Chapter 15
Linux Data Structures

This appendix lists the major data structures that Linux uses and which are described in this book. They have been edited slightly to fit the paper.

block_dev_struct

block_dev_struct data structures are used to register block devices as available for use by the buffer cache. They are held together in the blk_dev vector.

```
struct blk_dev_struct {
    void (*request_fn)(void);
    struct request * current_request;
    struct request   plug;
    struct tq_struct plug_tq;
};
```

buffer_head

The buffer_head data structure holds information about a block buffer in the buffer cache.

```
/* bh state bits */
#define BH_Uptodate  0   /* 1 if the buffer contains valid data      */
#define BH_Dirty     1   /* 1 if the buffer is dirty                 */
#define BH_Lock      2   /* 1 if the buffer is locked                */
#define BH_Req       3   /* 0 if the buffer has been invalidated     */
#define BH_Touched   4   /* 1 if the buffer has been touched (aging) */
#define BH_Has_aged  5   /* 1 if the buffer has been aged (aging)    */
#define BH_Protected 6   /* 1 if the buffer is protected             */
#define BH_FreeOnIO  7   /* 1 to discard the buffer_head after IO    */

struct buffer_head {
  /* First cache line: */
  unsigned long      b_blocknr;    /* block number                     */
  kdev_t             b_dev;        /* device (B_FREE = free)           */
  kdev_t             b_rdev;       /* Real device                      */
  unsigned long      b_rsector;    /* Real buffer location on disk     */
  struct buffer_head *b_next;      /* Hash queue list                  */
  struct buffer_head *b_this_page; /* circular list of buffers in one
                                      page                             */

  /* Second cache line: */
  unsigned long      b_state;      /* buffer state bitmap (above)      */
  struct buffer_head *b_next_free;
  unsigned int       b_count;      /* users using this block           */
  unsigned long      b_size;       /* block size                       */

  /* Non-performance-critical data follows. */
  char               *b_data;      /* pointer to data block            */
  unsigned int       b_list;       /* List that this buffer appears    */
  unsigned long      b_flushtime;  /* Time when this (dirty) buffer
                                    * should be written                */
  unsigned long      b_lru_time;   /* Time when this buffer was
```

```
                                    * last used.                    */
  struct wait_queue  *b_wait;
  struct buffer_head *b_prev;      /* doubly linked hash list        */
  struct buffer_head *b_prev_free; /* doubly linked list of buffers  */
  struct buffer_head *b_reqnext;   /* request queue                  */
};
```

