INTRODUCTION TO SUPERVISOR EMT TRAPS

The interface between a supervisor process and the kernel consists of a read only data segment called the process control block (PCB), approximately fifty EMT traps, and a few messages associated with creation and termination of a process and its children.

The *PCB* describes the process virtual address space (both supervisor and user modes), defines entry points for handling interrupts (events) and faults, and provides the scheduler with space for saving the state of the machine when switching between processes. The structure of the PCB is:

```
struct pcb {
        int     p_pnum;                 /*process number*/
        int     p_parent;               /*Process number of parent process */
        char    p_prior;                /*Initialschedulingpriority*/
        char    p_chan;                 /*Process control channel */
        char    p_name[8];              /* ASCII name of process */
        int     p_ttg;                  /*Scheduler saves time-to-go of preempted process time slice*/
        int     p_slice;                /*Process time slice in 1/60 seconds*/
        int     p_size;                 /*Total number of bytes in segment table p_tab*/
        struct {
                int ktime;              /*Total time spent in kernel mode while process was active*/
                int stime;              /*Total time spent in supervisor mode while process was ac-
                                        tive*/
                int utime;              /*Total time spent in user mode while process was active*/
        } p_times;
        char    p_timeout;              /*Inhibit context change flag for the scheduler*/
        char    p_wait;                 /*Hold in memory until time slice runs out flag to the
                                        scheduler*/
        char    p_tflag;                /* If non-zero a terminate message of type p_ttype will be sent
                                        to parent upon the death of this process*/
        char    p_ttype;                /*Message type to send to parent on death of this process*/
        char    *p_tident;              /* Message Ident to be sent to parent on death of process*/
        int     p_cwait;                /* Scheduler flag used for conditional roadblocks and condition-
                                        al waits */
        int     p_semafor;              /* not used */
        int     p_dummy;                /* not used */
        struct psdd {
                int     ps;
                char    *pc;
        }       p_topsd;                /*Scheduler save pc and ps of process on time out or preemp-
                                        tion*/
        struct {
                int     p_toreg[6];     /* r0-r5 saved on context change */
                int     p_ssp;          /*supervisor stack pointer*/
                int     p_usp;          /*User stack pointer*/
                int     p_kssr3;        /*Current state of supervisor and user D space register*/
                double                  p_fprg[6];/*Floating point registers */
                int     p_fps;
        }       p_tosave;               /*Scheduler saves the context of a process here*/
        int     p_event;                /*Event flags are stored here */
        struct  psdd p_evect;    /* ps and pc for vectoring to event handling routine*/
        struct  psdd p_evpsd;    /* Interrupted ps, pc are saved here */
        int     p_evmask;        /* Event mask word; a 1 in each bit position enables the
                                 corresponding event */
        struct  psdd p_fvect;    /* ps, pc for vectoring to supervisor fault handling routine*/
        int     p_fcode;         /* Type of fault */
        struct  psdd p_fpsd;     /* ps, pc save area */
        int     p_fsav[3];       /* Save area for fault dependent parameters - for segmentation
        faults ssr0, ssr1, and ssr2; for floating point faults FEC, FEA,
                                 and floating point status register */
        struct  psdd p_emtpsd;   /* ps, pc on EMT traps to the kernel */
        int     p_emtsave[6];    /* r0 -r5 are saved here on EMT traps to the kernel */
        struct p_ctab            {
            int     p_owner;     /* capability owner */
            int     p_cap;       /* capability */
        }       p_clist[P_NCAP]; /* capability list */
        struct {
            int     pstat;       /* Segment status word*/
            int     psegid;      /* Segment ID */
        } p_tab[];               /* segment table entries */
    }
```

The capability list *p_clist[]* is a list of the valid "capabilities" which the process has. A capability is a two-word entry which is put in the PCB by the memory manager process. A process may be given a capability only by the "owner" of the capability. Typically, upon opening a file, the file manager will send an "add capability" message to the memory manager process. The memory manager will bring the PCB into its address space, find an empty capability slot and put the owner (file manager process number (4)) into the owner field of the capability and the capability itself into the capability field. The capability for the file manager is encoded as follows: 8 bits for in-core inode, 2 bits for read/write permissions and 6 bits for inode usage value. When a read or write message is sent to the file manager, a capability must be specified; this is checked for valid access permissions on the file by the file manager. Upon closing a file, the file manager sends a "delete capability" message to the memory manager.

