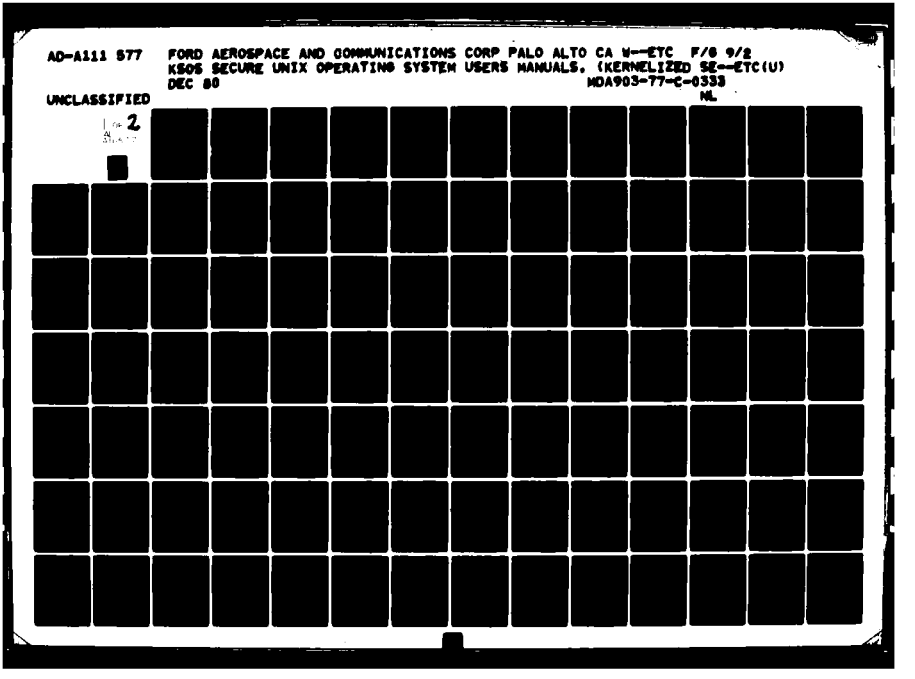

AD-A111 577
FORD AEROSPACE AND COMMUNICATIONS CORP PALO ALTO CA W--ETC F/G 9/2
KSOS SECURE UNIX OPERATING SYSTEM USERS MANUALS. (KERNELIZED SE--ETC(U)
DEC 80
MDA903-77-C-0333
UNCLASSIFIED
NL
2

11157

1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS

AD A111577

December 1980

(2)

SECURE MINICOMPUTER OPERATING SYSTEM (KSOS)

SECURE UNIX OPERATING SYSTEM

USERS MANUALS

Department of Defense Kernelized Secure Operating System

Contract MDA 903-77-C-0333
CDRL 0002AJ

DTIC
ELECTE
MAR 0 3 1982

Prepared for:

Defense Supply Service - Washington
Room 1D245, The Pentagon
Washington, DC 20310

DTIC FILE COPY

Approved for public release; distribution unlimited.

Ford Aerospace &
Communications Corporation
Western Development
Laboratories Division

3939 Fabian Way
Palo Alto, California 94303

NAME

acc_mode - access mode

SYNOPSIS

bits;

DESCRIPTION

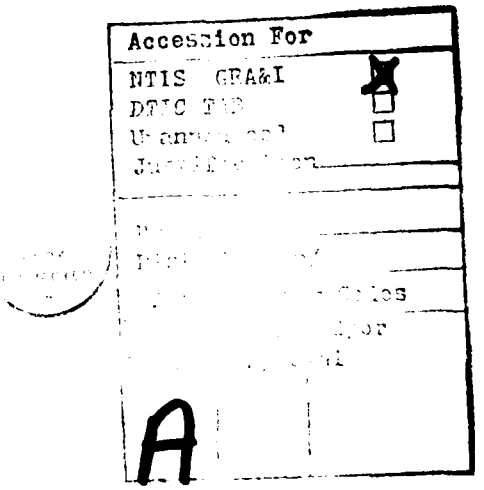
Acc mode describes the access allowed a process to a particular segment of its current segment set. An acc mode is composed of three bits indicating permission to read, to write, and to execute. They are equivalent to selected discretionary acc bits. The correspondences are:

```
CONST
        readAcc      = owner Read;
        writeAcc     = owner Write;
        executeAcc   = owner Execute;
```

The acc mode of a segment is initialized when its segment is made a part of the process's segment set using K build segment or K rendezvous segment. The acc mode may be changed using K set segment status or K remap. In no case does a segment's acc mode (which is always in relation to a process) allow more permissive access than that allowed the process by the segment's discretionary access.

SEE ALSO

K_remap (II), *K_rendezvous_segment* (II), *K_set_segment_status* (II), discr_access (I), seg_stat_block (I),



NAME

compart_set - set of security compartments

SYNOPSIS

bits32;

DESCRIPTION

A maximum of 32 security compartments may be used in a KSOS system. The type independent information (the tii_struct) of every object contains a compart set field that determines which security compartments the object belongs to, with each bit indicating one compartment. A process may not read an object if the object belongs to any compartments which the process does not belong to. A process may not write on an object if the process belongs to any compartments that the object does not belong to.

The 32 compartments are given arbitrarty names via an enumerated type as is shown below:

```
compartments = (
    SYSTEM,        compart_1,     compart_2,     compart_3,
    compart_4,     compart_5,     compart_6,     compart_7,
    compart_8,     compart_9,     compart_10,    compart_11,
    compart_12,    compart_13,    compart_14,    compart_15,
    compart_16,    compart_17,    compart_18,    compart_19,
    compart_20,    compart_21,    compart_22,    compart_23,
    compart_24,    compart_25,    compart_26,    compart_27,
    compart_28,    compart_29,    compart_30,    compart_31
    );
```

SEE

tii_struct(I)

NAME

discr_access - discretionary access

SYNOPSIS

bits;

DESCRIPTION

The discr access type describes the discretionary access to Kernel objects. Discretionary access is divided into three sets of three bits each. The first set refers to the object's owner, the next to others in the same user group, and the last to all others. Within each set the three bits indicate permission to read, to write, and to execute. The execute bits are meaningful only if a process has the privRealizeExecPermission privilege. Each execute bit is then equivalent to the corresponding read bit. Setuid and setgid are special bits used in conjunction with the execute bits. When a process image is created from an object with the setuid bit, the owner of the process is temporarily changed to be the owner of that object. Setgid is analogous to setuid, operating on group rather than owner.

The following table contains the bit definitions for the discretionary access field.

setuid	10
setgid	9
ownerRead	8
ownerWrite	7
ownerExecute	6
groupRead	5
groupWrite	4
groupExecute	3
allRead	2
allWrite	1
allExecute	0

SEE ALSO

K_build_segment (II), K_creat (II), K_set_da (II), priv (I)

NAME

file_stat_block - file status block

SYNOPSIS

```
file stat block = RECORD

     f size          : cardinal32;
     subtype         : seid;
     time last mod   : cardinal32;
     open at crash   : boolean
END;
```

DESCRIPTION

The file status block is used to return information about files, terminals, extents and devices.

F_size is the number of characters in the file. The kernel keeps internal track of the highest block number written in the file. The K device function SETFILESIZE can also be used to set this field.

Subtype is the Seid of the subtype of the I/O object or the null seid if the object is not subtyped.

Time_last_mod is the time of last modification of the text of a file. It is represented as the number of ticks since January 1, 1980. This time is only updated on K open and K close if the open mode allows write access. For all I/O objects other than files, this value is zero.

Open_at_crash is true if the file was open for writing when the system crashed. The next open for writing will clear open at crash. The file system recovery program(s) should be run on all files with this field set. For non-file objects, open at crash is always false. It should be noted that when open at crash is true, f size may be incorrect.

SEE ALSO

K_get_file_status (II), K_device_function (II), Seid (I)

NAME

integrity_cat_type - integrity categories

SYNOPSIS

```
integrity cat type = (      integrity cat 0,      integrity cat 1,
integrity cat 2,       integrity cat 3,       integrity cat 4,
integrity cat 5,       integrity cat 6,       integrity cat 7,
integrity cat 8,       integrity cat 9,       integrity cat 10,
integrity cat 11,      integrity cat 12,      integrity cat 13,
integrity cat 14,      integrity cat 15       );
```

DESCRIPTION

There are a 16 possible integrity categories on a KSOS system. These categories are defined in the integrity cat type enumerated type. The integrity categories form an ordered set which, along with the security category and security compartments, control access to and from the objects of the system. If object A has a higher integrity category than object B then:

```
A    cannot read    B
A    can write      B

B    can read       A
B    cannot write   A
```

SEE

compart_set(I) security_cat_type(I)

INTRODUCTION TO KERNEL INTERFACE TYPES

Section I of this manual describes all the types used by the Kernel Call Interface (section II). The kernel types defined in the interface are composed of 7 different basic types and 3 structure types. All the kernel type definitions are built from these types. Understanding these types is essential for using and building future kernel call interfaces.

char — is an eight bit unsigned quantity. It is used to represent variables with up to 256 distinct values. A char field must be and'ed with a 377(8) mask when converted to a 16 bit quantity because of the PDP-11 hardware sign extension on byte to word data transfer instructions.

boolean — is also an eight bit quantity. A boolean variable may have one of two values - true or false The false value is defined always to be zero. The true value is any nonzero value.

bits — is a 16 bit quantity. The bits are numbered right to left starting with zero (i.e. bit 0 is the rightmost (least significant) bit).

bits32 — as suggested by its name, is 32 bits long. However, the bits are numbered left to right starting with zero (i.e., bit 0 is the leftmost (most significant) bit). If bits32 is accessed as an array of bits, a[0] contains bits 0 through 15 and a[1] contains bits 16 to 31.

integer — is a 16 bit signed quantity in two's complement notation.

cardinal — is a 16 bit unsigned quantity in two's binary notation.

cardinal32 — is a 32 bit unsigned quantity. If cardinal32 is accessed as an array of cardinal, a[0] contains the high order word and a[1] contains the low order word.

The structured types are used to build data structures from the basic types. The structured types are enumerations, records and arrays.

Enumerations — An enumeration is a list of identifiers which denote the values constituting a data type. These identifiers may be used as constants in programs. They, and no other values, belong to the enumerated type. An ordering relation is defined on these values by their sequence in the enumeration.

Records — A record structure consists of a number of components where each component is identified by a unique field identifier. Field identifiers are known only within the record structure definition and within field designators, i.e., when they are preceded by a qualifying record variable identifier.

Arrays

An array structure consists of a number of components. Each component is identified by a number of indices whose range is specified in the declaration of the array structure.

SEE ALSO

Modula Specifications, NEWcalls.mod

NAME

ioStatus - Status of I/O operation

SYNOPSIS

```
ioStatus = RECORD

     devIndep: K_err_num;
     devDep  : bits
END;
```

DESCRIPTION

devIndep — contains the device independent status upon completion of an I/O operation. The status returned is one of the Kernel's exception numbers.

devDep — contains the device dependent status upon completion of an I/O operation. This field is only defined for special devices, such as the network devices, which have peculiar properties.

SEE ALSO

K_device_function (II), K_read (II), K_write (II), K_err_num (I)

NAME

ipc_block - inter-process communication messages

SYNOPSIS

```
msg_limit  = 11;

msg_struct = ARRAY 0:msg_limit OF char;

ipc_block = RECORD

      ipc_seid: seid;
      ipc_msg : msg_struct
END;
```

DESCRIPTION

An ipc_block is the general form for all the different types of inter-process communication messages (e.g. signals and IPC's). Messages can be generated by the kernel, by your process or by other processes.

The ipc_seid contains the seid of the process which sent the ipc_block.

The ipc_msg contains the text of the ipc_block.

The first byte of the ipc_msg, by convention, is the event type. This field and the ipc_seid, determines the message type. The following event types are predefined and can not be used for other purposes. All other event type numbers can be used by cooperating processes as they wish.

null_event	0	
memerr_event	1	memory error
bpt_event	2	break point trap instruction
iot_event	3	input output trap instruction
cpuerror_event	4	cpu error
illinst_event	5	illegal instruction
mm_event	6	memory management
fltpnt_event	7	floating point processor
ttoggle_event	10	timer toggle
talarm_event	11	timer alarm
emulcall_event	12	emulator call
iocomplete_event	13	I/O completion

Kernel Generated Messages

This type of message can only be generated by the kernel. These messages always come through the hardware pseudo interrupt-channel. The seid field contains the Kernel Seid. The event type must be one of the following:

memerr_event
bpt_event
iot_event

```
cpuerror_event
illinst_event
mm_event
fltpnt_event
```

The time of the pseudo-interrupt is placed in the last four bytes of the ipc msg.

The floating point processor message also has its status register stored in bytes 2 and 3 of the ipc msg.

Messages on the Behalf of Your Process

These messages can only be generated by some action to or by your process.

The seid field contains the Kernel Seid except I/O completion which contains the process Seid. Messages of this type have the following event types:

```
ttoggle_event
talarm_event
emulcall_event
iocomplete_event
```

The timer and I/O completion come through their respective pseudo-interrupt vectors. The emulator call pseudo-interrupts through at base level. This pseudo-interrupt will occur no matter what the pseudo-interrupt level is set at.

The emulator call message occurs synchronously when your process requests them. An I/O completion message occurs asynchronously after your process initiates an I/O requesting such a message. The timer messages can be caused either by your process or by another process setting the timer related fields in the process status block.

The last four bytes of both the timer messages and the emulator call contain the time that the pseudo interrupt occurred.

The I/O completion message has the following structure:

```
RECORD
    event_type : char;
    stat       : ioStatus
    byteCount  : cardinal;
    async      : cardinal;
    filler     : ARRAY 1:2 OF char;
END;
```

where :

event_type has the value of iocomplete event.

stat.devIndep contains the device independent completion status. This field is valid for all I/O completion messages.

stat.devDep contains the device dependent completion status. This field is valid only for special devices. The meaning of this field can be found with the supported device manual pages.

byteCount is the number of characters transferred by this asynchronous request.

async is the user supplied asynchronous identifier.

Messages generated by Processes

Messages of this type are sent using K_signal or K_post and are received either through the pseudo interrupt vectors or by K_receive. The seid field contains the sending process seid. The ipc_msg record can contain anything. By convention, the first byte contains the event type.

SEE ALSO

ioStatus (I), Seid (I), Pseudo Interrupts (I), K_post (II), K_signal (II), K_receive (II)

NAME

K_err_num - Kernel error numbers

SYNOPSIS

cardinal;

DESCRIPTION

This list of Kernel error numbers is subject to revision pending resolution of discrepancy reports concerning them. The actual numerical value for each exception may be found in NEWcalls.mod.

KPR related Kernel calls and the exception values they return follow:

XOK	No exception
XKemt	kernel generated EMT
XMapPr	bad mapping process
Xaborted	I/O request was aborted
Xasync	asynchronous I/O initiated
XatEot	OK but tape is now after EOT
Xattn	secure attention char input
XbAddr	bad address
XbEmt	bad emt
XbFn	bad function code
XbLcArgSg	bad location for arg seg
XbMapIn	cannot map in PCS
XbMapOut	cannot map out PCS
XbMovIn	bad PCS move in
XbMovO	bad PCS move out
XbNoMes	No message
XbParam	bad Kernel interface param
XbPcsRef	bad PCS reference
XbPcsSd	bad PCS seid
XbPm	
XbPrSd	bad process seid
XbPsLev	bad pseudo interrupt level
XbSchL	bad scheduler action
XbSgDes	bad seg des
XbSgRng	bad seg range
XbSgSd	bad seg seid
XbSgSz	seg size is not mult of 512
XbStPm	bad status parameter
XbSwp	bad PCS swap
XbSz	bad size parameter
XbSzArgSg	bad size for arg seg (too large)
XbTiiPm	bad tii parameter
XbadBlockNo	bad block number on I/O
XbadDa	SMXdap failure
XbadFs1	1: did not find expected slot kind
XbadFs2	2: bad checksum in slot
XbadFs3	3: reserved
XbadLink	K_link overflowed count (LEAK)
XbadModes	illegal combination of open modes

XbadNsp	
XbadOd	
XbadPriv	
XbadRefCount	
XbadSize	invalid size in pBlock
XbadSlot	bad upt slot
XbadStCap	
XbadSubtype	
XbadSubtypeMatch	
XbadVol	unacknowledged disk volume
XbdDm	bad domain
XbdKcl	bad Kernel call
XblAdVacMmu	block addr has vacant page reg
XblNtInSg	block not in seg
XbpT	break point trap
Xbusy	Exclusive use fails because busy
XcPsInt	cannot pseudo interrupt
XchgIDspc	attempt to change I/D space of seg
XchgSgOwn	attempt to change owner of seg
XcpuT	cpu error
XcritExcl	
XdapViol	
Xdown	Hardware is now down
XdupSg	attempt to duplicate use of a seg
Xempty	empty upt slot
XendOfFile	read would pass end of file
Xerror	Retryable hardware error
XexFile	Exclusive open on fork attempt
XexSpace	extent table full (LEAK)
Xfault	Non-retryable hardware error
XfloT	floating point unit trap
XiiT	illegal instruction
XinBuf	Input buffer illegal for fn
XinSgAldMap	incoming seg already mapped
XindMap	bad mapping index
XioT	I/O trap
Xmark	EOF mark read from magnetic tape
XmemT	memory error
XmmT	memory management trap
Xmoving	offline motion (rewind) going
XmtSpace	mount table full (LEAK)
XnPgSg	seg does not cross page boundary
XnPvLkSg	no priv to set mem_lock (swappable)
XnPvStkSg	no priv to set swap_lock (sticky)
XnSgDes	not a seg des (out of range)
XnWrtArgSg	non-writable arg seg
XncnDo	cannot say why
XnoAcc	no access
XnoAsync	asynchronous queue full (LEAK)
XnoClass	Object wrong class for this call
XnoExclWrite	
XnoFile	
XnoFunct	Hardware does not support this fn

XnoHelp	no swap help possible
XnoInit	mapping process not init
XnoMap	no existing map
XnoMode	No open mode given
XnoObj	object non-existent
XnoOwner	
XnoPriv	does not have privilege
XnoProc	no more process allowed
XnoSLev	no privSetLevel
XnoSPriv	no privSetPriv
XnoSpace	No file system space left
XnoStCap	
XnoTranquil	tranquility violation
XnotReadable	Object not open for reading
XnotWritable	
XnwOutSg	non-writable outgoing segment
XodSpace	
XopenFiles	Global open table full (LEAK)
XoutSgAldUmp	outgoing seg already unmapped
XpostEh	POST exhausted
XprvFn	privileged function not allowed
XsgMap	seg mapped when setting acc,loc
XsgNoAcc	seg no access
XsgSw	seg swapped out
XshSg	seg is sharable
XsltAl	bad PCS slot allocation
Xtimeout	Hardware did not respond in time
XunInt	cannot interrupt for K_signal
XvrtMmCfl	virtual memory conflict
XwOnlSg	attempt to make write-only seg

NAME

openDescriptor - open file handle

SYNOPSIS

integer;

DESCRIPTION

OpenDescriptors are used to reference open objects in the various I/O Kernel calls. When a K_create or a K_open is done, an openDescriptor is returned.

SEE ALSO

K_close (II), K_create (II), K_device_function (II), K_get_file_status (II), K_open (II), K_read (II), K_set_file_status (II), K_write (II)

NAME

openModes - open modes

SYNOPSIS

```
openModes = RECORD

        read            : boolean;
        write           : boolean;
        exclusive read  : boolean;
        exclusive write : boolean
    END;
```

DESCRIPTION

The OpenModes record is used to indicate the mode in which an object is to be opened or created. The list below describes each field.

read — when true, indicates that the object should be opened or created for reading.

write — when true, indicates that the object should be opened or created for writing.

exclusive_read — when true, indicates that the object should be opened or created for reading by the calling process only.

exclusive_write — when true, indicates that the object should be opened or created for writing by the calling process only.

The only valid combinations of open modes fields are:

		exclusive		
read	write	read	write	
true	false	false	false	Normal read.
false	true	false	false	Normal write.
true	true	false	false	Normal read and write.
true	true	false	true	Lock out all other writers.
true	true	true	true	Lock out everybody else.

All other combinations are invalid.

SEE ALSO

K_creat (II), K_open (II)

NAME

p_block - parameter block

SYNOPSIS

```
RECORD

  location: virt_loc;
  b_size  : cardinal;
END;
```

DESCRIPTION

P_block contains the I/O buffer description.

location is the virtual location of the I/O buffer.

b_size is the byte size of the I/O buffer.