device

Every network device in the system is represented by a `device` data structure.

```
struct device
{

  /*
   * This is the first field of the "visible" part of this structure
   * (i.e. as seen by users in the "Space.c" file).  It is the name
   * the interface.
   */
  char                    *name;

  /* I/O specific fields                                            */
  unsigned long           rmem_end;        /* shmem "recv" end     */
  unsigned long           rmem_start;      /* shmem "recv" start   */
  unsigned long           mem_end;         /* shared mem end       */
  unsigned long           mem_start;       /* shared mem start     */
  unsigned long           base_addr;       /* device I/O address   */
  unsigned char           irq;             /* device IRQ number    */

  /* Low-level status flags. */
  volatile unsigned char  start,           /* start an operation   */
                          interrupt;       /* interrupt arrived    */
  unsigned long           tbusy;           /* transmitter busy     */
  struct device           *next;

  /* The device initialization function. Called only once.         */
  int                     (*init)(struct device *dev);

  /* Some hardware also needs these fields, but they are not part of
     the usual set specified in Space.c. */
  unsigned char           if_port;         /* Selectable AUI,TP,   */
  unsigned char           dma;             /* DMA channel          */

  struct enet_statistics* (*get_stats)(struct device *dev);

  /*
   * This marks the end of the "visible" part of the structure. All
   * fields hereafter are internal to the system, and may change at
   * will (read: may be cleaned up at will).
   */

  /* These may be needed for future network-power-down code.       */
  unsigned long           trans_start;     /* Time (jiffies) of
                                              last transmit        */
  unsigned long           last_rx;         /* Time of last Rx      */
  unsigned short          flags;           /* interface flags (BSD)*/
  unsigned short          family;          /* address family ID    */
  unsigned short          metric;          /* routing metric       */
  unsigned short          mtu;             /* MTU value            */
  unsigned short          type;            /* hardware type        */
  unsigned short          hard_header_len; /* hardware hdr len     */
  void                    *priv;           /* private data         */

  /* Interface address info. */
  unsigned char           broadcast[MAX_ADDR_LEN];
  unsigned char           pad;
  unsigned char           dev_addr[MAX_ADDR_LEN];
  unsigned char           addr_len;        /* hardware addr len    */
  unsigned long           pa_addr;         /* protocol address     */
  unsigned long           pa_brdaddr;      /* protocol broadcast addr*/
  unsigned long           pa_dstaddr;      /* protocol P-P other addr*/
  unsigned long           pa_mask;         /* protocol netmask     */
  unsigned short          pa_alen;         /* protocol address len */

  struct dev_mc_list      *mc_list;        /* M'cast mac addrs     */
  int                     mc_count;        /* No installed mcasts  */

  struct ip_mc_list       *ip_mc_list;     /* IP m'cast filter chain */
  __u32                   tx_queue_len;    /* Max frames per queue   */

  /* For load balancing driver pair support */
  unsigned long           pkt_queue;       /* Packets queued       */
  struct device           *slave;          /* Slave device         */
  struct net_alias_info   *alias_info;     /* main dev alias info  */
  struct net_alias        *my_alias;       /* alias devs           */

  /* Pointer to the interface buffers. */
  struct sk_buff_head     buffs[DEV_NUMBUFFS];

  /* Pointers to interface service routines. */
  int                     (*open)(struct device *dev);
  int                     (*stop)(struct device *dev);
  int                     (*hard_start_xmit) (struct sk_buff *skb,
                                              struct device *dev);
  int                     (*hard_header) (struct sk_buff *skb,
                                          struct device *dev,
                                          unsigned short type,
                                          void *daddr,
                                          void *saddr,
                                          unsigned len);
  int                     (*rebuild_header)(void *eth,
                                          struct device *dev,
                                          unsigned long raddr,
                                          struct sk_buff *skb);
  void                    (*set_multicast_list)(struct device *dev);
  int                     (*set_mac_address)(struct device *dev,
                                          void *addr);
  int                     (*do_ioctl)(struct device *dev,
                                          struct ifreq *ifr,
                                          int cmd);
  int                     (*set_config)(struct device *dev,
                                          struct ifmap *map);
  void                    (*header_cache_bind)(struct hh_cache **hhp,
                                          struct device *dev,
                                          unsigned short htype,
                                          __u32 daddr);
  void                    (*header_cache_update)(struct hh_cache *hh,
                                          struct device *dev,
                                          unsigned char *  haddr);
  int                     (*change_mtu)(struct device *dev,
                                          int new_mtu);
  struct iw_statistics*   (*get_wireless_stats)(struct device *dev);
};
```

device_struct

`device_struct` data structures are used to register character and block devices (they hold its name and the set of file operations that can be used for this device). Each valid member of the `chrdevs` and `blkdevs` vectors represents a character or block device respectively.

```
struct device_struct {
    const char * name;
    struct file_operations * fops;
};
```

file

Each open file, socket etcetera is represented by a `file` data structure.

```
struct file {
  mode_t f_mode;
  loff_t f_pos;
  unsigned short f_flags;
  unsigned short f_count;
  unsigned long f_reada, f_ramax, f_raend, f_ralen, f_rawin;
  struct file *f_next, *f_prev;
  int f_owner;         /* pid or -pgrp where SIGIO should be sent */
  struct inode * f_inode;
  struct file_operations * f_op;
  unsigned long f_version;
  void *private_data;  /* needed for tty driver, and maybe others */
};
```

files_struct

The `files_struct` data structure describes the files that a process has open.

```
struct files_struct {
  int count;
  fd_set close_on_exec;
  fd_set open_fds;
  struct file * fd[NR_OPEN];
};
```

