

OpenBMC development of virtual media virtual-media service

原创

Lemon in the Rain

📅 Modified on 2024-10-30 17:21:26

👁 Read 2.7k

🌟 Collection 49

👍 Likes 53

Copyright CC 4.0 BY-SA

Category Column:

OpenBMC


 Article Tags:

media

linux

server

C++

 OpenBMC This column includes this content

8 articles

Subscribe to our column

Virtual Media (also known as Remote Media)

Problem Description

Virtual Media allows the user to remotely mount a given ISO/IMG drive image to a server host via the BMC. The remote drive appears as a USB storage device in the host and operates in RO mode or RW mode (keep in mind container restrictions and write protection switches). This can even be used to install an operating system on a bare metal system. This document highlights some of the redirection options, such as browser-based ISO/IMG image mounting and remote CIFS/HTTPS image mounting.

References

- Virtual media will use a network block device as the primary forwarder for disk images.
- NBDkit is used to serve images from remote storage, over HTTPS/CIFS protocol. USBGadget is used to expose media storage devices to the host.

need

none

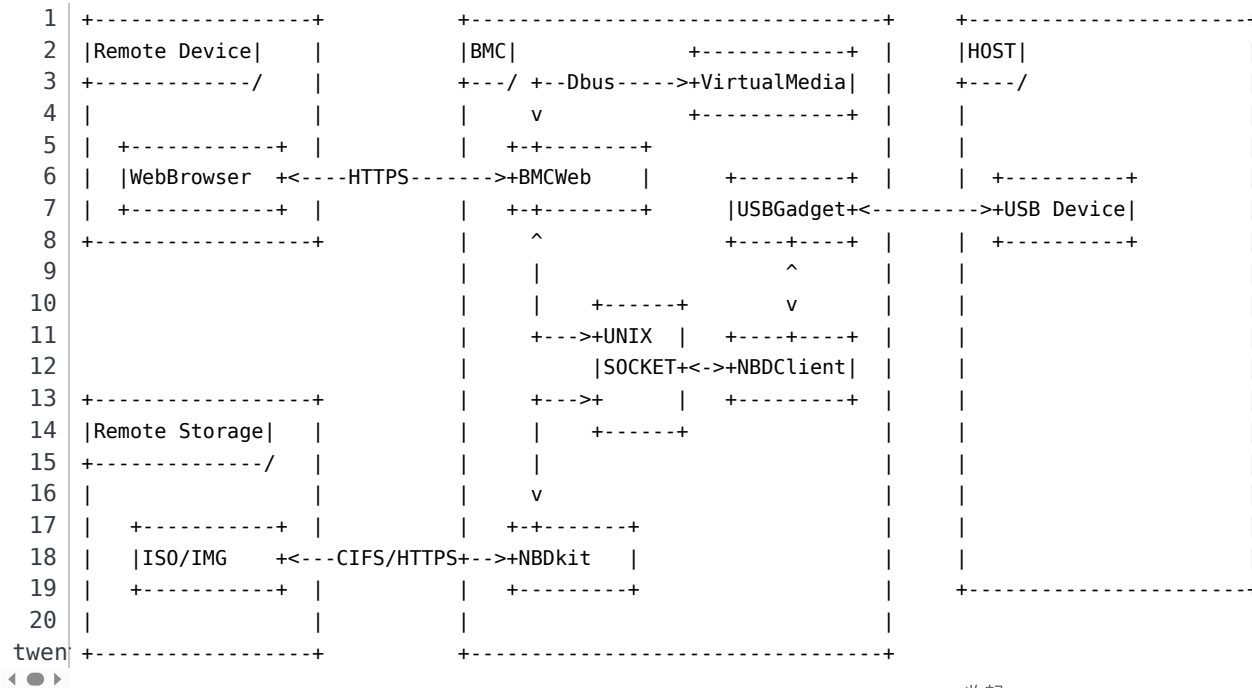
Proposed Design

Virtual media has two operating modes, called proxy mode and traditional mode.

- Proxy Mode - Works directly from your browser and uses JavaScript/HTML5 to communicate directly with an HTTPS endpoint hosted on the BMC over a secure WebSocket.
- Legacy Mode - Launched from a browser using Redfish defined VirtualMedia mode, the BMC process then connects to the external CIFS/HTTPS image specified during initialization.

