รายการอ้างอิง

Aitken , P.G. 1989. **Disk Interfaces for the High End**. PC TECH JOURNAL 1989 (February 1989) : 78 - 86.

American National Standards Institute, Inc. 1986. **Small Computer System Interface (SCSI) - ANSI X3.131-1986**. n.p.

Bach , M.J.. 1986. **THE DESIGN OF THE UNIX OPERATING SYSTEM**. New York : Prentice - Hall Press New York.

Buddine , L. and Young , E.. 1987. **THE BRADY GUIDE TO CD-ROM**. New York : Prentice - Hall Press New York.

Egan , J.I. and Teixeira , J.T.. 1988. **WRITING A UNIX DEVICE DRIVER**. New York: John Wiley & Sons.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. 1988. **Information processing - Volume and File structure of CD-ROM for Information interchange (ISO 9660-1988-09-01)**. n.p.

Jansson , P.. 1987. **Patterning CD-ROM**. PC TECH JOURNAL 1987 (July 1987) : 163-173.

The Santa Cruz Operation, Inc. 1989. **SCO UNIX SYSTEM V/386 - Development System : Device Driver Writer's Guide**. n.p.

ภาคผนวก ก.

การติดตั้งโปรแกรมย่อยควบคุมอุปกรณ์สากลเข้าในระบบปฏิบัติการ SCO UNIX SYSTEM V/386

ระบบปฏิบัติการ SCO UNIX SYSTEM V/386 มีระบบที่จะดำเนินการสร้างตารางโครงแบบ (configuration table) และติดตั้งโปรแกรมควบคุมอุปกรณ์ชุดใหม่เข้าไปในเคอร์เนลให้โดยผู้พัฒนาโปรแกรมควบคุมอุปกรณ์ต้องปฏิบัติตามขั้นตอนที่กำหนดไว้ให้ดังนี้

1. พัฒนาโปรแกรมย่อยควบคุมอุปกรณ์ โดยมี entry point ต่างๆ ให้ครบถ้วน โดยตั้งชื่อนำหน้า (Driver prefix) โปรแกรมย่อยควบคุมอุปกรณ์ตามข้อกำหนดของระบบปฏิบัติการดังนี้ คือ

ถ้าเป็นโปรแกรมสำหรับอุปกรณ์ประเภท	ต้องตั้งชื่อนำหน้าว่า
Bootable rootable hard disk drive	hd
Hard disk drive	Sdsk
Robot device for changing storage media	Smed
Optical memory device	Sopt
Processor device	Spr
Cartridge disk,read-only memory	Srom
Scanner device	Sscn
Tape drive	Stp
Communication device	Stty
Write-once-read-many drive (WORM)	Swrm

หลังจากนั้น compile โปรแกรมโดยใช้ option

-c เพื่อ create linkable object file

-K เพื่อ disable stack probes

-Zp4 เพื่อ Align program on quad-word boundaries

-DINKERNEL Required for conditional code in standard header files

-M3 ในกรณีที่ใช้ 80386 processor

2. หา Major device number ที่ว่าง โดยใช้คำสั่ง

/etc/conf/cf.d/configure -j NEXTMAJOR

3. เพิ่มรายการในแฟ้มข้อมูลระบบ ดังนี้

3.1 แฟ้มข้อมูล /etc/conf/cf.d/mdevice

1	2	3	4	5	6	7	8	9
Srom	Iocr	iHb	Srom	51	0	1	1	-1

คำอธิบาย :

ฟิลด์ 1 : Device name ชื่อ device

ฟิลด์ 2 : Function list แสดง entry point ต่างๆ ของโปรแกรมย่อยควบคุมอุปกรณ์ มีข้อมูลได้ดังนี้

o = open routine

c = close routine

r = read routine

w = write routine

i = ioctl routine

s = startup routine

I = init routine

h = halt routine

p = poll routine

E = enter routine

(ถ้าเป็น Block device driver จะมี strategy routine และ print routine เป็น default อยู่แล้ว)

ฟิลด์ 3 : Characteristics of driver แสดงคุณลักษณะของอุปกรณ์ มีข้อมูลได้ดังนี้

i = โปรแกรมย่อยควบคุมอุปกรณ์นี้ installable

c = อุปกรณ์นี้เป็น character device

b = อุปกรณ์นี้เป็น block device

t = อุปกรณ์นี้เป็น tty

o = อุปกรณ์นี้มีข้อมูลใน sdevice เพียงรายการเดียว

r = จะต้องมีอุปกรณ์นี้ในทุก configuration ของ kernel
(สำหรับ device driver ที่มากับ Base system)

S = โปรแกรมย่อยควบคุมอุปกรณ์นี้เป็นแบบ STREAMS

H = โปรแกรมย่อยควบคุมอุปกรณ์นี้ใช้ควบคุมอุปกรณ์จริง (มี Hardware)

G = อุปกรณ์นี้ไม่ได้ใช้ interrupt ถึงแม้จะมีการระบุ interrupt line ใน sdevice (ใช้ในกรณีที่อุปกรณ์นี้จะเชื่อมกับกลุ่มของอุปกรณ์)

D = ระบุไว้แสดงว่าโปรแกรมย่อยควบคุมอุปกรณ์นี้ใช้ DMA channel ร่วมกับโปรแกรมย่อยควบคุมอุปกรณ์อื่นได้

O = ระบุไว้แสดงว่าโปรแกรมย่อยควบคุมอุปกรณ์นี้มีช่วง IOA (I/O Address) คาบเกี่ยวกับอุปกรณ์อื่น

ฟิลด์ 4 : Handle prefix ฟิลด์นี้เตรียมไว้สำหรับใช้เป็นชื่อนำหน้าโปรแกรมย่อยต่างๆของโปรแกรมควบคุมอุปกรณ์นี้
(มีความยาวได้ไม่เกิน 4 ตัวอักษร)