fs_struct

```
struct fs_struct {
  int count;
  unsigned short umask;
  struct inode * root, * pwd;
};
```

gendisk

The `gendisk` data structure holds information about a hard disk. They are used during initialization when the disks are found and then probed for partitions.

```
struct hd_struct {
    long start_sect;
    long nr_sects;
};

struct gendisk {
    int major;               /* major number of driver */
    const char *major_name;  /* name of major driver */
    int minor_shift;         /* number of times minor is shifted to
                                get real minor */
    int max_p;               /* maximum partitions per device */
    int max_nr;              /* maximum number of real devices */

    void (*init)(struct gendisk *);
                             /* Initialization called before we
                                do our thing */
    struct hd_struct *part;  /* partition table */
    int *sizes;              /* device size in blocks, copied to
                                blk_size[] */
    int nr_real;             /* number of real devices */

    void *real_devices;      /* internal use */
    struct gendisk *next;
};
```

inode

The VFS inode data structure holds information about a file or directory on disk.

```
struct inode {
    kdev_t                       i_dev;
    unsigned long                i_ino;
    umode_t                      i_mode;
    nlink_t                      i_nlink;
    uid_t                        i_uid;
    gid_t                        i_gid;
    kdev_t                       i_rdev;
    off_t                        i_size;
    time_t                       i_atime;
    time_t                       i_mtime;
    time_t                       i_ctime;
    unsigned long                i_blksize;
    unsigned long                i_blocks;
    unsigned long                i_version;
    unsigned long                i_nrpages;
    struct semaphore             i_sem;
    struct inode_operations      *i_op;
    struct super_block           *i_sb;
    struct wait_queue            *i_wait;
    struct file_lock             *i_flock;
    struct vm_area_struct        *i_mmap;
    struct page                  *i_pages;
    struct dquot                 *i_dquot[MAXQUOTAS];
    struct inode                 *i_next, *i_prev;
    struct inode                 *i_hash_next, *i_hash_prev;
    struct inode                 *i_bound_to, *i_bound_by;
    struct inode                 *i_mount;
    unsigned short               i_count;
    unsigned short               i_flags;
    unsigned char                i_lock;
    unsigned char                i_dirt;
    unsigned char                i_pipe;
    unsigned char                i_sock;
    unsigned char                i_seek;
    unsigned char                i_update;
    unsigned short               i_writecount;
    union {
        struct pipe_inode_info   pipe_i;
        struct minix_inode_info  minix_i;
        struct ext_inode_info    ext_i;
        struct ext2_inode_info   ext2_i;
        struct hpfs_inode_info   hpfs_i;
        struct msdos_inode_info  msdos_i;
        struct umsdos_inode_info umsdos_i;
        struct iso_inode_info    isofs_i;
        struct nfs_inode_info    nfs_i;
        struct xiafs_inode_info  xiafs_i;
        struct sysv_inode_info   sysv_i;
        struct affs_inode_info   affs_i;
        struct ufs_inode_info    ufs_i;
        struct socket            socket_i;
        void                     *generic_ip;
    } u;
};
```

ipc_perm

The ipc_perm data structure describes the access permissions of a System V IPC object .

```
struct ipc_perm
{
  key_t  key;
  ushort uid;   /* owner euid and egid */
  ushort gid;
  ushort cuid;  /* creator euid and egid */
  ushort cgid;
  ushort mode;  /* access modes see mode flags below */
  ushort seq;   /* sequence number */
};
```

irqaction

The `irqaction` data structure is used to describe the system's interrupt handlers.

```
struct irqaction {
  void (*handler)(int, void *, struct pt_regs *);
  unsigned long flags;
  unsigned long mask;
  const char *name;
  void *dev_id;
  struct irqaction *next;
};
```

linux_binfmt

Each binary file format that Linux understands is represented by a `linux_binfmt` data structure.

```
struct linux_binfmt {
  struct linux_binfmt * next;
  long *use_count;
  int (*load_binary)(struct linux_binprm *, struct  pt_regs * regs);
  int (*load_shlib)(int fd);
  int (*core_dump)(long signr, struct pt_regs * regs);
};
```