Both methods inherit the default Redfish/BMCWeb authentication and privilege mechanisms.

The following component diagram shows a high-level overview of virtual media.



The Virtual Media feature supports multiple simultaneous connections in two modes.

Asynchronicity

Mounting and unmounting remote devices may take some time. Virtual Media should support asynchronicity at the DBus and optionally Redfish API level.

The default timeout for DBus is 25 seconds and the default timeout for Redfish is 60 seconds, which may not be enough to perform mount and unmount in some cases.

Asynchronous responses are described in the appropriate sections.

Network Block Device (NBD)

You may notice that most of the connections in the diagram are based on network block devices. According to the Sourceforge project description:

After compiling this in the kernel, Linux can use the remote server as one of its block devices. Each time a client computer wants to read /dev/nbd0, it sends a request over TCP to the server, and the server replies with the requested data. This can be used for workstations with low disk space (or even diskless - if you use initrd) to borrow disk space from other computers. Unlike NFS, you can put any filesystem on it. However (unlike NFS), if someone mounts NBD as read/write, you have to make sure that no one else will mount it.

– <https://nbd.sourceforge.io/>

In the Virtual Media use case, it is used to pull data from a remote client and present it to `/dev/nbdXX` a device that is not mounted to the BMC. The block device is then provided to the host via a USB gadget.

USB Gadget

Part of the Linux kernel that makes emulation of specific USB device classes possible via USB "On-The-Go" , emulating USB mass storage connected to the host if appropriately connected to the host. In the case of Virtual Media, it emulates USB mass storage connected to the host. The source or redirection is a block device created by nbd-client `/dev/nbdXX` .

