QEMU AS A USB MTP RESPONDER

Bandan Das <bsd@redhat.com>

KVM Forum 2016

MULTIPLE WAYS TO SHARE FOLDERS, SUCH AS:

- Network based - NFS/Samba/SSHFS
- Device based
  - Virtio - 9pfs, virtio-serial
  - usb-mtp

ADVANTAGES/DISADVANTAGES

- Configuration
  - Is there a firewall ?
- Availability of services and support
  - Does guest support this device ?
- Present usb-mtp as another option
  - More options = good ?

MEDIA TRANSFER PROTOCOL

- Introduced by Microsoft as an extension to Picture Transfer Protocol (PTP)
- What is PTP ?
  - Protocol for transferring digital images from cameras
  - Application layer protocol
- New names : Initiator (Client), Responder (Server)
- Atomic operations, controlled by the server
  - One operation at a time

MEDIA TRANSFER PROTOCOL

- Limited file operations support
- Supports many transport layers (TCP/IP, Firewire) although USB is common
  - USB device class
- Supports DRM
- Good adoption - Android, Windows, Linux
  - Plug and Play in most cases!

MTP VS USB MASS STORAGE

- Storage still in control of the device
- File corruption is minimized
- Interesting tidbit:
    - Default in Android
    - Let's Android not having to use VFAT
    - Prevents OEM from providing users with little application space

MTP HIGH LEVEL WORKFLOW

	Device connected
Transport layer discovery	
Device information/OpenSession	
	Device capability, name etc
List contents	
	Object handles
Object properties such as file size	
	Send object metadata
Object Exchange Request	Response

QEMU AND MTP

- Exposed to the guest as a USB device
- Example usage:

```
... -usb -device usb-mtp,x-root=usbdrive,desc=mtp-share
```

- One file operation at a time
- Supports notification of file changes to the guest
- Supports > 4G files
- No write support yet (Copy to device)

THE TRANSPORT LAYER

- MTP runs on top of USB
- USB communication is through endpoints
  - Each endpoint is a data pipe
  - One control endpoint
  - IN endpoints (device -> host, or responder to initiator)
  - OUT endpoints (host-> device, or initiator to responder)
- Types of endpoints
  - Control
  - Bulk (Storage data)
  - Isochronous (Streaming data)
  - Interrupts (IN endpoint, host polls this endpoint)

A LOOK AT THE DATA STRUCTURES

```
  /* Device structure corresponding to OpenSession */
  struct MTPState {
      USBDevice dev;
      ...
      MTPData    *data_in;
      MTPData    *data_out;
      MTPControl *result;
      ...
#ifdef CONFIG_INOTIFY1
      /* inotify descriptor */
      int        inotifyfd;
      QTAILQ_HEAD(events, MTPMonEntry) events;
#endif
  }
  ...
  /* Response Dataset from Responder to Initiator */
  struct MTPData {
      uint16_t code;
      uint32_t trans;
      ...
  }
```

A LOOK AT THE DATA STRUCTURES

```
/*
 * Request Dataset from Initiator to Responder
 * Formatted by usb-mtp
 */
struct MTPControl {
    uint16_t code;
    uint32_t trans;
    int argc;
    uint32_t argv[5];
}
...
/* Struct that defines contents */
struct MTPObject {
    uint32_t handle;
    uint16_t format;
    char     *name;
    ...
}
```

USB AND MTP INTERACTION

```
static void usb_mtp_handle_data(USBDevice *dev, USBPacket *p)
{
 ...
 /* Responses from device, including data transfers, error propagation */
 case EP_DATA_IN:
 /* Requests from host */
 case EP_DATA_OUT:
 /* Events such as file change notifications */
 case EP_EVENT:
```

MTP IMPLEMENTATION INTERNALS

- Object Enumeration
- Notification changes
- Data transfer
- File Operations - Write/Copy/Delete etc.

OBJECT ENUMERATION

- Initiator sets up a "MTPControl" packet with argv[2] = 0 or 0xffffffff
- Initiator sends CMD_GET_OBJECT_HANDLES
- Responder does sanity checks on MTPControl packet
- Responder does readdir() on root folder and fills MTPObject structs recursively
- Responder sends a MTPData packet with the MTPObject uint32_t handles array

DATA TRANSFER

- Initiator sends CMD_GET_OBJECT_INFO (Optional)
- Responder replies with a MTPData packet with object details such as name, size
- Initiator sends CMD_GET_OBJECT with MTPControl argv[0] set to the handle
- Responder does sanity checks on object handle and looks up the entry
- Responder reads file and fills up a MTPData packet
  - Responder keeps track of offset if size > usb payload

NOTIFICATION CHANGES (EP_EVENT)

- Convention: device interrupts the host when it needs attention
  - With USB, host polls for events
- Events are propagated when the host polls the interrupt endpoint
- Uses inotify (only works with Linux hosts)
  - Register inotify handlers to all files in the folders
  - Call object enumeration when new file is added
  - Store inotify events
  - When host (Initiator) polls this EP, deliver one event at a time

MTP WRITE

- MTP does not support edit/write directly
- Host(Initiator) can copy file, edit and copy it back
  - Or create a new file
- Support for SendObjectInfo that sends a ObjectInfo dataset (I->R)
  - ObjectInfo sanity checks detremine if device can accept the object
- Support for SendObject that follows the above

OUTREACHY INTERNSHIP PROJECT

- Isaac Lozano, adding features and fixing bugs
- Adding support for > 4G file transfers
    - Support for Device properties
- Microsoft specific data fields
    - Not mentioned in spec - conventional values
- Adding write support

TODO ITEMS

- Adding asynchronous operations
  - Support for multiple sessions
- Performance audit, synchronous operations eats up CPU
- Support all MTP file operations - Write/Move/Delete/Copy etc.
- Testing with different guest configurations

THANK YOU

- Questions ?