SEE ALSO

virt_loc (I), K_device_function (II), K_read (II), K_write (II)

NAME

priv_struct - set of privileges

SYNOPSIS

bits32;

DESCRIPTION

Each bit corresponds to a privilege in the priv names enumeration. See the B-5 specifications for the meaning of the different privileges.

```
priv_names = (                          (*bit number*)

        privFileUpdateStatus,           (* 00 *)
        privLink,                       (* 01 *)
        privLockSeg,                    (* 02 *)
        privModifyPriv,                 (* 03 *)
        privMount,                      (* 04 *)
        privSetLevel,                   (* 05 *)
        privStickySeg,                  (* 06 *)
        privSetPath,                    (* 07 *)
        privViolSimpSecurity,           (* 08 *)
        privViolStarSecurity,           (* 09 *)
        privViolSimpIntegrity,          (* 10 *)
        privViolStarIntegrity,          (* 11 *)
        privViolDiscrAccess,            (* 12 *)
        privRealizeExecPermission,      (* 13 *)
        privSignal,                     (* 14 *)
        privWalkPTable,                 (* 15 *)
        privHalt,                       (* 16 *)
        privKernelCall,                 (* 17 *)
        privViolCompartments,           (* 18 *)
        privSetComm,                    (* 19 *)
        privImmigrate,                  (* 20 *)
        privViolTranquility             (* 21 *)
    );
```

SEE ALSO

K_set_priv (II)

NAME

pseudo_int_vector - pseudo-interrupt vector

SYNOPSIS

```
pseudo_int_vector = RECORD
            interrupted_pc          : cardinal;
            interrupted_ps          : cardinal;
            interrupted_p_level     : cardinal;
            int_msg                 : ipc_block;
            new_pc                  : cardinal;
            new_ps                  : cardinal
END;
```

DESCRIPTION

interrupted_pc
: pc of process when (before) pseudo-interrupt asserted

interrupted_ps
: ps of process when (before) pseudo-interrupt asserted

interrupted_p_level
: pseudo-interrupt level of process when (before) pseudo-interrupt asserted

int_msg
: pseudo-interrupt message

new_pc
: pc of process after pseudo-interrupt asserted

new_ps
: ps of process after pseudo-interrupt asserted

pseudo interrupt levels :

```
base_lev   = 0;        (* emulator call        *)
ipc_lev    = 1;        (* IPC pseudo int       *)
time_lev   = 2;        (* timer pseudo int     *)
sig_lev    = 3;        (* K_signal pseudo int  *)
io_lev     = 4;        (* I/O pseudo int       *)
hard_lev   = 5;        (* hardware pseudo int  *)
```

NAME

proc_stat_block - process status block

SYNOPSIS

```
proc_stat_block = RECORD

        own     : seid;
        parent  : seid;
        family  : cardinal;

        owner   : cardinal;
        group   : cardinal;

        aprio   : cardinal;
        pseint  : cardinal;
        ps      : cardinal;

        alarm   : cardinal;
        clock   : cardinal32;
        tt      : boolean;

        stiming : cardinal;
        utiming : cardinal
END;
```

DESCRIPTION

This record contains process dependent information.

A fields with an asterisk by its name is a read only variable.

*own — is the seid of the process associated with this status block.

*parent — is the seid of the process which created the process, named by own

family — is the family name controlled by the NKSR. This field may only be set by processes with the privilege privSetLevel.

owner — is the real user identifier. This field may only be set by processes with the privilege privSetLevel.

group — is the real group identifier. This field may only be set by processes with the privilege privSetLevel.

aprio — is the advisory priority of the process. A process in user domain can set this field between 0 and 63. A process in supervisor domain can set this field between 0 and 127. A process with privSetLevel can set this field between 0 and 255. Priority 0 is the lowest priority.

pseint	is the current pseudo-interrupt level of the process.
ps	is the processor status word of the process.
alarm	is the value of the real-time timer alarm in clock ticks. Setting this field activates the real-time timer which will generate a timer pseudo-interrupt. Setting this field to zero cancels the alarm.
clock	is the value of the time of day clock in clock ticks. This field is read-only except for the Initial Process.
tt	is the timer toggle clock. Setting this field to true causes a timer toggle pseudo-interrupt on every clock tick.
*stiming	is the amount of time spent in the supervisor domain in clock ticks.
*utiming	is the amount of time spent in the user domain in clock ticks.

SEE ALSO

priv_struct (I), seid (I) tii_struct (I), K_get_process_status (II), K_set_process_status (II)

NAME

security_cat_type - security categories

SYNOPSIS

```
security cat type = (       security cat 0,       security cat 1,
security cat 2,         security cat 3,         security cat 4,
security cat 5,         security cat 6,         security cat 7,
security cat 8,         security cat 9,         security cat 10,
security cat 11,        security cat 12,        security cat 13,
security cat 14,        security cat 15         );
```

DESCRIPTION

There are a 16 possible security categories on a KSOS system. These categories are defined in the security cat type enumerated type. The security categories form an ordered set which, along with the integrity category and security compartments, control access to and from the objects of the system. If object A has a higher security category than object B then:

A	can read	B
A	cannot write	B
B	cannot read	A
B	can write	A

SEE

compart_set(I), integrity_cat_type(I)

NAME

seid - secure entity identifier

SYNOPSIS

```
nsp_type = char;

seid = RECORD

        nsp     : nsp_type;
        uniq_id0: char;
        uniq_id1: cardinal
END;
```

DESCRIPTION

A seid uniquely identifies each ksos object. A seid has two fields, the name space partition (nsp) and a 24 bit field. Each kernel object class is given its own nsp. The 24 bit field is used to identity the individual instances of the kernel object. (This field is subdivided into uniq ido and unil idi pieces.)

The following table contains the valid name space partitions.

null_nsp	0	null
extent_nsp	1	extents
terminal_nsp	2	terminal
device_nsp	3	devices
process_nsp	4	processes
segment_nsp	5	segments
subtype_nsp	6	subtypes
kernel_nsp	7	kernel
same_nsp	128	reserved for directories
root_nsp	129	ROOT file system
low_file_nsp	129	lower limit on file system nsp
high_file_nsp	255	upper limit on file system nsp

The root file system has the distinguished value of 129. All other file systems have nsp values from 130 to 255, which are returned by the kernel when the file system is mounted.

The description given is sufficient for all users except system programmers. The 24 bit quantity has many different fields discriminated by the nsp.

Null Seid

The null seid is a distinguished value. Uniq id0 and uniq id1 have values of zero. In general the null seid acts as a place holder. The null seid is always translated by the kernel interface to be the seid of the calling process.

Process Seid

Each process has a seid which is valid for the life of the process. Uniq id0 contains the process table index. Uniq id1 contains a random number to make the process seid unique over some period of time.

Segment Seid

Each segment has a seid which is valid for the life of the segment. Uniq id0 contains the global segment table index, while uniq id1 contains a random number to make the segment seid unique over some period of time.

Kernel Seid

The kernel seid is a distinguished value used to identify messages sent by the kernel. Both uniq id0 and uniq id1 contain zero values.

Device Seid

Each device has a seid which is valid for the life of KSOS. Changing these seids will cause compatibility problems between KSOS systems. Uniq id0 contains the device class. The device class identifies the type of device being used. The following device class assignments have been made.

disks	1-9
tapes	11-19
asynchronous communication devices	21-29
synchronous communication devices	31-39
network communications devices	41-49
paper tape devices	51-59
card readers	61-69
printers	71-79

The following device class numbers have been assigned to the following devices.

disks	RK05	1	RWP04	2
	RWP05	3	PWP06	4
	RWS04	5		
tapes	TWE16	11	TM11	12
	TU56	13		
network	IMP11B	41	LHDH	42
paper tape	PR11	51	PC11	52
printers	LP11	71		

Uniq id1 distinguishes multiple devices of the same type. The first device has a value of zero and all the rest are numbered sequentially. This field can not be greater than 256.

Extent Seid

All block addressable devices contain extents. Each extent has its own seid. For each device, the first four extents are predefined across all KSOS systems. Uniq id0 is the same as uniq.id0 on the device seids for the device upon which the extent rides. Uniq id1 however, has the low order byte containing Uniq id1 of the device seid. The high order byte contains the extent number.

Subtype Seid

The subtype seids are valid across all KSOS systems. Subtype seids are always predefined. The following subtype seids have been defined.

seid	uniq_id0	uniq_id1
UNIX directory	100	0
KSOS file system	100	1
Network Files	100	2

Terminal Seid

Each terminal has a seid which is valid across all KSOS systems. Two types of terminal devices are supported, the DL and DH. DL devices have uniq id0 values of 0 through 49. The value of 0 is reserved for the console device. DH devices have their uniq id0 starting with 50. Uniq id1 is always the terminal path number. The path number is assigned each time a user changes to a new level by the NKSR. The path numbers 0 and 1 are reserved for the secure server and the secure services respectively.

File Seid

A file has a seid which is valid for the life of the file. The nsp indicates which file system the file is on. The uniq id0 and uniq id1 fields contain the file's 24 bit Jnode number.

NAME

seg_stat_block - segment status block

SYNOPSIS

```
seg_stat_block = RECORD

        mem_lock     : boolean;
        mem_advise   : boolean;
        swap_lock    : boolean;
        sharable     : boolean;
        growth       : direction;
        stat_size    : seg_size;
        stat_seid    : seid;
        stat_des     : seg_des;
        stat_mapped  : boolean;
        stat_acc     : acc_mode;
        stat_loc     : virt_loc;
END;
```

DESCRIPTION

Fields marked with an asterisk (*) are read only. Fields marked with a double asterisk (**) are read only while mapped in.

mem_lock — is true if a segment is locked into memory (may not be swapped out). The privilege privLockSeg is required to lock a segment into memory.

mem_advise — is true if the segment is liable to be locked down in memory. I/O segments should be given this attribute by user programs.

swap_lock — is true if the segment is locked into swap space (sticky: must not be deleted). The privilege privStickySeg is required to lock a segment into swap space.

sharable — is true if the segment may be shared with other processes. A segment's discretionary access controls whether it is sharable. A sharable segment can not have its global attributes changed. For example, once a segment is shared, it can not be made unsharable. However, it can still be mapped anywhere in a process's address space.

growth — indicates the growth direction of the segment. A downward-growing segment starts at a high memory address and ends at a low memory address. Its virtual address must be the high byte address (i.e. the address is odd). An upward growing segment starts at a low memory address and ends at high memory address. Its virtual address must be the low byte address(i.e. the address is even).

*stat_size is the length of the segment in bytes. This must be a multiple of 512.

*stat_seid is the segment's seid.

*stat_des is the segment's open descriptor (process-local segment number).

*stat_mapped is true if the segment is currently mapped into the process's address space.

**stat_acc is the access allowed the calling process to a segment. The access allowed must always be a subset of the access allowed this process by the TII of the segment.

**stat_loc is the last known virtual location (in the process's address space) of a segment. If the segment is upward-growing, this address must be a multiple of 64; if downward-growing, this address must be one less than a multiple of 64.

SEE ALSO

K_build_segment (II), K_get_segment_status (II), K_set_segment_status (II), priv_struct (I), tii_struct (I)

NAME

tii_struct - type independent information

SYNOPSIS

```
priv_struct = bits32;

compart_set = bits32;

access_level_type = RECORD
      security_category       : security_cat_type;
      integrity_category      : integrity_cat_type;
      security_compart        : compart_set
    END;

tii_struct = RECORD

      access_level    : access_level_type;
      owner           : cardinal;
      group           : cardinal;
      da              : discr_access;
      tii_priv        : priv_struct
END;
```

DESCRIPTION

access_level contains the security and integrity levels, and the security compartment of the object.

owner contains the (effective) user identifier for the object.

group contains the (effective) group identifier for the object.

da contains the discretionary access bits for the object.

tii_priv contains the privileges belonging to the object.

SEE ALSO

K_get_object_level (II), K_set_object_level (II), priv_struct (I), compart_set (I), discr_access (I)

NAME

virt_loc - virtual location

SYNOPSIS

```
domain_type = (         (* domains *)
                        null_domain,
                        kernel_domain,
                        supervisor_domain,
                        user_domain
              );

mem_type    = (         (* memory space divisions *)
                        null_space,
                        d_space,
                        i_space
              );

virt_loc    = RECORD    (* virtual location *)
                        domain  : domain_type;
                        space   : mem_type;
                        address : cardinal;
              END;
```

DESCRIPTION

A virtual location specifies the domain, space, and address of where the location exists in the user process.

domain — contains either the user domain or the supervisor domain. The null domain is a distinguished value which the kernel translates to the domain from which a kernel call has been made.

space — contains which space within the domain the data buffer is located. The null space is a distinguished value which is converted to data space on separate I/D space programs, and otherwise to instruction space.

address — contains the address of the location involved.

SEE ALSO

K_remap (II), K_rendezvous_segment (II)

NAME

C-Kernel_Interface - C Interface to the Kernel Calls

DESCRIPTION

The C-Kernel interface consists of 38 C routines. One interface routine exists per Kernel call. The interface routines are used in conjunction with the types defined in KERcalls.h

The general procedure followed in the interface routines is to put the arguments of the interface routine into one contiguous block of memory. Then through an assembly language subroutine, a pointer to the block with the arguments is passed to the appropriate kernel call. Upon completion, the kernel call places any return values into the argument block and returns an exception. This exception is then passed upward by the assembly language subroutine. The interface routine unpacks the return values from the argument block and returns the exception to the calling C program.

The 38 C interface routines are contained in 38 seperate files, named K00.c through K37.c. The assembly language subroutines are contained on files named EMT00.s through EMT37.s. The C routines are archived into one file (EKL.a) and the assembly subroutines into another (EMT.a).

To use the C-Kernel interface routines put the archive file names on your compile command line. An example of compiling a program, "test", that uses the interface routines is:

```
cc test EKL.a EMT.a
```

FILES

K00.c - K37.c, EMT00.s - EMT37.s, EKL.a, EMT.a

SEE ALSO

KERcalls.h

NAME

K_boot

MODULA SYNOPSIS

```
CONST b_seg: seg_des;
VAR stat: K_err_num;
. . .
stat := NK_boot(b_seg);
```

C SYNOPSIS

```
seg_des b_seg;
int stat;
. . .
stat = K_boot(b_seg);
```

DESCRIPTION

K_boot releases (as with K_release_segment) the segment indicated by the argument and maps in (as with K_remap) all other segments attached to the process. Execution of the process continues at location 0 of the process's supervisor I-space.

SEE ALSO

K_release_segment(II), K_remap(II)

NAME

K_build_segment

MODULA SYNOPSIS

```
CONST status: seg stat block;
CONST da: discr access;
VAR   segSeid: seid;
VAR   segDes: seg des;
VAR   stat: K err num;
. . .
stat := NK build segment(status, da, segSeid, segDes);
```

C SYNOPSIS

```
seg stat block status;
discr access da;
seid segSeid;
seg des segDes;
int stat;
. . .
stat = K build segment (&status, da, &segSeid, &segDes);
```

DESCRIPTION

K build segment creates a new process segment and maps it into the process's virtual address space. Its operation includes the following actions:

* A global segment descripter is allocated and swap space is allocated.
* A process-local segment descripter is allocated and associated with the created segment.
* Page registers for the process are chosen according to the address range specified for the segment, and are partially initialized.
* Physical memory is allocated for the segment and the physical memory address of the segment is set up in its page registers. The segment's memory is guaranteed to be all zeros.

Information needed to build the segment is taken from the parameters. From status (see seg_stat_block(I)) are taken:

mem_lock	whether to lock the segment in memory; the privilege privLockSeg is required for this action
mem_advise	whether the segment may be locked in memory, particularly by I/O operations
swap_lock	whether to lock the segment in swap space: the privilege privStickySeg is required for this action
sharable	whether to allow sharing of the segment with other processes
growth	segment's direction of growth
stat_size	size of the segment in bytes
stat_acc	access the calling process is to have to the segment
stat_loc	virtual location of the segment in the process's address space

The da parameter supplies the discretionary access for the segment, to be placed in its type-independent information (see tii_struct(I)). The ac-

cess to the segment allowed by the stat acc field of the status parameter must be a subset of the access allowed by the owner part of the da parameter. Write-only access (write without read) may not be specified in either the stat acc field or the da parameter. The remaining fields of the type-independent information for the segment are taken from those of the calling process.

Values produced by the call are returned in segSeid (the SEID created for the segment); and segDes (the process-local segment number).

Segments may be shared between processes providing that the security, integrity and discretionary access checks would allow such sharing. Sharing of segments requires that the first process to desire to share the segment create it. Subsequent to creation, the segment SEID must be made available to processes wishing to share the segment, typically either via placing it in a mutually agreed upon file, or by passing it via an IPC message. The other processes would then issue K rendezvous segment calls (below) to gain access to the segment.

It is the responsibility of the processes sharing the segment to see to it that it is properly initialized, either by the Kernel's guarantee of all zeros, or by explicit initialization. The initialization may require cooperation or mutual exclusion to be completed successfully. This is particularly true in the case of shared pure text segments which are to be resident in the Instruction Space on the PDP-11/70. Since such segments cannot be written into, they must be created as writable segments, initialized from the appropriate image file, and then have their status changed to make them reside in the correct space and be non-writable. Care should be taken in the design of programs like the Process Bootstrapper which perform such initializations to assure that duplicate initialization attempts or multiple copies of the same shared text segment do not occur.

The virtual address and domain parameters shall be for the calling process only. Other processes sharing the segment may map it into their virtual address spaces as desired through their use of the K rendezvous segment calls. Of course, some segments, e.g. text segments, may operate correctly only if mapped starting at a specific virtual address.

EXCEPTIONS

XbSgRng	Bad segment range: set of addresses specified for the segment would lie outside a 64K address space.
XbSgSz	Bad segment size: segment size of zero specified
XnPgSg	No page in segment: segment address range does not cross or is not adjacent to some multiple of 8K (address range must include an 8K multiple or have top address that one less than an 8K multiple).
XnPvLkSg	No privilege to lock segment: mem_lock specified, but without necessary privilege
XnPvStkSg	No privilege to make sticky segment: swap_lock specified, but without necessary privilege
XncnDo	Cannot do: global resource exhaustion

XpostEh	Process open segment table exhaustion: too many process segments
XvrtMmCfl	Virtual memory conflict: some subset of the address range of this segment would use a page register(s) already in use by an existing segment (high-order three bits the same).
XwOnlSg	Write-only segment: attempt to create write-only segment

NAME

K_close - close a file

MODULA SYNOPSIS

```
CONST ed: openDescriptor;
VAR   status: K_err_num;
...
status := NK_close(ed);
```

C SYNOPSIS

```
char ed;
int status;
...
status = K_close(ed);
```

DESCRIPTION

K_close closes the file, terminal, extent, or device identified by the given open descriptor (previously returned from a K_create or a K_open). Upon process exit, close of all open files, extents, terminals, and devices is performed automatically.

K_close does not cause any device action, such as tape rewinding.

SEE ALSO

K_create(II), K_open(II)

NAME

K_create

MODULA SYNOPSIS

```
CONST protoSeid: seid;
CONST om: openModes;
CONST da: discr access;
CONST stCap: openDescripter;
VAR fSeid: seid;
VAR od: openDescripter;
VAR stat: K err num;
. . .
stat := NK create(protoSeid, om, da, stCap, fSeid, od);
```

C SYNOPSIS

```
seid protoSeid;
open mode om;
discr access da;
openDescriptor stCap;
seid fSeid;
openDescriptor od;
int stat;
. . .
stat = K create (protoSeid, om, da, stCap, &fSeid, &od);
```

DESCRIPTION

K_create creates and opens a new file on the same file system as the file protoSeid. protoSeid need not refer to an actual file; only the name space partition part of the seid is examined to determine which file system is to be used.

The file is created with size zero, the subtype specified by stCap, and is opened with the specified open modes, which must be valid open modes as defined for K_open(II). The file is created with a link count of zero. This implies that the file will be deleted when closed by the creating process unless the link count of the file is first incremented by K_link(II). Use of K_link(II) is normally restricted to the UNIX directory manager, which thus restricts the creation of permanent files to those under the control of the directory manager.

The seid of the new file, fSeid, is chosen pseudo-randomly by the system. The concept of re-creating an existing file is thus not meaningful.

The open descriptor od is a local name which the process can subsequently use to access the created file.

SEE ALSO

K_open(II), K_close(II)

NAME

K_device_function - perform arbitrary I/O function

MODULA SYNOPSIS

```
CONST od: openDescriptor;
CONST funct: ioFunction;
CONST blockNo: fBlockNumber;
CONST inP: p_block;
CONST outP: p_block;
CONST id: asyncId;
VAR errs: ioStatus;
VAR bytesDone: cardinal;
VAR status: K_err_num;
...
status := NK_device_function(od, funct, blockNo, inP, outP, id, errs, bytesDone);
```

C SYNOPSIS

```
openDescriptor od;
int funct;
long blockNo;
pblock inp;
pblock outp;
int id;
ioStatus errs;
int bytes;
int status;
...
status = K_device_function(od, funct, blockNo, &inp, &outp, id, &errs, &bytes);
```

DESCRIPTION

K_device_function can be used for performing all I/O functions. There are a number of I/O functions, some of which have meaning only for certain devices. The parameter blocks inP and outP define the memory areas to be used for input (into memory) and output (from memory). Either or both parameter blocks may be null (size = 0) depending on the requirements of the I/O function.

SPECIFIC FUNCTIONS

Functions are specified in the funct argument.

READ	Read. (same as K_read_block(II)) Valid inP and open for read required.
WRITE	Write. (same as K_write_block(II)) Valid outP and open for write required.
SETEOMCHARS	Sets the set of characters considered to end a single input request for a terminal. See KSOS Input/Output Guide.
REWIND	Rewind (tape only). Open for write required.
UNLOAD	Unload removable medium. Rewinds tapes to off-line condition, stops disk drives where hardware permits.

Open for write required.

WRITEMARK — Write tape mark. Open for write required.

SETDENSITY — Set tape writing density. Open for write required. The density (800 or 1600) is placed in the blockNo argument.

SETTERMMODES1 — Set raw mode, echo, etc. for terminals - see KSOS Input/Output Guide. Open for read and write required.

SETTERMMODES2 — Set parity, density, etc. See KSOS Input/Output Guide. Open for read and write required, and the caller must have the privilege to set these security-related modes. Open for read and write required. [requires privSetComm]

GETTERMMODES — Get terminal modes - see KSOS Input/Output Guide. Open for read and valid inP required.

SETFILESIZE — Set byte size of file. The size is submitted in blockNo, and must be compatible with the current high written block of the file. (i.e. (bytes + 511) DIV 512 = blocks) This function does not allocate or release file space. Open for write required.

ERASEFILE — Get rid of all space in a file. Open for write required.

VOLUMEVALID — Mark removable medium as usable. Open for read and write required. [Requires privImmigrate]

VOLUMEINVALID — Mark removable medium as unusable. Where hardware permits, performs an UNLOAD. Open for read and write required.