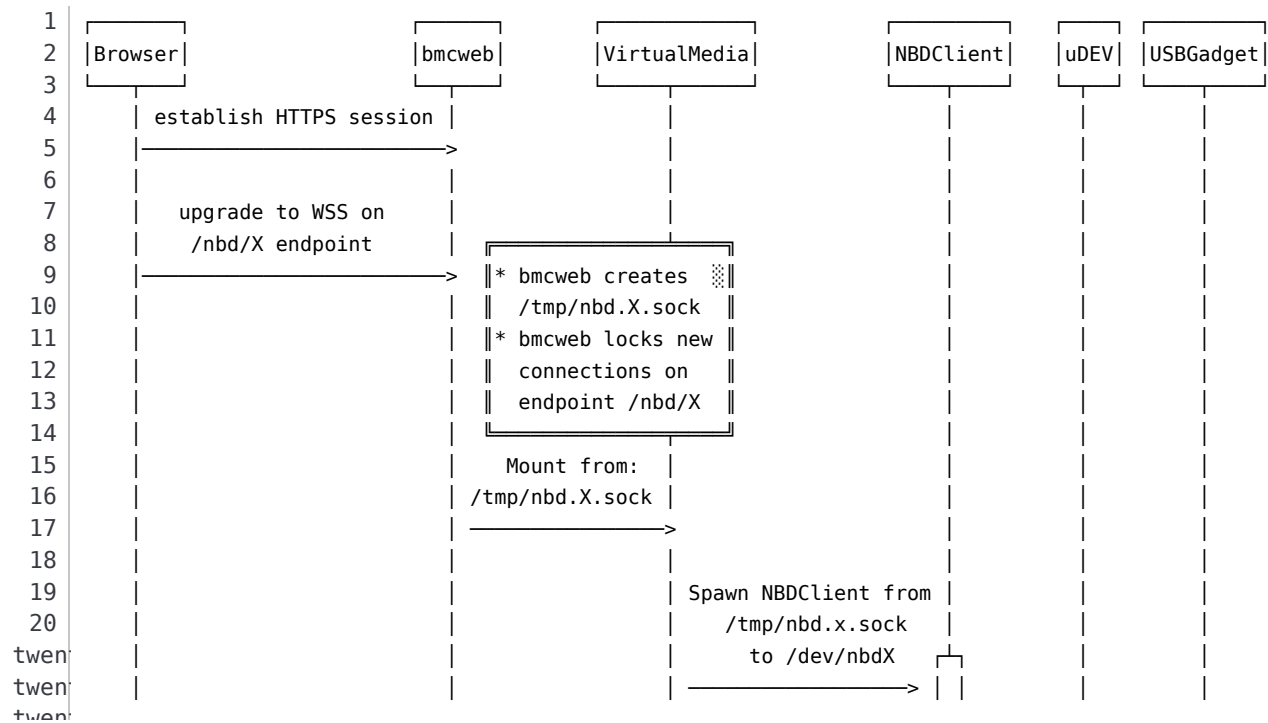
Proxy Mode

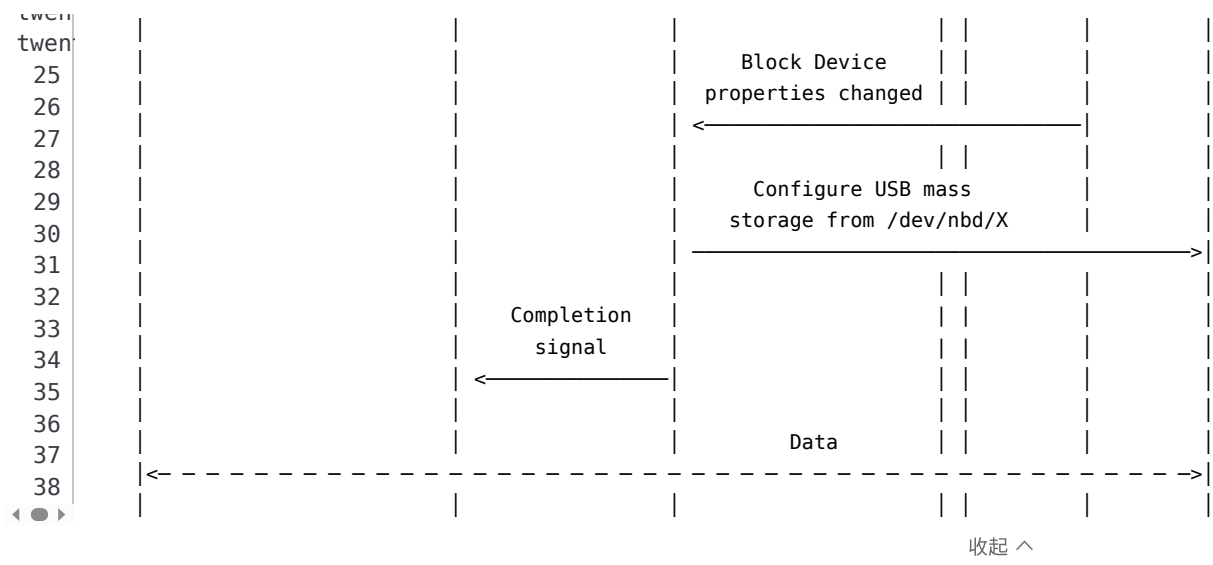
Proxy mode uses browser JavaScript and WebSocket support to create a JS NBD server. The browser is responsible for creating an HTTPS session, authenticating the user, and receiving the given permissions, and then upgrading the HTTPS session to WSS through the mechanism described in RFC6455 . Since the upgrade to WSS, the JS application is responsible for handling all NBD server commands required by the specification.

To open multiple simultaneous connections per URI, define the number of simultaneous connections available in the HTTPS server. The configuration file described in the next section defines the number of simultaneous connections available.

The proxy's encryption is supported through HTTPS/WSS channels and inherits the encryption mechanism directly from the HTTPS server. All data transactions go through bmcweb.

The connection initialization will be as shown below:





