ASPI Router for OS/2

Daniel Dorau (woodst@cs.tu-berlin.de)

Version 1.01 – June 1997

Licence and Warranty

Contents

1 Overview 3
1.1 Why ASPI Router? 3
1.2 Why ASPI? 4
1.3 What does ASPI Router? 4
1.4 The ASPILIB library 4
1.5 Current limitations 4

2 Interface to ASPI Router 4
2.1 Opening the driver 4
2.2 Semaphore initialization 5
2.3 Buffer initialization 5
2.4 Host adapter inquiry 6
2.5 Get Device Type 7
2.6 Abort SRB 7
2.7 Reset SCSI device 8
2.8 Issuing SCSI commands 8
2.9 Closing the driver 9
2.10 Summary of ASPI Rounter function codes 10

3 The ASPILIB library 10
3.1 Initializing the library 10
3.2 Using library funtions 10
3.3 Closing the library 10

4 Other Information 11
4.1 People I want to thank 11
4.2 Contact, Reporting bugs 11

1 Overview

1.1 Why ASPI Router?

In OS/2 it is not possible for Ring-3 applications to call `OS2SCSI.DMD` or `OS2ASPI.DMD` directly, therefore any application which wants to access the SCSI bus has to provide its own driver to access the SCSI bus via `OS2SCSI.DMD` or `OS2ASPI.DMD`. Since many users want to write their own SCSI applications but shy away from writing their own driver, I decided to write a simple driver which provides access to `OS2ASPI.DMD` for Ring-3 applications. Since it only routes SRBs (SCSI request blocks) to `OS2ASPI.DMD` I named it “ASPI Router”.

1.2 Why ASPI?

Besides ASPI, in OS/2 there also exists the possibility to issue SCSI commands through `OS2SCSI.DMD`. So why did I decide to use `OS2ASPI.DMD`?
I decided to use ASPI because a) it seemed to be easier to use for the application developer as well as for the device driver developer (me) and b) I hope that libraries for other platforms using ASPI can be ported more easily.

1.3 What does ASPI Router?

The interface of ASPI Router merely is an extension of the interface of `OS2-ASPI.DMD` which expects a SRB and a callback function to call when the SRB was processed. ASPI Router simply takes this SRB from your application via `DosDevIOCtl` and calls `OS2ASPI.DMD` with this SRB after conversion of virtual to physical addresses. Additionaly, ASPI Router provides a callback function for `OS2ASPI.DMD`. If this function gets called by `OS2ASPI.DMD`, ASPI Router posts an event semaphore opened by your application before. So before ASPI Router can process the first SRB, the application has to pass the handle of this semaphore to the driver. ASPI Router closes its own handle of the semaphore automatically when the application closes the driver.

1.4 The ASPILIB library

The accompaning library consisting of the files `aspilib.cpp`, `aspilib.h` and `srb.h` is intended as an example of how to use the ASPI Router device driver. You may use this library or parts of it in your own programs or modify parts of it so that it meets your requirements.

1.5 Current limitations

ASPI Router does not support scatter/gather or SRB linking. The maximum size of data transferred per SRB is 64K (=65536 bytes). Only one application can use this driver at a time. Therefore you need to open it with the `OPEN_SHARE_DENYREADWRITE` flag to prevent other applications from opening the driver while already in use.

2 Interface to ASPI Router

All functions of ASPI Router can be accessed with `DosDevIOCtl` after opening the driver with `DosOpen`. The name of the driver handle is `ASPIROU$`.

2.1 Opening the driver

The driver can be opened with the following call:

```
ULONG rc;                                       // return value
ULONG ActionTaken;                              // return value

rc = DosOpen((PSZ) "aspirou$",                  // open driver
             &driver_handle,
             &ActionTaken,
             0,
             0,
             FILE_OPEN,
             OPEN_SHARE_DENYREADWRITE |
             OPEN_ACCESS_READWRITE,
             NULL);
```

2.2 Semaphore initialization

Before any SRBs can be processed, ASPI Router needs a handle of an event semaphore. ASPI Router will post this semaphore for every SRB which has been processed. To create this semaphore and to send its handle to ASPI Router, the following code can be used:

```
ULONG  rc;                                    // return value
USHORT openSemaReturn;                        // return value
HEV    postSema;                              // event
                                              // semaphore
unsigned long cbreturn;
unsigned long cbParam;

rc = DosCreateEventSem(NULL, &postSema,       // create event
                       DC_SEM_SHARED, 0);     // semaphore

rc = DosDevIOCtl(driver_handle, 0x92,         // pass
                 0x03,                        // semaphore
                 (void*) &postSema,           // handle to
                 sizeof(HEV), &cbParam,       // driver
                 (void*) &openSemaReturn,
                 sizeof(USHORT), &cbreturn);
```

If ASPI Router could open the semaphore, `openSemaReturn` will be set to zero. Otherwise an error occured.

