```
/*
 *  hosts.h Copyright (C) 1992 Drew Eckhardt
 *          Copyright (C) 1993, 1994, 1995 Eric Youngdale
 *
 *  mid to low-level SCSI driver interface header
 *      Initial versions: Drew Eckhardt
 *      Subsequent revisions: Eric Youngdale
 *
 *  <drew@colorado.edu>
 *
 *       Modified by Eric Youngdale eric@aib.com to
 *       add scatter-gather, multiple outstanding request, and other
 *       enhancements.
 *
 *  Further modified by Eric Youngdale to support multiple host adapters
 *  of the same type.
 */

#ifndef _HOSTS_H
#define _HOSTS_H

/*
    $Header: hosts.h,v 1.3 1993/09/24 12:21:00 drew Exp drew $
*/

#include <linux/proc_fs.h>

/* It is senseless to set SG_ALL any higher than this - the performance
 *  does not get any better, and it wastes memory
 */
#define SG_NONE 0
#define SG_ALL 0xff

#define DISABLE_CLUSTERING 0
#define ENABLE_CLUSTERING 1

/* The various choices mean:
 * NONE: Self evident.  Host adapter is not capable of scatter-gather.
 * ALL:  Means that the host adapter module can do scatter-gather,
 *       and that there is no limit to the size of the table to which
 *       we scatter/gather data.
 * Anything else:  Indicates the maximum number of chains that can be
 *       used in one scatter-gather request.
 */

/*
 * The Scsi_Host_Template type has all that is needed to interface
 * with a SCSI host in a device independent matter.  There is one
 * entry for each different type of host adapter that is supported on
 * the system.
 */

typedef struct scsi_disk Disk;

typedef struct  SHT
{

    /* Used with loadable modules so we can construct a linked list. */
    struct SHT * next;

```

```
    /*
     * Used with loadable modules so that we know when it is safe to
     * unload.
     */
    long * usage_count;

    /* The pointer to the /proc/scsi directory entry */
    struct proc_dir_entry *proc_dir;

    /*
     * proc-fs info function.
     * Can be used to export driver statistics and other infos to the
     * world outside the kernel ie. userspace and it also provides an
     * interface to feed the driver with information. Check
     * eata_dma_proc.c for reference
     */
    int (*proc_info)(char *, char **, off_t, int, int, int);

    /*
     * The name pointer is a pointer to the name of the SCSI
     * device detected.
     */
    const char *name;

    /*
     * The detect function shall return non zero on detection,
     * indicating the number of host adapters of this particular
     * type were found.  It should also
     * initialize all data necessary for this particular
     * SCSI driver.  It is passed the host number, so this host
     * knows where the first entry is in the scsi_hosts[] array.
     *
     * Note that the detect routine MUST not call any of the mid level
     * functions to queue commands because things are not guaranteed
     * to be set up yet.  The detect routine can send commands to
     * the host adapter as long as the program control will not be
     * passed to scsi.c in the processing of the command.  Note
     * especially that scsi_malloc/scsi_free must not be called.
     */
    int (* detect)(struct SHT *);

    /*
     * Used with loadable modules to unload the host structures.
     * Note: there is a default action built into the modules code
     * which may be sufficient for most host adapters.  Thus you may
     * not have to supply this at all.
     */
    int (*release)(struct Scsi_Host *);

    /*
     * The info function will return whatever useful
     * information the developer sees fit.  If not provided, then
     * the name field will be used instead.
     */
    const char *(* info)(struct Scsi_Host *);

    /*
     * The command function takes a target, a command (this is a SCSI
     * command formatted as per the SCSI spec, nothing strange), a
     * data buffer pointer, and data buffer length pointer.  The return
```

```
         * is a status int, bit fielded as follows :
         * Byte What
         * 0    SCSI status code
         * 1    SCSI 1 byte message
         * 2    host error return.
         * 3    mid level error return
         */
        int (* command)(Scsi_Cmnd *);

        /*
         * The QueueCommand function works in a similar manner
         * to the command function.  It takes an additional parameter,
         * void (* done)(int host, int code) which is passed the host
         * # and exit result when the command is complete.
         * Host number is the POSITION IN THE hosts array of THIS
         * host adapter.
         */
        int (* queuecommand)(Scsi_Cmnd *, void (*done)(Scsi_Cmnd *));

        /*
         * Since the mid level driver handles time outs, etc, we want to
         * be able to abort the current command.  Abort returns 0 if the
         * abortion was successful.  The field SCpnt->abort reason
         * can be filled in with the appropriate reason why we wanted
         * the abort in the first place, and this will be used
         * in the mid-level code instead of the host_byte().