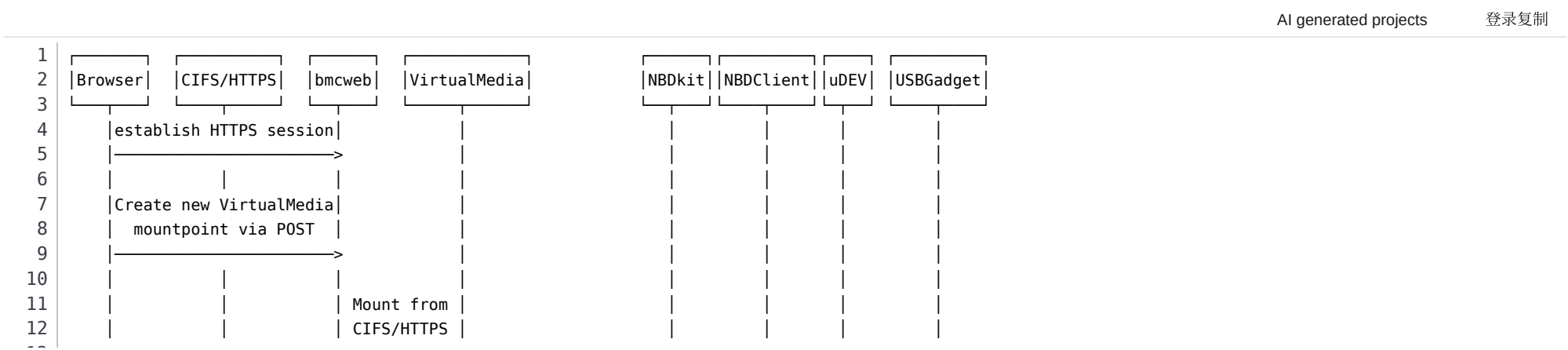
Legacy Mode

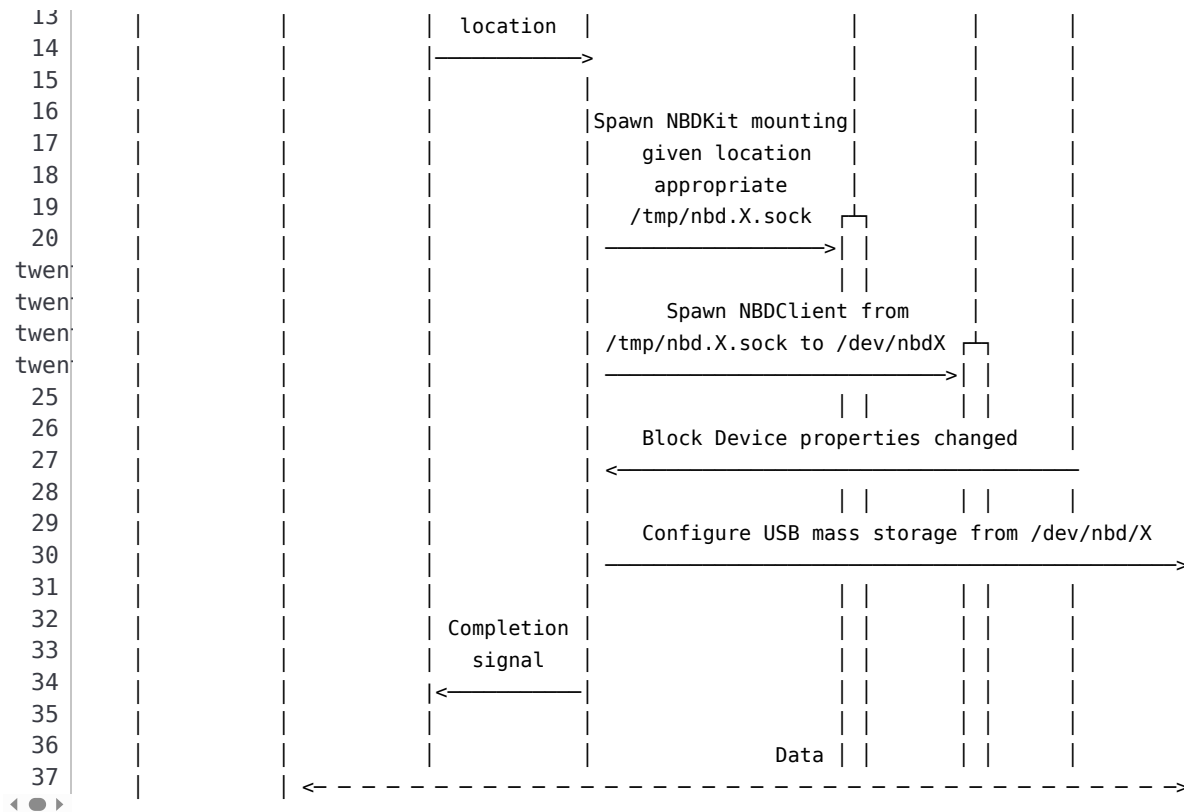
Legacy mode uses the VirtualMedia schema defined by DMTF to mount external CIFS/HTTPS images. The current implementation only supports streaming mounts. In this case, Redfish is only used as a mechanism for Virtual Media initialization and is not responsible for data transmission. For data, there is a separate component responsible for handling CIFS/HTTPS traffic, called NBDkit.