2.3 Buffer initialization

After passing the semaphore handle to the driver you still need to do one step before you can issue SCSI commands. You have to initialize the buffer. You have to allocate a buffer in your application where the driver can access the data to be transferred. The driver has not only to convert virtual to physical addresses, it also has to ensure that the virtual address is in physical memory (e.g. not swapped out) and that it does not change its position in physical memory. To achieve this, it calls a helper function which locks the buffer down in memory. Another important point is that the buffer must not cross a 64K-boundary in physical memory. Your task is to allocate this buffer and pass its address to the driver. You can do this as follows:

```
PVOID  buffer                              // our buffer
ULONG  rc;                                 // return value
USHORT lockSegmentReturn;                  // return value
unsigned long cbreturn;
unsigned long cbParam;

rc = DosAllocMem(&buffer, bufsize,         // allocate the
                 OBJ_TILE | PAG_READ       // buffer at a
                 | PAG_WRITE               // 64K boundary
                 | PAG_COMMIT);
rc = DosDevIOCtl(driver_handle, 0x92,      // pass buffer
                 0x04, (void*) buffer,     // pointer
                 sizeof(PVOID),            // to driver
                 &cbParam,
                 (void*) &lockSegmentReturn,
                 sizeof(USHORT), &cbreturn);
```

If `DosDevIOCtl` returns with a non-zero value of `lockSegmentReturn` something went wrong with locking the buffer.

2.4 Host adapter inquiry

Do determine the number of host adapters available and to retrieve information about them you can issue the "Host adapter Inquiry" command. Refer to the "Storage Device Driver Reference" [1] to get more information about it. Here is an example:

```
ULONG rc;                           // return value
unsigned long cbreturn;
unsigned long cbParam;

SRBlock.cmd=SRB_Inquiry;            // host adapter inquiry
SRBlock.ha_num=ha;                  // host adapter number
SRBlock.flags=0;                    // no flags set

rc = DosDevIOCtl(driver_handle, 0x92, 0x02,
                 (void*) &SRBlock, sizeof(SRB), &cbParam,
```

```
                  (void*) &SRBlock, sizeof(SRB), &cbreturn);
```

2.5 Get Device Type

To get the device type there is the "Get Device Type" command. Refer to the "Storage Device Driver Reference" [1] to get more information about it. Here is an example:

```
ULONG rc;                                 // return value
unsigned long cbreturn;
unsigned long cbParam;

SRBlock.cmd=SRB_Device;                   // get device type
SRBlock.ha_num=0;                         // host adapter number
SRBlock.flags=0;                          // no flags set
SRBlock.u.dev.target=id;                  // target id
SRBlock.u.dev.lun=lun;                    // target LUN

rc = DosDevIOCtl(driver_handle, 0x92, 0x02,
                  (void*) &SRBlock, sizeof(SRB), &cbParam,
                  (void*) &SRBlock, sizeof(SRB), &cbreturn);
```

2.6 Abort SRB

With this command you tell `OS2ASPI.DMD` to abort the previous SRB. You have to specify a second SRB with which you pass the address of the SRB to be aborted to the driver. Aborting a SRB does not have to be successful. Watch the status values and sense code of the SRB you aborted. For further information about this command refer to the "Storage Device Driver Reference" [1]. Here is an example:

```
ULONG rc;                                 // return value
unsigned long cbreturn;
unsigned long cbParam;

AbortSRB.cmd=SRB_Abort;                   // abort SRB
AbortSRB.ha_num=0;                        // host adapter number
AbortSRB.flags=0;                         // no flags set
AbortSRB.u.abt.srb=&SRBlock;              // SRB to abort

rc = DosDevIOCtl(driver_handle, 0x92, 0x02,
                  (void*) &AbortSRB, sizeof(SRB), &cbParam,
                  (void*) &AbortSRB, sizeof(SRB), &cbreturn);
```

2.7 Reset SCSI device

A SCSI device can be reset with the "Reset SCSI device" command as described in the "Storage Device Driver Reference" [1]. Here is an example:

```
ULONG rc;                                   // return value
unsigned long cbreturn;
unsigned long cbParam;

SRBlock.cmd=SRB_Reset;                      // reset device
SRBlock.ha_num=0;                           // host adapter number
SRBlock.flags=SRB_Post;                     // posting enabled
SRBlock.u.res.target=id;                    // target id
SRBlock.u.res.lun=lun;                      // target LUN

rc = DosDevIOCtl(driver_handle, 0x92, 0x02,
                 (void*) &SRBlock, sizeof(SRB), &cbParam,
                 (void*) &SRBlock, sizeof(SRB), &cbreturn);
```