INPUTWAIT — This function is used to wait until input is available from a terminal. It is only valid as an asynchroneous request, and only for a terminal. When an INPUTWAIT request completes, a following K read block request for input from the terminal will return immediately with input data. Thus, the effect of asynchroneous terminal reads (which are not supported) is available by performing an INPUTWAIT to wait until input is available, and then when the asynchronous completion message is received, performing a K read block request. Open for read is required.

SETPATH — Connect physical terminal to a different logical terminal path. The desired path number is placed in the blockNo argument. Open for read and write required. [Requires privSetPath]

ASYNCHRONOUS REQUESTS

I/O and computation (and to a limited extent I/O and I/O) may be overlapped within a single process. Normal calls to K device function, K read block , and K write block should have a zero in the id argument. Such requests wait until the I/O operation is complete. If anything else is placed in the id argument, the request is asynchroneous and the call may return before the operation is complete. When the operation completes, an I/O completion message (see the ioCompletionMessagetype) will be sent to the process which did the K device function (or K read block or K_write_block) call.

Only when status is XOK and errs.devIndep is Xasync should an I/O completion message be expected. On asynchronous requests, some errors are reported through errs.devIndep in the call and some are reported through the errs.devIndep field in the I/O completion message. It is generally possible to have several requests in progress for different devices from the same process, but an asynchronous request for a busy device will be delayed until the device is available.

SEE ALSO

K_read_block(II), K_write_block(II)

NAME

K_fork

MODULA SYNOPSIS

```
VAR own: seid;
VAR other: seid;
VAR stat: K_err_num;
. . .
stat := NK_fork(own, other);
```

C SYNOPSIS

```
seid own;
seid other:
int stat;
. . .
stat = K_fork (&own, &other);
```

DESCRIPTION

The K_fork primitive creates a process. The new process is remembered as a child of the caller (parent). The child is an exact duplicate of the parent. The process id of the parent is returned to the child. The process id of the child is returned to the parent. The program counter of the parent is not adjusted as it is in standard UNIX. The returned process id will be sufficient to distinguish parent and child. The type independent information of the child process is identical to the type independent information of the parent process.

The non-sharable segments of the process are copied into new segment instances for the child. The reference counts of sharable segments are incremented. The process local segment names are the same in both the parent and child, although in the case of non-sharable segments they refer to a different segment instance (and therefore, a different segment SEID) for the child than for the parent.

The child inherits the open objects of the parent. That is, each object that is open in the parent is opened in the child, and has the same local open object descriptor. The open counts of the open objects so inherited are incremented to reflect the fact that another process (the child) has them open. If the parent has an object open for exclusive use, the K_fork call fails, preventing two processes from having simultaneous access to the same exclusive use object.

An error is returned in stat if the pool of available processes has been exhausted.

NAME

K_get_file_status

MODULA SYNOPSIS

```
CONST fSeid: seid;
CONST stCap: openDescriptor;
VAR status: file stat block;
VAR stat: K err num;
. . .
stat := NK get file status(fSeid, stCap, status);
```

C SYNOPSIS

```
seid fSeid;
openDescriptor stCap;
file stat block status;
int stat;
. . .
stat = K get file status (fSeid, stCap, &status);
```

DESCRIPTION

K get file status returns the type dependent information associated with a file, terminal, extent, or device. This information includes the size of the object in bytes (zero for non-addressable devices), and the time of the last open for writing (meaningful for files only). The invoking process must have read access to the file with respect to the mandatory security and integrity rules only. No discretionary access checking is performed.

The stCap arguement to this call is ignored.

NAME

K_get_object_level

MODULA SYNOPSIS

```
CONST objSeid: seid;
VAR level: tii struct;
VAR stat: K err num;
. . .
stat := NK get object level(objSeid, level);
```

C SYNOPSIS

```
seid objSeid;
tii struct level;
int stat;
. . .
stat = K get object level (objSeid, &level);
```

DESCRIPTION

Given objSeid, the primitive K get object level returns the type independent information for an object in level.

NAME

K_get_process_status

MODULA SYNOPSIS

```
CONST procSeid: seid;
VAR status: proc stat block;
VAR stat: K err num;
. . .
stat := NK get process status(procSeid, status);
```

C SYNOPSIS

```
seid procSeid;;
proc stat block status;
int stat:
. . .
stat = K get process status (procSeid, &status);
```

DESCRIPTION

The K get process status call returns, in the status parameter, the type dependent information about the process specified by procSeid. A process may successfully request information of processes that it can access given its level and the level of the target process.

NAME

K_get_segment_status

MODULA SYNOPSIS

```
CONST segSeid: seid;
CONST segDES: seg_des;
VAR status: seg_stat_block;
VAR stat: K_err_num;
. . .
stat := NK_get_segment_status(segSeid, status);
```

C SYNOPSIS

```
seid segSeid;
seg_des segDes;
seg_stat_block status;
int stat;
. . .
stat = K_get_segment_status (segSeid, &status);
```

DESCRIPTION

K_get_segment_status returns, in the parameter status, the type dependent information associated with a segment segSeid. A process may successfully obtain type dependent status about any segment from which information could flow to the process. No discretionary access checking shall be performed.

EXCEPTIONS

XbSgDes	Bad segment designator: This segment designator is inactive.
XbSgSd	Bad segment seid: segSeid is not the seid of an existing segment or process does not have manditory (security) access to the segment.
XnSgDes	Not a segment designator: This number is outside the set of segment designators.

NAME

K_halt

MODULA SYNOPSIS

```
VAR stat: K_err_num;
. . .
stat := NK_halt;
```

C SYNOPSIS

```
int stat;
. . .
stat = K_halt ( );
```

DESCRIPTION

K_halt causes the entire system to halt. Use of this call is restricted to processes with the privHalt privilege.

SEE ALSO

priv_struct(I)

NAME

K_interrupt_return

MODULA SYNOPSIS

```
VAR stat: K_err_num;
. . .
stat := NK_interrupt_return;
```

C SYNOPSIS

```
int stat;
. . .
stat = K_interrupt_return ( );
```

DESCRIPTION

K_interrupt_return provides an atomic return operation from pseudo interrupts. It can be thought of as being analogous to the PDP-11 RTI and RTT instructions. When a pseudo interrupt occurs, the program counter, processor status word, and current pseudo interrupt level are saved in a pseudo interrupt vector for the particular type of pseudo interrupt which occurred. In the PDP-11 implementation, these vectors are located at fixed locations in the supervisor domain. The K_interrupt_return call restores the process state from these saved values. Because the interrupted process state is accessible to the process, the K_interrupt_return call checks the saved state prior to restoring it. The process is not be permitted to increase its privileges or accessible domains. (Similar checking takes place in the processing of the pseudo interrupt itself.)

NAME

K_invoke

MODULA SYNOPSIS

```
CONST immSeid: seid;
CONST arg: seg des:
VAR stat: K err num;
. . .
stat := NK invoke(immSeid, arg);
```

C SYNOPSIS

```
seid immSeid;
seg des arg;
int stat;
. . .
stat = K invoke (immSeid, arg);
```

DESCRIPTION

The purpose of the K invoke primitive is the invocation of potentially privileged software. The effect of this call is to replace the existing segment map (including the executing text segment) by a new process image. All segments will be released except for the argument segment specified by arg. The new process image has only the intermediary segment and the argument segment active (mapped in). Arguments for use by the intermediary process may be placed in the argument segment. The exact format of the argument segment is determined by the particular intermediary specified in the call. The argument segment may be used by the intermediary as a scratch pad as the intermediary builds any other segments it requires. It is the responsibility of the newly executing program (the intermediary) to create its own working segments. The privileges of the process are set to those associated with the intermediary segment. In the PDP-11/70 implementation, the intermediary is mapped into location 0 of the supervisor domain instruction space, and the argument segment is mapped out. The intermediary may perform any arbitrary function. Thus, applications of the KSOS Kernel may elect to create specialized intermediaries to perform specific functions. The only predefined intermediary is the Process Bootstrapper, described next.

The Process Bootstrapper segments implement a trusted process whose sole purpose is the creation of other, potentially trusted, environments by replacing itself with image from the prototype file whose name is passed in as an argument. The Process Bootstrapper has the following privileges:

to set privileges

to set the effective owner

to set the security and integrity level

to realize execute permissions (i.e. use the execute permissions for read access attempts)

Using the parameters specified in the argument segment, the Process Bootstrapper builds a new set of process segments conforming to the process prototype file. The privileges for the new environment is obtained from the process prototype file. If the prototype file has no privileges associated with it, the new environment is unprivileged. If the prototype file specifies that it is to execute in a different discretionary access domain, the bootstrap changes the effective user and/or group of the process to the owner of the prototype file. The new trusted process is then set into execution by the Process Bootstrapper. Note that a completely trusted path exists from the K_invoke call, through process construction, to the execution of the trusted software.

The use of the K_invoke call is not limited to the invocation of trusted processes. Untrusted processes may also be executed through the K_invoke call. If change of discretionary access domain or privilege is not required by the type dependent information associated with the process prototype file, the process bootstrap simply removes all privileges prior to setting the new image into execution.

The privilege information associated with a process prototype file is controlled by the Privilege Control Process, a restricted use program discussed in the Non-Kernel Security-Related Software CPCI Specification [NKSR 78].

SEE ALSO

PBB(III), K_spawn(II)

NAME

K_link

MODULA SYNOPSIS

```
CONST fSeid: seid;
VAR stat: K_err_num;
. . .
stat := NK_link(fSeid);
```

C SYNOPSIS

```
seid fSeid;
int stat;
. . .
stat = K_link (fSeid);
```

DESCRIPTION

K_link increments the reference count of a Kernel file specified by the seid fSeid. The reference count is normally used to indicate the number of UNIX directory entries which point to this SEID. Applications of the KSOS Kernel which do not use the UNIX directory structure and semantics may use the reference count for other purposes. The reference count may only be incremented by processes with the privLink privilege. Such processes should be carefully designed to reduce the bandwidth of the resultant confinement channels and to preserve the integrity of the higher level directory structure, if any. The security and integrity checking on K_link is as if the user were reading and writing the file. No discretionary access checking is performed. Thus, the processes privileged to use K_link may implement whatever discretionary checking they choose.

SEE ALSO

K_create(II), K_unlink(II).

NAME

K_mount

MODULA SYNOPSIS

```
CONST extSeid: seid;
CONST readOnly: boolean;
VAR nsp: nsp type;
VAR stat: K err num;
. . .
stat := NK mount(extSeid, readOnly, nsp);
```

C SYNOPSIS

```
seid extSeid;
nsp type nsp;
boolean readOnly;
int stat;
. . .
stat = K mount (extSeid, readOnly, &nsp);
```

DESCRIPTION

The K mount call performs the function of associating a particular file name space partition with the extent, extSeid, making it possible to access files in the mounted file system.

Use of this call requires the privilege privMount which normally restricts the use of this call to the NKSR 'mount' program. It is the responsibiity of the privileged program to insure that the file system being mounted is a valid file system, that the Immigration Officer function has approved its use, and that the user invoking 'mount' is authorized to operate on the file system involved.

Each mounted KSOS file system belongs to a different name space partition. The Kernel assures this by assigning a name space partition to the file system when the file system is mounted. The Immigration Officer software [NKSR 78] maintains a data base of file systems currently immigrated. When a extent is mounted, the Kernel shall update an internal data base which tells it on which extent the SEIDs in the mounted name space partition may be found. The nsp value returned by the K mount call determines the name space partition which the Kernel will expect in operations referring to files in the newly mounted file system.

The first K mount after startup of the Kernel will always return the same value, and by convention this value is associated with the "root" file system.

It is possible to mount a file system as read only by setting readOnly to true in which case no file on the file system can be opened for writing, new files cannot be created, and existing files cannot be altered or deleted. The physical device may be placed in write-protect mode without interfering with read-only mounts.

Each file system contains type independent information. The security and integrity levels of the file system shall be interpreted as the maximum levels allowed for any file on the extent. The Kernel prevents K_create operations by any process not permitted data flow to the file system under the security model.

Note that extents may contain data structures other than KSOS file systems. A given extent may be assigned for private, non-file system use. However, only extents belonging to the subtype 'file system' may be mounted.

SEE ALSO

K_create(II)

NAME

K_nap

MODULA SYNOPSIS

```
CONST timeOut: cardinal;
VAR stat: K err num;
. . .
stat := NK nap(timeOut);
```

C SYNOPSIS

```
cardinal timeOut;
int stat;
. . .
stat = K nap (timeOut);
```

DESCRIPTION

K nap is a mechanism for explicitly giving up the processor when a higher level blocking condition occurs. This situation occurs when, for example, processes implementing semaphores on top of the Kernel become logically blocked on a semaphore. K nap provides an alternative to busy waiting for the semaphore. The timeOut argument is the incremental time (in 1/60th second clock ticks) before which the process should not be rescheduled by the Kernel. Processes using K nap should check that the logical condition for which they were waiting has occurred when they are activated.

NAME

K_open - open a file, terminal, extent, or device

MODULA SYNOPSIS

```
CONST fSeid: seid;
CONST om: OpenModes;
CONST stCap: openDescriptor;
VAR od: openDescriptor:
    status: K_err_num;
...
status := NK_open(fSeid, om, stCap, od);
```

C SYNOPSIS

```
seid fSeid;
KopenModes om;
char stCap;
char od;
int status;
...
status = K_open(fSeid, om, stCap, &od);
```

DESCRIPTION

K_open opens the file, terminal, extent, or device specified by fSeid. The open descriptor is returned in the arguement od.

No initialization of the device occurs when a device is opened. Devices which are not ready (no tape mounted, etc.) can be opened, but devices which are not physically present cannot be opened.

om contains the requested open modes, which are

om.read	Open for reading
om.write	Open for writing
om.exclusive_read	Lock out all other readers
om.exclusive_write	Lock out all other writers

Only the following combinations of modes are permitted:

		exclusive		
read	write	read	write	
true	false	false	false	Normal read.
false	true	false	false	Normal write.
true	true	false	false	Normal read and write.
true	true	false	true	Lock out all other writers.
true	true	true	true	Lock out everybody else.

When an open request fails because of an exclusive use blockage, an exception is returned. There is no blocking or delay associated with

exclusive use failure. Note, though, that exclusive use is available only to those with the ability to open for writing.

SUBTYPES

Subtypes are predefined openable objects which control access to other objects. If an object is subtyped, a requestor can open it for writing only if the subtype is already open for writing to that process and the open descriptor of the subtype is submitted in the stCap argument. A similar rule applies for reading. Files may be created in a subtype by providing the subtype, opened for reading and writing, to K_create(II). Subtypes are primarily used by the directory manager to protect directories and are of limited use to most users.

SEE ALSO

K_create(II), K_close(II)

NAME

K_post

MODULA SYNOPSIS

```
CONST receiver: seid;
CONST psInt: boolean;
CONST msg: msg struct;
VAR stat: K err num;
. . .
stat := NK post(receiver, psInt, msg);
```

C SYNOPSIS

```
seid receiver;
boolean psInt;
msg struct msg;
int stat:
. . .
stat = K post (receiver, psInt, &msg);
```

DESCRIPTION

K post sends a short message to another process specified by the seid receiver. A pseudo interrupt is asserted at the destination process, if selected, and if the receiving process has IPC pseudo interrupts enabled (i.e. that its pseudo-interrupt level is sufficiently low to allow pseudo interrupts). A header is attached to the message indicating the SEID of the originating process.

The type independent information for the two processes is used to determine rights of the originating process to communicate with the the destination.

NAME

K_read_block - perform read

MODULA SYNOPSIS

```
CONST od: openDescripter;
CONST blockNo: fBlockNumber;
CONST inP: p_block;
CONST id: asyncId;
VAR errs: ioStatus;
VAR bytesRead: cardinal;
VAR status: K_err_num;
...
status := NK_read_block(od, blockNo, inP, id, errs, bytesRead);
```

C SYNOPSIS

```
openDescripter od;
long blockNo;
pblock inp;
int id;
ioStatus errs;
int bytes;
int status;
...
status = K_read_block(od, blockNo, &inp, id, &errs, &bytes);
```

DESCRIPTION

K_read_block is used to request reading from files, terminals, extents, and devices. The parameter block inP defines the memory area to be used for input.

Files and extents are stored in units of 512 byte blocks. From 1 to 64 blocks can be read from a file or extent with one request. A single big request is much faster than many small requests. Files which contain "holes" (unwritten blocks) are treated on read as if the unwritten block contained all zero bytes. Transfers are always in multiples of 512 bytes, regardless of the byte size of the file.

Terminals may be read and written in sizes from 1 to 128

Devices may have different rules for each device on block size. See the specific device in (VI). KSOS Input/Output Guide

ASYNCHRONOUS REQUESTS

See K_device_function(II). To make a normal, synchronous, request, the id argument should be zero.

SEE ALSO

K_device_function(II), K_write_block(II)

EXCEPTIONS

See K device function(II).

ERROR CODES

Error information is returned in errs. After an operation which did not return an exception, errs.devIndep contains one of the values given below, and, for operations on devices, errs.devDep contains 16 bits of hardware status as described in (VI) under the specific device. See K device function(II) for further details.

NAME

K_receive

MODULA SYNOPSIS

```
CONST timeOut: cardinal;
CONST n pil: pseudo int levels;
VAR msg: ipc block;
VAR stat: K err num;
. . .
stat := NK receive(timeOut, n pil, msg);
```

C SYNOPSIS

```
cardinal timeOut;
pseudo int levels n pil;
ipc block msg;
int stat;
. . .
stat = K receive (timeOut, n pil,  &msg);
```

DESCRIPTION

K receive suspends the execution of a process until the receipt of an IPC message or until a time out. The return value indicates the condition which caused the process to be restarted.

The first message in the queue of received IPC messages is returned. If more than timeOut 'clock ticks' expire before any IPC messages are received, no message is returned and the error code so indicates. The n pil parameter sets the pseudo interrupt level of the process before beginning the wait. This is analogous to a K set pil call.

NAME

K_release_process

MODULA SYNOPSIS

```
CONST procSeid: seid;
VAR stat: K_err_num;
. . .
stat := NK_release_process(procSeid);
```

C SYNOPSIS

```
seid procSeid;
int stat;
. . .
stat = K_release_process (procSeid);
```

DESCRIPTION

K_release_process deallocates all of the Kernel level resources associated with the named process. A K_release_process call with the null_seid argument releases the calling process. Only a process with the same owner or a process privileged to change its owner may issue a K_release_process for another process. The effects of K_close for all open files and of a K_release_segment for all the segments of the process occur. Shared segments remain intact unless the reference count to the segment has reached zero. Segments with a zero reference count are deallocated unless they have been created to be 'sticky'. Files are deallocated if their link counts and open counts are zero. The process seid becomes unknown.

NAME

K_release_segment

MODULA SYNOPSIS

```
CONST seg: seg_des;
VAR stat: K_err_num;
. . .
stat := NK_release_segment(seg);
```

C SYNOPSIS

```
seg_des seg;
int stat;
. . .
stat = K_release_segment (seg);
```

DESCRIPTION

The primitive K_release_segment releases the Kernel level resources associated with the specified segment. The segment is not deleted if other processes are still using it or if its swap_lock (sticky) bit is set.

SEE ALSO

seg_stat_block (I)

EXCEPTIONS

XbSgDes	Bad segment seid: segSeid is not the seid of an existing segment or process does not have manditory (security) access to the segment.
XnSgDes	Not a segment designator: This number is outside the set of segment designators.

NAME

K_remap

MODULA SYNOPSIS

```
CONST inSeg: seg des;
CONST inLoc: virt loc;
CONST inAcc: acc mode;
CONST outSeg: seg des;
CONST outSize: seg size;
CONST choice: selector;
VAR stat: K err num;
. . .
stat := NK remap(inSeg, inLoc, inAcc, outSeg, outSize, choice):
```

C SYNOPSIS

```
seg des inSeg;
virt loc inLoc;
acc mode inAcc;
seg des outSeg;
seg size outSize;
selector choice;
int stat;
. . .
stat = K remap (inSeg, inLoc, inAcc, outSeg, outSize, choice);
```

DESCRIPTION

The K remap primitive permits the process to change its segment map. The outgoing segment is no longer directly addressable by the process through machine instructions. The incoming segment becomes directly addressable by the process. The outgoing segment is not released. However, the memory management hardware of the segment to be removed from the current mapped set may be used to satisfy the hardware requirements of the incoming segment. When a process segment is mapped into the current addressable set of segments, it occuppies the virtual address vector defined by the arguments to K_remap. Either or both of the segment designators may be null. If both are null the call has no effects. The incoming segment must fit into the virtual memory and memory management resources available after the outbound segment is unmapped. If it does not, or if any of the other error conditions occur, or if both segment designators are null, the call has no effect on the segment mapping.

If the alter virtual location flag (vlFlg) within the choice parameter is TRUE, the incoming segment is mapped into the location specified as arguments to the call, and its status information adjusted to reflect this as a permanent change. Otherwise, the segment is mapped into the location specified in its permanent status information.

If the alter discretionary access information flag (daFlg) within the choice parameter is TRUE, the modes in which this process will access the segment are checked against the permitted access modes for the segment, and if allowed, will become the access modes for the segment. This may

alter the settings of memory management hardware when the segment is mapped back in.

If the alter size flag (osFlg) within the parameter choice is TRUE, the size of the outbound segment are set to the value outsize. The expansion or truncation of the segment is performed at the end of segment specified by the growth attribute of the segment specified when built. Expanded parts of segments are filled with zeros. The size change can only be applied to segments that are not sharable.

EXCEPTIONS

XbSgRng	Bad segment range: set of addresses specified for the segment would lie outside a 64 K address space.
XbSgDes	Bad segment designator: this segment designator is inactive.
XinSgAldMp	Incoming segment already mapped.
XncnDo	Cannot do: global resource exhaustion.
XnoAcc	No access: cannot access this object.
XnPgSg	No page in segment: segment address range does not cross or is not adjacent to some multiple of 8 K (address range must include an 8K multiple or have top address that one less than an 8K multiple).
XoutSgAldUmp	Outgoing segment already unmapped.
XvrtMmCfl	Virtual memory conflict: some subset of the address range of this segment would use a page register(s) already in use by an existing segment (high-order three bits the same).