Multiple simultaneous connections are supported by spawning additional nbkit instances. The number of available CIFS/HTTPS instances is configured and described in detail in the next section.

Encryption is based on the remote storage connection and follows Intel's best security practices as long as the remote server supports such encryption (e.g. HTTPS requires SSL, plain HTTP is not supported, for CIFS protocol version 3.0 encryption is allowed and will be provided).

The process is as follows:





Redfish support

Virtual media services will be exposed as DMTF defined Redfish VirtualMedia endpoints. Below are some examples.

Virtual Media Collection schema

The members of the collection will be defined according to the configuration file described in the next section, and will be visible regardless of whether media is inserted or not.

Members in collection will be defined based on configuration file described in next sections. And will be visible despite media is inserted or not.

```
1 {
2   "@odata.type": "#VirtualMediaCollection.VirtualMediaCollection",
3   "Name": "Virtual Media Services",
4   "Description": "Redfish-BMC Virtual Media Service Settings",
5   "Members@odata.count": 2,
6   "Members": [
7
```

```

8      {
9        "@odata.id": "/redfish/v1/Managers/BMC/VirtualMedia/IS00"
10      },
11      {
12        "@odata.id": "/redfish/v1/Managers/BMC/VirtualMedia/1"
13      }
14    ],
15    "@odata.context": "/redfish/v1/$metadata#VirtualMediaCollection.VirtualMediaCollection",
16    "@odata.id": "/redfish/v1/Managers/BMC/VirtualMedia"
  }

```

收起 ^

Virtual Media schema

AI generated projects

登录复制

```

1  {
2    "@odata.type": "#VirtualMedia.v1_1_0.VirtualMedia",
3    "Id": "IS00",
4    "Name": "Virtual Removable Media",
5    "Actions": {
6      "#VirtualMedia.InsertMedia": {
7        "target": "/redfish/v1/Managers/bmc/VirtualMedia/IS00/Actions/VirtualMedia.InsertMedia"
8      },
9      "#VirtualMedia.EjectMedia": {
10       "target": "/redfish/v1/Managers/bmc/VirtualMedia/IS00/Actions/VirtualMedia.EjectMedia"
11     }
12   },
13   "MediaTypes": ["CD", "USBStick"],
14   "Image": "https://192.168.0.1/Images/os.iso",
15   "ImageName": "Os",
16   "ConnectedVia": "URI",
17   "Inserted": true,
18   "WriteProtected": false,
19   "@odata.context": "/redfish/v1/$metadata#VirtualMedia.VirtualMedia",
20   "@odata.id": "/redfish/v1/Managers/BMC/VirtualMedia/IS00",
21   "Oem": {
22     "OpenBMC": {
23       "@odata.type": "#OemVirtualMedia.v1_0_0.VirtualMedia",
24       "WebSocketEndpoint": "/nbd/0"
25     }
26   }
27 }

```

收起 ^

The architecture of the proxy model and the traditional model look similar. Here are some key differences:

Field Name	Proxy Mode	Legacy Mode	Comment
InsertMedia	N/A	action as described by DMTF spec	Action can return Task object if process is time consuming
Image	N/A	image location	
ImageName	N/A	image name	
ConnectedVia	"Applet"	as described by DMTF spec	applies only for connected media
TransferMethod	"Stream"	"Stream"	"upload" is not supported by design
TransferProtocolType	"OEM"	as described by DMTF spec	

Virtual Media OEM Extensions

Virtual Media mode is an adaptation of the traditional mode, where the image is provided directly by the user through the Redfish operation and the entire connection is handled between the service and **the web server** .

For **proxy mode** , nbd data is provided by the client web browser. In order to establish a connection, the client needs information about the location of the WebSocket created by the web server. This value is exposed as the OEM "WebSocketEndpoint" property of each project.

Inactivity timeout

Virtual media supports inactivity timeout, which will disconnect virtual media connection after a certain period of inactivity. Since nbdclient has a mechanism to cache images and kernel also has home buffer mechanism for block devices, the idea is to prepare a patch on USBGadget driver which will write USB gadget statistics under /proc/USBGadget/lun.X file. Virtual media application will observe these statistics.