ฟิลด์ 5 : Block Major number ฟิลด์นี้ควรใส่เป็น "0" ใน DSP (Distribute softwared Package) ถ้าอุปกรณ์เป็น block device โปรแกรม idinstall จะเป็นโปรแกรมที่กำหนดค่าให้เอง

ฟิลด์ 6 : Character Major number ฟิลด์นี้ควรใส่เป็น "0" ใน DSP (Distribute softwared Package) ถ้าอุปกรณ์เป็น character device โปรแกรม idinstall จะเป็นโปรแกรมที่กำหนดค่าให้เอง

ฟิลด์ 7 : Minimum units ฟิลด์นี้จะระบุจำนวนอุปกรณ์ที่น้อยที่สุดที่สามารถกำหนดใน sdevice

ฟิลด์ 8 : Maximum units ฟิลด์นี้จะระบุจำนวนอุปกรณ์ที่มากที่สุดที่สามารถกำหนดใน sdevice

ฟิลด์ 9 : DMA channel ฟิลด์นี้ใส่ DMA channel ที่จะถูกใช้โดยอุปกรณ์นี้ ถ้าอุปกรณ์ไม่ใช้ DMA channel ให้ใส่ -1 ในฟิลด์นี้
(อุปกรณ์หลายตัวสามารถใช้ DMA channel เดียวกันได้)

3.2 /etc/conf/cf.d/sdevice.d/Srom

1	2	3	4	5	6	7	8	9	10
Srom	Y	1	5	0	0	0	0	0	0

คำอธิบาย :

ฟิลด์ 1 : Device name ชื่อ device ที่ใช้เป็น prefix ของทุก entry point (ต้องตรง กับชื่อในฟิลด์แรกของ device เดียวกันที่ปรากฏใน mdevice) สำหรับโปรแกรมย่อยควบคุมอุปกรณ์ของ CD-ROM ใช้ชื่อ "Srom"

ฟิลด์ 2 : Configure ระบุว่าจะ install device ใน kernel หรือไม่ สำหรับโปรแกรมย่อยควบคุมอุปกรณ์ CD-ROM ใส่ "Y"

ฟิลด์ 3 : Unit แสดงจำนวนของ subdevices ที่ติดต่อกับ controller ของ device นี้ สำหรับโปรแกรมย่อยควบคุมอุปกรณ์ CD-ROM ใส่ "1"

ฟิลด์ 4 : Ipl คือ interrupted priority level ของ โปรแกรมย่อยควบคุมอุปกรณ์ CD-ROM ในที่นี้ใช้เหมือนฮาร์ดดิสค์ คือ "5"

ฟิลด์ 5 : Type แสดงประเภทของ interrupt scheme ของอุปกรณ์นั้น ในกรณีของโปรแกรมย่อยควบคุมอุปกรณ์ SCSI บนระบบปฏิบัติการ SCO UNIX SYSTEM V/386 จะไม่รับ interrupted line โดยตรง เนื่องจาก Host adapter driver จะเป็นโปรแกรมที่รับ interrupted line ของ device ไว้แล้วจึงส่งต่อการทำงานไปยัง SCSI device driver อีกครั้งหนึ่งดังนั้นสำหรับโปรแกรมย่อยควบคุมอุปกรณ์ CD-ROM ใส่ "0"

ฟิลด์ 6 : Vector แสดง interrupted vector number ของอุปกรณ์ สำหรับโปรแกรมย่อยควบคุมอุปกรณ์ CD-ROM ใส่ "0" เหตุผลเดียวกับฟิลด์ 5

ฟิลด์ 7 : SIOA (Start IO Address) สำหรับโปรแกรมย่อยควบคุมอุปกรณ์ CD-ROM ใส่ "0" เหตุผลเดียวกับฟิลด์ 5

ฟิลด์ 8 : EIOA (End IO Address) สำหรับโปรแกรมย่อยควบคุมอุปกรณ์ CD-ROM ใส่ "0" เหตุผลเดียวกับฟิลด์ 5

ฟิลด์ 9 : SCMA (Start controller Memory address) สำหรับโปรแกรมย่อยควบคุมอุปกรณ์ CD-ROM ใส่ "0" เหตุผลเดียวกับฟิลด์ 5

ฟิลด์10 : ECMA (End controller Memory Address) สำหรับโปรแกรมย่อยควบคุมอุปกรณ์ CD-ROM ใส่ "0" เหตุผลเดียวกับฟิลด์ 5

3.3 /etc/conf/cf.d/mscsi

1	2	3	4	5
ad	Srom	0	4	0

คำอธิบาย :

ฟิลด์ 1 : Host adapter driver prefix ในที่นี้คือ ad (Adaptec Host adapter)

ฟิลด์ 2 : Supported device driver name ชื่อ โปรแกรมย่อยควบคุมอุปกรณ์ CD-ROM

ฟิลด์3 :Host Adapter number หมายเลขของ Host adapter (SCO UNIX SYSTEM V/386 ยอมให้มี Host Adapter ได้ 2 การ์ด) ในกรณีนี้มีเพียงการ์ดเดียวใส่ "1"

ฟิลด์ 4 : SCSI bus device ID number คือ SCSI ID ของอุปกรณ์

ฟิลด์ 5 : SCSI bus device LUN คือ Logical unit number ของอุปกรณ์บน SCSI ID ที่ระบุ

3.4 เพิ่มแฟ้มข้อมูลชื่อ /etc/conf/node.d/Srom ซึ่งมีข้อมูลดังนี้

1	2	3	4
Srom	cd0	b	0

คำอธิบาย :

