Using MDLs

An I/O buffer that spans a range of contiguous virtual memory addresses can be spread over several physical pages, and these pages can be discontiguous. The operating system uses a *memory descriptor list* (MDL) to describe the physical page layout for a virtual memory buffer.

An MDL consists of an **MDL** structure that is followed by an array of data that describes the physical memory in which the I/O buffer resides. The size of an MDL varies according to the characteristics of the I/O buffer that the MDL describes. System routines are available to calculate the required size of an MDL and to allocate and free the MDL.

An MDL structure is semi-opaque. Your driver should directly access only the **Next** and **MdlFlags** members of this structure. For a code example that uses these two members, see the following Example section.

The remaining members of an MDL are opaque. Do not access the opaque members of an MDL directly. Instead, use the following macros, which the operating system provides to perform basic operations on the structure:

> **MmGetMdlVirtualAddress** returns the virtual memory address of the I/O buffer that is described by the MDL.
>
> **MmGetMdlByteCount** returns the size, in bytes, of the I/O buffer.
>
> **MmGetMdlByteOffset** returns the offset within a physical page of the beginning of the I/O buffer.

You can allocate an MDL with the **IoAllocateMdl** routine. To free the MDL, use the **IoFreeMdl** routine. Alternatively, you can allocate a block of nonpaged memory and then format this block of memory as an MDL by calling the **MmInitializeMdl** routine.

Neither **IoAllocateMdl** nor **MmInitializeMdl** initializes the data array that immediately follows the MDL structure. For an MDL that resides in a driver-allocated block of nonpaged memory, use **MmBuildMdlForNonPagedPool** to initialize this array to describe the physical memory in which the I/O buffer resides.

For pageable memory, the correspondence between virtual and physical memory is temporary, so the data array that follows the MDL structure is valid only under certain circumstances. Call **MmProbeAndLockPages** to lock the pageable memory into place and to initialize this data array for the current layout. The memory will not be paged out until the caller uses the **MmUnlockPages** routine, at which point the contents of the data array are no longer valid.

The **MmGetSystemAddressForMdlSafe** routine maps the physical pages that are described by the specified MDL to a virtual address in system address space, if they are not already mapped to system address space. This virtual address is useful for drivers that might have to look at the pages to perform I/O, because the original

virtual address might be a user address that can be used only in its original context and can be deleted at any time.

Note that when you build a partial MDL by using the IoBuildPartialMdl routine, the caller should use **MmGetMdlVirtualAddress** instead of the **MmGetSystemAddressForMdlSafe** routine when determining the virtual address to pass in. **IoBuildPartialMdl** uses the address that **MmGetMdlVirtualAddress** returns from the source MDL to determine the offset for the target MDL. If the addresses are different (for example, when the first address is a user address), passing the address that **MmGetSystemAddressForMdlSafe** returns can cause data corruption or a bug check.

When a driver calls **IoAllocateMdl**, it can associate an IRP with the newly allocated MDL by specifying a pointer to the IRP as the *Irp* parameter of **IoAllocateMdl**. An IRP can have one or more MDLs associated with it. If the IRP has a single MDL associated with it, the IRP's **MdlAddress** member points to that MDL. If the IRP has multiple MDLs associated with it, **MdlAddress** points to the first MDL in a linked list of MDLs that are associated with the IRP, known as an *MDL chain*. The MDLs are linked by their **Next** members. The **Next** member of the last MDL in the chain is set to **NULL**.

If, when the driver calls **IoAllocateMdl**, it specifies **FALSE** for the *SecondaryBuffer* parameter, the IRP's **MdlAddress** member is set to point to the new MDL. If *SecondaryBuffer* is **TRUE**, the routine inserts the new MDL at the end of the MDL chain.

When the IRP is completed, the system unlocks and frees all the MDLs that are associated with the IRP. The system unlocks the MDLs before it queues the I/O completion routine and frees them after the I/O completion routine executes.

Drivers can traverse the MDL chain by using the **Next** member of each MDL to access the next MDL in the chain. Drivers can manually insert MDLs into the chain by updating the **Next** members.

MDL chains are typically used to manage an array of buffers that are associated with a single I/O request. (For example, a network driver could use one buffer for each IP packet in a network operation.) Each buffer in the array has its own MDL in the chain. When the driver completes the request, it combines the buffers into a single large buffer. The system then automatically cleans up all the allocated MDLs for the request.

The I/O manager is a frequent source of I/O requests. When the I/O manager completes an I/O request, the I/O manager frees the IRP and frees any MDLs that are attached to the IRP. Some of these MDLs might have been attached to the IRP by drivers that are located beneath the I/O manager in the device stack. Similarly, if your driver is the source of an I/O request, your driver must clean up the IRP and any MDLs that are attached to the IRP when the I/O request completes.

Example

The following code example is a driver-implemented function that frees an MDL chain from an IRP:

```
VOID MyFreeMdl(PMDL Mdl)
{
```

```
    PMDL currentMdl, nextMdl;

    for (currentMdl = Mdl; currentMdl != NULL; currentMdl = nextMdl)
    {
        nextMdl = currentMdl->Next;
        if (currentMdl->MdlFlags & MDL_PAGES_LOCKED)
        {
            MmUnlockPages(currentMdl);
        }
        IoFreeMdl(currentMdl);
    }
}
```

If the physical pages that are described by an MDL in the chain are locked, the example function calls the **MmUnlockPages** routine to unlock the pages before it calls **IoFreeMdl** to free the MDL. However, the example function does not need to explicitly unmap the pages before it calls **IoFreeMdl**. Instead, **IoFreeMdl** automatically unmaps the pages when it frees the MDL.

For a summary of the system routines that allocate, free, and manage MDLs, see **Address Mappings and MDLs**.

Send comments about this topic to Microsoft