mem_map_t

The `mem_map_t` data structure (also known as `page`) is used to hold information about each page of physical memory.

```
typedef struct page {
  /* these must be first (free area handling) */
  struct page        *next;
  struct page        *prev;
  struct inode       *inode;
  unsigned long      offset;
  struct page        *next_hash;
  atomic_t           count;
  unsigned           flags;     /* atomic flags, some possibly
                                   updated asynchronously */
  unsigned           dirty:16,
                     age:8;
  struct wait_queue  *wait;
  struct page        *prev_hash;
  struct buffer_head *buffers;
  unsigned long      swap_unlock_entry;
  unsigned long      map_nr;    /* page->map_nr == page - mem_map */
} mem_map_t;
```

mm_struct

The `mm_struct` data structure is used to describe the virtual memory of a task or process.

```
struct mm_struct {
  int count;
  pgd_t * pgd;
  unsigned long context;
  unsigned long start_code, end_code, start_data, end_data;
  unsigned long start_brk, brk, start_stack, start_mmap;
  unsigned long arg_start, arg_end, env_start, env_end;
  unsigned long rss, total_vm, locked_vm;
  unsigned long def_flags;
  struct vm_area_struct * mmap;
  struct vm_area_struct * mmap_avl;
  struct semaphore mmap_sem;
};
```

pci_bus

Every PCI bus in the system is represented by a pci_bus data structure.

```
struct pci_bus {
  struct pci_bus  *parent;     /* parent bus this bridge is on */
  struct pci_bus  *children;   /* chain of P2P bridges on this bus */
  struct pci_bus  *next;       /* chain of all PCI buses */

  struct pci_dev  *self;       /* bridge device as seen by parent */
  struct pci_dev  *devices;    /* devices behind this bridge */

  void    *sysdata;            /* hook for sys-specific extension */

  unsigned char  number;       /* bus number */
  unsigned char  primary;      /* number of primary bridge */
  unsigned char  secondary;    /* number of secondary bridge */
  unsigned char  subordinate;  /* max number of subordinate buses */
};
```

pci_dev

Every PCI device in the system, including PCI-PCI and PCI-ISA bridge devices is represented by a pci_dev data structure.

```
/*
 * There is one pci_dev structure for each slot-number/function-number
 * combination:
 */
struct pci_dev {
  struct pci_bus  *bus;      /* bus this device is on */
  struct pci_dev  *sibling;  /* next device on this bus */
  struct pci_dev  *next;     /* chain of all devices */

  void    *sysdata;          /* hook for sys-specific extension */

  unsigned int  devfn;       /* encoded device & function index */
  unsigned short  vendor;
  unsigned short  device;
  unsigned int  class;       /* 3 bytes: (base,sub,prog-if) */
  unsigned int  master : 1;  /* set if device is master capable */
  /*
   * In theory, the irq level can be read from configuration
   * space and all would be fine.  However, old PCI chips don't
   * support these registers and return 0 instead.  For example,
   * the Vision864-P rev 0 chip can uses INTA, but returns 0 in
   * the interrupt line and pin registers.  pci_init()
   * initializes this field with the value at PCI_INTERRUPT_LINE
   * and it is the job of pcibios_fixup() to change it if
   * necessary.  The field must not be 0 unless the device
   * cannot generate interrupts at all.
   */
  unsigned char  irq;        /* irq generated by this device */
};
```

request

`request` data structures are used to make requests to the block devices in the system. The requests are always to read or write blocks of data to or from the buffer cache.

```
struct request {
    volatile int rq_status;
#define RQ_INACTIVE            (-1)
#define RQ_ACTIVE              1
#define RQ_SCSI_BUSY           0xffff
#define RQ_SCSI_DONE           0xfffe
#define RQ_SCSI_DISCONNECTING  0xffe0

    kdev_t rq_dev;
    int cmd;        /* READ or WRITE */
    int errors;
    unsigned long sector;
    unsigned long nr_sectors;
    unsigned long current_nr_sectors;
    char * buffer;
    struct semaphore * sem;
    struct buffer_head * bh;
    struct buffer_head * bhtail;
    struct request * next;
};
```