ฟิลด์ 1 : Device Driver Prefix name ชื่อนำหน้าโปรแกรมย่อยควบคุมอุปกรณ์ทุกโปรแกรม

ฟิลด์ 2 : Device node name ชื่อแฟ้มข้อมูลพิเศษที่จะใช้เป็นช่องทางติดต่อกับอุปกรณ์

ฟิลด์ 3 : Device type หมายถึงประเภทของแฟ้มอุปกรณ์ ว่าเป็นแบบบล็อกดีไวซ์ หรือคาแรคเตอร์ดีไวซ์

ฟิลด์ 4 : Minor unit number หมายเลขอุปกรณ์ย่อย

4. ทำการสำรองแฟ้มข้อมูลต่างๆ ใน directory ต่อไปนี้
   - /etc/conf/cf.d/mdevice
   - /etc/conf/sdevice.d
   - /etc/conf/pack.d
5. สร้าง directory /etc/conf/pack.d/Srom
6. ใส่ Object code ทั้งหมดเข้าไปใน directory นี้
7. สร้าง kernel ใหม่โดยใช้ shell script
   /etc/conf/cf.d/link_unix ซึ่งบรรจุ shell script ดังนี้

ภาคผนวก ข.

Source code ของโปรแกรมย่อยควบคุมอุปกรณ์ต่างๆ และแฟ้มรายการกำหนด (Header file)