         * If non-zero, the code passed to it
         * will be used as the return code, otherwise
         * DID_ABORT  should be returned.
         *
         * Note that the scsi driver should "clean up" after itself,
         * resetting the bus, etc.  if necessary.
         */
        int (* abort)(Scsi_Cmnd *);

        /*
         * The reset function will reset the SCSI bus.  Any executing
         * commands should fail with a DID_RESET in the host byte.
         * The Scsi_Cmnd  is passed so that the reset routine can figure
         * out which host adapter should be reset, and also which command
         * within the command block was responsible for the reset in
         * the first place.  Some hosts do not implement a reset function,
         * and these hosts must call scsi_request_sense(SCpnt) to keep
         * the command alive.
         */
        int (* reset)(Scsi_Cmnd *, unsigned int);

        /*
         * This function is used to select synchronous communications,
         * which will result in a higher data throughput.  Not implemented
         * yet.
         */
        int (* slave_attach)(int, int);

        /*
         * This function determines the bios parameters for a given
         * harddisk.  These tend to be numbers that are made up by
         * the host adapter.  Parameters:
         * size, device number, list (heads, sectors, cylinders)
         */
```

```
        int (* bios_param)(Disk *, kdev_t, int []);

        /*
         * This determines if we will use a non-interrupt driven
         * or an interrupt driven scheme,  It is set to the maximum number
         * of simultaneous commands a given host adapter will accept.
         */
        int can_queue;

        /*
         * In many instances, especially where disconnect / reconnect are
         * supported, our host also has an ID on the SCSI bus.  If this is
         * the case, then it must be reserved.  Please set this_id to -1 if
         * your setup is in single initiator mode, and the host lacks an
         * ID.
         */
        int this_id;

        /*
         * This determines the degree to which the host adapter is capable
         * of scatter-gather.
         */
        short unsigned int sg_tablesize;

        /*
         * True if this host adapter can make good use of linked commands.
         * This will allow more than one command to be queued to a given
         * unit on a given host.  Set this to the maximum number of
         * command blocks to be provided for each device.  Set this to 1
         * for one command block per lun, 2 for two, etc.  Do not set this
         * to 0.  You should make sure that the host adapter will do the
         * right thing before you try setting this above 1.
         */
        short cmd_per_lun;

        /*
         * present contains counter indicating how many boards of this
         * type were found when we did the scan.
         */
        unsigned char present;

        /*
         * true if this host adapter uses unchecked DMA onto an ISA bus.
         */
        unsigned unchecked_isa_dma:1;

        /*
         * true if this host adapter can make good use of clustering.
         * I originally thought that if the tablesize was large that it
         * was a waste of CPU cycles to prepare a cluster list, but
         * it works out that the Buslogic is faster if you use a smaller
         * number of segments (i.e. use clustering).  I guess it is
         * inefficient.
         */
        unsigned use_clustering:1;

    } Scsi_Host_Template;

    /*
     * The scsi_hosts array is the array containing the data for all
```

```
 * possible <supported> scsi hosts.   This is similar to the
 * Scsi_Host_Template, except that we have one entry for each
 * actual physical host adapter on the system, stored as a linked
 * list.  Note that if there are 2 aha1542 boards, then there will
 * be two Scsi_Host entries, but only 1 Scsi_Host_Template entry.
 */

struct Scsi_Host
{
    struct Scsi_Host * next;
    unsigned short extra_bytes;
    volatile unsigned char host_busy;
    char host_no;  /* Used for IOCTL_GET_IDLUN, /proc/scsi et al. */
    unsigned long last_reset;
    struct wait_queue *host_wait;
    Scsi_Cmnd *host_queue;
    Scsi_Host_Template * hostt;

    /*
     *  These three parameters can be used to allow for wide scsi,
     *  and for host adapters that support multiple busses
     *  The first two should be set to 1 more than the actual max id
     *  or lun (i.e. 8 for normal systems).
     */
    unsigned int max_id;
    unsigned int max_lun;
    unsigned int max_channel;

    /*
     * Pointer to a circularly linked list - this indicates the hosts
     * that should be locked out of performing I/O while we have an
     * active command on this host.