The segment table *p_tab[]* defines the supervisor and user virtual address spaces. The table contains a minimum of three entries which, in the order they appear, are: the PCB segment, a supervisor stack segment, and a code segment. The maximum number of entries is currently limited to 48 (no more than 32 can be active at one time). Each table entry defines how a segment is to be accessed by the process (see *pstat* below). The segments specified by *psegid* are contiguous pieces of memory and (swap space on disk) which vary from 32 to 32K words in 32 word increments. The segment *ID* is a pointer to a table in the kernel (the *RSDE* table) which has the structure:

```
    struct      rsde             {
        char    *r_ptr;          /* Pointer to memory management tables*/
        int     *r_leng;         /* Segment length in 32 word blocks */
        char    r_ucnt;          /* Number of users */
        char    r_status;        /* Segment status flags */
        char    *r_disk;         /* Starting block on the swap device */
        int     r_name[2]        /* Unique 32 bit name */
    };
```

The sharing of segments by independent cooperating processes is accomplished via the r_name entry in the RSDE table. The processes need only create a segment with the same unique name in order to share the same physical segment. The system convention for establishing unique names for segments is based on the premise that shared segments will contain some initial data, and this data will reside in a file. The name is simply the absolute disk address (major, minor device and block number) of the first block of the initialized data. For processes which have a parent child relationship, sharing is accomplished by passing a segment ID between parent and child.

The bits of *pstat* are defined as follows:

bit 15	1 if segment is active. Symbolic name *pcbnn* for need now.
bit 14	1 if segment is needed next time the process is activated. Symbolic name *pcbnxt* for next.
bit 13:12	Address space: 1 for supervisor, 2 for user
bit 11	1 if D-space
bit 10:8	Starting segmentation register number.
bit 7	1 if segment has been made non-swap by the process
bit 6	1 if memory manager did not load this segment (even though *pcbnn* or *pcnxt* were set) because of insufficient swap space. Symbolic name *pcbnold.*
bit 5	1 if segment is sharable. Symbolic name *pcbshare.*
bit 4	1 if segment is writeable by the process. Symbolic name *pcbwrite.*
bit 3	1 if segment is a stack segment. Symbolic name *pcbed.*
bit 2:0	Access setting for the segment.

In this section of the manual, the statement "segment indexed by *segnum*" means that the kernel will access the PCB segment table:

```
struct pcb p;
.....
p.p_tab[ segnum ].pstat ...
```

The kernel procedure on all traps from supervisor and user mode, except break point traps from supervisor (a core dump is produced in /cdmp/p_name), is:

1) The psd saved on the stack is put into the PCB at *p_fpsd.*
2) For floating point or segmentation traps, the appropriate registers are saved in *p_fsav.*
3) The fault code is saved in *p_fcode:*

0	bus error
1	illegal instruction
2	trace trap
3	IOT trap
4	power fault
5	EMT (from user only)
6	trap trap
8	floating point exception
9	segmentation fault
10	invalid emt (from supervisor mode only)

4) The new ps and pc are extracted from *p_fvect* and put on the kernel stack.
5) The c-bit in the new ps is set if the trap is a segmentation fault from supervisor mode.
6) If *f_code* = 10, *p_fvect* is cleared.
7) The kernel executes an rti to give control to the supervisor trap handling routine.

The kernel procedure on the arrival of an event is:

1) The appropriate bit in *p_event* is set.
2) If the corresponding bit in *p_evmask* is set and the psd in *p_event,* is defined, the interrupted ps and pc are posted in *p_evpsd* and an rti is executed to the psd in *p_evpsd.*

The description of all the EMT calling sequences are given for C. In most cases no reference to the assembly language calling sequence is given. One can translate the C call to the assembly call by putting the address of the argument list in r0 and substituting c-bit set for -1 returns. In cases where the assembly language returns are more complex than simple success or fail, a complete description is included.

The message formats for creating and terminating processes are discussed in section C of the manual.