DRIVER.C Friday, May 3, 1991 1:25 pm

```
/*
*************************************************************************
****                                                                 ****
****      T H E  D E V E L O P M E N T  O F  C D - R O M             ****
****                                                                 ****
****      D E V I C E  D R I V E R  S U B P R O G R A M S            ****
****                                                                 ****
****-------------------------------------------------------------------****
****                                                                 ****
****   Thesis creator :  Anuchai  Triraroungchaisri                  ****
****   Thesis advisor :  Associated Professor Somchai  Thayarnyong   ****
****                                                                 ****
****   Date submitted :  5 May 1991                                  ****
*************************************************************************
*/
/*
*************************************************************************
****                                                                 ****
****       G E N E R A L  I N C L U D E  F I L E S                   ****
****                                                                 ****
*************************************************************************
*/
#include "sys/conf.h"
#include "sys/cmn_err.h"
#include "sys/types.h"
#include "sys/param.h"
#include "sys/sysmacros.h"
#include "sys/systm.h"
#include "sys/buf.h"
#include "sys/iobuf.h"

#include "sys/cram.h"

#include "sys/dir.h"
#include "sys/file.h"
#include "sys/signal.h"
#include "sys/seg.h"
#include "sys/page.h"
#include "sys/immu.h"
#include "sys/region.h"
#include "sys/proc.h"
#include "sys/x.out.h"
#include "sys/user.h"
#include "sys/errno.h"
#include "sys/open.h"
#include "sys/cmn_err.h"
#include "sys/dma.h"
#include "sys/vendor.h"
#include "sys/fcntl.h"
```

```
/*
*****************************************************************************
****                                                                     ****
****            S C S I   I N C L U D E   F I L E S                      ****
****                                                                     ****
*****************************************************************************
*/
#include "sys/scsi.h"
#include "sys/scsisup.h"
#include "sys/adaptec.h"
#include "Srom.h"

/*
*****************************************************************************
****                                                                     ****
****            E X T E R N A L   V A R I A B L E S                      ****
****                                                                     ****
*****************************************************************************
*/
unsigned int    cdbopened = 0;              /* count of opens */

REQ_IO          req_io[MAX_REQ_BLCK],
                *head_req_io;

struct iobuf  Sromtab;
struct buf    rSrombuf;

/* Device driver buffer for maintain the diff. of CD block and BUFFER size */
struct  buf  Drv_buf_head,*drv_bp;
char    Drv_buf_data[CD_BLOCK_SIZE+1];
int     buf_in_use = 0;

daddr_t     max_blck;           /* Maximum block number        */
daddr_t         last_blck;                  /* Last block access by user    */

DEVCFG      *ptr_d_cfg;         /* Pointer to device config data       */
HACFG       *ptr_ha_cfg;        /* Pointer to host adpater data        */
INQ_DATA        *ptr_inq_data;
MS_DATA         *ptr_ms_data;

/*
*****************************************************************************
****                                                                     ****
****            C D - R O M   D E V I C E   D R I V E R                  ****
****                                                                     ****
*****************************************************************************
*/
Srominit()
```

```
{
    int i;
    for ( i = 0 ; i < MAX_REQ_BLCK ; i++ ) {
                if ( (i+1) < MAX_REQ_BLCK ) {
                req_io[i].req_forw      =   &req_io[i+1];
                } else {
                req_io[i].req_forw              =   NULLPTR;
                }
    }

    head_req_io             =   &req_io[0];

        /* Set up Driver buffer pointer         */
        drv_bp          =       &Drv_buf_head;

        /* Set Driver buffer data                       */
        Drv_buf_head.b_un.b_addr        =       Drv_buf_data;

    return;
}

Sromintr(ptr_req)
REQ_IO  *ptr_req;
{
        register        struct  buf *bp;
        caddr_t         *ptr_drv_data,*ptr_bp_data;
        RD_CAP_DATA     *ptr_rd_cap;

        extern REQ_IO   *free_req_io();

        if ( ptr_req->host_sts )
                Chk_host_sts(ptr_req);

        if ( ptr_req->target_sts )
                Chk_target_sts(ptr_req);

        if ( !ptr_req->host_sts && !ptr_req->target_sts ) {
                switch ( ptr_req->opcode ) {
                        case READ_CMD:          /* 0x08 */
                                bp      =       ptr_req->rbuf;

                                /* Transfer parameters from drv_bp to bp */
                                trans_buf(drv_bp,bp);

                                /*      Copy data from Driver buffer to bp      */
                                ptr_bp_data             =       (caddr_t *) bp->b_un.b_addr;

                                /*  Check whether data is upper or lower section in buffer */
                                if ( (bp->b_blkno/2)%2  ==      0 ) {
```

```
                        ptr_drv_data    =       (caddr_t *)     &Drv_buf_data[0];
                    } else {
                        ptr_drv_data    =       (caddr_t *)     &Drv_buf_data[1024];
                    }

                    /* Transfer data from Driver buffer to buffer   */
                    bcopy(ptr_drv_data,ptr_bp_data,1024);

                    /* Remember last block access by user */
                    last_blck       =       ptr_req->data_blk;

                    /* Setup parameters in iobuf structure          */
                    Sromtab.b_actf          =       bp->av_forw;
                    Sromtab.b_errcnt        =       0;
                    Sromtab.b_active        =       0;
                    iodone(bp);
                    break;

                case  READ_CAP_CMD:     /* 0x25 */
                    ptr_rd_cap      =  (RD_CAP_DATA *) ptok(ptr_req->data_ptr);
                    max_blck        =       scsi_stol(ptr_rd_cap->blck_addr);
                    break;
            }

            ptr_req->internal       =       0;
            wakeup(ptr_req);

            buf_in_use                  =       0;
            wakeup(&buf_in_use);