     */
    struct Scsi_Host * block;
    unsigned wish_block:1;

    /* These parameters should be set by the detect routine */
    unsigned char *base;
    unsigned int  io_port;
    unsigned char n_io_port;
    unsigned char irq;
    unsigned char dma_channel;

    /*
     * This is a unique identifier that must be assigned so that we
     * have some way of identifying each detected host adapter properly
     * and uniquely.  For hosts that do not support more than one card
     * in the system at one time, this does not need to be set.  It is
     * initialized to 0 in scsi_register.
     */
    unsigned int unique_id;

    /*
     * The rest can be copied from the template, or specifically
     * initialized, as required.
     */

    int this_id;
    int can_queue;
    short cmd_per_lun;
```

```
    short unsigned int sg_tablesize;
    unsigned unchecked_isa_dma:1;
    unsigned use_clustering:1;
    /*
     * True if this host was loaded as a loadable module
     */
    unsigned loaded_as_module:1;

    void (*select_queue_depths)(struct Scsi_Host *, Scsi_Device *);

    unsigned long hostdata[0];  /* Used for storage of
                                 * host specific stuff */
};

extern struct Scsi_Host * scsi_hostlist;
extern struct Scsi_Device_Template * scsi_devicelist;

extern Scsi_Host_Template * scsi_hosts;

extern void build_proc_dir_entries(Scsi_Host_Template  *);


/*
 *  scsi_init initializes the scsi hosts.
 */

/*
 * We use these goofy things because the MM is not set up when we init
 * the scsi subsystem.  By using these functions we can write code that
 * looks normal.  Also, it makes it possible to use the same code for a
 * loadable module.
 */

extern void * scsi_init_malloc(unsigned int size, int priority);
extern void scsi_init_free(char * ptr, unsigned int size);

extern int next_scsi_host;

extern int scsi_loadable_module_flag;
unsigned int scsi_init(void);
extern struct Scsi_Host * scsi_register(Scsi_Host_Template *, int j);
extern void scsi_unregister(struct Scsi_Host * i);

#define BLANK_HOST {"", 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

struct Scsi_Device_Template
{
    struct Scsi_Device_Template * next;
    const char * name;
    const char * tag;
    long * usage_count;           /* Used for loadable modules */
    unsigned char scsi_type;
    unsigned char major;
    unsigned char nr_dev;         /* Number currently attached */
    unsigned char dev_noticed;    /* Number of devices detected. */
    unsigned char dev_max;        /* Current size of arrays */
    unsigned blk:1;               /* 0 if character device */
    int (*detect)(Scsi_Device *); /* Returns 1 if we can attach
                                   * this device */
    int (*init)(void);                /* Sizes arrays based upon number
```

```
                                     *  of devices detected */
    void (*finish)(void);      /* Perform initialization after
                                     *  attachment */
    int (*attach)(Scsi_Device *); /* Attach devices to arrays */
    void (*detach)(Scsi_Device *);
};

extern struct Scsi_Device_Template sd_template;
extern struct Scsi_Device_Template st_template;
extern struct Scsi_Device_Template sr_template;
extern struct Scsi_Device_Template sg_template;

int scsi_register_device(struct Scsi_Device_Template * sdpnt);

/* These are used by loadable modules */
extern int scsi_register_module(int, void *);
extern void scsi_unregister_module(int, void *);

/* The different types of modules that we can load and unload */
#define MODULE_SCSI_HA 1
#define MODULE_SCSI_CONST 2
#define MODULE_SCSI_IOCTL 3
#define MODULE_SCSI_DEV 4


/*
 * This is an ugly hack.  If we expect to be able to load devices at
 * run time, we need to leave extra room in some of the data
 * structures. Doing a realloc to enlarge the structures would be
 * riddled with race conditions, so until a better solution is
 * discovered, we use this crude approach
 */
#define SD_EXTRA_DEVS 2
#define ST_EXTRA_DEVS 2
#define SR_EXTRA_DEVS 2
#define SG_EXTRA_DEVS (SD_EXTRA_DEVS + SR_EXTRA_DEVS + ST_EXTRA_DEVS)

#endif
/*
 * Overrides for Emacs so that we follow Linus's tabbing style.  Emacs
 * will notice this stuff at the end of the file and automatically
 * adjust the settings for this buffer only.  This must remain at the
 * end of the file.
 * ---------------------------------------------------------------------
 * Local variables:
 * c-indent-level: 4
 * c-brace-imaginary-offset: 0
 * c-brace-offset: -4
 * c-argdecl-indent: 4
 * c-label-offset: -4
 * c-continued-statement-offset: 4
 * c-continued-brace-offset: 0
 * indent-tabs-mode: nil
 * tab-width: 8
 * End:
 */
```