INTRODUCTION TO KERNEL EMT CALLS

Section B of this manual lists all the kernel EMT entries into the kernel from the kernel mode processes. In most cases two calling sequences are specified, one of which is usable from assembly language, and the other from C. Most of these calls have an error return. From assembly language an erroneous call is always indicated by turning on the c-bit of the condition codes. The presence of an error is most easily tested by the instructions *bes* and *bec* ("branch on error set (or clear)"). These are synonyms for the *bcs* and *bcc* instructions.

From C, an error condition is indicated by an otherwise impossible returned value. Almost always this is −1; the individual sections specify the details.

A kernel process is defined in the kernel in a *DCT* (dispatcher control table) entry. The structure of a DCT entry is:

```
struct dct {
        int     d_stat;         /* process status bits */
        int     d_link;         /* pointer to next process on this queue at this hardware priority */
        int     d_sleep;        /* sleep bit pattern */
        int     *d_msg;         /* pointer to first message on input queue */
        int     d_timeout;      /* time-out value in 60ths of a second */
        int     d_events;       /* event flags */
        int     d_prc;          /* process number */
        char    d_chan;         /* control channel */
        char    d_ucnt;         /* process user count */
        int     d_pcb;          /* physical block address of start of code for kernel process, PCB
                                   segment ID for supervisor-user process */
        char    d_iprior;       /* initial priority (not used for kernel process) */
        char    d_cprior;       /* current priority (not used for kernel process) */
        char    d_msgcnt;       /* count of messages on process input queue */
        char    d_age;          /* used by scheduler */
}
```

A kernel mode process runs at its specified hardware priority in *d_stat* and uses I-space only to provide protection for the kernel tables in D-space. The code for a kernel process must begin at address 060000 and may be up to 12K words long. Kernel base register 3 (KBR3) is set up to the beginning of the code. KBR4 and KBR5 are only set up if required. However KBR5 is used by the *iomap* emt call to set up a transfer directly into a supervisor or user segment for character I/O. KBR6 is always set up for access to the kernel library routines, such as the general tty routines. A kernel process uses the kernel stack. It also has access to the I/O address space by means of KBR7 and to the kernel message buffers by means of KBR0.

Each kernel process has a 27 word header preceding the actual code with the following structure:

```
struct k_pcb {
        struct {
                int sar;        /* segment address register */
                int sdr;        /* segment descriptor register */
        } kr[6];                /* kernel base register settings for KBR3, KBR4, KBR5 (I and D)
```

```
                        */
        int     k_segid[6];     /* segment ID's (I and D) */
        int     k_sav;          /* pointer to base register save routine in kernel */
        int     k_ent[3];       /* process entry points:
                                     event entry point
                                     emt entry point
                                     fault entry point */
        int     k_pn;           /* kernel process number */
        char    k_intflg;       /* interrupt flag set by kernel when interrupt occurs for this process
                                */
        char    k_nubmap;       /* number of UNIBUS map registers used by this process */
        int     *k_fubmap;      /* pointer to first UNIBUS address map register */
        int     k_rsave[3];     /* register save area upon entry to this kernel process */
}
```

The variable *k_pn* is externally referred to by *process* in a kernel-mode process driver. The variable *process* is globally defined. The kernel process is dispatched to by means of an interrupt or by the PIR (programmed interrupt request) triggered by an event. A kernel process does not need to be attached to an interrupt vector. In this case the process is dispatched to by means of an EMT call from the supervisor (EMT code >=192.). Upon entry to the kernel process, the kernel saves the current settings of KBR3 and of KBR4 and KBR5 only if used. The base registers are restored on exit.

The kernel processes as well as the supervisor processes communicate via messages. However, kernel processes may read and write the message buffer area directly. EMT trap calls are provided to allocate and free messages in the kernel buffer area. The message consists of a six word header and up to 106 words of data. The size of the message may be a multiple of 16 words up to a total of 112 words. The layout of the message is defined by the structure:

```
struct {
        int     *mslink;        /* link to next message on input queue */
        int     msfrom;         /* process from which message was  received */
        int     msto;           /* process to which message is to be sent */
        char    mssize;         /* bits 0-2:  size in multiples of 16 words
                bit 3:          allocated bit
                bit 4:          acknowledgement bit
                bit 5:          iolock bit
                bit 6:          message contains capability
        char    mstype;         /* message type */
        int     msident;        /* message identification wordonly used by sender */
        char    msstat;         /* status byte set by receiver in acknowledgment message or by sys-
                                   tem if receiver process does not exist */
        char    msseqnum;       /* message sequence number */
}
```

The data in the body of the message must be filled in directly by the sender of the message.

ALOCMSG(b) ALOCMSG(b)

NAME

alocmsg – allocate message buffer

SYNOPSIS

(alocmsg = 12.)
alocmsg(nwords)
int nwords; /* number of words */

DESCRIPTION

Alocmsg allocates a contiguous piece of kernel memory in the message buffer pool. The size of message allocated is a multiple of 16 words and equal to or greater than *nwords* plus the size of the message header *msghdr*. The allocate bit and message size bits are set in the *mssize* byte of the message header. The rest of the message buffer is zeroed out. A pointer to the beginning of the message is returned in C.

In assembly language, r0 must contain the size of message buffer required in words. A pointer to the message buffer is returned in r0.

SEE ALSO

queuem(b), messink(b), dequeuem(b), freemsg(b), dqtype(b), queuemn(b)

DIAGNOSTICS

A 0 is returned from C if no message buffer space exists or if the size of message requested is greater than 112 words minus the message header size (6 words).

In assembly language, the c-bit is set to indicate an error.

FUTURE AND DMERT DIAGNOSTICS

A null pointer is returned if no message buffer space exists or if the size of message requested is greater than MAXMSG bytes minus the message header size .