NAME

K_rendezvous_segment

MODULA SYNOPSIS

```
CONST segSeid: seid;
CONST location: virt loc:
CONST access: acc mode;
VAR segDes: seg des;
VAR stat: K err num;
. . .
stat := NK rendezvous segment(segSeid, location, access, segDes);
```

C SYNOPSIS

```
seid segSeid;
virt loc location;
acc mode access;
seg des segDes;
int stat;
. . .
stat = K rendezvous segment (segSeid, location, access, &segDes);
```

DESCRIPTION

The Kernel call K rendezvous segment is the mechanism by which processes are able to share segments. If the segment requested exists and is accessible, it is mapped into the processes address space as requested, *providing that the requested* mapping information is valid. The Kernel will check that the segment may be mapped into the process issuing the K rendezvous segment call. The checks include:

that the segment seid is active

that the segment may be shared

that the security/integrity level of the process allows it to access the segment

that the discretionary access for the segment allows it to be accessed in the requested way

that the virtual address supplied is valid

EXCEPTIONS

XbSgRng	Bad segment range: set of addresses specified for the segment would lie outside a 64 K address space.
XbSgSd	Bad segment seid: segSeid is not the seid of an existing segment or process does not have manditory (security) access to the segment.

XdupSg	Duplicate segment: some process-local segment designator is already attached to the segment.
XncnDo	Cannot do: global resource exhaustion.
XnPgSg	No page in segment: segment address range does not cross or is not adjacent to some multiple of 8 K (address range must include an 8K multiple or have top address that one less than an 8K multiple).
XpostEh	Process open segment table exhaustion: too many process segments.
XsgNoAcc	Segment no access: discretionary access of the segment does not allow the requested access.
XvrtMmCfl	Virtual memory conflict: some subset of the address range of this segment would use a page register(s) already in use by an existing segment (high-order three bits the same).

NAME

K_secure_terminal_lock

MODULA SYNOPSIS

```
CONST tSeid: seid;
VAR stat: K err num;
. . .
stat := NK secure terminal lock(tSeid);
```

C SYNOPSIS

```
seid tSeid;
int stat;
. . .
stat = K secure terminal lock (tSeid);
```

DESCRIPTION

This Kernel call has been deleted.

NAME

K_set_da

MODULA SYNOPSIS

```
CONST objSeid: seid;
CONST da: discr access;
VAR stat: K err num;
. . .
stat := NK set da(objSeid, da);
```

C SYNOPSIS

```
seid objSeid;
discr acces da;
int stat;
. . .
stat = K set da (objSeid, da);
```

DESCRIPTION

K set da sets the discretionary access of the object specified by the first argument to that given in the second argument.

NAME

K_set_file_status

DESCRIPTION

This Kernel call has been deleted.

NAME

K_set_real_id

MODULA SYNOPSIS

```
VAR stat: K_err_num;
. . .
stat := NK_set_real_id;
```

C SYNOPSIS

```
int stat;
. . .
stat = K_set_real_id();
```

DESCRIPTION

K_set_real_id sets the process' effective id to its real id. The effective id is set by doing a K_set_object_level call, while the real id is set by doing a K_set_process_status call.

NAME

K_set_object_level

MODULA SYNOPSIS

```
CONST objSeid: seid;
CONST level: tii struct;
VAR stat: K err num;
. . .
stat := NK set object level(objSeid, level, choice);
```

C SYNOPSIS

```
seid objSeid;
tii struct level;
int stat;
. . .
stat = K set object level (objSeid, &level);
```

DESCRIPTION

The K set object level primitive sets the security relevant type independent information for an object.

Processes with the privilege to set object level shall be capable of changing

- the user which owns the object
- the group which owns the object
- the security level (security category and compartments)
- the integrity level (integrity category and (presently null) compartments.)

NAME

K_set_pil

MODULA SYNOPSIS

```
CONST new pil: pseudo int levels;
VAR old pil: pseudo int levels;
VAR stat: K err num;
. . .
stat := NK set pil(new pil, old pil);
```

C SYNOPSIS

```
pseudo int levels new pil;
pseudo init levels old pil;
int stat;
. . .
stat = K set pil (new pil, &old pil);
```

DESCRIPTION

K set pil sets the process' pseudo interupt level to the first argument. The process' old pseudo interupt level is returned in the old_pil field.

NAME

K_set_priv

MODULA SYNOPSIS

```
CONST objSeid: seid;
CONST priv: priv struct;
VAR stat: K err num;
. . .
stat := NK set priv(objSeid, priv);
```

C SYNOPSIS

```
seid objSeid;
priv struct priv;
int stat;
. . .
stat = K set priv(objSeid, priv);
```

DESCRIPTION

K set priv sets the privileges of the object specified by the first argument to the privileges specified by the second argument.

NAME

K_set_process_status

MODULA SYNOPSIS

```
CONST procSeid: seid;
CONST status: proc stat block;
CONST choice: selector;
VAR stat: K err num;
. . .
stat := NK set process status(procSeid, status, choice);
```

C SYNOPSIS

```
seid procSeid;
proc stat block status;
selector choice;
int stat;
. . .
stat = K set process status (procSeid, &status, choice);
```

DESCRIPTION

The K set process status call permits the process to change those type dependent parameters that are not controlled by other primitives.

The K set process status Kernel call supplies an advisory scheduling priority to the Kernel level scheduler. The Kernel may elect to adjust the advisory priority to guarantee equitable service to all processes.

The notion of real and effective user identification shall be retained at the Kernel level because these identifiers determine the access permissions extended to a process. The effective user and group ID's are part of the type independent information for the process, because they are what determine the discretionary access rights. The real user and group ID's are part of this type dependent information and require the privilege privSetLevel to modify.

The timer toggle and pseudo interrupt level control the pseudo interrupt mechanism. If the timer toggle is TRUE, a pseudo interrupt shall be generated every clock tick (machine dependent time unit). This mechanism may be used for periodic sampling of user mode program counter values for the construction of execution profiles. The pseudo interrupt level is analogous to the hardware interrupt level. Pseudo interrupts shall be transmitted to the process only if the level of the pseudo interrupt is above the level of the process.

NAME

K_set_segment_status

MODULA SYNOPSIS

```
CONST segSeid: seid;
CONST status: seg stat block;
CONST choice: selector;
VAR stat: K err num;
. . .
stat := NK set segment status(segSeid, status, choice);
```

C SYNOPSIS

```
seid segSeid;
seg stat block status;
choice selector;
int stat;
. . .
stat = K set segment status (segSeid, &status, selector);
```

DESCRIPTION

K set segment status supports modification of the type dependent information of a segment. The invoking process shall have appropriate privilege in order to modify the "sticky" flag or the "lock" flag.

NAME

K_signal

MODULA SYNOPSIS

```
CONST procSeid: seid;
CONST sigMsg: msg struct;
VAR stat: K err num;
. . .
stat := NK signal(procSeid, sigMsg);
```

C SYNOPSIS

```
seid procSeid;
msg struct sigMsg;
int stat;
. . .
stat = K signal (procSeid, &sigMsg);
```

DESCRIPTION

The K signal primitive provides a means for privileged processes to transmit a high priority pseudo-interrupt to a process. K signal differs from the K post IPC mechanism in several ways. First, K signal always generates a pseudo interrupt. The pseudo-interrupt level of the K signal pseudo-interrupt is above that of normal IPC. Second, the K signal pseudo interrupt will abort long running Kernel calls (i.e. terminal I/O) which receiving the K post mechanism does not. The intended use of K signal is to provide a mechanism for a privileged process to "get through" to another process, typically to ask it to terminate. The calling process must have the privilege privSignal.

NAME

K_spawn

MODULA SYNOPSIS

```
CONST immSeid: seid;
CONST arg: seg des;
VAR child: seid;
VAR stat: K err num;
. . .
stat := NK spawn(immSeid, arg, child);
```

C SYNOPSIS

```
seid immSeid;
seg des arg;
seid child;
int stat;
. . .
stat = K spawn (immSeid, arg, &child);
```

DESCRIPTION

The Kernel primitive K spawn combines the functions of K fork and K invoke into one operation. The K spawn primitive permits process creation without the cost of copying the parent process image to the child process. The effect of K spawn is to create a new process and to force the effect of a K invoke call upon the newly created process. The parent process may therefore completely specify the contents of the child process image.

The parameters to K spawn are the same as the parameters to the K invoke primitive. These parameters are used to determine the effect of the K invoke call forced upon the child process. (See K invoke above for a discussion of this primitive.) The full semantics of K invoke are implemented. Hence, a child process may acquire more privilege than the parent and may operate in a different discretionary access domain.

SEE ALSO

K_invoke(II)

NAME

K_special_function

DESCRIPTION

This Kernel call has been deleted.

NAME

K_unlink

MODULA SYNOPSIS

```
CONST fSeid: seid;
VAR stat: K err num;
. . .
stat := NK unlink(fSeid);
```

C SYNOPSIS

```
seid fSeid;
int stat;
. . .
stat = K unlink (fSeid);
```

DESCRIPTION

K unlink decrements the file reference count of the specified file. When the file reference count is zero and no process has the file open, the file is deleted. When the count is decremented from one to zero, the file becomes logically nonexistent. If a file is logically nonexistent, but the file has not been deleted because some process still has it open, it cannot be opened again, and the file does not exist for Kernel calls which take file SEIDs as arguments, such as K link(II). When a file is created with K create(II) it has a reference count of zero, but does have logical existence and thus K link(II) can be used to increment its count.

The security and integrity checking are as if the file is being opened for reading and writing, except that no discretionary access checking is done by the Kernel, allowing processes privileged to use this primitive to perform whatever checking they choose to.

The use of K unlink requires the privilege privLink. This privilege is normally restricted to the UNIX directory manager.

NAME

K_unmount

MODULA SYNOPSIS

```
CONST nsp: nsp type;
VAR stat: K err num;
. . .
stat := NK unmount(nsp);
```

C SYNOPSIS

```
nsp type nsp;
int stat;
. . .
stat = K unmount (nsp);
```

DESCRIPTION

The Kernel primitive K unmount logically unmounts the file system specified by the name space partition nsp. The following checks must be satisfied before the Kernel will unmount a file system:

- the process must have the privilege to issue the call
- the device must have file system mounted on it
- the extent must be tranquil (no open files)

After normal completion of the Kernel call, the disk has been returned to the 'unmounted' condition and can be mounted again in the future without performing file recovery.

Should a disk device fail and have to be shut down, it is still possible to perform a K unmount to inform the Kernel that the file system is now unmounted. Although the K unmount will return an I/O error exception (Xerror or Xfault) the Kernel's internal database will still be purged of information about the file system. This allows mounting the disk on another drive and (after file system recovery, if required) remounting the file system.

SEE ALSO

KSOS Operator / Administrator Reference Manual [reference to be supplied]

NAME

K_walk_process_table

MODULA SYNOPSIS

```
CONST index: cardinal;
VAR p_seid: seid;
VAR stat: K_err_num;
. . .
stat := NK_walk_process_table(index, p_seid);
```

C SYNOPSIS

```
cardinal index;
seid p_seid;
int stat;
. . .
stat = K_walk_process_table (index, &p_seid);
```

DESCRIPTION

The K_walk_process_table primitive is a means for privileged software to obtain the SEIDs of active processes. The primitive returns the SEID of the process which occupies slot index of the global process table. This SEID can then be used in K_get_object_level or K_get_process_status calls. The call fails if the process does not possess the privilege to issue it, or if index is not a valid index number for the process table.

NAME

K_write_block - perform write

MODULA SYNOPSIS

```
CONST od: openDescriptor;
CONST blockNo: fBlockNumber;
CONST outP: p_block;
CONST id: asyncId;
VAR errs: ioStatus;
VAR status: K_err_num;
...
status := NK_write_block(od, blockNo, outP, id, errs);
```

C SYNOPSIS

```
openDescriptor od:
long blockNo;
pblock outp;
int id;
ioStatus errs;
int status;
...
status = K_write_block(od, blockNo, &outp, id, &errs);
```

DESCRIPTION

K_write_block is used to request writing to files, terminals, extents, and devices. The parameter block outP defines the memory area to be used for output.

Writing to a file will cause file space to be allocated as required. A write which increases the highest block number of the file sets the byte size of the file to (high block x 512). The user may later indicate, via the SETFILESIZE function of K_device_function(II), that the size of the file in bytes is up to 511 less. This will not prevent the entire last block from being read.

Files may contain "holes", (unwritten blocks) but extremely sparse files are inefficient.

ASYNCHRONOUS REQUESTS

See K_device_function(II). To make a normal, synchronous, request, id should be zero.

SEE ALSO

K_device_function(II), K_read_block(II)

NAME

Modula_Kernel_Interface - Modula Interface to the Kernel Calls

DESCRIPTION

Modula interface routines for the Kernel calls are available in the file NKcalls.mod. One interface routine exists per Kernel call. The interface routines are used in conjunction with the types and low level interface procedures defined in NEWcalls.mod

The general procedure followed in the interface routines is to put the arguments into one contiguous block of memory. Then through an assembly language subroutine, a pointer to the block with the arguments is passed to the appropriate kernel call. Upon completion, the kernel call places any return values into the argument block and returns an exception. This exception is then passed upward by the assembly language subroutine. The interface routine unpacks the return values from the argument block and returns the exception to the calling Modula program.

To use the Modula Kernel interface routines include the file NKcalls.mod at the beginning of your Modula program. The file NEWcalls.mod is explicitly included in NKcalls.mod.

FILES

NKcalls.mod, NEWcalls.mod

NAME

acp_op - operator interface to the audit capture process

SYNOPSIS

acp_op flag [file_name]

DESCRIPTION

Acp_op is an operator interface to the audit capture process. This interface requires one of the following flags:

-c Changes the device to which the audit capture messages are written. If the messages are currently written to a file, the file will be closed and the messages will be diverted to the console. This function enables the operator to close all acp files and, for example, unmount the root file system.

-i Identifies the file in which the acp messages are currently being placed.

-p Prints out the acp file given in the file_name field.

-r Removes the acp file specified in the file_name field.

-s Switches the file in which the acp messages are placed to a new file. The name of the current file is printed out. The file is then closed and a new file is opened. The name of the new acp file is also printed.

DIRECTORIES

/sys/sysAudit

NAME

btcp - boot copy program

SYNOPSIS

btcp packseid [-s] [-0 lev0boot] [-1 lev1boot] [-k kernelimage] [-u initialimage]

DESCRIPTION

Btcp copies files required to boot a KSOS system to their correct place on the specified initialized KSOS pack. Btcp is spawned by the secure server at the request of a user running at OPERATOR or higher security level.

s	copy the system security map to extent 4 of the pack.
0	copy the specified level 0 boot program to extent 1 of the pack.
1	copy the specified level 1 boot program to extent 2 of the pack.
k	copy the specified kernel image to its proper place on the pack - extent 5 beginning at block 0.
u	copy the specified initial process image to its proper place on the pack - extent 5 beginning at block 314.

FILES

/sys/dataBases/security	system security map.

SEE ALSO

exi, pki

NAME

cal - change access level

SYNOPSIS

cal [pathname]

DESCRIPTION

Change access level, cal, allows a user to create an environment at a new security level, or to return to a previously interrupted environment. If the "pathname" argument is given, the level of the environment will be that of the file specified by "pathname"; otherwise cal will prompt for a new access level. If an environment already exists at the requested level, cal will revert to that environment; otherwise a new environment will be created.

Like login, cal should create the new environment by spawning the user/supervisor domain programs given for the user's login id in the user access authentication database. However, as an interim measure, cal will prompt for the pathname of the supervisor domain program to be spawned. This program will normally be a UNIX emulator.

Each user environment corresponds to a different terminal path. There are a fixed number (currently 3) of paths on which user environments can be created. One of these is used by login for the user's initial environment; the remainder are available for allocation by cal.

FILES

/sys/dataBases/user	user access authentication database
/sys/dataBases/group	group access authentication database
/sys/dataBases/terminal	terminal profile database
/sys/dataBases/system	system profile database

SEE ALSO

SSP(III), login(III), user(IV), group(IV), terminal(IV), system(IV)

DIAGNOSTICS

"object not found" "pathname" does not exist
"no free paths"
"you can not change to that level"

NAME

dpe - device profile database editor

SYNOPSIS

dpe

DESCRIPTION

Dpe interactively edits the device profile database and is invoked from the secure server. The editor commands include add, change, delete, find, next, print, view, and quit. A description of command action follows. The character preceding the closing parenthesis, ')', is the command code. Any unrecognized character causes printing of the command list.

a)dd prompts the user for all information and appends the new record to the end of the database. The following questions are asked:

```
Enter name (max 8 char):
Enter desired device name space:
Enter desired device type (high byte):
Enter desired device unit number (low word):
Enter user name of owner (max 8 char):
Enter login name of group (max 8 char):
Does this device allow Valid_required ? (y or n) :
Can user assign with assign function ? (y or n) :
Enter desired discretionary access for owner:
Enter desired discretionary access for group:
Enter desired discretionary access for all:
ENTER ACCESS LEVEL DEFAULT
SECURITY CATEGORY
Enter desired SECURITY category:
INTEGRITY CATEGORY
Enter desired INTEGRITY category:
Enter numbers of desired security compartments
separated by spaces (carriage return for NULL):
```

c)hange changes a specific field in the current record. Change accepts the following commands:

```
ex)it
v)iew
n)ame
s)eid
g)roup id
d)iscret access
a)ccess level
r)valrequr
l)assignment allowed
```

d)elete the current record by asking "Do you want to delete [current record name] (y or n) :"

p)rint the current database records, including modifications, to the lineprinter.

v)iew outputs the current record to the terminal

f)ind searches the database for the specified name and sets the current pointer to the record. The user is prompted for the name; dpe responds "Record is not found", if the name not in the *database*.

n)ext moves the current pointer is to the next record. If the pointer is currently pointing to the last record, it is moved to the first record.

q)uit ends execution of the editor. If modifications were made, the question "Do you want to save the updates?" is asked and a y or n is expected in response.

FILES

/sys/dataBases/device device profile database
/sys/dataBases/security system security map

SEE ALSO

SME(III), UCE(III), GAA(IV), device(IV), security(IV), user(IV), group(IV)

ERRORS

can't open device database
can't create tempfile
can't open user database
can't open group database
can't *open security_map*

NAME

exi - initialize pack extents

SYNOPSIS

exi deviceSEID [-rv]

DESCRIPTION

Exi is a pack initialization tool which enables the administrative user to interactively view, define and modify KSOS pack extent map slots. It is a very powerful tool and should be used carefully. Exi is spawned by the secure server at the request of a user running at OPERATOR or higher security level. It is commonly used to initialize KSOS file systems. The argument deviceSeid specifies the pack which is to be operated on (e.g. d1/0). The -r option indicates a read only mode of operation in which extent map slots may be viewed but not modified. The -v option causes exi to interact with the user in a verbose manner.

Commands which operate on a particular extent map slot may be optionally preceded by the extent number. Specifying extent number 0 allows modification of the pack master mount item. If no extent number is supplied the current extent is assumed. Recognized commands include:

[n]v View the specified extent map slot. A formatted dump is produced.

[n]f Free the extent.

l List extents. The extent label, first block number, last block number and extent size for each extent on the pack are displayed.

[n]a Add extent. The user is prompted for all the information necessary to create a new extent. If desired, exi will prompt the user for information needed to initialize the extent as a KSOS file system.

e Exit exi.

[n]m Modify extent map slot. This command places the user in modification mode. Both the pack master mount item and extent items may be modified. After receiving the prompt, the user types the control character corresponding to the slot field to be modified and exi responds by asking for specific data. To return to normal mode, type an empty line (<CR> only). Control characters for each type of slot are listed below.

extent item:
- v view extent item.
- a modify access rights, security and integrity.
- l modify label field.
- s modify subtype field.

mount item:
- v view mount item slot.
- a modify access, security and integrity information.
- l modify label field.

SEE

pki, mce

BUGS

Currently EXI opens the whole pack unexclusively. It opens the whole pack because EXI contains the functionality to initialize file systems. It does not open the pack for exclusive use because, at least initially, EXI must be able to modify extents residing on a pack with a mounted file system. EXI should be split into two programs - one for extent modification which exclusively accesses the extent map extent, and a separate program for file system initialization.

NAME

fam - file access modifier

SYNOPSIS

fam [-v] [[key argument] ...] filename

DESCRIPTION

Fam allows file access modifications of files owned by the user. The argument -v puts fam into an interactive mode where the user will be prompted for commands, if no other arguments other than the filename are given interactive mode is assumed, also the -v flag must always appear before any key arguments. Key flags must always be followed by an appropriate argument.

The following are descriptions of the keys and their arguments.

-d modify the discretionary access. The argument following this flag must be an octal number.

-g change the group id of the file. The argument following this flag must be a legal group name.

-o change the owner of the file. The argument following this flag must be a legal user name.

-s change the security level of the file. The argument following the flag must be a legal security level name.

-i change the integrity level of the file. The argument following the flag must be a legal integrity level name.

-c delete a compartment from the compartment set. The argument following the flag must be a legal compartment name. mce.

+c add a compartment to the compartment set. The argument following the flag must be a legal compartment name.

Fam will never allow the user to modify files he does not own or can not access. Also the integrity level of a file can never be raised above that of the user. It should be noted that when a user request that a file be given a lower security level (by either changing the security category to a lower one, or deleting a compartment) the entire file will be displayed before the change is allowed.

BUGS

Fam does not check to see if the filesystem maximum level is lower than a security request being made.