        } else {
            switch ( ptr_req->opcode ) {
                case READ_CMD:          /* 0x08 */
                    Sromtab.b_actf          =       bp->av_forw;
                    Sromtab.b_errcnt        =       0;
                    Sromtab.b_active        =       0;
                    bp->b_flags             |=      B_ERROR;
                    bp->b_error                     =       EIO;
                    bp->b_resid                     =       bp->b_bcount;
                    iodone(bp);
                    break;

                case READ_CAP_CMD:  /* 0x25 */
                    max_blck        =       0;
                    break;
            }
            ptr_req->internal       =       0;
            wakeup(ptr_req);
```

```
                buf_in_use                      =       0;
                wakeup(&buf_in_use);

                return;
        }

        /* Check for another request */
        Sromstart();
}

Sromopen(dev,mode,flag)
dev_t   dev;
int     mode,flag;
{
        int             unit;
        REQ_IO                  *ptr_req;
        extern REQ_IO   *get_req_io();
        extern REQ_IO   *free_req_io();
        union scsi_cdb  *ptr_scsi_cmd;
        SENSE_DATA              *ptr_scsi_sense;
        EXTENSION               extendata;
        long                    array[2];

        /* Temporary declaration        */
        int                             i;

        /*      Variable for calling send_scsi_cmd                      */
        char    inquiry_buf[INQUIRY_LEN];
        char    sense_buf[SENSE_LEN];
        char    rd_cap_buf[RD_CAP_LEN];
        char    void_buf[VOID_LEN];

        /*    Variable for calling Host adapter infomation      */
        HA_INFO ha_info_data,*ptr_ha_info;

        /*        Variable for calling Read capacity command    */
        RD_CAP_DATA     *ptr_rd_cap;

        /*    Variable for receive old priority level           */
        register        lev;

        /* unit is minor number of device node */
        unit = minor(dev);

        /* get configure data of HOST ADAPTER ADAPTEC */
        ptr_ha_cfg      =  &ahacfg[unit];

        /* find configuration data of device */
        ptr_d_cfg = scsi_getdev(unit,&Sromcfg[0],Sromopen,"Srom");
```

```
        if ( ptr_d_cfg == NULLPTR ){
                seterror(ENODEV);
                return;
        }

        /* check for correct open mode */
        if ( (mode & 3) != 1 ) {
                seterror(EACCES);
                return;
        }

        /* check for correct device type */
        if ( flag == BLKDEV ) {
                if ( cdbopened == 0 ) {
                        printcfg("Srom",0x0,0x0,0,0,"type=S ha=%d id=%d lun=%d",
                                ptr_d_cfg->ha_num,ptr_d_cfg->id,ptr_d_cfg->lun);
                }
                cdbopened++;            /* increment count for open */
        } else {
                cmn_err(CE_WARN,"Err: Not supported character interface..\n");
                seterror(ENOTBLK);
                return;
        }

        /* Test if necessary to call SCSI_INIT and READ CAPACITY command        */
        if ( cdbopened  == 1 ) {

                Init_host_adapter();

                /* Get a REQUEST BLOCK  */
                ptr_req          = get_req_io();
                ptr_scsi_cmd     = &ptr_req->scsi_cmd;

                /* Initial I/O request block    */
                Init_io_req(ptr_req,ptr_ha_cfg,ptr_d_cfg);

                /* Call Read capacity command loop until no attention   */
                Init_array(array , rd_cap_buf , RD_CAP_LEN);

                ptr_scsi_sense  = (SENSE_DATA *)(&ptr_scsi_cmd->raw[10]);
                ptr_scsi_sense->key     =       SS_UATT;
                while (ptr_scsi_sense->key      ==      SS_UATT) {
                        if(send_scsi_cmd(ptr_req,READ_CAP_CMD,array,2,ptr_d_cfg)==SEND_CMD_ERR){
                                cmn_err(CE_WARN,"HOST: Fault Detect in HOST ADAPTER");
                                seterror(EIO);
                                break;
                        }
                        if ( ptr_scsi_sense->key == SS_UATT )
                                Init_host_adapter();
```

```
                }

                /* Free a REQUEST BLOCK to pool */
                free_req_io( ptr_req );
        }
}

Sromclose()
{
        cdbopened = 0;
}

Sromstrategy(bp)
register        struct  buf     *bp;
{
        caddr_t         *ptr_drv_data,*ptr_bp_data;
        register        lev;

        /* Check valid block */
        if (( (bp->b_blkno/4) < 0 ) || ( (bp->b_blkno/4) > max_blck ) ) {
                bp->b_flags     |=      B_ERROR;
                bp->b_error     =       ENXIO;
                bp->b_resid     =       bp->b_bcount;
                iodone(bp);
                return;
        }

        /*  Check if buffer available for use   */
        lev     =       spl5();
        while ( buf_in_use )
                sleep(&buf_in_use,PRIBIO);
        splx(lev);

        buf_in_use      =       1;

        /*  Check for access the same block as last time */
        if ( (bp->b_blkno/4) == last_blck ) {

                ptr_bp_data             =       (caddr_t *) bp->b_un.b_addr;
                /*  Check whether data is upper or lower section in buffer */
                if ( (bp->b_blkno/2)%2  ==      0 ) {
                        ptr_drv_data    =       (caddr_t *)     &Drv_buf_data[0];
                } else {
                        ptr_drv_data    =       (caddr_t *)     &Drv_buf_data[1024];
                }

                /* Transfer data from Driver buffer to buffer   */
                bcopy(ptr_drv_data,ptr_bp_data,1024);
```

```
                /* complete request and send data to user              */
                iodone(bp);

                buf_in_use      =       0;
                wakeup(&buf_in_use);
                return;
        }

        buf_in_use      =       0;
        wakeup(&buf_in_use);

        /* queue the request to the Sromtab            */
        /* start the device if necessary               */
        bp->b_flags             &=      ~B_DONE;                /* reset done flag */

        lev     =       spl5();
        disksort(&Sromtab,bp);
        if ( Sromtab.b_active   ==   0 )
                Sromstart();
        splx(lev);

}

Sromstart()
{
        register struct buf     *bp;
        register                lev;
        caddr_t                 *ptr_drv_data,*ptr_bp_data;
        union  scsi_cdb *ptr_scsi_cmd;
        REQ_IO                  *ptr_req;
        SENSE_DATA              *ptr_scsi_sense;
        long                    array[2];

        /* Check for any buffer left for process       */
        if (( bp = Sromtab.b_actf ) == NULL ) {