rtable

Each `rtable` data structure holds information about the route to take in order to send packets to an IP host. `rtable` data structures are used within the IP route cache.

```
struct rtable
{
    struct rtable     *rt_next;
    __u32             rt_dst;
    __u32             rt_src;
    __u32             rt_gateway;
    atomic_t          rt_refcnt;
    atomic_t          rt_use;
    unsigned long     rt_window;
    atomic_t          rt_lastuse;
    struct hh_cache   *rt_hh;
    struct device     *rt_dev;
    unsigned short    rt_flags;
    unsigned short    rt_mtu;
    unsigned short    rt_irtt;
    unsigned char     rt_tos;
};
```

semaphore

Semaphores are used to protect critical data structures and regions of code. y

```
struct semaphore {
    int count;
    int waking;
    int lock ;                /* to make waking testing atomic */
    struct wait_queue *wait;
};
```

sk_buff

The sk_buff data structure is used to describe network data as it moves between the layers of protocol.

```
struct sk_buff
{
  struct sk_buff      *next;       /* Next buffer in list                   */
  struct sk_buff      *prev;       /* Previous buffer in list               */
  struct sk_buff_head *list;       /* List we are on                        */
  int                 magic_debug_cookie;
  struct sk_buff      *link3;      /* Link for IP protocol level buffer chains */
  struct sock         *sk;         /* Socket we are owned by                */
  unsigned long       when;        /* used to compute rtt's                 */
  struct timeval      stamp;       /* Time we arrived                       */
  struct device       *dev;        /* Device we arrived on/are leaving by   */
  union
  {
      struct tcphdr   *th;
      struct ethhdr   *eth;
      struct iphdr    *iph;
      struct udphdr   *uh;
      unsigned char   *raw;
      /* for passing file handles in a unix domain socket */
      void            *filp;
  } h;

  union
  {
      /* As yet incomplete physical layer views */
      unsigned char   *raw;
      struct ethhdr   *ethernet;
  } mac;

  struct iphdr        *ip_hdr;     /* For IPPROTO_RAW                       */
  unsigned long       len;         /* Length of actual data                 */
  unsigned long       csum;        /* Checksum                              */
  __u32               saddr;       /* IP source address                     */
  __u32               daddr;       /* IP target address                     */
  __u32               raddr;       /* IP next hop address                   */
  __u32               seq;         /* TCP sequence number                   */
  __u32               end_seq;     /* seq [+ fin] [+ syn] + datalen         */
  __u32               ack_seq;     /* TCP ack sequence number               */
  unsigned char       proto_priv[16];
  volatile char       acked,       /* Are we acked ?                        */
                      used,        /* Are we in use ?                       */
                      free,        /* How to free this buffer               */
                      arp;         /* Has IP/ARP resolution finished        */
  unsigned char       tries,       /* Times tried                           */
                      lock,        /* Are we locked ?                       */
                      localroute,  /* Local routing asserted for this frame */
                      pkt_type,    /* Packet class                          */
                      pkt_bridged, /* Tracker for bridging                  */
                      ip_summed;   /* Driver fed us an IP checksum          */
#define PACKET_HOST         0        /* To us                                 */
#define PACKET_BROADCAST    1        /* To all                                */
#define PACKET_MULTICAST    2        /* To group                              */
#define PACKET_OTHERHOST    3        /* To someone else                       */
  unsigned short      users;       /* User count - see datagram.c,tcp.c     */
  unsigned short      protocol;    /* Packet protocol from driver.          */
  unsigned int        truesize;    /* Buffer size                           */
  atomic_t            count;       /* reference count                       */
  struct sk_buff      *data_skb;   /* Link to the actual data skb           */
  unsigned char       *head;       /* Head of buffer                        */
  unsigned char       *data;       /* Data head pointer                     */
  unsigned char       *tail;       /* Tail pointer                          */
  unsigned char       *end;        /* End pointer                           */
  void                (*destructor)(struct sk_buff *); /* Destruct function */
  __u16               redirport;   /* Redirect port                         */
};
```