NAME

fsd - incremental file system dump

SYNOPSIS

fsd filesystem [-0vc] [-e extentseid] [-b blocks] [-d days] [-f device] [-h hours]

DESCRIPTION

Fsd makes an incremental dump of all files on the specified KSOS file system which were changed after a certain date. Fsd is spawned by the secure server at the request of a user running at OPERATOR or higher integrity level. The save medium may be either tape or an existing KSOS extent. Fsd opens the file system for exclusive use, thus the file system must be unmounted.

b	The next argument is the maximum size of the save tape (or extent) in blocks.
c	If the dump tape overflows, increment the minor device number and continue. Normally, you are asked to change tapes.
e	Dump to the specified defined extent instead of to tape.
d	The next argument specifies the dump date as some number of days prior to the current date.
f	Use the next argument as the save device instead of the default (device_nsp, 11, 0).
v	Print out the information in the dump header.
h	The next argument specifies the dump date as some number of hours prior to the current date.
0	Dump from the beginning of time.

DIAGNOSTICS

Generally errors are fatal. Files found in an unSAFE condition are not dumped, but the jnode is dumped and the high block field is set to 0. A message to this effect is printed for each bad file encountered.

FILES

/sys/dataBases/security system security map

SEE ALSO

fsr(1),dump(IV)

BUGS

The d and h options are not implemented - all dumps are from the beginning of time. When dumping to an extent, fsd could recover from most errors. Unfortunately, fsd's approach is to exit if anything is wrong.

NAME

fsr - incremental file system restore

SYNOPSIS

fsr filesystem [-citr] [-e extentseid] [-f device]

DESCRIPTION

Fsr is used to restore files dumped using the fsd command. It is spawned by the secure server at the request of a user running at OPERATOR or higher level. The dump tape (or extent) is read and files are copied to the file system specified. The jnode number of a restored file will be equal to its number before it was dumped. The latest incremental dump must be restored first onto a clear file system. At this time, jnodes are created for all files on the filesystem and they are restore locked to prevent their use for other purposes (such as indirect slots). As previous dumps are restored, only files with a jnode in the restore locked state are actually copied from the dump medium. Thus, to restore a file system, the incremental dumps must be restored in reverse order of that in which they were made. Optional arguments include:

c If the tape overflows, increment the minor device number and continue on the new drive.

e Restore from the specified extent and not from tape.

r Reconstruct the system space of the file system. The first restore to a clear extent must be done with this option. This should not be done lightly since any existing information on the extent will be lost.

f Read the dump from the tape drive specified by the next argument instead of from the default drive d11/0.

t Print the numbers of all jnodes restored.

SEE ALSO

fsd(1),dump(IV)

DIAGNOSTICS

If the security map on the dump tape and the system to which it is being restored do not agree fsr sends an audit capture message and aborts.

BUGS

The c option is not yet implemented.

No audit capture messages are sent, ever.

NAME

gaa - group access authentication database editor

SYNOPSIS

gaa

DESCRIPTION

Gaa interactively edits the group access authentication database and is invoked from the secure server. The editor commands include add, change, delete, find, next, print, view, and quit. A description of command action follows. The character preceding the closing parenthesis, ')', is the command code. Any unrecognized character causes printing of the command list.

a)dd prompts the user for all information and appends the new record to the end of the database. The following questions are asked:

```
Enter name (max 8 char):
Enter password (max 10 char):
Enter password again to verify :
Enter group identification number:
Enter user name of administrator (max 8 char):
ENTER MAXIMUM ACCESS LEVEL
SECURITY CATEGORY
Enter desired SECURITY category:
INTEGRITY CATEGORY
Enter desired INTEGRITY category:
Enter numbers of desired security compartments
separated by spaces (carriage return for NULL):
```

c)hange changes a specific field in the current record. Change accepts the following commands:

```
ex)it
v)iew
p)assword
g)roup id
a)dmin
m)ax access level
```

d)elete the current record by asking "Do you want to delete [current record name] (y or n) :"

p)rint the current database records, including modifications, to the lineprinter.

v)iew outputs the current record to the terminal

f)ind searches the database for the specified name and sets the current pointer to the record. The user is prompted for the name; gaa responds "Record is not found", if the name not in the database.

n)ext moves the current pointer is to the next record. If the pointer is currently pointing to the last record, it is moved to the first record.

q)uit ends execution of the editor. If modifications were made, the question "Do you want to save the updates?" is asked and a y or n is expected in response.

FILES

/sys/dataBases/user	user access authentication database
/sys/dataBases/group	group access authentication database
/sys/dataBases/security	system security map

SEE ALSO

UCE(III), SME(III)

ERRORS

can't open gaadb
can't create tempfile
can't open uaa
can't open security_map

NAME

login - sign onto KSOS

DESCRIPTION

The login command is used when a user initially signs onto KSOS. When the user hits the attention key at a terminal which is not logged in, the secure server (SSP) invokes login at that terminal.

Login prompts for a user name and password. Echoing is turned off (if possible) during the typing of the password, so it will not appear on the written record of the session.

After a successful login, the user and supervisor domain programs specified in the user access authentication database (user) should be entered. This will normally be the UNIX emulator. However, as an interim measure, login will prompt for the pathname of an emulator.

The user's security level will normally be the default level specified in the user access authentication database for the given login name. If this level is higher than the maximum level of the terminal or the current maximum level of the system, the user's level will be lowered accordingly and a diagnostic message will be issued.

FILES

/sys/dataBases/user	user access authentication database
/sys/dataBases/group	group access authentication database
/sys/dataBases/terminal	terminal profile database
/sys/dataBases/system	system profile database

SEE ALSO

SIP(III), SSP(III), user(IV), group(IV), terminal(IV), system(IV)

DIAGNOSTICS

"Login denied," if user does not exist or wrong password is given.
"Your maximum level is too low to login."

NAME

logout - sign off from KSOS

DESCRIPTION

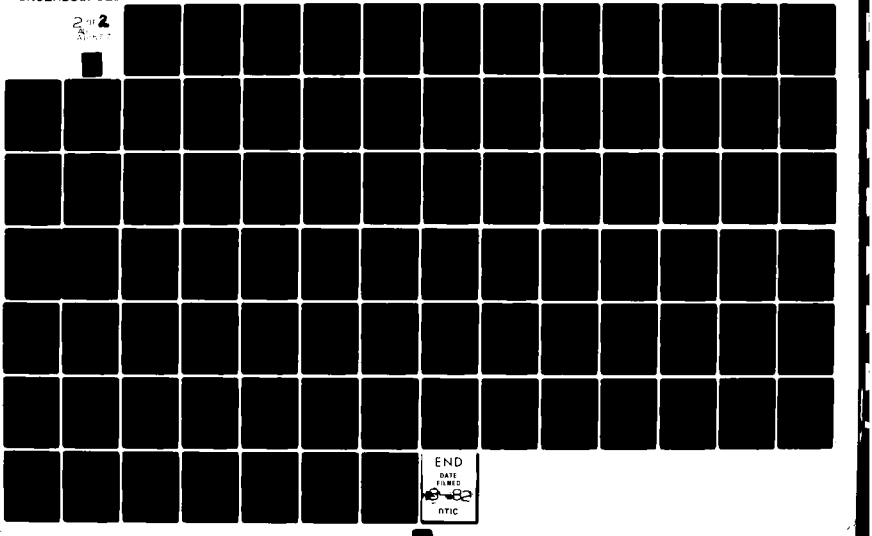
The logout command is used to sign off from KSOS. If the user has any active processes, these are immediately killed.

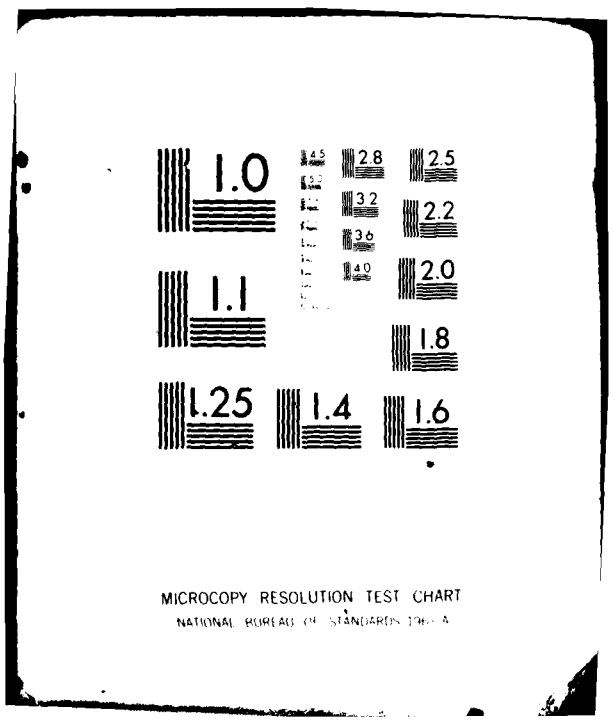
SEE ALSO

SSP(III), login(III)

AD-A111 877 FORD AEROSPACE AND COMMUNICATIONS CORP PALO ALTO CA W--ETC F/G 9/2
KSOS SECURE UNIX OPERATING SYSTEM USERS MANUALS. (KERNELIZED SE--ETC(U)
DEC 80 MDA903-77-C-0333

UNCLASSIFIED NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

NAME

mce - modify KSOS file system control entries

SYNOPSIS

mce deviceseid extent [-rvp]

DESCRIPTION

Mce is a file system maintenance tool which can be used to interactively view and modify KSOS file system control slots. The argument deviceseid specifies the device on which the file system resides (e.g. d1/0). Extent is a decimal integer specifying the extent on which the file system resides. By default mce is rather verbose and leads the user by the hand; the -v option instructs mce to interact with the user in an even more verbose mode. The -p option instructs mce to print out the direct and indirect pointer table values when viewing a jnode. The -r option indicates read only operation where slots may be viewed but not modified. This option allows the user to view a mounted file system. Only an unmounted file system may be modified.

Commands which operate on a particular slot may be optionally preceded by the slot number. If no slot number is supplied the current slot is assumed. Mce considers all numbers to be decimal, except where explicitly stated otherwise. Recognized commands include:

+ Increment current slot by one.

- Decrement current slot by one.

[n]v View slot. A formatted dump of the contents of the specified slot is produced. Slots of type jnode, indirect, reserved, free, mount item, extent item, and allocation item are recognized.

[n]i[m] Set current slot to the slot number pointed to by indirect pointer number m of the jnode or indirect slot number n.

[n]f Free the slot. Returns slot to free space.

e Exit mce.

[n]m Modify slot. This command places the user in modification mode. At this time jnodes, indirects and mount items may be modified. A special prompt (*) indicates modification mode. After receiving the prompt, the user types the control character corresponding to the slot field to be modified and mce responds by asking for specific data. To return to normal mode, type an empty line (<CR> only). Modifications are buffered until leaving modification mode, at which time the user may elect to save the changes or throw them away. Control characters for each type of slot are listed below.

jnode:
v view jnode.

```
        a   modify access rights, security and integrity.
        c   modify condition field.
        l   modify link count field.
        h   modify high block field.
        p   modify privileges.
        t   modify tail count field.
        s   modify self field.
        i   modify indirect pointer table.
        d   modify direct pointer table.

    indirect slot:
        v   view indirect slot.
        h   modify home jnode field.
        p   modify parent field.
        t   modify treeN field.
        i   modify indirect pointer table.
        d   modify direct pointer table.

    mount item:
        v   view mount item.
        a   modify access, security and integrity information.
        m   modify mounted field.
```

PRIVILEGES

```
    privMount                (* temporary *)
    privViolStarSecurity
    privViolStarIntegrity
    privViolDiscrAccess
```

WARNINGS

Mce is a powerful tool. Because of its power, it is also a very dangerous tool. A malicious user, or even a well-intentioned user who mistypes a character could potentially invalidate an entire file system or worse.

BUGS

MCE sends no audit capture messages.

NAME

mnt - mounts a file system

SYNOPSIS

mnt [filesystem_name drive_no mount_on_entry] [-r]

DESCRIPTION

Mnt logically mounts the given file system on the mount_on_entry. To be able to mount a particular file system, that file system must be listed in the immigration data base.

The first argument, filesystem_name, contains the name of the file system to be mounted. The drive_no argument gives the drive where the filesystem is located. The mount_on_entry argument is the complete pathname to the entry on which the file system is to be mounted. This entry must already exist. The -r flag, if present, indicates that the file system is to be mounted read only. Finally, if no arguments are given, a list of the currently mounted file systems is given.

The user must be at OPERATOR level or above to actually mount a file system.

FILES

/sys/dataBases/mountTable
/sys/dataBases/immigration

SEE

umt(III)

NAME

NKcopy - copy program

SYNOPSIS

nkcopy

DESCRIPTION

NKcopy copies information from a given input seid to a given output seid. It is used primarily to copy from tape to a file; however it can be used to copy between any two objects. This program is provided for development purposes only.

NKcopy excepts a single or multi-file tape. Each file should be written to tape in 512 byte blocks terminated by a tape mark (end of file). Under UNIX the files can just be cat'ed or cp'ed to tape. Once the tape is made it should be mounted on a TU16 tape drive.

NKcopy can be invoked by the secure server to copy the files on the tape to the desired files. When invoked NKcopy prompts for input and output seids and block lengths. Seids should be given in the following format.

name space/ unique id 0/ uniq id 1

Well known namespaces:

r - root name space.
d - device name space. The seid of the TU16 drive is d/11/unit number (0, 1)
n - null name space.

If a null output seid is given then a file is created and the seid of the newly created file is printed.

NKcopy also has the ability to mount another file system and copy the file to that file system.

BUGS

NKcopy does not use the directory manager. Presently it is necessary to create a file with the directory manager test frame (udm_tf), then remembering the seid, copy the input file to the newly created file file.

SEE

UDM_TF - UDM Test Frame

NAME

PBB - Process Bootstrapper

DESCRIPTION

The Process Bootstrapper may only be used by a process executing in supervisor space. It is an intermediary that is brought into execution via the K_invoke and K_spawn Kernel calls. The function of the PBB is to replace the segments of its process with segments filled from the image files specified in the argument segment passed to the Bootstrapper. After this replacement has been accomplished, the PBB sets the privileges, sets the effective user and group id's, and transfers control to the supervisor domain at a well-defined location.

The steps performed when an invoke or spawn is executed, using PBB as the intermediary, are given below.

a. The calling process (via the spawn/invoke interfaces used by the NKSR) will construct the argument segment for the process bootstrapper. The argument block is built in location 0, user domain, d-space.

b. It will then map the argument segment out of its' address space.

c. K_invoke or K_spawn will be called. (K_invoke will release all of the calling segments except the argument segment. K_spawn will only instantiate the argument segment.)

d. K_invoke/K_spawn will then rendezvous with a copy of the intermediary segment and put it at supervisor domain, i-space, address 0. The current pc/ps will be set to address 0 in the supervisor domain. This set up is required for the process bootstrapper to run non-separate I&D.

e. The process bootstrapper will map the argument segment into its supervisor i-space, thus maintaining non-separate I&D.

f. The process bootstrapper will then build the invoked supervisor domain image in the user domain and then map it out, and use K_set_segment_status to make the vitual address be supervisor domain instead of user domain.

g. The process bootstrapper will then build the invoked user domain image in the user domain.

h. The process bootstrapper will release the argument segment.

i. The process bootstrapper will then issue a K_boot call with the segment descriptor of the intermediary segment. The K_boot Kernel call will release the intermediary segment and then map in the supervisor domain segments to where they belong. The pc/ps will be set to address 0 in the supervisor domain.

The invoked process image now exists and is executing.

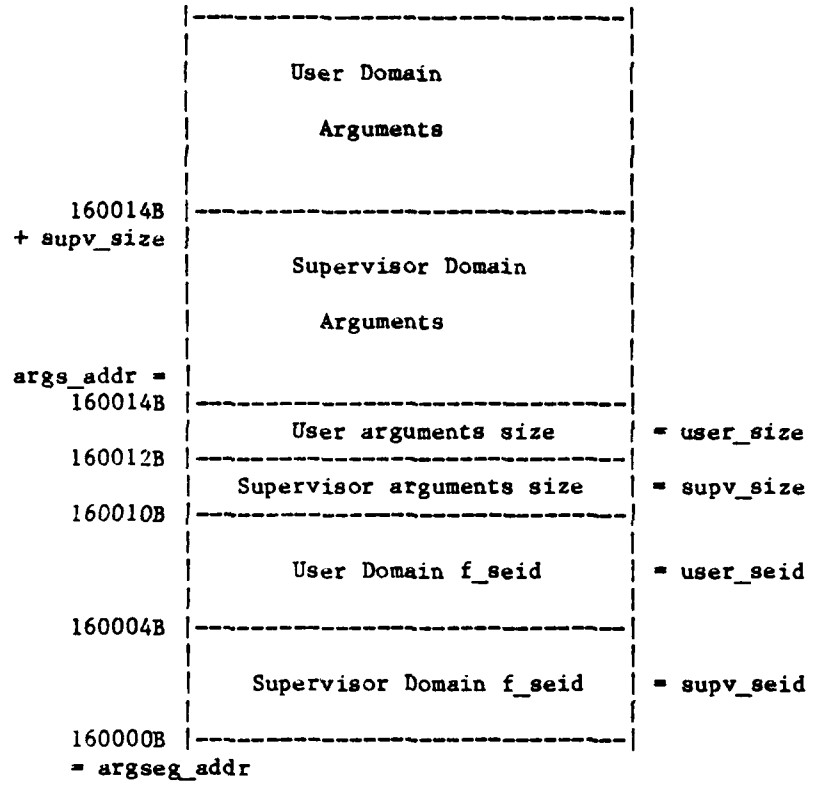
The initial process builds two bootstrapper segments: a user process bootstrapper and a secure server bootstrapper segment.

The user process boostrapper segment is available to all users. The new process runs at the same level as the parent process, and is given the privileges of its' a.out file.

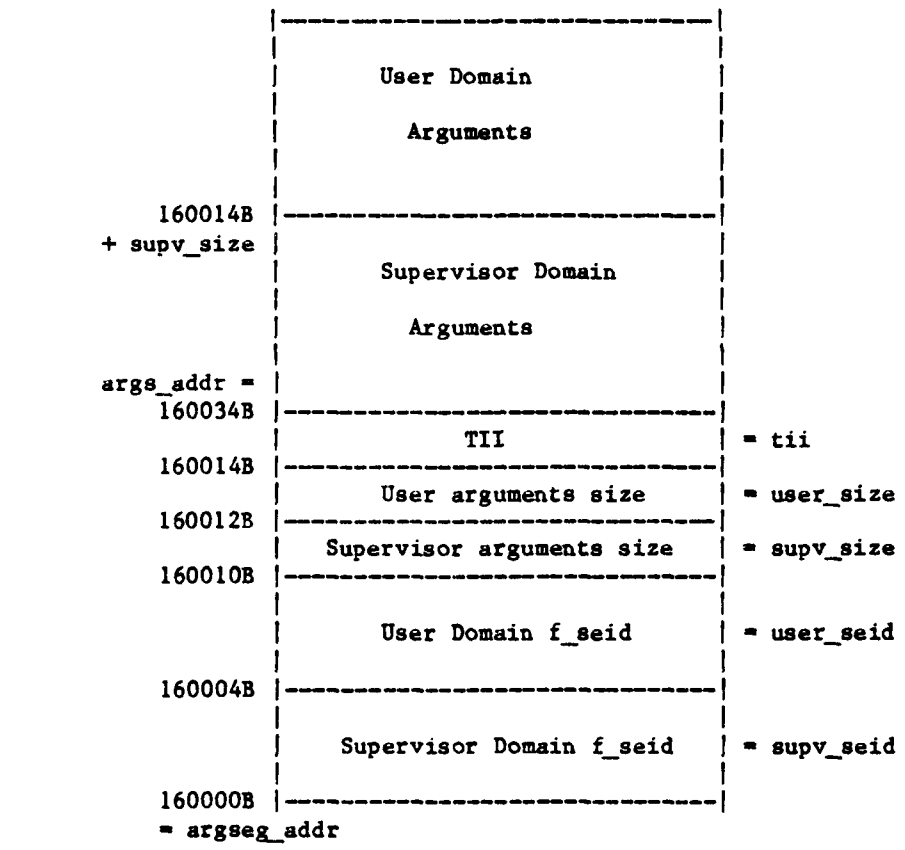
The secure server process bootstrapper segment is only used by the NKSR and exists at system high level. It takes a tii structure as part of the argument block. The child process runs with a tii equal to that which was passed in the argument block.

Once the process bootstrapper segments are built, the secure initiator process (SIP) makes directory entries "/sys/sysbin/userPBB" and "/sys/sysbin/serverPBB".

The user process bootstrapper argument segment has the following format:



The secure server process bootstrapper argument segment has the following format:



SEE ALSO
SIP(III)

NAME

pki - pack initialization program

SYNOPSIS

pki packseid [-a]

DESCRIPTION

Pki initializes the pack reserved extents on an uninitialized pack. Pki is spawned by the secure server at the request of a user running at OPERATOR or higher security level. The pack master slot is initialized and the pack is given the proper access level. The pack reserved extents (extents 1-4) are created and given the system subtype and an access level of SYSTEMTII. The -a option indicates that the user should be prompted for the pack's access level. The default level is: syshi security, syslow integrity, all compartments, NKSR owner and group, and a discretionary access of rwx---.

SEE ALSO

exi(I), btcp(I)

NAME

rkget - Get files from RK05 pack.

SYNOPSIS

rkget

DESCRIPTION

Rkget copies files from a specially formatted rk05 pack to the specified KSOS filenames. The user is prompted for an rk file number and a KSOS destination file name. If the KSOS destination file already exists it is written over; otherwise a new file is created. Rkget is normally used to retrieve files written to the rk pack using the *UNIX* cprk program.