                Sromtab.b_active        =       0;
                return;
        }

        Sromtab.b_active        =       1;

        /*  Check if buffer available for use   */
        lev     =       spl5();
        while ( buf_in_use )
                sleep(&buf_in_use,PRIBIO);
        splx(lev);

        buf_in_use      =       1;
```

```
        /* Transfer data from bp to drv_bp      */
        trans_buf(bp,drv_bp);

        /* Get a REQUEST BLOCK  */
        ptr_req         = get_req_io();
        ptr_scsi_cmd    = &ptr_req->scsi_cmd;

        Init_io_req(ptr_req,ptr_ha_cfg,ptr_d_cfg);
        ptr_req->data_len       =       CD_BLOCK_SIZE;
        ptr_req->data_ptr       =       vtop(paddr(drv_bp),drv_bp->b_proc);
        ptr_req->data_blk       =       drv_bp->b_blkno/4;
        ptr_req->rbuf           =       bp;

        /* Call Read command */
        ptr_scsi_sense  = (SENSE_DATA *)(&ptr_scsi_cmd->raw[6]);
        ptr_scsi_sense->key     =       SS_UATT;
        while (ptr_scsi_sense->key      ==      SS_UATT) {
                if (send_scsi_cmd(ptr_req,READ_CMD,array,2,ptr_d_cfg) == SEND_CMD_ERR) {

                        lev     =       spl5();
                        Sromtab.b_actf          =       bp->av_forw;
                        Sromtab.b_errcnt        =       0;
                        Sromtab.b_active        =       0;
                        bp->b_flags             |=      B_ERROR;
                        bp->b_error                     =       EIO;
                        bp->b_resid                     =       bp->b_bcount;
                        iodone(bp);
                        splx(lev);

                        /*  set buffer available for use  */
                        buf_in_use              =       0;
                        wakeup(&buf_in_use);

                        return;
                }
                if ( ptr_scsi_sense->key == SS_UATT )
                        Init_host_adapter();
        }

        /* Free a REQUEST BLOCK to pool */
        free_req_io( ptr_req );
}

Sromread()
{
        return;
}

Sromioctl()
```

```
{
        return;
}

Sromprint()
{
        return;
}

/*
*****************************************************************************
****                                                                     ****
****             U T I L I T Y   F U N C T I O N S                       ****
****                                                                     ****
*****************************************************************************
*/

/*  Init_host_adapter() :
**      For INITIAL and PREPARE host adapter for subsequence command
*/
Init_host_adapter()
{
        REQ_IO                  *ptr_req;
        extern REQ_IO   *get_req_io();
        extern REQ_IO   *free_req_io();
        union scsi_cdb  *ptr_scsi_cmd;

        /* Get a REQUEST BLOCK  */
        ptr_req         = get_req_io();
        ptr_scsi_cmd    = &ptr_req->scsi_cmd;

        /* Initial I/O request block    */
        Init_io_req(ptr_req,ptr_ha_cfg,ptr_d_cfg);

        /* Call SCSI_INIT to prepare the adapter to send commands       */
        ptr_req->req_type = SCSI_INIT;
        ptr_req->hacmd    = SCSI_NO_INFO;

        while ((*ptr_d_cfg->adapter_entry)(ptr_req) ) {
                ++Sromtab.b_errcnt;
                if ( Sromtab.b_errcnt = MADRETRY ) {
                        cmn_err(CE_WARN,"HOST: Fault Detect in HOST ADAPTER");
                        seterror(EIO);
                        Sromtab.b_errcnt        =       0;
                        free_req_io( ptr_req );
                        return;
                }
        }
```

```
        /* Free a REQUEST BLOCK to pool */
        free_req_io( ptr_req );
}

/*  Init_io_req() :
**      For INITIAL and PREPARE some fields of request block
*/
Init_io_req(ptr_req,ptr_ha_cfg,ptr_d_cfg)
REQ_IO  *ptr_req;
HACFG   *ptr_ha_cfg;
DEVCFG  *ptr_d_cfg;
{
        ptr_req->req_type       =       0;
        ptr_req->ext_p          =       0;
        ptr_req->opcode         =       0;
        ptr_req->id                     =       ptr_d_cfg->id;
        ptr_req->lun            =       ptr_d_cfg->lun;
        ptr_req->ha_num         =       ptr_d_cfg->ha_num;
        ptr_req->dir            =       0;
        ptr_req->cmdlen         =       0;

        ptr_req->data_len       =       0;
        ptr_req->data_ptr       =       0;
        ptr_req->data_blk       =       0;

        ptr_req->link_ptr       =   0;
        ptr_req->link_id        =       0;
        ptr_req->sense_len      =       0;
        ptr_req->scsi_sense     =       0;

        ptr_req->req_id         =       0;
        ptr_req->internal       =       0;
        ptr_req->io_intr        =       Srcmintr;
        ptr_req->ctlr           =       0;
        ptr_req->req_status     =       0;
        ptr_req->adapter        =       0;
        ptr_req->r_count        =       0;
        ptr_req->hacmd          =       0;
        ptr_req->rbuf           =       0;
}

/*  Init_array() :
**      For INITIAL and PREPARE array for sending SCSI command
*/
Init_array(array,inquiry_buf,data_len)
long    array[];
char    inquiry_buf[];
int             data_len;
{
```

```
        array[0]        = (long) ktop(inquiry_buf);
        array[1]        = (long) data_len;
}

/* get_req_io() :
**      For get a REQUEST IO BLOCK from request block poll
*/
REQ_IO *
get_req_io()
{
        REQ_IO          *ptr_req;
        register        lev;            /* Interrupted priority level */

        while ( head_req_io == NULLPTR )
                sleep( &head_req_io, PRIBIO );

        lev                             =       spl5();
        ptr_req                         =       head_req_io;
                head_req_io                     =               head_req_io->req_forw;
        splx(lev);

        return( ptr_req );
}

/* free_req_io() :
**      For release a REQUEST IO BLOCK to request blaock poll
*/
REQ_IO *
free_req_io(ptr_req)
REQ_IO  *ptr_req;
{
        register        lev;            /* Interrupted priority level */

        lev     =       spl5();
        ptr_req->req_forw   =   head_req_io;
                head_req_io                     =       ptr_req;
        splx(lev);

        wakeup( &head_req_io );
                return;
}

/* send_scsi_cmd() :
**      For prepare request io block and command block
**      (**  REPLACE SCO scsi_get_gen_cmd()  **)
*/
int
send_scsi_cmd(ptr_req,cmd,args,nargs,ptr_d_cfg)
REQ_IO  *ptr_req;
```

```
int     cmd,nargs;
paddr_t *args;
DEVCFG  *ptr_d_cfg;
{
        int                     size,
                                i;
        register        lev;
        char            swaparray[4];
        char            *ptr_char;
        union scsi_cdb  *ptr_scsi_cmd   = &ptr_req->scsi_cmd;