sock

Each sock data structure holds protocol specific information about a BSD socket. For example, for an INET (Internet Address Domain) socket this data structure would hold all of the TCP/IP and UDP/IP specific information.

```
struct sock
{
    /* This must be first. */
    struct sock             *sklist_next;
    struct sock             *sklist_prev;

    struct options          *opt;
    atomic_t                wmem_alloc;
    atomic_t                rmem_alloc;
    unsigned long           allocation;       /* Allocation mode */
    __u32                   write_seq;
    __u32                   sent_seq;
    __u32                   acked_seq;
    __u32                   copied_seq;
    __u32                   rcv_ack_seq;
    unsigned short          rcv_ack_cnt;      /* count of same ack */
    __u32                   window_seq;
    __u32                   fin_seq;
    __u32                   urg_seq;
    __u32                   urg_data;
    __u32                   syn_seq;
    int                     users;            /* user count */
  /*
   *    Not all are volatile, but some are, so we
   *     might as well say they all are.
   */
    volatile char           dead,
                            urginline,
                            intr,
                            blog,
                            done,
                            reuse,
                            keepopen,
                            linger,
                            delay_acks,
                            destroy,
                            ack_timed,
                            no_check,
                            zapped,
                            broadcast,
                            nonagle,
                            bsdism;
    unsigned long           lingertime;
    int                     proc;

    struct sock             *next;
    struct sock             **pprev;
    struct sock             *bind_next;
    struct sock             **bind_pprev;
    struct sock             *pair;
    int                     hashent;
    struct sock             *prev;
    struct sk_buff          *volatile send_head;
    struct sk_buff          *volatile send_next;
    struct sk_buff          *volatile send_tail;
    struct sk_buff_head     back_log;
    struct sk_buff          *partial;
    struct timer_list       partial_timer;
    long                    retransmits;
    struct sk_buff_head     write_queue,
                            receive_queue;
    struct proto            *prot;
    struct wait_queue       **sleep;
    __u32                   daddr;
    __u32                   saddr;            /* Sending source */
    __u32                   rcv_saddr;        /* Bound address */
    unsigned short          max_unacked;
    unsigned short          window;
    __u32                   lastwin_seq;      /* sequence number when we last
                                                 updated the window we offer */
    __u32                   high_seq;         /* sequence number when we did
                                                 current fast retransmit */
    volatile unsigned long  ato;              /* ack timeout */
    volatile unsigned long  lrcvtime;         /* jiffies at last data rcv */
    volatile unsigned long  idletime;         /* jiffies at last rcv */
    unsigned int            bytes_rcv;
/*
 *    mss is min(mtu, max_window)
 */
    unsigned short          mtu;              /* mss negotiated in the syn's */