Virtual Media Service

The Virtual Media Service is a standalone application that will co-exist with DBus. It will be initialized from a configuration json file and expose all the functions that can be used to provide virtual media objects. Virtual Media is responsible for:

- Expose the current virtual media configuration to DBus users.
- Spawns an nbdclient for the proxy connection, and monitors its lifecycle.
- Spawn nbdkit for CIFS/HTTPS connections and monitor their lifecycle.
- Monitor uDEV for all NBD-related block device changes and configure/unconfigure USB Gadgets accordingly.
- Monitor NBD devices for inactivity to support inactivity timeouts.

Configuration

When the process starts, the virtual media reads its configuration file, which has the following structure:

```
1  "InactivityTimeout": 1800,          # Timeout of inactivity on device in seconds, that will lead to automatic disconnection
2  "MountPoints": {
3      "IS00": {
4          "EndpointId": "/nbd/0",      # bmcweb endpoint (URL) configured for this type of connection
5          "Mode": 0,                  # 0 - Proxy Mode, 1 - Legacy Mode
6          "NBDDDevice": "/dev/nbd0",  # nbd endpoint on device usually matches numeric value with EndpointId
7          "UnixSocket": "/tmp/nbd.sock", # defines which Unix socket will be occupied by connection
8          "Timeout": 30,              # timeout in seconds passed to nbdclient
9          "BlockSize": 512,          # Block size passed to nbdclient
10     }
11 },
```

收起 ^

DBus Interface

Virtual Media will expose the following object structure. All object paths are representations of the configuration files described above.

```
1  /xyz/openbmc_project/VirtualMedia/Proxy/IS00
2  /xyz/openbmc_project/VirtualMedia/Proxy/1
3  /xyz/openbmc_project/VirtualMedia/Legacy/0
4  /xyz/openbmc_project/VirtualMedia/Legacy/1
```

Each object will implement `xyz.openbmc_project.VirtualMedia.Process` the interface, which will be defined as follows:

Name	type	input	return	description
Active	Property	-	BOOLEAN	<code>True</code> , if object is occupied by active process, <code>False</code> otherwise
ExitCode	Property	-	INT32	If process terminates this property will contain returned exit code

Each object will also expose its own configuration `xyz.openbmc_project.VirtualMedia.MountPoint` under (all properties are read-only).

Name	type	input	return	description
EndPointId	Property	-	STRING	As per configuration
Mode	Property	-	BYTE	As per configuration

Name	type	input	return	description
Device	Property	-	STRING	As per configuration
Socket	Property	-	STRING	As per configuration
Timeout	Property	-	UINT16	As per configuration
BlockSize	Property	-	UINT16	As per configuration
RemainingInactivityTimeout	Property	-	UINT16	Seconds to drop connection by server, for activated endpoint, 0 otherwise
ImageUrl	Property	-	STRING	URL to mounted image
WriteProtected	Property	-	BOOLEAN	'True', if the image is mounted as read only, 'False' otherwise

Each object also exposes `xyz.openbmc_project.VirtualMedia.Stats` the following statistics interface (all properties are read-only):

Name	type	input	return	description
ReadIO	Property	-	UINT64	Number of read IOs since image mounting
WriteIO	Property	-	UINT64	Number of write IOs since image mounting

Depending on the object path, the object will expose different interfaces to mount the image.

Mounting can be a time-consuming task, so an event-driven mechanism must be introduced. Mount and unmount calls will trigger asynchronous operations and end immediately, giving appropriate signals containing the task completion status.