NAME

setvv - set volume valid

SYNOPSIS

setvv deviceseid

DESCRIPTION

Setvv is a utility used to set the volume-valid condition on the device specified by deviceseid (e.g. setvv d4/0). This marks a removable medium as usable. The volume valid operation is accomplished using the K_device_function kernel call. Setvv is spawned by the secure server at the request of a user running at operator or higher security level.

PRIVILEGES

privImmigrate

NAME

gaa - security map editor

SYNOPSIS

sme

DESCRIPTION

Sme interactively edits the security map database and is invoked from the secure server. The editor commands include add, change, delete, level, print, view, and quit. A description of command action follows. The character preceding the closing parenthesis, ')', is the command code. Any unrecognized character causes printing of the command list.

a)dd prompts the user for all information and appends the new record to the end of the database. The following questions are asked:

Enter entry number where addition is to be placed:
Enter short name (max 12 char):
Enter long name (max 50 char):
Is this entry to be active (yes or no):

c)hange asks for the entry number to be changed. Change accepts the following commands:

ex)it
v)iew
s)hort name
l)ong name
a)ctive

d)elete asks for the entry number to be deleted.

p)rint the current database records, including modifications, to the lineprinter.

v)iew outputs the current level records to the terminal

q)uit ends execution of the editor. If modifications were made, the question "Do you want to save the updates?" is asked and a "y" or "n" is expected in response.

FILES

/sys/dataBases/security system security map

SEE ALSO

security(IV)

ERRORS

can't open security map database

NAME

spe - system profile editor

SYNOPSIS

spe

DESCRIPTION

Spe interactively edits the system profile database and is invoked from the secure server. The editor commands include quit, view, print, and change of various fields. A description of command action follows. The character preceding the closing parenthesis, ')', is the command code. Any unrecognized character causes printing of the command list.

q)uit ends execution of the editor. If modifications were made, the question "Do you want to save the updates?" is asked and a "y" or "n" is expected in response.

v)iew outputs the record to the terminal

p)rint has not been implemented.

Change accepts the following commands:

```
s)ys name
i)nst name
sys n)um
ve(r)sion num
gen d)ate
m)ax acc lev
c)urr max acc lev
min l)ogin acc lev
```

The three access level fields have the following dialog with the user:

```
SECURITY CATEGORY
Enter desired SECURITY category:
INTEGRITY CATEGORY
Enter desired INTEGRITY category:
Enter numbers of desired security compartments
separated by spaces (carriage return for NULL):
ENTER MAXIMUM ACCESS LEVEL
SECURITY CATEGORY
Enter desired SECURITY category:
INTEGRITY CATEGORY
Enter desired INTEGRITY category:
Enter numbers of desired security compartments
separated by spaces (carriage return for NULL):
```

FILES

```
/sys/dataBases/system   system profile database
/sys/dataBases/security system security map
```

SEE ALSO
SME(III), system(IV), security(IV)

ERRORS
can't open sysdb
can't create tempfile
can't open security_map

NAME

SSP - secure server process

DESCRIPTION

The secure server is essentially a rudimentary command interpreter which allows a user to execute programs ("services"). One secure server is spawned for each configured terminal on the system. When the secure attention key is struck, the secure server responds by either invoking login or by prompting for a command if someone is already logged in. The server prompt is "> ". The environment active when the secure attention key is struck is suspended. Typing a carriage return in response to the server prompt will resume the interrupted environment.

Server commands are just a single line of input, the first word specifying the particular service to be performed. The remainder of the line is passed, uninterpreted by the server, as an argument to the requested service. The server provides limited editing capabilities on terminal input: backspacing will erase single characters, and an '@' will erase the whole line.

Some commands such as change access level (cal) and logout are interpreted directly by the server. The remainder are the file names of programs which are in the server's program directories.

The server only permits one service to be executing at any one time, and an attempt to execute more than one concurrently will produce a diagnostic message. An interim feature of the server is the kill command, which peremptorily kills any service the user is currently executing.

FILES

/sys/dataBases/user	user access authentication database
/sys/dataBases/group	group access authentication database
/sys/dataBases/terminal	terminal profile database
/sys/dataBases/system	system profile database
/sys/server/admin	a server program directory
/sys/server/operator	a server program directory
/sys/server/user	a server program directory

SEE ALSO

SIP(III), login(III), cal(III), logout(III), user(IV), group(IV), terminal(IV), system(IV)

DIAGNOSTICS

"service still executing"

NAME

stc - storage consistency check

SYNOPSIS

stc deviceseid extent

DESCRIPTION

Stc examines the KSOS file system residing on the specified extent and reports any inconsistencies.

Diagnostic messages report on the following inconsistencies:

- Blocks claimed by more than one jnode or the free list.
- Blocks or slots outside the range of the file system.
- Lost blocks and slots.
- Bad file condition.
- Incorrect file size.

Upon completion stc outputs the following summary information:

Total files on the system.
Total blocks on the system.
blocks allocated.
of free blocks.
# of lost blocks.
# of duplicated blocks.
Total slots defined.
slots allocated.
free slots.
lost slots.
duplicate slots.

SEE ALSO

mce

BUGS

No scratch file is used, so the size of a system which can be checked is limited.
No audit capture messages are ever sent.

NAME

tpe - terminal profile database editor

SYNOPSIS

tpe

DESCRIPTION

Tpe interactively edits the group access authentication database and is invoked from the secure server. The editor commands include add, change, delete, find, next, print, view, and quit. A description of command action follows. The character preceding the closing parenthesis, ')', is the command code. Any unrecognized character causes printing of the command list.

a)dd prompts the user for all information and appends the new record to the end of the database. The following questions are asked:

```
Enter tty id:
Is the terminal configured ? (y or n) :
Is the terminal to be treated as a console ? (y or n) :
Enter default transmission baud rate:
Enter default receiveing baud rate:
Enter default parity (even, odd, or none) :
Enter clear screen sequence (max 8 chars) :
ENTER MAXIMUM ACCESS LEVEL
SECURITY CATEGORY
Enter desired SECURITY category:
INTEGRITY CATEGORY
Enter desired INTEGRITY category:
Enter numbers of desired security compartments
separated by spaces (carriage return for NULL):
```

c)hange changes a specific field in the current record. Change accepts the following commands:

```
ex)it
v)iew
i)d
f)configured
c)onsole
t)ransmit rate
r)eceive rate
p)arity
s)creen clear
m)ax access level
```

d)elete the current record by asking "Do you want to delete [current record name] (y or n) :"

p)rint the current database records, including modifications, to the lineprinter.

v)iew outputs the current record to the terminal

f)ind searches the database for the specified id and sets the current pointer to the record. The user is prompted for the id; tpe responds "Record is not found", if the id is not in the database.

n)ext moves the current pointer is to the next record. If the pointer is currently pointing to the last record, it is moved to the first record.

q)uit ends execution of the editor. If modifications were made, the question "Do you want to save the updates?" is asked and a y or n is expected in response.

FILES

/sys/dataBases/terminal terminal profile database
/sys/dataBases/security system security map

SEE ALSO

SME(III), terminal(IV), security(IV)

ERRORS

can't open tpdb
can't create tempfile
can't open security_map

NAME

uce - user access authentication database editor

SYNOPSIS

uce

DESCRIPTION

Uce interactively edits the user access authentication database and is invoked from the secure server. The editor commands include add, change, delete, find, next, print, view, and quit. A description of command action follows. The character preceding the closing parenthesis, ')', is the command code. Any unrecognized character causes printing of the command list.

a)dd prompts the user for all information and appends the new record to the end of the database. The following questions are asked:

```
Enter name (max 8 char):
Enter password (max 10 char):
Enter password again to verify :
Can this owner login ? (y or n) :
Enter owners id number :
Enter owners group name :
ENTER ACCESS LEVEL FOR LOGIN
SECURITY CATEGORY
Enter desired SECURITY category:
INTEGRITY CATEGORY
Enter desired INTEGRITY category:
Enter numbers of desired security compartments
separated by spaces (carriage return for NULL):
ENTER MAXIMUM ACCESS LEVEL
SECURITY CATEGORY
Enter desired SECURITY category:
INTEGRITY CATEGORY
Enter desired INTEGRITY category:
Enter numbers of desired security compartments
separated by spaces (carriage return for NULL):
Enter directory pathname (max 63 char) :
Enter shell pathname (max 63 char) :
Enter emulator pathname (max 63 char) :
```

c)hange changes a specific field in the current record. Change accepts the following commands:

```
ex)it
v)iew
p)assword
l)ogin ok
o)wner id
g)roup name
m)ax access level
curr a)ccess level
d)ir path
```

s)hell path
e)mul path

d)elete the current record by asking "Do you want to delete [current record name] (y or n) :"

p)rint the current database records, including modifications, to the lineprinter.

v)iew outputs the current record to the terminal

f)ind searches the database for the specified name and sets the current pointer to the record. Receives search string from "Enter name :". Responds "Record not found", if name not in database.

n)ext moves the current pointer to the next record. Pointer moved to the first record if currently pointing to the last record.

q)uit ends execution of the editor. If modifications were made, the question "Do you want to save the updates?" is asked and a y or n is expected in response.

FILES

/sys/dataBases/user	user access authentication database
/sys/dataBases/group	group access authentication database
/sys/dataBases/security	system security map

SEE ALSO

GAA(III), SME(III), user(IV)

ERRORS

can't open uaadb
can't create tempfile
can't open gaa
can't open security_map

NAME

UDM - UNIX Directory Manager.

DESCRIPTION

UDM maintains a UNIX-like directory structure on top of a KSOS file system. All directory operations, creating entries, removing entries and searching directories is supported by the directory manager and directory manager interface procedures.

Directories are of a distinguished subtype known as the directory subtype. The directory subtype is used to insure that the UDM is the only process that is allowed to write directories. However, other processes may open directories for reading by presenting a read directory subtype open descriptor to K_open.

Access to the directory manager is provided by the directory manager interfcae, which must be compiled with the program which plans to use it. The directory manager interface provides a procedure call interface to the UDM. The interface handles the packing of a UDM argument block, the building an argument segment, the spawning of the UDM and the waiting for IPC status return.

DEFINITIONS

Path Name - A path name is a character array of directory names, where the character "/" is used to separate directory names.

Leaf Name - The leaf component of a path name is the last name in a path name.

Starting Directory - Directory operations take a path name and a starting directory (seid) as arguments. If a rooted path name (a path name that begins with "/") is given to a directory manager interface procedure, the directory operation assumes the root directory as the starting directory regardless of the starting directory given.

Directory Subtype - The directory subtype is a well known seid.

seid (subtype_nsp, char(100) cardinal (0))

All directories are of directory subtype.

Root Directory - The seid of the first mounted file system, which is know as the root file system, is:

seid (root_nsp, char(0) cardinal (5))

UDM Event Type - The directory manager process communicates status information to its parent, the directory manager interface procedure, via an IPC. The first byte of the message portion of the IPC contains the event type of the IPC. The IPC returned from the directory manager shall always have event type 26 decimal.

UDM_error - UDM error is an enumerated type that is declared in UDM interface. Its MODULA definition is:

```
UDM_error = (UDM_no_error,
     UDM_cannot_do,
     UDM_cannot_link,
     UDM_cannot_unlink,
     UDM_entry_exists,
     UDM_entry_does_not_exist,
     UDM_cannot_open_directory,
     UDM_seid_refers_to_a_directory,
     UDM_cannot_create_directory,
     UDM_cannot_remove_directory,
     UDM_directory_not_empty,
     UDM_not_directory,
     UDM_directory_not_writable,
     UDM_not_executable,
     UDM_no_path,
     UDM_not_found,
     UDM_cannot_link_across_file_systems,
     UDM_cannot_mount,
     UDM_cannot_unmount);
```

Note that an enumerated type in MODULA begins at zero.

INTERFACE PROCEDURE

UDM_mkentry - make a directory entry

MODULA SYNOPSIS

```
CONST     SEID_of_dir       : seid;
CONST     SEID_of_entry     : seid:
CONST     path_name         : ARRAY integer OF char;
CONST     wait_flag         : boolean;
CONST     UDM_stat          : UDM_error;

UDM_stat := UDM_mkentry ( SEID_of_dir,
                          SEID_of_entry,
                          path_name,
                          wait_flag);
```

DESCRIPTION

The UDM mkentry procedure causes a directory entry to be made. Starting at the directory specified by SEID of dir, the components of the path name path name are used to find subordinate directories until only a leaf component of the path name remains. A directory entry is then made in the parent directory of the leaf name. This directory entry is composed of the SEID of entry and the leaf component of path name. Seids with any name space may be used as a SEID of entry. However seids with a file name space shall be linked.

The wait flag specifies if the directory operation is to be synchronous or asynchronous. If the wait flag is true then the procedure shall not return until a IPC status block is received from the UDM.

Note that files are created with a zero link count and remain in existence so long as they are open or have a non-zero link count. Therefore, the proper way to make a directory entry for a newly created file is to create the file, call UDM mkentry, and then close the file.

DIAGNOSTICS

UDM mkentry returns the following error codes:

- UDM_no_error
- UDM_cannot_link
- UDM_entry_exists
- UDM_cannot_open_directory
- UDM_seid_refers_to_a_directory
- UDM_cannot_link_across_file_systems

INTERFACE PROCEDURE

UDM_mkdir - create a directory

MODULA SYNOPSIS

```
CONST     SEID_of_dir     : seid;
CONST     dir_name        : ARRAY integer OF char;
CONST     wait_flag       : boolean;
VAR       UDM_stat        : UDM_error;

   ...

UDM_stat := UDM_mkdir ( SEID_of_dir,
                        dir_name,
                        wait_flag);
```

DESCRIPTION

UDM_mkdir creates a directory. Starting at the directory specified by SEID_of_dir the components of the path name dir_name are used to find subordinate directories until the last parent directory component is found. A directory is then created in the parent directory with the name of the leaf name. This newly created directory shall have access modes of read, write, execute by owner and read, execute by group and others. Also, this directory shall contain two distinguished directory entries, "." and "..". The "." directory entry shall refer to the newly created directory and the ".." directory entry shall refer to the parent directory.

The wait_flag specifies if the directory operation is to be synchronous or asynchronous. If the wait_flag is true then the interface procedure shall not return control to the caller until an IPC status block is received from the UDM.

DIAGNOSTICS

UDM_mkdir returns the following error codes:

- UDM_no_error
- UDM_entry_exists
- UDM_cannot_open_directory
- UDM_cannot_create_directory

INTERFACE PROCEDURE

UDM_rm - remove a directory entry

MODULA SYNOPSIS

```
CONST     SEID_of_dir     : seid;
CONST     path_name       : ARRAY integer OF char;
CONST     wait_flag       : boolean;
VAR       UDM_stat        : UDM_error;

 ...

UDM_stat := UDM_rm ( SEID_of_Dir,
                     path_name,
                     wait_flag);
```

DESCRIPTION

The UDM_rm procedure causes a directory entry, which may be a file or directory, to be removed. Starting at the directory specified by SEID_of_Dir the components of the path name path_name are used to find subordinate directories until only a leaf component of the path name remains. The leaf name directory entry is then removed. If the directory entry to be removed is a file, K_unlink is called. If the directory entry to be removed is an empty directory, consisting of only "." and ".." entries, the directory is removed.

The wait_flag specifies if the directory operation is to be synchronous or asynchronous. If the wait_flag is true then the procedure shall not return until a IPC status block is received from the UDM.

DIAGNOSTICS

UDM_rm returns the following error codes:

- UDM_no_error
- UDM_not_found
- UDM_cannot_unlink
- UDM_directory_not_empty
- UDM_cannot_open_directory
- UDM_cannot_remove_directory

INTERFACE PROCEDURE

UDM_find - Path Interpretation.

MODULA SYNOPSIS

```
CONST    starting_dir_seid     : seid;
CONST    path_name             : ARRAY integer OF char;
CONST    operation             : pathOp;
VAR      parent_dir_seid       : seid;
VAR      entry_seid            : seid;
VAR      entry_name            : ARRAY integer OF char;
VAR      UDM_stat              : UDM_error;

...

UDM_stat := UDM_find (starting_dir_seid,
                      path_name,
                      operation,
                      parent_dir_seid,
                      entry_seid,
                      entry_name);
```

DESCRIPTION

UDM_find performs path interpretation operations. Starting at the directory specified by starting_dir_seid, the components of the path name path_name are used to find subordinate directories until only a leaf component of the path name remains. The path operation specified by operation is then performed.

For all directory operations, if the path name to the parent directory of the leaf name is legal, then the parent_dir_seid is filled with the seid of the parent directory and the entry_name is filled with the leaf name. If the entry_name directory entry exists then entry_seid is filled with the seid of that directory entry.

The following path operations are defined:

po_mkentry - checks if a directory entry of the specified name can be made. A status of UDM_no_error is returned if the process has write permissions in the parent directory and if the entry of the desired name does not exist.

po_remove - determines if the directory entry of the specified name can be removed. A status of UDM_no_error is returned if the process has write permissions in the parent directory and the leaf name directory entry exists. If the leaf name is a directory that directory must only contain the "." and ".." directory entries.

po_locate - determines if a directory entry of the specified name exists. A status of UDM_no_error is returned if the entry is found.

po_chdir - checks if the specified directory entry is a directory and that the process has search/execute permissions for that directory. If the above checks are satisfied a status of UDM no error is returned.

po_exec - checks if the specified directory entry exists and is an executable file. A status of UDM no error is returned if the process has execute permissions for that file and if the file contains the magic numbers of 407, 410 or 411 (octal).

DIAGNOSTICS

UDM_find returns the following error codes:

UDM_no_error
UDM_no_path
UDM_cannot_do
UDM_not_found
UDM_entry_exists
UDM_not_directory
UDM_cannot_open_directory
UDM_directory_not_writable

USING THE UDM INTERFACE

Two UDM interfaces exist, one for supervisor MODULA programs and one for supervisor C programs.

To use the UDM interface for MODULA one must:

a) Define the following cpp constants: k_oldseg, k_invoke, k_spawn, k_getps, k_relseg, k_getss, k_setss, k_setps, and k_setda. Defining these constants allows the related kernel calls to be conditionally compiled with the source.

b) Include KERcalls.mod.

c) Define the cpp constant ipc_on. This cpp constant allows the IPC handling procedure to be compiled.

d) Declare the procedure IPC_valid. This procedure is imported into the pseudo-interrupt handling module and it is used to check the validity of an ipc before it is put on the IPC queue. This procedure allows a process to guarantee that its IPC queue will not get filled with unwanted IPC's. The procedure, IPC_valid should take an ipc_block as a parameter and should return a boolean result. True if the IPC is to be placed on the queue and false if the IPC is to be discarded.

e) Include the the pseudo-interrupt handling module npi.mod. f) Include the UDM directory manager interface for MODULA.

FILES

KERcalls.mod	- Kernel interface
udm_lib.mod	- UDM interface for MODULA
EDI.h	- UDM interface for C
npi.mod	- Pseudo-interrupt handling module

SEE ALSO

ipc_block (I)
seid (I)

NAME

udmtf - UDM Test Frame

SYNOPSIS

udmtf

DESCRIPTION

The udmtf allows one to exercise the directory manager. It is often used to create files, which can be overwritten by NKcopy. Udmtf prompts with a ">" and accepts the following commands:

ls [-lrT] [directory name] provides a similar functionality to the UNIX ls.

- l signifies that a long listing is to be performed.
- r signifies that a the subdirectories are to be recursively listed.
- T signifies that the security information is to be printed.

mkentry creates a file.

rm removes a file or a directory.

mkdir creates a directory.

quit causes the udmtf process to exit.

BUGS

Erase and kill processing is not performed on input.

NAME

umt - unmounts a file system

SYNOPSIS

umt filesystem_name

DESCRIPTION

Umt logically unmounts the file system given in the filesystem_name argument. The name must consist of a full pathname to the file system.

The user must be at OPERATOR level or above to actually unmount a file system.

FILES

/sys/dataBases/mountTable
/sys/dataBases/immigration

SEE

mnt(III)

NAME

device - device profile database

DESCRIPTION

The device profile data base contains the owner and maximum access level for each device on the system.

The modula definition of a device profile database record is:

```
RECORD                                          SIZE
  deviceName        : ARRAY 0:7 OF char;        8 bytes
  deviceSeid        : seid;                     4 bytes
  device tii        : tii struct;              16 bytes
  valid required    : boolean;                  1 byte
  assign required   : boolean;                  1 byte
END;                                           -------
                                               30 bytes
```

deviceName — Name of the device.

deviceSeid — Seid of the device.

device_tii — Includes device owner, group and maximum access level of the device.

valid_required — This device is a disk. A VOLUME_VALID K_device_function is required on this device before the disk can be accessed.

assign_required — This device can only be accessed after it has been assigned.

The device profile database can be modified with the device profile editor (DPE). All fields of the device profile record can be modified with this editor.

This data base is read by the secure initiator (SIP), assign (ASG) and deassign (DSG).

FILES

/sys/dataBases/device

SEE ALSO

DPE(III), ASG(III), DSG(III), tii_struct(I), seid(I)

NAME

dump - incremental dump format

DESCRIPTION

The fsd and fsr programs are used to incrementally dump and restore KSOS file systems. The dump format consists of four sections, each of which is an integral number of 512-byte blocks long: a one block dump master, the file system's security map, a dump map and, finally, the jnodes and data blocks for each file dumped. The complete dump is written out as one long record composed of logical 512-byte blocks. The first block has the following structure:

```
dump_master = RECORD
   mount : mountItem        (* fs mount item - 102 bytes *)
   from_date : timeStamp;   (* incremental dump date *)
   dump_date : timeStamp;   (* date this dump was taken *)
   size : cardinal32;       (* blocks used on save device *)
   secmap_size : cardinal;  (* security map size (blocks) *)
   filler : ARRAY 1:DM_FILLSZ OF char; (* filler *)
   checksum : cardinal;     (* block checksum *)
END;
```

The security map is a copy of the security map database for the KSOS system on which the file system resides. Its size is specified by secmap_size and is normally 8 blocks long. The dump map contains one boolean element for each slot in the system space of the file system. It indicates which file system slots contain jnodes. It is essentially a copy of the file system allocation map minus references to indirect items. Its size (in blocks) is equal to (total system slots)/(512*8 bits per block) rounded up. The rest of the tape is made up of the data blocks for each dumped file. Each set of data blocks is immediately preceded by a block containing the jnode for the file. These jnode blocks each contain a dump block number and a checksum. The final block on the tape has a dump block number of 0.