    volatile unsigned short mss;              /* current eff. mss - can change */
    volatile unsigned short user_mss;         /* mss requested by user in ioctl */
    volatile unsigned short max_window;
    unsigned long           window_clamp;
    unsigned int            ssthresh;
    unsigned short          num;
    volatile unsigned short cong_window;
    volatile unsigned short cong_count;
    volatile unsigned short packets_out;
    volatile unsigned short shutdown;
    volatile unsigned long  rtt;
    volatile unsigned long  mdev;
    volatile unsigned long  rto;

    volatile unsigned short backoff;
    int                     err, err_soft;    /* Soft holds errors that don't
                                                 cause failure but are the cause
                                                 of a persistent failure not
                                                 just 'timed out' */
    unsigned char           protocol;
    volatile unsigned char  state;
    unsigned char           ack_backlog;
    unsigned char           max_ack_backlog;
    unsigned char           priority;
    unsigned char           debug;
    int                     rcvbuf;
    int                     sndbuf;
    unsigned short          type;
    unsigned char           localroute;       /* Route locally only */
/*
 *    This is where all the private (optional) areas that don't
 *    overlap will eventually live.
 */
    union
    {
          struct unix_opt   af_unix;
#if defined(CONFIG_ATALK) || defined(CONFIG_ATALK_MODULE)
        struct atalk_sock   af_at;
#endif
#if defined(CONFIG_IPX) || defined(CONFIG_IPX_MODULE)
        struct ipx_opt      af_ipx;
#endif
#ifdef CONFIG_INET
        struct inet_packet_opt  af_packet;
#ifdef CONFIG_NUTCP
        struct tcp_opt      af_tcp;
#endif
#endif
    } protinfo;
/*
 *    IP 'private area'
 */
    int                     ip_ttl;           /* TTL setting */
    int                     ip_tos;           /* TOS */
    struct tcphdr           dummy_th;
    struct timer_list       keepalive_timer;  /* TCP keepalive hack */
    struct timer_list       retransmit_timer; /* TCP retransmit timer */
    struct timer_list       delack_timer;     /* TCP delayed ack timer */
    int                     ip_xmit_timeout;  /* Why the timeout is running */
    struct rtable           *ip_route_cache;  /* Cached output route */
    unsigned char           ip_hdrincl;       /* Include headers ? */
#ifdef CONFIG_IP_MULTICAST
    int                     ip_mc_ttl;        /* Multicasting TTL */
    int                     ip_mc_loop;       /* Loopback */
    char                    ip_mc_name[MAX_ADDR_LEN]; /* Multicast device name */
    struct ip_mc_socklist   *ip_mc_list;      /* Group array */
#endif

/*
 *    This part is used for the timeout functions (timer.c).
 */
    int                      timeout;         /* What are we waiting for? */
    struct timer_list        timer;           /* This is the TIME_WAIT/receive
                                               * timer when we are doing IP
                                               */
    struct timeval           stamp;
 /*
  *    Identd
  */
    struct socket            *socket;
  /*
   *    Callbacks
   */
    void                     (*state_change)(struct sock *sk);
    void                     (*data_ready)(struct sock *sk,int bytes);
    void                     (*write_space)(struct sock *sk);
    void                     (*error_report)(struct sock *sk);

};
```