        bzero(ptr_scsi_cmd,sizeof(*ptr_scsi_cmd));
        ptr_scsi_cmd->six.opcode        = cmd;
        ptr_req->req_type       = SCSI_SEND;

        switch(cmd) {
        case READ_CMD:          /* 0x08 */
                ptr_req->opcode = READ_CMD;
                ptr_req->cmdlen = sizeof ptr_req->scsi_cmd.six;
                ptr_scsi_cmd->six.lun           =       ptr_req->lun;

                /*      Fill in data block number       */
                ptr_char                                        =       (char *)&ptr_req->data_blk;

                for ( i = 0; i < 4 ; i++ ) {
                        swaparray[i]    =       *ptr_char;
                        ptr_char++;
                }

                scsi_swap4(swaparray);

                ptr_scsi_cmd->six.misc          =       swaparray[1];
                ptr_scsi_cmd->six.data[0]       =       swaparray[2];
                ptr_scsi_cmd->six.data[1]       =       swaparray[3];

                /*  Fill in transfer length             */
                ptr_scsi_cmd->six.data[2]       =       1;
                break;

        case READ_CAP_CMD:              /* 0x25 */
                ptr_req->opcode         = READ_CAP_CMD;
                ptr_req->dir            = SCSI_IN;

                ptr_req->cmdlen = sizeof ptr_req->scsi_cmd.ten;
                ptr_scsi_cmd->ten.lun   =       ptr_req->lun;
                ptr_scsi_cmd->ten.block =       0;
                ptr_scsi_cmd->ten.length=       0;
                ptr_req->data_ptr               =       args[0];
                ptr_req->data_len               =       RD_CAP_LEN;
```

```
            break;

        case TEST_CMD:
            ptr_req->opcode = TEST_CMD;
            ptr_req->cmdlen = sizeof ptr_req->scsi_cmd.six;
            ptr_scsi_cmd->six.lun          =      ptr_req->lun;
            break;

        case SENSE_CMD:         /* 0x03 */
            ptr_req->opcode = SENSE_CMD;
            ptr_req->cmdlen = sizeof ptr_req->scsi_cmd.six;
            ptr_scsi_cmd->six.lun          =      ptr_req->lun;
            ptr_scsi_cmd->six.data[2]      =      SENSE_LEN;
            break;

        case REWIND_CMD:        /* 0x01 */
            ptr_req->opcode = REWIND_CMD;
            break;

        case SEEK_CMD:          /* 0x0B */
            ptr_req->opcode = SEEK_CMD;
            break;

        case INQUIRY_CMD:       /* 0x12 */
            ptr_req->opcode        = INQUIRY_CMD;
            ptr_req->cmdlen        = sizeof ptr_req->scsi_cmd.six;
            ptr_scsi_cmd->six.lun          =      ptr_req->lun;
            ptr_scsi_cmd->six.data[2]      =      INQUIRY_LEN;
            ptr_req->data_ptr      =      args[0];
            ptr_req->data_len      =      INQUIRY_LEN;
            break;

        case MODESELECT_CMD:    /* 0x15 */
            ptr_req->opcode = MODESELECT_CMD;
            break;

        case MODESENSE_CMD:             /* 0x1A */
            ptr_req->opcode        = MODESENSE_CMD;
            ptr_req->cmdlen        = sizeof ptr_req->scsi_cmd.six;
            ptr_req->dir           = SCSI_IN;
            ptr_scsi_cmd->six.lun          =      ptr_req->lun;
            ptr_scsi_cmd->six.data[2]      =      SENSE_LEN;
            ptr_req->data_ptr      =      args[0];
            ptr_req->data_len      =      SENSE_LEN;
            break;
        |

        /*
           Check whether have one of two posible error
```

```
                1. HOST ADAPTER can not accept additional command
                2. HOST ADAPTER have fault condition
           Loop retry for error case 1.
        */

        while ((*ptr_d_cfg->adapter_entry)(ptr_req) ) {
                ++Sromtab.b_errcnt;
                if ( Sromtab.b_errcnt = NADRETRY ) {
                        return(SEND_CMD_ERR);
                }
        }

        /* Loop waiting for event */
        ptr_req->internal       = 1;

        lev     = spl5();
        while ( ptr_req->internal )
                sleep(ptr_req,PRIBIO);
        splx(lev);

        return(SEND_CMD_COMPLETE);

}

/*  Chk_host_sts() :
**      For checking HOST STATUS
*/
Chk_host_sts(ptr_req)
REQ_IO  *ptr_req;
{
        switch(ptr_req->host_sts)
        {
                /* selection timeout            */
                case HOST_SEL_TO:
                        cmn_err(CE_WARN,"HOST:selection timeout on id %d-lun %d"
                                ,ptr_req->id,ptr_req->lun);
                        break;

                /* linked command completed ok  */
                case HOST_LINKED_OK:
                        cmn_err(CE_NOTE,"HOST: linked command completed ok");
                        break;

                /* linked command completed bad */
                case HOST_LINKED_BAD:
                        cmn_err(CE_NOTE,"HOST: linked command completed bad");
                        break;

                /* data overrun                 */
```

```
                case HOST_DATA_OVER:
                        cmn_err(CE_WARN,"HOST: data overrun");
                        break;

                /* unexpected bus free          */
                case HOST_BUS_FREE:
                        cmn_err(CE_WARN,"HOST: host bus free");
                        break;

                /* target bus phase seq error   */
                case HOST_PHASE_ERR:
                        cmn_err(CE_WARN,"HOST: host phase free");
                        break;

                /* MBO cmd error                */
                case HOST_CMD:
                        cmn_err(CE_WARN,"HOST: host MBO cmd error");
                        break;

                /* invalid CCB                  */
                case HOST_CCB_ERROR:
                        cmn_err(CE_WARN,"HOST: invalid CCB");
                        break;

                /* linked CCB to another LUN    */
                case HOST_LINK_ERR:
                        cmn_err(CE_WARN,"HOST: linked CCB to another LUN");
                        break;

        }
}

/*  Chk_target_sts() :
**      For checking TARGET STATUS
*/
Chk_target_sts(ptr_req)
REQ_IO  *ptr_req;
{
        switch(ptr_req->target_sts)
        {
                /* status bits mask             */
                case UNIT_MASK:
                        cmn_err(CE_WARN,"TARGET: status bits mask");
                        break;

                /* check condition              */
                case UNIT_CHECK:
                        Chk_cond(ptr_req);
                        break;
```

```
                /* condition met                */
                case UNIT_MET:
                        cmn_err(CE_WARN,"TARGET: condition met");
                        break;

                /* busy                         */
                case UNIT_BUSY:
                        cmn_err(CE_WARN,"TARGET: busy");