NAME

group - group access authentication database

DESCRIPTION

The group access authentication database contains group identification, administrator and maximum access level information.

The modula definition of a group access authentication database record is:

```
RECORD                                          SIZE
  groupName      : ARRAY 0:7 OF char;           8 bytes
  groupPassword  : ARRAY 0:10 OF char;          12 bytes
  groupid        : cardinal;                    2 bytes
  maxLevel       : access level type;           6 bytes
  groupAdmin     : integer;                     2 bytes
END;                                            -------
                                                30 bytes
```

groupName — name of the group

groupPassword — encoded password of the group

groupid — unique group identification number

maxLevel — maximum access level of the group

groupAdmin — a user that serves as the group administrator

The group access authentication database can be modified with the user control editor (UCE). All fields of the group access authentification record can be modified with this editor.

The data base is read by the secure server (SSP), and file access modification (FAM).

FILES

/sys/dataBases/group

SEE ALSO

UCE(III), SSP(III), FAM(III), access_level_type(I)

NAME

security - security map database

DESCRIPTION

The security map database specifies the defined security levels, integrity levels and security compartments of the system.

The security map database is divided into three sections. There are 16 security level entries, 16 integrity level entries, and 32 security compartment entries.

The modula definition of a security map database record is:

```
RECORD                                          SIZE
  shortName  : ARRAY 0:11 OF char;              12 bytes
  longName   : ARRAY 0:49 OF char;              50 bytes
  fillnull   : char;                             1 byte
  active     : char;                             1 byte
END;                                            --------
                                                64 bytes
```

shortName — is in uppercase and does not need to end with null.

longName — is uppercase, may include blanks, and must end with null.

active — is 'A' active or 'I' inactive.

The map has 64 entries: 0 - 15 are security level categories, 16 - 31 are integrity level categories, and 32 - 63 are security compartments.

The security map database can be modified with the security map editor (SME). All fields of the security map record can be modified with this editor.

FILES

/sys/dataBases/security

SEE ALSO

SME(III)

NAME

system - system profile database

DESCRIPTION

The system profile database contains system identification and access level information.

The modula definition of a system profile database record is:

```
RECORD                                          SIZE
  sys name       :ARRAY 0:59 OF char;          60 bytes
  inst name      :ARRAY 0:59 OF char;          60 bytes
  sys no         :cardinal;                     2 bytes
  version        :versionType;                  4 bytes
  gen date       :cardinal32;                   4 bytes
  systemMax      :access level type;            6 bytes
  currentMax     :access level type;            6 bytes
  currentMinMax  :access level type;            6 bytes
END;                                          --------
                                              148 bytes
```

sys name	system name for this system
inst name	name of installation where this system is located
sys no	unique KSOS system number
version	KSOS OS version number (major/minor)
gen date	KSOS OS system generation date (ticks since January 1, 1980)
systemMax	max level ever permitted for this system
currentMax	max level currently permitted on this system
currentMinMax	defines the lowest maximum level needed to login in (i.e. if a user's max level is less than this level he can not login)

The system profile database can be modified with the system profile editor (SPE). All fields of the system profile record can be modified with this editor.

The data base is read by file access modification (FAM) and secure server (SSP).

FILES

/sys/dataBases/system

SEE ALSO

SPE(III), FAM(III), SSP(III), access_level_type(I)

NAME

terminal - terminal profile database

DESCRIPTION

The terminal profile database contains terminal configuration and access level information.

The modula definition of a terminal profile database record is:

```
RECORD                                          SIZE
  ttyid          :char;                         1 byte
  config         :boolean;                      1 byte
  console        :boolean;                      2 bytes
  xmitBaud       :cardinal;                     2 bytes
  recBaud        :cardinal;                     2 bytes
  parity         :cardinal;                     2 bytes
  clrScreen      :ARRAY 1:8 OF char;            8 bytes
  maxLevel       :access level type;            6 bytes
END;                                            -------
                                                24 bytes
```

ttyid	unique terminal id
config	terminal is configured
console	terminal is to be treated as a console
xmitBaud	default transmit baud rate
recBaud	default recieve baud rate
parity	default parity
clrScreen	sequence required to clear the screen
maxLevel	maximum access level for this terminal

The terminal profile database can be modified with the terminal profile editor (TPE). All fields of the terminal profile record can be modified with this editor.

The data base is read by the secure server (SSP) and file access modification (FAM).

FILES

/sys/dataBases/terminal

SEE ALSO

TPE(III), SSP(III), FAM(III)

NAME

user - user access authentication database

DESCRIPTION

The user access authentication database contains user identification and access level information.

The modula definition of a user access authentication database record is:

```
RECORD                                              SIZE
  userName       : ARRAY 0:7 OF char;               8 bytes
  userPassword   : ARRAY 0:10 OF char;              12 bytes
  loginOk        : boolean;                          2 bytes
  maxLevel       : access level type;                6 bytes
  tii            : tii struct;                      16 bytes
  loginDir       : ARRAY 0:63 OF char;              64 bytes
  loginShell     : ARRAY 0:63 OF char;              64 bytes
  loginEmul      : ARRAY 0:63 OF char;              64 bytes
  filler         : ARRAY 0:19 OF char;              20 bytes
END;                                                ---------
                                                   256 bytes
```

userName	name of the user
userPassword	password of the user
loginOk	true if user can login
maxLevel	maximum access level
tii	login access level
loginDir	login directory pathname
loginShell	login shell pathname
loginEmul	login emulator pathname

The user access authentication database can be modified with the user control editor (UCE). All fields of the user access authentication record can be modified with this editor.

The data base is read by the secure server (SSP) and file access modification (FAM).

FILES

/sys/dataBases/user

SEE ALSO

UCE(III), SSP(III), FAM(III), tii_struct(I), access_level_type(I)

NAME

break, brk, sbrk - change core allocation

SYNOPSIS

```
(break = 17.)
sys break; addr

char *brk(addr)

char *sbrk(incr)
```

DESCRIPTION

Break sets the system's idea of the lowest location not used by the program (called the break) to addr (rounded up to the next multiple of 512 bytes). Locations not less than addr and below the stack pointer are not in the address space and will thus cause a memory violation if accessed.

From C, brk will set the break to addr. The old break is returned.

In the alternate entry sbrk, incr more bytes are added to the program's data space and a pointer to the start of the new area is returned.

When a program begins execution via exec the break is set at the highest location defined by the program and data storage areas. Ordinarily, therefore, only programs with growing data areas need to use break.

SEE ALSO

exec (II), alloc (III), end (III)

DIAGNOSTICS

The c-bit is set if the program requests more memory than the system limit or if more than 8 segmentation registers would be required to implement the break. From C, -1 is returned for these errors.

BUGS

Setting the break in the range 0177700 to 0177777 is the same as setting it to zero.

KSOS

Note that in KSOS, the system's memory grain size is 512 bytes as opposed to the 64 bytes of UNIX.

NAME

chdir - change working directory

SYNOPSIS

```
(chdir = 12.)
sys chdir; dirname

chdir(dirname)
char *dirname;
```

DESCRIPTION

Dirname is the address of the pathname of a directory, terminated by a null byte. Chdir causes this directory to become the current working directory.

SEE ALSO

chdir (I)

DIAGNOSTICS

The error bit (c-bit) is set if the given name is not that of a directory or is not readable. From C, a -1 returned value indicates an error, 0 indicates success.

KSOS

NAME

chmod - change mode of file

SYNOPSIS

(chmod = 15.)

```
sys chmod; name; mode

chmod(name, mode)
char *name;
```

DESCRIPTION

The file whose name is given as the null-terminated string pointed to by name has its mode changed to mode. Modes are constructed by ORing together some combination of the following:

```
4000 set user ID on execution
2000 set group ID on execution
0400 read by owner
0200 write by owner
0100 execute (search on directory) by owner
0070 read, write, execute (search) by group
0007 read, write, execute (search) by others
```

Only the owner of a file may change its mode.

SEE ALSO

chmod (I)

DIAGNOSTIC

Error bit (c-bit) set if name cannot be found or if current user is not the owner of the file. From C, a -1 returned value indicates an error, 0 indicates success.

KSOS

Note that uid and gid do not work as in UNIX. In particular, from the Emulator it is not possible to change the setting of the set user or set group ID bits.

NAME

chown - change owner and group of a file

SYNOPSIS

```
(chmod = 16.)
sys chown; name; owner

chown(name, owner)
char *name;
```

DESCRIPTION

The file whose name is given by the null-terminated string pointed to by name has its owner and group changed to the low and high bytes of owner respectively.

SEE ALSO

hown (VIII), chgrp (VIII), passwd (V)

DIAGNOSTICS

The error bit (c-bit) is set on illegal owner changes. From C a -1 returned value indicates error, 0 indicates success.

KSOS

This system call is subsumed by the NKSR; attempted use under the UNIX Emulator will result in an error.

NAME

close - close a file

SYNOPSIS

```
(close = 6.)
(file descriptor in r0)
sys close

close(fildes)
```

DESCRIPTION

Given a file descriptor such as returned from an open, creat, or pipe call, close closes the associated file. A close of all files is automatic on exit, but since processes are limited to 15 simultaneously open files, close is necessary for programs which deal with many files.

SEE ALSO

creat (II), open (II), pipe (II)

DIAGNOSTICS

The error bit (c-bit) is set for an unknown file descriptor. From C a -1 indicates an error, 0 indicates success.

KSOS

NAME

creat - create a new file

SYNOPSIS

```
(creat = 8.)
sys creat; name; mode
(file descriptor in r0)

creat(name, mode)
char *name;
```

DESCRIPTION

Creat creates a new file or prepares to rewrite an existing file called name, given as the address of a null-terminated string. If the file did not exist, it is given mode mode. See chmod (II) for the construction of the mode argument.

If the file did exist, its mode and owner remain unchanged but it is truncated to 0 length.

The file is also opened for writing, and its file descriptor is returned (in r0).

The mode given is arbitrary; it need not allow writing. This feature is used by programs which deal with temporary files of fixed names. The creation is done with a mode that forbids writing. Then if a second instance of the program attempts a creat, an error is returned and the program knows that the name is unusable for the moment.

SEE ALSO

write (II), close (II), stat (II)

DIAGNOSTICS

The error bit (c-bit) may be set if: a needed directory is not searchable; the file does not exist and the directory in which it is to be created is not writable; the file does exist and is unwritable; the file is a directory; there are already too many files open.

From C, a -1 return indicates an error.

KSOS

NAME

csw - read console switches

SYNOPSIS

```
(csw = 38.; not in assembler)
sys     csw

getcsw( )
```

DESCRIPTION

The setting of the console switches is returned (in r0).

KSOS

This call is not supported by KSOS. Attempted use will result in an error.

NAME

dup - duplicate an open file descriptor

SYNOPSIS

```
(dup = 41.; not in assembler)
(file descriptor in r0)
sys dup

dup(fildes)
int fildes;
```

DESCRIPTION

Given a file descriptor returned from an open, pipe, creat, or port call, dup will allocate another file descriptor synonymous with the original. The new file descriptor is returned in r0.

Dup is used more to reassign the value of file descriptors than to genuinely duplicate a file descriptor. Since the algorithm to allocate file descriptors returns the lowest available value, combinations of dup and close can be used to manipulate file descriptors in a general way. This is handy for manipulating standard input and/or standard output.

SEE ALSO

creat (II), open (II), close (II), pipe (II)

DIAGNOSTICS

The error bit (c-bit) is set if: the given file descriptor is invalid; there are already too many open files. From C, a -1 returned value indicates an error.

KSOS

NAME

exec, execl, execv - execute a file

SYNOPSIS

```
(exec = 11.)
sys exec; name; args
...
name: <...\0>
...
args: arg0; arg1; ...; 0
arg0: <...\0>
arg1: <...\0>
   ...

execl(name, arg0, arg1, ..., argn, 0)
char *name, *arg0, *arg1, ..., *argn;

execv(name, argv)
char *name;
char *argv[ ];
```

DESCRIPTION

Exec overlays the calling process with the named file, then transfers to the beginning of the core image of the file. There can be no return from the file; the calling core image is lost.

Files remain open across exec calls. Ignored signals remain ignored across exec, but signals that are caught are reset to their default values.

Each user has a real user ID and group ID and an effective user ID and group ID. The real ID identifies the person using the system; the effective ID determines his access privileges. Exec changes the effective user and group ID to the owner of the executed file if the file has the ''set-user-ID'' or ''set-group-ID'' modes. The real user ID is not affected.

The form of this call differs somewhat depending on whether it is called from assembly language or C; see below for the C version.

The first argument to exec is a pointer to the name of the file to be executed. The second is the address of a null-terminated list of pointers to arguments to be passed to the file. Conventionally, the first argument is the name of the file. Each pointer adoresses a string terminated by a null byte.

Once the called file starts execution, the arguments are available as follows. The stack pointer points to a word containing the number of arguments. Just above this number is a list of pointers to the argument strings. The arguments are placed as high as possible in core.

```
sp->  nargs
      arg0
      ...
      argn
      -1

arg0: <arg0\0>
      ...
argn: <argn\0>
```

From C, two interfaces are available. execl is useful when a known file with known arguments is being called; the arguments to execl are the character strings constituting the file and the arguments; as in the basic call, the first argument is conventionally the same as the file name (or its last component). A 0 argument must end the argument list.

The execv version is useful when the number of arguments is unknown in advance; the arguments to execv are the name of the file to be executed and a vector of strings containing the arguments. The last argument string must be followed by a 0 pointer.

When a C program is executed, it is called as follows:

```
main(argc, argv)
inc argc;
char **argv;
```

where argc is the argument count and argv is an array of character pointers to the arguments themselves. As indicated, argc is conventionally at least one and the first member of the array points to a string containing the name of the file.

Argv is not directly usable in another execv, since argv[argc] is -1 and not 0.

SEE ALSO

fork (II)

DIAGNOSTICS

If the file cannot be found, if it is not executable, if it does not have a valid header (407, 410, or 411 octal as first word), if maximum memory is exceeded, or if the arguments require more than 512 bytes a return from exec constitutes the diagnostic; the error bit (c-bit) is set. From C the returned value is -1.

BUGS

Only 512 characters of arguments are allowed.

KSOS

Set user id and set group id are not supported by the Emulator. At present shared and separate text files (types 410 and 411) are not supported by the Emulator.

NAME

exit - terminate process

SYNOPSIS

```
(exit = 1.)
(status in r0)
sys exit

exit(status)
int status;
```

DESCRIPTION

Exit is the normal means of terminating a process. Exit closes all the process's files and notifies the parent process if it is executing a wait. The low byte of r0 (resp. the argument to exit) is available as status to the parent process.

This call can never return.

SEE ALSO

wait (II)

DIAGNOSTICS

None.

KSOS

NAME

fork - spawn new process

SYNOPSIS

```
(fork = 2.)
sys fork
(new process return)
(old process return)

fork( )
```

DESCRIPTION

Fork is the only way new processes are created. The new process's core image is a copy of that of the caller of fork. The only distinction is the return location and the fact that r0 in the old (parent) process contains the process ID of the new (child) process. This process ID is used by wait.

The two returning processes share all open files that existed before the call. In particular, this is the way that standard input and output files are passed and also how pipes are set up.

From C, the child process receives a 0 return, and the parent receives a non-zero number which is the process ID of the child; a return of -1 indicates inability to create a new process.

SEE ALSO

wait (II), exec (II), sfork (II)

DIAGNOSTICS

The error bit (c-bit) is set in the old process if a new process could not be created because of lack of process space. From C, a return of -1 (not just negative) indicates an error.

KSOS

NAME

fstat - get status of open file

SYNOPSIS

```
(fstat = 28.)
(file descriptor in r0)
sys fstat; buf

fstat(fildes, buf)
struct inode *buf;
```

DESCRIPTION

This call is identical to stat, except that it operates on open files instead of files given by name. It is most often used to get the status of the standard input and output files, whose names are unknown.

SEE ALSO

stat (II)

DIAGNOSTICS

The error bit (c-bit) is set if the file descriptor is unknown; from C, a -1 return indicates an error, 0 indicates success.

KSOS

Not all fields of the status structure are meaningful in KSOS. The fstat call supplies zeroes in such fields.

NAME

getgid - get group identifications

SYNOPSIS

(getgid = 47.; not in assembler)

sys getgid

getgid( )

DESCRIPTION

Getgid returns a word (in r0), the low byte of which contains the real group ID of the current process. The high byte contains the effective group ID of the current process. The real group ID identifies the group of the person who is logged in, in contradistinction to the effective group ID, which determines his access permission at the moment. It is thus useful to programs which operate using the ''set group ID'' mode, to find out who invoked them.

SEE ALSO

setgid (II)

DIAGNOSTICS

-

KSOS

KSOS group IDs are 16 bits each, but are mapped into 8 bits by using only the low order byte.

NAME

getpid - get process identification

SYNOPSIS

```
(getpid = 20.; not in assembler)
sys getpid
(pid in r0)

getpid( )
```

DESCRIPTION

Getpid returns the process ID of the current process. Most often it is used to generate uniquely-named temporary files.

SEE ALSO

-

DIAGNOSTICS

-

KSOS

Process ids are unique only within an Emulator family.

NAME

getuid - get user identifications

SYNOPSIS

```
(getuid = 24.)
sys getuid

getuid( )
```

DESCRIPTION

Getuid returns a word (in r0), the low byte of which contains the real user ID of the current process. The high byte contains the effective user ID of the current process. The real user ID identifies the person who is logged in, in contradistinction to the effective user ID, which determines his access permission at the moment. It is thus useful to programs which operate using the ''set user ID'' mode, to find out who invoked them.

SEE ALSO

setuid (II)

DIAGNOSTICS

-

KSOS

KSOS user IDs are 16 bits each, but are mapped into 8 bits by using only the low order byte.

NAME

gtty - get terminal status

SYNOPSIS

```
(gtty = 32.)
(file descriptor in r0)
sys gtty; arg
...
arg: .=.+6

gtty(fildes, arg)
int arg[3];
```

DESCRIPTION

Gtty stores in the three words addressed by arg the status of the typewriter whose file descriptor is given in r0 (resp. given as the first argument). The format is the same as that passed by stty.

SEE ALSO

stty (II)

DIAGNOSTICS

Error bit (c-bit) is set if the file descriptor does not refer to a typewriter. From C, a -1 value is returned for an error, 0, for a successful call.

KSOS

As the manipulation of terminal speeds (and parity) is an NKSR function under KSOS, the terminal speed information returned by this call is meaningless.

NAME

indir - indirect system call

SYNOPSIS

(indir = 0.; not in assembler)
sys indir; syscall

DESCRIPTION

The system call at the location syscall is executed. Execution resumes after the indir call.

The main purpose of indir is to allow a program to store arguments in system calls and execute them out of line in the data segment. This preserves the purity of the text segment.

If indir is executed indirectly, it is a no-op. If the instruction at the indirect location is not a system call, the executing process will get a fault.

SEE ALSO

-

DIAGNOSTICS

-

KSOS

INTRODUCTION TO SYSTEM CALLS

Section II of this manual lists all the entries into the KSOS Unix emulator. 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.

In both cases an error number is also available. In assembly language, this number is returned in r0 on erroneous calls. From C, the external variable errno is set to the error number. Errno is not cleared on successful calls, so it should be tested only after an error has occurred. There is a table of messages associated with each error, and a routine for printing the message. See perror (III).

The possible error numbers are not recited with each writeup in section II, since many errors are possible for most of the calls. Here is a list of the error numbers, their names inside the system (for the benefit of system-readers), and the messages available using perror. A short explanation is also provided.

0 - (unused)

1 EPERM Not owner and not super-user
Typically this error indicates an attempt to modify a file in some way forbidden except to its owner. It is also returned for attempts by ordinary users to do things allowed only to the super-user.

2 ENOENT No such file or directory
This error occurs when a file name is specified and the file should exist but doesn't, or when one of the directories in a path name does not exist.

3 ESRCH No such process
The process whose number was given to signal does not exist, or is already dead.

4 EINTR Interrupted system call
An asynchronous signal (such as interrupt or quit), which the user has elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.

5 EIO I/O error
Some physical I/O error occurred during a read or write. This error may in some cases occur on a call following the one to which it actually applies.

6 ENXIO No such device or address
I/O on a special file refers to a subdevice which does not exist, or beyond the limits of the device. It may also occur when, for example, a tape drive is not dialled in or no disk pack is loaded on a drive.

7 E2BIG Arg list too long
An argument list longer than 512 bytes (counting the null at the end of each argument) is presented to exec.

8 ENOEXEC Exec format error
A request is made to execute a file which, although it has the appropriate permissions, does not start with one of the magic numbers 407 or 410.

9 EBADF Bad file number
Either a file descriptor refers to no open file, or a read (resp. write) request is made to a file which is open only for writing (resp. reading).

10 ECHILD No children
Wait and the process has no living or unwaited-for children.

11 EAGAIN No more processes
In a fork, the system's process table is full and no more processes can for the moment be created.

12 ENOMEM Not enough core
During an exec or break, a program asks for more core than the system is able to supply. This is not a temporary condition; the maximum core size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments is such as to require more than the existing 8 segmentation registers.

13 EACCES Permission denied
An attempt was made to access a file in a way forbidden by the protection system.

14 - (unused)

15 ENOTBLK Block device required
A plain file was mentioned where a block device was required, e.g. in mount.

16 EBUSY Mount device busy
An attempt to mount a device that was already mounted or an attempt was made to dismount a device on which there is an open file or some process's current directory.