2.8 Issuing SCSI commands

SCSI commands can be issued by sending ASPI SRBs (SCSI Request Blocks) to the ASPI Router. The data structure definition of the SRB and it's constant definitions can be found in srb.h. The `cdb_st` field has to be filled with a valid SCSI command. Refer to the SCSI command reference [2] for a description of the different SCSI devices and their commands. An example of sending a SRB to ASPI Router follows below:

```
ULONG rc;                                       // return value
SRB   SRBlock;

unsigned long cbreturn;
unsigned long cbParam;

SRBlock.cmd=SRB_Command;                        // execute SCSI
                                                // command
SRBlock.ha_num=0;                               // host adapter
                                                // number
SRBlock.flags=SRB_Read | SRB_Post;              // data transfer,
                                                // posting enabled
SRBlock.u.cmd.target=6;                         // Target SCSI ID
SRBlock.u.cmd.lun=0;                            // Target SCSI LUN
SRBlock.u.cmd.data_len=512*transfer;            // # of bytes
                                                // transferred
SRBlock.u.cmd.sense_len=32;                     // length of
```

```
                                              // sense buffer
SRBlock.u.cmd.data_ptr=NULL;                  // pointer to
                                              // data buffer
SRBlock.u.cmd.link_ptr=NULL;                  // pointer to
                                              // next SRB
SRBlock.u.cmd.cdb_len=6;                      // SCSI command
                                              // length
SRBlock.u.cmd.cdb_st[0]=0x08;                 // read command
SRBlock.u.cmd.cdb_st[1]=1;                    // fixed length
SRBlock.u.cmd.cdb_st[2]=(transfer >> 16)      // transfer
                        & 0xFF;               // length MSB
SRBlock.u.cmd.cdb_st[3]=(transfer >> 8)       // transfer
                        & 0xFF;               // length
SRBlock.u.cmd.cdb_st[4]=transfer & 0xFF;      // transfer
                                              // length LSB
SRBlock.u.cmd.cdb_st[5]=0;

rc = DosDevIOCtl(driver_handle, 0x92, 0x02,
                (void*) &SRBlock, sizeof(SRB), &cbParam,
                (void*) &SRBlock, sizeof(SRB), &cbreturn);
```

After the SRB is being processed you may read the status fields out of the SRB and decide whether the SRB was processed successfully or not. Notice that the field `data_ptr` is always set to `NULL`. Since you initialize the data buffer before any SRBs can be processed the driver knows the correct physical address and inserts it automatically. Note that the valid bit in sense data does not always represent its current state. Refer to the target status field of the SRB to determine whether the sense data is valid or not.

2.9 Closing the driver

After you're done you have to close the driver. The driver will close its semaphore handle and unlock the buffer (otherwise you won't be able to free the memory). Then don't forget to close the semaphore in your app and free the buffer memory. This shows how it can be done:

```
ULONG  rc;                          // return value

rc = DosCloseEventSem(postSema);    // close event semaphore
rc = DosClose(driver_handle);       // close driver
rc = DosFreeMem(buffer);            // free our buffer
```

Be careful to close the driver before you free the memory. Attempting to free locked memory does not work.

2.10 Summary of ASPI Rounter function codes

ASPI Router provides the following functions:

Category	Function	Description
92h	02h	**Route SRB to ASPI driver** The driver expects the address of a SRB. The 'Data Buffer Pointer' field needs not to be filled.
92h	03h	**Pass semaphore handle** The driver expects the pointer to an event semaphore handle to notify the application that a SRB has be processed.
92h	04h	**Pass Data Buffer Pointer** The driver expects the pointer to a data buffer. The buffer must not cross a 64K-boundary.

3 The ASPILIB library

The ASPILIB library is written as a C++ object to be used with (hopefully) any C++ compiler. I developed it as an example library that shows how to use the driver. Since I need it for my tape drive, only functions I need for this application are implemented. If you need additional functions, add them.

3.1 Initializing the library

First, create a variable of the object type `scsiObj`. Then you have to call the `init(ULONG bufsize)` function to which you have to pass the size of the data buffer you wish to use. Note that the maximum size supported by the driver is 64K (=65536 bytes). If the call returns with `TRUE`, all went ok, otherwise something went wrong.

3.2 Using library funtions

After initializing the library you can use any public member functions of your SCSI object. Be careful to check if your calls return successful and check the error codes or sense data if necessary.

3.3 Closing the library

Before you quit your application you have to close the library. This is done by calling the `close()` function.

4 Other Information

4.1 People I want to thank

Big thanks go to Alger Pike (acp107@psu.edu) who wrote the Hello World Device Driver. Without this I wouldn't have known where to start.

4.2 Contact, Reporting bugs

If you use my driver and/or the library, please tell me what you think about it. Furthermore you're welcome to report any bugs to me you encounter. My email address is `woodst@cs.tu-berlin.de`.
If you want to use PGP for your email, you can retrieve my public key by sending me a mail with `send pgp key` in the subject line.

References

[1] Storage Device Driver Reference, part of IBM's Device Driver Kit (DDK), freely available at `http://service.boulder.ibm.com/ddk`

[2] SCSI command reference, for those who have Internet access: I found one in the WWW at `http://abekas.com:8080/SCSI2/`