                        break;

                /* intermediate cmd good        */
                case UNIT_INTERMED:
                        cmn_err(CE_WARN,"TARGET: intermediate cmd good");
                        break;

                /* intermediate cond met        */
                case UNIT_INTMED_GD:
                        cmn_err(CE_WARN,"TARGET: intermediate cond met");
                        break;

                /* reservation conflict         */
                case UNIT_RESERV:
                        cmn_err(CE_WARN,"TARGET: reservation conflict");
                        break;

        }
}

/*  Chk_cond() :
**      For checking sense data return from target
*/
Chk_cond(ptr_req)
REQ_IO  *ptr_req;
{
        union scsi_cdb  *ptr_scsi_cmd   = &ptr_req->scsi_cmd;
        SENSE_DATA      *ptr_scsi_sense;

        ptr_scsi_sense = (SENSE_DATA *)(&ptr_scsi_cmd->raw[ptr_req->cmdlen]);

        switch (ptr_scsi_sense->key)
        {
                case SS_SOFT:
                        cmn_err(CE_NOTE,"TARGET:recoverable error on id %d-lun %d"
                                ,ptr_req->id,ptr_req->lun);
                        break;
                case SS_NOTRDY:
                        cmn_err(CE_NOTE,"TARGET:unit not ready on id %d-lun %d"
                                ,ptr_req->id,ptr_req->lun);
```

```
                break;
        case SS_MEDERR:
                cmn_err(CE_NOTE,"TARGET:media error on id %d-lun %d"
                        ,ptr_req->id,ptr_req->lun);
                break;
        case SS_HARDERR:
                cmn_err(CE_NOTE,"TARGET:hardware error on id %d-lun %d"
                        ,ptr_req->id,ptr_req->lun);
                break;
        case SS_ILLREQ:
                cmn_err(CE_NOTE,"TARGET:illegal request on id %d-lun %d"
                        ,ptr_req->id,ptr_req->lun);
                break;
        case SS_UATT:
                cmn_err(CE_NOTE,"TARGET:unit attention on id %d-lun %d"
                        ,ptr_req->id,ptr_req->lun);
                break;
        case SS_WPROT:
                cmn_err(CE_NOTE,"TARGET:data protect on id %d-lun %d"
                        ,ptr_req->id,ptr_req->lun);
                break;
        case SS_BLANK:
                cmn_err(CE_NOTE,"TARGET:blank check on id %d-lun %d"
                        ,ptr_req->id,ptr_req->lun);
                break;
        case SS_COPYAB:
                cmn_err(CE_NOTE,"TARGET:copy aborted on id %d-lun %d"
                        ,ptr_req->id,ptr_req->lun);
                break;
        case SS_CMDAB:
                cmn_err(CE_NOTE,"TARGET:command aborted on id %d-lun %d"
                        ,ptr_req->id,ptr_req->lun);
                break;
        case SS_VOFLOW:
                cmn_err(CE_NOTE,"TARGET:volume overflow on id %d-lun %d"
                        ,ptr_req->id,ptr_req->lun);
    }

    return;
}

/*  trans_buf() :
**      For     transfer data from source buffer to target buffer
*/
trans_buf(src_buf,trg_buf)
struct buf *src_buf,*trg_buf;
{
        trg_buf->b_flags        =       src_buf->b_flags;
        trg_buf->b_forw         =       src_buf->b_forw;
```

```
        trg_buf->b_back         =       src_buf->b_back;
        trg_buf->av_forw        =       src_buf->av_forw;
        trg_buf->av_back        =       src_buf->av_back;
        trg_buf->b_dev          =       src_buf->b_dev;
        trg_buf->b_bcount       =       src_buf->b_bcount;
/*      trg_buf->b_un.b_addr    =       src_buf->b_un.b_addr;
        trg_buf->b_un.b_words   =       src_buf->b_un.b_words;
        trg_buf->b_un.b_daddr   =       src_buf->b_un.b_daddr;
#ifdef  PPS
        trg_buf->b_un.b_filsys  =       src_buf->b_un.b_filsys;
#endif
*/
        trg_buf->b_blkno        =       src_buf->b_blkno;
        trg_buf->b_error        =       src_buf->b_error;
        trg_buf->b_res          =       src_buf->b_res;
        trg_buf->b_cylin        =       src_buf->b_cylin;
        trg_buf->b_resid        =       src_buf->b_resid;
        trg_buf->b_sector       =       src_buf->b_sector;
        trg_buf->b_start        =       src_buf->b_start;
        trg_buf->b_proc         =       src_buf->b_proc;
        trg_buf->b_reltime      =       src_buf->b_reltime;
#ifdef  PPS
        trg_buf->b_s2           =       src_buf->b_s2;
        trg_buf->b_s3           =       src_buf->b_s3;
#endif
        trg_buf->b_want         =       src_buf->b_want;
}
```

```
/*
*****************************************************************************
****                                                                     ****
****  I N C L U D E  F I L E  F O R  C D - R O M  D R I V E R            ****
****                                                                     ****
*****************************************************************************
*/

/* defines for block/char specification to blckopen */
#define CHRDEV  OTYP_CHR        /* 3rd parameter to device open routine */
#define BLKDEV  OTYP_BLK        /* specifying block or char device      */

#define NULLPTR (void *)0
extern  DEVCFG  Sromcfg[];
extern  HACFG   ahacfg[];

#define SCSI_INFO    4     /* return information about capabilities */
#define SCSI_NO_INFO 0xFF  /* do not want any DISK INFORMATION      */

#define MAX_REQ_BLCK    8    /* Maximum request block at one time     */
#define CD_BLOCK_SIZE   2048 /* Data of Driver buffer                 */
#define CDSHIFT         11

#define SEND_CMD_COMPLETE 0  /* Error code of send_scsi_cmd           */
#define SEND_CMD_ERR    1    /* Error code of send_scsi_cmd           */

/*  Add sense key definitions  */
#define SS_NOTRDY  0x2  /* drive not ready      */
#define SS_MEDERR  0x3  /* media error          */
#define SS_HARDERR 0x4  /* hardware error       */
#define SS_ILLREQ  0x5  /* illegal request      */
#define SS_BLANK   0x8  /* blank check          */
#define SS_COPYAB  0xA  /* copy aborted         */
#define SS_CMDAB   0xB  /* command aborted      */

#define RD_CAP_LEN 8
#define VOID_LEN   8

struct read_cap_data {
      unsigned char blck_addr[4];
      unsigned char blck_len[4];
};
typedef struct read_cap_data RD_CAP_DATA;

struct scsi_ha_info {
      unsigned int do_sg:1;
      unsigned int do_buffer:1;
      unsigned int reserved:30;
};
```

```
typedef struct scsi_ha_info    HA_INFO;

typedef struct scsi_mode_sense MS_DATA;

typedef struct scsi_sense      SENSE_DATA;
```

ประวัติผู้เขียน

นาย อนุชัย ธีระเรืองไชยศรี เกิดเมื่อวันที่ 17 กรกฎาคม พ.ศ. 2508 ที่กรุงเทพมหานคร สำเร็จการศึกษาเภสัชศาสตรบัณฑิต จาก คณะเภสัชศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2529 หลังจากนั้นเข้าทำงานในศูนย์คอมพิวเตอร์ทางเภสัชศาสตร์ คณะเภสัชศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิทยาศาสตร์คอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อ พ.ศ. 2531