socket

Each socket data structure holds information about a BSD socket. It does not exist independently; it is, instead, part of the VFS inode data structure.

```
struct socket {
  short                type;         /* SOCK_STREAM, ...             */
  socket_state         state;
  long                 flags;
  struct proto_ops     *ops;         /* protocols do most everything */
  void                 *data;        /* protocol data                */
  struct socket        *conn;        /* server socket connected to   */
  struct socket        *iconn;       /* incomplete client conn.s     */
  struct socket        *next;
  struct wait_queue    **wait;       /* ptr to place to wait on      */
  struct inode         *inode;
  struct fasync_struct *fasync_list; /* Asynchronous wake up list    */
  struct file          *file;        /* File back pointer for gc     */
};
```

task_struct

Each task_struct data structure describes a process or task in the system.

```
struct task_struct {
/* these are hardcoded - don't touch */
  volatile long        state;          /* -1 unrunnable, 0 runnable, >0 stopped */
  long                 counter;
  long                 priority;
  unsigned             long signal;
  unsigned             long blocked;   /* bitmap of masked signals */
  unsigned             long flags;     /* per process flags, defined below */
  int errno;
  long                 debugreg[8];    /* Hardware debugging registers */
  struct exec_domain   *exec_domain;
/* various fields */
  struct linux_binfmt  *binfmt;
  struct task_struct   *next_task, *prev_task;
  struct task_struct   *next_run,  *prev_run;
  unsigned long        saved_kernel_stack;
  unsigned long        kernel_stack_page;
  int                  exit_code, exit_signal;
  /* ??? */
  unsigned long        personality;
  int                  dumpable:1;
  int                  did_exec:1;
  int                  pid;
  int                  pgrp;
  int                  tty_old_pgrp;
  int                  session;
  /* boolean value for session group leader */
  int                  leader;
  int                  groups[NGROUPS];
  /*
   * pointers to (original) parent process, youngest child, younger sibling,
   * older sibling, respectively.  (p->father can be replaced with
   * p->p_pptr->pid)
   */
  struct task_struct   *p_opptr, *p_pptr, *p_cptr,
                       *p_ysptr, *p_osptr;
  struct wait_queue    *wait_chldexit;
  unsigned short       uid,euid,suid,fsuid;
  unsigned short       gid,egid,sgid,fsgid;
  unsigned long        timeout, policy, rt_priority;
  unsigned long        it_real_value, it_prof_value, it_virt_value;
  unsigned long        it_real_incr, it_prof_incr, it_virt_incr;
  struct timer_list    real_timer;
  long                 utime, stime, cutime, cstime, start_time;
/* mm fault and swap info: this can arguably be seen as either
   mm-specific or thread-specific */
  unsigned long        min_flt, maj_flt, nswap, cmin_flt, cmaj_flt, cnswap;
  int swappable:1;
  unsigned long        swap_address;
  unsigned long        old_maj_flt;    /* old value of maj_flt */
  unsigned long        dec_flt;        /* page fault count of the last time */
  unsigned long        swap_cnt;       /* number of pages to swap on next pass */
/* limits */
  struct rlimit        rlim[RLIM_NLIMITS];
  unsigned short       used_math;
  char                 comm[16];
/* file system info */
  int                  link_count;
  struct tty_struct    *tty;           /* NULL if no tty */
/* ipc stuff */
  struct sem_undo      *semundo;
  struct sem_queue     *semsleeping;
/* ldt for this task - used by Wine.  If NULL, default_ldt is used */
  struct desc_struct *ldt;
/* tss for this task */
  struct thread_struct tss;
/* filesystem information */
  struct fs_struct     *fs;
/* open file information */
  struct files_struct  *files;
/* memory management info */
  struct mm_struct     *mm;
/* signal handlers */
```

```
  struct signal_struct *sig;
#ifdef __SMP__
  int                   processor;
  int                   last_processor;
  int                   lock_depth;     /* Lock depth.
                                           We can context switch in and out
                                           of holding a syscall kernel lock... */
#endif
};
```

timer_list

`timer_list` data structure's are used to implement real time timers for processes.

```
struct timer_list {
  struct timer_list *next;
  struct timer_list *prev;
  unsigned long expires;
  unsigned long data;
  void (*function)(unsigned long);
};
```

tq_struct

Each task queue (`tq_struct`) data structure holds information about work that has been queued. This is usually a task needed by a device driver but which does not have to be done immediately.

```
struct tq_struct {
    struct tq_struct *next;   /* linked list of active bh's */
    int sync;                 /* must be initialized to zero */
    void (*routine)(void *);  /* function to call */
    void *data;               /* argument to function */
};
```

vm_area_struct

Each `vm_area_struct` data structure describes an area of virtual memory for a process.

```
struct vm_area_struct {
  struct mm_struct * vm_mm;  /* VM area parameters */
  unsigned long vm_start;
  unsigned long vm_end;
  pgprot_t vm_page_prot;
  unsigned short vm_flags;
/* AVL tree of VM areas per task, sorted by address */
  short vm_avl_height;
  struct vm_area_struct * vm_avl_left;
  struct vm_area_struct * vm_avl_right;
/* linked list of VM areas per task, sorted by address */
  struct vm_area_struct * vm_next;
/* for areas with inode, the circular list inode->i_mmap */
/* for shm areas, the circular list of attaches */
/* otherwise unused */
  struct vm_area_struct * vm_next_share;
  struct vm_area_struct * vm_prev_share;
/* more */
  struct vm_operations_struct * vm_ops;
  unsigned long vm_offset;
  struct inode * vm_inode;
  unsigned long vm_pte;      /* shared mem */
};
```

File translated from TEX by TTH, version 1.0.

Top of Chapter, Table of Contents, Show Frames, No Frames

© 1996-1999 David A Rusling copyright notice.