For the proxy mode, the interface `xyz.openbmc_project.VirtualMedia.Proxy` is defined as follows:

Name	type	input	return	description
Mount	Method	-	BOOLEAN	Perform an asynchronous operation of mounting to HOST on given object.
Unmount	Method	-	BOOLEAN	Perform an asynchronous operation of unmount from HOST on given object
Completion	Signal	-	INT32	Returns 0 for success or errno on failure after background operation completes

For the traditional mode, the interface `xyz.openbmc_project.VirtualMedia.Legacy` is defined as follows:

Name	type	input	return	description
Mount	Method	STRING	BOOLEAN	Perform an asynchronous operation of mounting to HOST on given object, with location given as STRING parameter
Mount	Method	STRING BOOLEAN VARIANT<UNIX_FD,INT>	BOOLEAN	Perform an asynchronous operation of mounting to HOST on given object, with parameters: STRING : url to image. It should start with either smb:// or https:// prefix BOOLEAN : RW flag for mounted gadget (should be consistent with remote image capabilities) VARIANT<UNIX_FD, INT> : file descriptor of named pipe used for passing null-delimited secret data (username and password). When there is no data to pass -1 should be passed as INT
Unmount	Method	-	BOOLEAN	Perform an asynchronous operation of unmounting from HOST on given object
Completion	Signal	-	INT32	Returns 0 for success or errno on failure after background operation completes

Mount and unmount operations return true if the asynchronous operation is started, and false if the pre-check encounters an error. They can also indicate an appropriate DBus error.

Alternatives Considered

Existing implementation in OpenBMC

Impact

Shall not affect usability of current Virtual Media implementation

Testing

TBD

<https://github.com/openbmc/docs/blob/master/designs/virtual-media.md>

The data flow is as follows:

1. The user opens the Virtual Media page in the client browser and sends an HTTP request to the OpenBMC Virtual Media service.
2. The OpenBMC Virtual Media service receives the HTTP request and calls the Virtual Media process through DBus.
3. After receiving the request, the Virtual Media process creates an nbd-client instance and connects to the nbd-server. If necessary, it also generates a proxy address and Websocket URL and returns them to the client browser.
4. Once the connection is established, the client browser can use the NBD protocol to read/write the contents of the block device from the OpenBMC device.
5. When a user disconnects a Virtual Media session in the client browser, an HTTP request is sent to the OpenBMC Virtual Media service to end the session.
6. The OpenBMC Virtual Media service calls the Virtual Media process through DBus and notifies it to end the nbd-client instance.
7. The Virtual Media process closes the nbd-client instance and, if necessary, stops the nbd-server instance.

In the implementation of OpenBMC Virtual Media, the relationship between NBDKit, NBDServer and NBDClient is as follows:

1. NBDKit is an open source tool for creating and managing Network Block Devices (NBD). In OpenBMC, NBDKit is a dependency of NBDServer and is used to create NBD servers and manage NBD block devices.
2. NBDServer is a server application running on OpenBMC, used to provide virtual media image files to clients. NBDServer uses NBDKit to create NBD servers and forward client requests to NBD block devices.
3. NBDClient is a client application used to connect to NBDServer and map NBD block devices to the local system. In OpenBMC Virtual Media, NBDClient runs in the client's web browser and is used to transfer virtual media image files to OpenBMC and load them into NBD block devices.

Therefore, NBDKit is a tool for creating and managing NBD servers, NBDServer is an NBD server application running on OpenBMC, and NBDClient is an NBD client application running in the client browser. These three work together to form the core components of OpenBMC Virtual Media.

Openbmc Dbus----->VirtualMedia Explanation

OpenBMC Virtual Media is a DBus-based service that aims to provide the ability to connect to virtual media on the BMC (Baseboard Management Controller) over a network. DBus is an inter-process communication mechanism used to pass messages between different processes.

The Virtual Media service will register an object on DBus, and other processes can access the Virtual Media service API through the DBus Object Path. The Virtual Media service API includes:

- Setting Virtual Media Configuration
- Get the current Virtual Media configuration
- Connect/Disconnect Virtual Media
- Get the status of the Virtual Media connection

The format of the DBus API call is as follows:

AI generated projects 登录复制

```
1 pythonCopy codeMethodCall("xyz.openbmc_project.VirtualMedia",
2     "/xyz/openbmc_project/virtual_media/1",
3     "xyz.openbmc_project.VirtualMedia.Process",
4     "Mount",
5     "s",
6     imagePath)
```

Among them, the first parameter indicates DBus Service Name, the second parameter indicates DBus Object Path, the third parameter indicates DBus Interface Name, the fourth parameter indicates the name of the called method, the fifth parameter indicates the parameter type of the method, and the sixth parameter indicates the parameters of the method.

The Virtual Media service communicates with other processes through DBus. Therefore, other processes only need to understand how to call the DBus API to use the functions provided by the Virtual Media service.

