremove /dev/vgtest/lvtest
```

7.11 openLDAP 服务压力规格说明

在客户端使用SSSD的场景下，openLDAP 服务端短时间内收到一个客户端的10000次查询请求，所需CPU资源大约在60%～80%。

当openLDAP 服务端进程可用的CPU资源在20%以下时，SSSD客户端有极大概率会出现请求超时。

由于openLDAP服务端所处的环境存在不确定性，建议将SSSD客户端的ldap_search_timeout配置项设置为10s，以保证在openLDAP服务端所处OS压力过大，其服务进程能使用的CPU资源较小的情况下，SSSD也能正常工作（减小超时的可能性）。

此外，大量的IO读写也会拖慢服务端的响应速度（openLDAP服务需要读取磁盘上的数据库），大量写日志既会产生大量IO也会使得journal服务的CPU使用率极大上升抢占IO和CPU资源，建议服务端的日志级别配置项olcLogLevel不要小于32（数值越小日志量越大）。

7.12 qemu-kvm 创建虚拟机增加 cdrom 设备后使用 UUID 方式启动概率出现卡死的现象

问题现象

在用qemu-kvm创建虚拟机时，在xml文件增加cdrom设备，并在启动参数命令行配置root=UUID="xxxx"启动虚拟机，经反复重启后会概率出现卡死的现象。

```
[?[32m  OK  ?[0m] Reached target Basic System.
[  241.297329] INFO: task systemd-udevd:236 blocked for more than 120 seconds.
[  241.300624] "echo 0 > /proc/sys/kernel/hung_task_timeout_secs" disables this message.
[  361.304321] INFO: task systemd-udevd:236 blocked for more than 120 seconds.
[  361.307643] "echo 0 > /proc/sys/kernel/hung_task_timeout_secs" disables this message.
[  361.312234] INFO: task systemd-udevd:238 blocked for more than 120 seconds.
[  361.315496] "echo 0 > /proc/sys/kernel/hung_task_timeout_secs" disables this message.
```

原因分析

由于libata的自身tag机制实现存在缺陷，修改方案造成内核非静态函数接口和对外导出接口名称发生变更，数据结构成员发生变更，影响较大，所以此问题未做修复。

解决方法

1. 重启虚拟机。
2. 升级到EulerOS V200R007C00SPC300及以后的版本。

7.13 nfs 的客户端上 mount 相关进程卡住

问题现象

使用nfs时，mount默认采用hard模式挂载。在mount成功以后，服务端发生重启。但在服务端重启完成之后，客户端在访问之前挂载的目录时，仍然卡住。

原因分析

服务端重启之前配置了多个ip地址，客户端采用hard模式挂载了多个ip的共享目录。在服务端重启完成之后，部分ip没有恢复配置。导致客户端在访问原先挂载的共享目录时，访问失败，nfs request会一直重试，导致客户端mount相关进程卡住。

解决方法

在使用hard模式挂载时，如果客户端出现mount相关进程卡住，需要排查服务端的网络、nfs服务、rpcbind服务等是否已经恢复正常。在服务端恢复正常以后，客户端进程即可自动恢复。 另外，建议在使用hard模式时增加intr选项来硬挂载目录，这时当有某个进程进入了重试循环，则允许用户使用键盘将其中断。

说明

1. mount命令在挂载时默认采用hard模式。
2. 在nfs客户端采用hard模式挂载时，可以保证客户端和服务端数据的一致性，不会出现静默数据错误。但当服务端出现异常的时候，客户端会一直向服务端发出请求，直到服务端恢复正常，进而导致客户端进程一直阻塞。
3. 在nfs客户端采用soft模式挂载时，可以通过timeo和retry参数配置超时时间。服务端出现异常时，客户端会向服务器端重发请求。当超过配置的超时时间时，则返回错误，不会一直阻塞。但在soft模式下可能会出现静默数据错误。

8 命令参考

EulerOS的命令类似于Readhat，CentOS7.1 大多命令可以直接在EulerOS 上运行。

表 8-1 Euler OS 常用帮助

命令	功能	支持/扩展
uname -a	查看当前内核版本信息	/
cat /etc/euler-release	Euler OS 发布信息	/
cat /etc/euleros-release	Euler OS 版本信息	/

更多命令参考 《EulerOS V2.0SP2 管理员指南》

