Isochronous USB I/O on Windows

Although the issue does not come up often, it seems that everyone who tries to do isochronous USB I/O in a Windows driver gets it wrong. Isochronous URBs do work very differently from bulk and interrupt requests (which, as it happens, are identical). This article is an attempt to point out the differences.

Bulk - Buffer-oriented

Bulk and interrupt requests are buffer-oriented. If you want to read 10,000 bytes, you submit an URB where the TransferBuffer points to a 10,000 byte buffer, and you set TransferBufferLength to 10,000. The host controller hardware will hold on to your request until those 10,000 bytes are all handled, no matter how long it takes or how many retries it requires. You don't particular need to worry about the endpoint's packet size, although a careful driver writer will always ensure tht the buffer is an even multiple of the packet size (bulk packets in a high-speed device are always 512 bytes).

Isochronous -- Packet-oriented

Isochronous requests are packet-oriented, and YOU are responsible to chop up your buffer into packet-sized chunks. This is done with the IsoPacket array in the URB_ISOCH_TRANSFER structure. Each element in the IsoPacket array must include the offset of the start of the packet within the TransferBuffer, and the size of this packet. Further the size of each element in the array must be the same size as the packet size of your endpoint (or larger, although there is little point in doing so).

So, for a device with 512-byte packets, where we want 8 packets per URB, we might set:

```
URB_ISOCH_TRANSFER urb;
RtlZeroMemory( &urb, sizeof(URB_ISOCH_TRANSFER) );
...
urb.TransferBuffer = ExAllocatePool( ..., 4096 );
urb.TransferBufferLength = 4096;
urb.NumberOfPackets = 8;
urb.IsoPacket[0].Offset = 0;
urb.IsoPacket[0].Length = 512;
urb.IsoPacket[1].Offset = 512;
urb.IsoPacket[1].Length = 512;
urb.IsoPacket[2].Offset = 1024;
urb.IsoPacket[2].Length = 512;
...
urb.IsoPacket[7].Offset = 3584;
urb.IsoPacket[7].Length = 512;
```

Of course, you would usually do that setup in a loop, instead of explicitly, like that. That's especially true if you have a device with several alternate settings for different bandwidth requirements, which is implemented by having different packet sizes for the isochronous endpoint.

When the request is completed and your completion handler is called, the Length and Error fields in the IsoPacket array will have been modified by the host controller driver.

Full-speed Packeting

Isochronous endpoints on a full-speed device can have a maximum packet size up to 1023 bytes. Yes, that's 1023 bytes, not 1024. The reasons for this oddness have been lost in the mists of antiquity. A single URB can cover up to 255 frames.

High-speed Packeting

For a high-speed device, there are a couple of twists. The USB spec allows a high-speed isoch endpoint to specify 1, 2, or 3 transfers per microframe, which increases the bandwidth for the endpoint. As far as a driver is concerned, this just increases the size of the packet. A maximum bandwidth isochronous endpoint has 3 transfers of 1024 bytes per microframe. This means that your device has 3072-byte packets, so your IsoPacket array must be set up in 3072-byte units.

Further, each isochronous URB for a high-speed device must cover a multiple of full frames, meaning a multiple of 8 microframes. So, if your device specifies an interval of 1 (bInterval == 1), meaning that bandwidth is reserved in every microframe, your URB must have a multiple of 8 packets. If the interval is 2 (bInterval == 2), your URB must have a multiple of 4 packets. If the interval is 4 (bInterval == 3), your URB must have an even number of packets. If the interval is 8 or more (bInterval >= 4), there is no limitation on the number of packets.

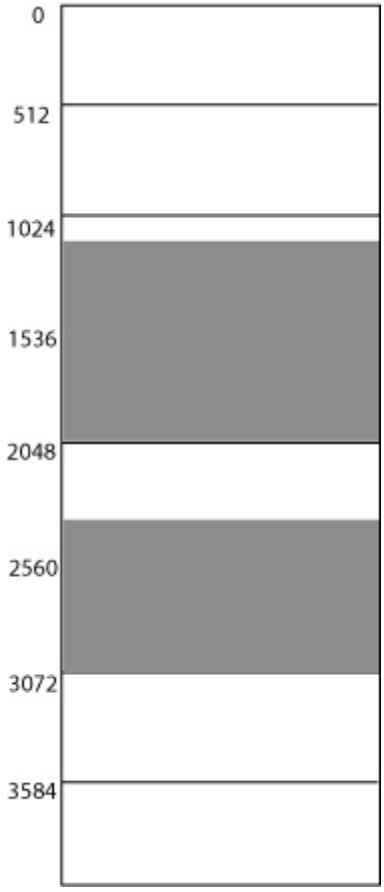
Packet Array

The packets in your array will be associated with one frame or microframe, one at a time, in order. If you have 10 packets in your array, your URB will last for exactly 10 frames (or microframes) and then be completed. This has several important implications for handling the buffer in your completion handler. Isochronous requests do not automatically retry; if a bus error occurs during an isochronous transfer, that packet is discarded. In that case, the Length fields in the corresponding IsoPacket array element will be 0. Further, it's quite common for a device to return less than the maximum packet size in a single transfer (especially for audio devices). Again, the Length field will tell you how many bytes were actually transferred during that frame or microframe.

However, the hardware does not pack the data in the buffer. This is one of the biggest surprises for people just starting into isochronous work.

Here's an example. Let's start from the request we set up above, with 8 packets of 512 bytes. Now, let's say the device transfered two full packets, followed by a short packet of 128 bytes, followed by a skipped frame, followed by a short packet of 300 bytes, followed by another skipped packet, followed by two more full packets. So, the size of the packets are 512, 512, 128, 0, 300, 0, 512, 512.

The start of our data buffer will have 1152 bytes of data (512+512+128), followed by 896 bytes of empty space, followed by 300 bytes of data, followed by 724 bytes of empty space, followed by 1024 bytes of data. So, even though the packet contains 2476 bytes of data, that data is spread sparsely through the buffer. If you need to return a contiguous buffer to your caller (a common need), you will have to copy the fragments yourself into an output buffer. In the following diagram, the white areas are regions that contain good data, while the shaded areas contain unknown garbage.



Written by Tim Roberts.