17 EEXIST File exists
An existing file was mentioned in an inappropriate context, e.g. link.

18 EXDEV Cross-device link
A link to a file on another device was attempted.

19 ENODEV No such device

An attempt was made to apply an inappropriate system call to a device; e.g. read a write-only device.

20 ENOTDIR Not a directory
A non-directory was specified where a directory is required, for example in a path name or as an argument to chdir.

21 EISDIR Is a directory
An attempt to write on a directory.

22 EINVAL Invalid argument
Some invalid argument: currently, dismounting a non-mounted device, mentioning an unknown signal in signal, and giving an unknown request in stty to the TIU special file.

23 ENFILE File table overflow
The system's table of open files is full, and temporarily no more opens can be accepted.

24 EMFILE Too many open files
Only 15 files can be open per process.

25 ENOTTY Not a typewriter
The file mentioned in stty or gtty is not a typewriter or one of the other devices to which these calls apply.

26 ETXTBSY Text file busy
An attempt to execute a pure-procedure program which is currently open for writing (or reading!). Also an attempt to open for writing a pure-procedure program that is being executed.

27 EFBIG File too large
An attempt to make a file larger than the maximum of 32768 blocks.

28 ENOSPC No space left on device
During a write to an ordinary file, there is no free space left on the device.

29 ESPIPE Seek on pipe
A seek was issued to a pipe. This error should also be issued for other non-seekable devices.

30 EROFS Read-only file system
An attempt to modify a file or directory was made on a device mounted read-only.

31 EMLINK Too many links
An attempt to make more than 127 links to a file.

32 EPIPE Write on broken pipe
A write on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.

33 EMTTY Too many open TTYs in the family
An attempt to open too many terminals within the same emulator process family. This error is a KSOS UNIX Emulator addition.

100 ENOSYS Nonexistent system call
This error indicates an attempt to make a nonexistent system call.

KSOS
Due to the differing internals of UNIX and the KSOS UNIX Emulator, the error codes returned from failed Emulator calls are frequently only approximations to the corresponding UNIX error codes.

NAME

kill - send signal to a process

SYNOPSIS

```
(kill = 37.; not in assembler)
(process number in r0)
sys kill; sig

kill(pid, sig);
```

DESCRIPTION

Kill sends the signal sig to the process specified by the process number in r0. See signal (II) for a list of signals.

The sending and receiving processes must have the same effective user ID.

If the process number is 0, the signal is sent to all other processes which have the same controlling typewriter and user ID.

In no case is it possible for a process to kill itself.

SEE ALSO

signal (II), kill (I)

DIAGNOSTICS

The error bit (c-bit) is set if the process does not have the same effective user ID, or if the process does not exist. From C, -1 is returned.

KSOS

Signals have effect only within the Emulator family of the sender.

NAME

link - link to a file

SYNOPSIS

```
(link = 9.)
sys link; name1; name2

link(name1, name2)
char *name1, *name2;
```

DESCRIPTION

A link to name1 is created; the link has the name name2. Either name may be an arbitrary path name.

SEE ALSO

link (I), unlink (II)

DIAGNOSTICS

The error bit (c-bit) is set when name1 cannot be found; when name2 already exists; when the directory of name2 cannot be written; when an attempt is made to link to a directory; when an attempt is made to link to a file on another file system; when more than 127 links are made. From C, a -1 return indicates an error, a 0 return indicates success.

KSOS

NAME

mknod - make a directory or a special file

SYNOPSIS

(mknod = 14.; not in assembler)

```
sys  mknod; name; mode; addr

mknod(name, mode, addr)
char *name;
```

DESCRIPTION

Mknod creates a new file whose name is the null-terminated string pointed to by name. The mode of the new file (including directory and special file bits) is initialized from mode. The first physical address of the file is initialized from addr. Note that in the case of a directory, addr should be zero. In the case of a special file, addr specifies which special file.

SEE ALSO

mkdir (I), mknod (VIII), fs (V)

DIAGNOSTICS

Error bit (c-bit) is set if the file already exists. From C, a -1 value indicates an error.

KSOS

Manipulation of special files is an NKSR function; hence, in the UNIX Emulator, this call can only be used to make directories.

NAME

mount - mount file system

SYNOPSIS

(mount = 21.)
sys mount; special; name; rwflag

mount(special, name, rwflag)
char *special, *name;

DESCRIPTION

Mount announces to the system that a removable file system has been mounted on the block-structured special file special; from now on, references to file name will refer to the root file on the newly mounted file system. Special and name are pointers to null-terminated strings containing the appropriate path names.

Name must exist already. Its old contents are inaccessible while the file system is mounted.

The rwflag argument determines whether the file system can be written on; if it is 0 writing is allowed, if non-zero no writing is done. Physically write-protected and magnetic tape file systems must be mounted read-only or errors will occur when access times are updated, whether or not any explicit write is attempted.

SEE ALSO

mount (VIII), umount (II)

DIAGNOSTICS

Error bit (c-bit) set if: special is inaccessible or not an appropriate file; name does not exist; special is already mounted; name is in use; there are already too many file systems mounted.

BUGS

-

KSOS

this call is subsumed by the NKSR. Attempted use under the UNIX Emulator will result in an error.

NAME

nice - set advisory program priority

SYNOPSIS

(nice = 34.)
(priority in r0)
sys nice

nice(priority)

DESCRIPTION

The scheduling priority of the process is changed to the argument. Positive priorities get less service than normal; 0 is default. The valid range of priority is 20 to -220. The value of 4 is recommended to users who wish to execute long-running programs without flak from the administration.

The effect of this call is passed to a child process by the fork system call. The effect can be cancelled by another call to nice with a priority of 0.

The actual running priority of a process is the priority argument plus a number that ranges from 100 to 119 depending on the cpu usage of the process.

SEE ALSO

nice (I)

DIAGNOSTICS

The error bit (c-bit) is set if the user requests a priority outside the range of 0 to 20 and is not the super-user.

KSOS

This call is not yet implemented. The description of priorities given above is not accurate for KSOS.

NAME

open - open for reading or writing

SYNOPSIS

```
(open = 5.)
sys open; name; mode
(file descriptor in r0)

open(name, mode)
char *name;
```

DESCRIPTION

Open opens the file name for reading (if mode is 0), writing (if mode is 1) or for both reading and writing (if mode is 2). Name is the address of a string of ASCII characters representing a path name, terminated by a null character.

The returned file descriptor should be saved for subsequent calls to read, write, and close.

SEE ALSO

creat (II), read (II), write (II), close (II)

DIAGNOSTICS

The error bit (c-bit) is set if the file does not exist, if one of the necessary directories does not exist or is unreadable, if the file is not readable (resp. writable), or if too many files are open. From C, a -1 value is returned on an error.

KSOS

NAME

pipe - create an interprocess channel

SYNOPSIS

```
(pipe = 42.)
sys pipe
(read file descriptor in r0)
(write file descriptor in r1)

pipe(fildes)
int fildes[2];
```

DESCRIPTION

The pipe system call creates an I/O mechanism called a pipe. The file descriptors returned can be used in read and write operations. When the pipe is written using the descriptor returned in r1 (resp. fildes[1]), up to 4096 bytes of data are buffered before the writing process is suspended. A read using the descriptor returned in r0 (resp. fildes[0]) will pick up the data.

It is assumed that after the pipe has been set up, two (or more) cooperating processes (created by subsequent fork calls) will pass data through the pipe with read and write calls.

The Shell has a syntax to set up a linear array of processes connected by pipes.

Read calls on an empty pipe (no buffered data) with only one end (all write file descriptors closed) return an end-of-file. Write calls under similar conditions generate a fatal signal (signal (II)); if the signal is ignored, an error is returned on the write.

SEE ALSO

sh (I), read (II), write (II), fork (II)

DIAGNOSTICS

The error bit (c-bit) is set if too many files are already open. From C, a -1 returned value indicates an error. A signal is generated if a write on a pipe with only one end is attempted.

BUGS

KSOS

This call is not yet available.

NAME

profil - execution time profile

SYNOPSIS

(profil = 44.; not in assembler)
sys profil; buff; bufsiz; offset; scale

profil(buff, bufsiz, offset, scale)
char buff[ ];
int bufsiz, offset, scale;

DESCRIPTION

Buff points to an area of core whose length (in bytes) is given by bufsiz. After this call, the user's program counter (pc) is examined each clock tick (60th second); offset is subtracted from it, and the result multiplied by scale. If the resulting number corresponds to a word inside buff, that word is incremented.

The scale is interpreted as an unsigned, fixed-point fraction with binary point at the left: 177777(8) gives a 1-1 mapping of pc's to words in buff; 77777(8) maps each pair of instruction words together. 2(8) maps all instructions onto the beginning of buff (producing a non-interrupting core clock).

Profiling is turned off by giving a scale of 0 or 1. It is rendered ineffective by giving a bufsiz of 0. Profiling is also turned off when an exec is executed but remains on in child and parent both after a fork.

SEE ALSO

monitor (III), prof (I)

DIAGNOSTICS

-

KSOS

This call is not yet implemented.

NAME

ptrace - process trace

SYNOPSIS

(ptrace = 26.; not in assembler)
(data in r0)
sys ptrace; pid; addr; request
(value in r0)

ptrace(request, pid, addr, data);

DESCRIPTION

Ptrace provides a means by which a parent process may control the execution of a child process, and examine and change its core image. Its primary use is for the implementation of breakpoint debugging, but it should be adaptable for simulation of non-UNIX environments. There are four arguments whose interpretation depends on a request argument. Generally, pid is the process ID of the traced process, which must be a child (no more distant descendant) of the tracing process. A process being traced behaves normally until it encounters some signal whether internally generated like ``illegal instruction'' or externally generated like ``interrupt.'' See signal (II) for the list. Then the traced process enters a stopped state and its parent is notified via wait (II). When the child is in the stopped state, its core image can be examined and modified using ptrace. If desired, another ptrace request can then cause the child either to terminate or to continue, possibly ignoring the signal.

The value of the request argument determines the precise action of the call:

0 This request is the only one used by the child process; it declares that the process is to be traced by its parent. All the other arguments are ignored. Peculiar results will ensue if the parent does not expect to trace the child.

1,2 The word in the child process's address space at addr is returned (in r0). Request 1 indicates the data space (normally used); 2 indicates the instruction space (when I and D space are separated). addr must be even. The child must be stopped. The input data is ignored.

3 The word of the system's per-process data area corresponding to addr is returned. Addr must be even and less than 512. This space contains the registers and other information about the process; its layout corresponds to the user structure in the system.

4,5 The given data is written at the word in the process's address space corresponding to addr, which must be even. No useful value is returned. Request 4 specifies data space (normally used), 5 specifies instruction space. Attempts to write in pure procedure result in termination of the child, instead of going through or causing an error for the parent.

6 The process's system data is written, as it is read with request 3. Only a few locations can be written in this way: the general registers, the floating point status and registers, and certain bits of the processor status word.

7 The data argument is taken as a signal number and the child's execution continues as if it had incurred that signal. Normally the signal number will be either 0 to indicate that the signal which caused the stop should be ignored, or that value fetched out of the process's image indicating which signal caused the stop.

8 The traced process terminates.

As indicated, these calls (except for request 0) can be used only when the subject process has stopped. The wait call is used to determine when a process stops; in such a case the ''termination'' status returned by wait has the value 0177 to indicate stoppage rather than genuine termination.

To forestall possible fraud, ptrace inhibits the set-user-id facility on subsequent exec (II) calls.

SEE ALSO

wait (II), signal (II), cdb (I)

DIAGNOSTICS

From assembler, the c-bit (error bit) is set on errors; from C, -1 is returned and errno has the error code.

BUGS

The request 0 call should be able to specify signals which are to be treated normally and not cause a stop. In this way, for example, programs with simulated floating point (which use ''illegal instruction'' signals at a very high rate) could be efficiently debugged.

Also, it should be possible to stop a process on occurrence of a system call; in this way a completely controlled environment could be provided.

KSOS

This call is not yet implemented.

NAME

read - read from file

SYNOPSIS

```
(read = 3.)
(file descriptor in r0)
sys read; buffer; nbytes

read(fildes, buffer, nbytes)
char *buffer;
```

DESCRIPTION

A file descriptor is a word returned from a successful open, creat, dup, pipe, or port call. Buffer is the location of nbytes contiguous bytes into which the input will be placed. It is not guaranteed that all nbytes bytes will be read; for example if the file refers to a typewriter at most one line will be returned. In any event the number of characters read is returned (in r0).

If the returned value is 0, then end-of-file has been reached.

SEE ALSO

open (II), creat (II), dup (II), pipe (II)

DIAGNOSTICS

As mentioned, 0 is returned when the end of the file has been reached. If the read was otherwise unsuccessful the error bit (c-bit) is set. Many conditions can generate an error: physical I/O errors, bad buffer address, preposterous nbytes, file descriptor not that of an input file. From C, a -1 return indicates the error.

KSOS

NAME

seek - move read/write pointer

SYNOPSIS

(seek = 19.)
(file descriptor in r0)
sys seek; offset; ptrname

seek(fildes, offset, ptrname)

DESCRIPTION

The file descriptor refers to a file open for reading or writing. The read (resp. write) pointer for the file is set as follows:

if ptrname is 0, the pointer is set to offset.

if ptrname is 1, the pointer is set to its current location plus offset.

if ptrname is 2, the pointer is set to the size of the file plus offset.

if ptrname is 3, 4 or 5, the meaning is as above for 0, 1 and 2 except that the offset is multiplied by 512.

If ptrname is 0 or 3, offset is unsigned, otherwise it is signed.

SEE ALSO

open (II), creat (II)

DIAGNOSTICS

The error bit (c-bit) is set for an undefined file descriptor. From C, a -1 return indicates an error.

KSOS

NAME

setgid - set process group ID

SYNOPSIS

(setgid = 46.; not in assembler)
(group ID in r0)
sys setgid

setgid(gid)

DESCRIPTION

The group ID of the current process is set to the argument. Both the effective and the real group ID are set. This call is only permitted if the argument is the real group ID.

SEE ALSO

getgid (II)

DIAGNOSTICS

Error bit (c-bit) is set as indicated; from C, a -1 value is returned.

KSOS

This call is not yet implemented.

NAME

setuid - set process user ID

SYNOPSIS

(setuid = 23.)
(user ID in r0)
sys setuid

setuid(uid)

DESCRIPTION

The user ID of the current process is set to the argument. Both the effective and the real user ID are set. This call is only permitted if the argument is the real user ID.

SEE ALSO

getuid (II)

DIAGNOSTICS

Error bit (c-bit) is set as indicated; from C, a -1 value is returned.

KSOS

This call is not yet implemented.

NAME

signal - catch or ignore signals

SYNOPSIS

```
(signal = 48.)
sys  signal; sig; label
(old value in r0)

signal(sig, func)
int (*func)( );
```

DESCRIPTION

A signal is generated by some abnormal event, initiated either by user at a typewriter (quit, interrupt), by a program error (bus error, etc.), or by request of another program (kill). Normally all signals cause termination of the receiving process, but this call allows them either to be ignored or to cause an interrupt to a specified location. Here is the list of signals:

1	hangup
2	interrupt
3*	quit
4*	illegal instruction (not reset when caught)
5*	trace trap (not reset when caught)
6*	IOT instruction
7*	EMT instruction
8*	floating point exception
9	kill (cannot be caught or ignored)
10*	bus error
11*	segmentation violation
12*	bad argument to system call
13	write on a pipe with no one to read it

In the assembler call, if label is 0, the process is terminated when the signal occurs; this is the default action. If label is odd, the signal is ignored. Any other even label specifies an address in the process where an interrupt is simulated. An RTI or RTT instruction will return from the interrupt. Except as indicated, a signal is reset to 0 after being caught. Thus if it is desired to catch every such signal, the catching routine must issue another signal call.

In C, if func is 0, the default action for signal sig (termination) is reinstated. If func is 1, the signal is ignored. If func is non-zero and even, it is assumed to be the address of a function entry point. When the signal occurs, the function will be called. A return from the function will continue the process at the point it was interrupted. As in the assembler call, signal must in general be called again to catch subsequent signals.

When a caught signal occurs during certain system calls, the call terminates prematurely. In particular this can occur during a read or write on a slow device (like a typewriter; but not a file); and during sleep or

wait. When such a signal occurs, the saved user status is arranged in such a way that when return from the signal-catching takes place, it will appear that the system call returned a characteristic error status. The user's program may then, if it wishes, re-execute the call.

The starred signals in the list above cause a core image if not caught or ignored.

The value of the call is the old action defined for the signal.

After a fork (II) the child inherits all signals. Exec (II) resets all caught signals to default action.

SEE ALSO
kill (I), kill (II), ptrace (II), reset (III)

DIAGNOSTICS
The error bit (c-bit) is set if the given signal is out of range. In C, a -1 indicates an error; 0 indicates success.

BUGS

KSOS

NAME

sleep - stop execution for interval

SYNOPSIS

```
(sleep = 35.; not in assembler)
(seconds in r0)
sys sleep

sleep(seconds)
```

DESCRIPTION

The current process is suspended from execution for the number of seconds specified by the argument.

SEE ALSO

sleep (I)

DIAGNOSTICS

-

KSOS

NAME

stat - get file status

SYNOPSIS

(stat = 18.)

```
sys stat; name; buf

stat(name, buf)
char *name;
struct inode *buf;
```

DESCRIPTION

Name points to a null-terminated string naming a file; buf is the address of a 36(10) byte buffer into which information is placed concerning the file. It is unnecessary to have any permissions at all with respect to the file, but all directories leading to the file must be readable. After stat, buf has the following structure (starting offset given in bytes):

```
struct inode {
   char  minor;          /* +0: minor device of i-node */
   char  major;          /* +1: major device */
   int   inumber;        /* +2 */
   int   flags;          /* +4: see below */
   char  nlinks;         /* +6: number of links to file */
   char  uid;            /* +7: user ID of owner */
   char  gid;            /* +8: group ID of owner */
   char  size0;          /* +9: high byte of 24-bit size */
   int   size1;          /* +10: low word of 24-bit size */
   int   addr[8];        /* +12: block numbers or device number */
   int   actime[2];      /* +28: time of last access */
   int   modtime[2];     /* +32: time of last modification */
};
```

The flags are as follows:

```
100000   i-node is allocated
060000   2-bit file type:
     000000   plain file
     040000   directory
     020000   character-type special file
     060000   block-type special file.
010000   large file
004000   set user-ID on execution
002000   set group-ID on execution
001000   save text image after execution
000400   read (owner)
000200   write (owner)
000100   execute (owner)
000070   read, write, execute (group)
000007   read, write, execute (others)
```

SEE ALSO
ls (I), fstat (II), fs (V)

DIAGNOSTICS
Error bit (c-bit) is set if the file cannot be found. From C, a -1 return indicates an error.

KSOS
Not all fields are meaningful under KSOS. The stat call supplies zeroes in such fields, which are: nlinks, addr, and, actime.

NAME

stime - set time

SYNOPSIS

```
(stime = 25.)
(time in r0-r1)
sys stime

stime(tbuf)
int tbuf[2];
```

DESCRIPTION

Stime sets the system's idea of the time and date. Time is measured in seconds from 0000 GMT Jan 1 1970. Only the super-user may use this call.

SEE ALSO

date (I), time (II), ctime (III)

DIAGNOSTICS

Error bit (c-bit) set if user is not the super-user.

KSOS

This call has been subsumed by the NKSR. Attempted use under the UNIX Emulator will result in an error.

NAME

stty - set mode of typewriter

SYNOPSIS

```
(stty = 31.)
(file descriptor in r0)
sys stty; arg
...
arg:  .byte ispeed, ospeed; .byte erase, kill; mode

stty(fildes, arg)
struct {
        char    ispeed, ospeed;
        char    erase, kill;
        int     mode;
} *arg;
```

DESCRIPTION

Stty sets mode bits and character speeds for the typewriter whose file descriptor is passed in r0 (resp. is the first argument to the call). First, the system delays until the typewriter is quiescent. The input and output speeds are set from the first two bytes of the argument structure as indicated by the following table, which corresponds to the speeds supported by the DH-11 interface. If DC-11, DL-11 or KL-11 interfaces are used, impossible speed changes are ignored.

0	(hang up dataphone)
1	50 baud
2	75 baud
3	110 baud
4	134.5 baud
5	150 baud
6	200 baud
7	300 baud
8	600 baud
9	1200 baud
10	1800 baud
11	2400 baud
12	4800 baud
13	9600 baud
14	External A
15	External B

In the current configuration, only 110, 150 and 300 baud are really supported on dial-up lines, in that the code conversion and line control required for IBM 2741's (134.5 baud) must be implemented by the user's program, and the half-duplex line discipline required for the 202 dataset (1200 baud) is not supplied.

The next two characters of the argument structure specify the erase and kill characters respectively. (Defaults are # and @.)

The mode contains several bits which determine the system's treatment of the typewriter:

```
100000 Select one of two algorithms for backspace delays
040000 Select one of two algorithms for form-feed and vertical-tab
       delays
030000 Select one of four algorithms for carriage-return delays
006000 Select one of four algorithms for tab delays
001400 Select one of four algorithms for new-line delays
000200 even parity allowed on input (e. g. for M37s)
000100 odd parity allowed on input
000040 raw mode: wake up on all characters
000020 map CR into LF; echo LF or CR as CR-LF
000010 echo (full duplex)
000004 map upper case to lower on input (e. g. M33)
000002 echo and print tabs as spaces
000001 hang up (remove 'data terminal ready,' lead CD) after last
       close
```

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay.

Backspace delays are currently ignored but will be used for Terminet 300's.

If a form-feed/vertical tab delay is specified, it lasts for about 2 seconds.

Carriage-return delay type 1 lasts about .08 seconds and is suitable for the Terminet 300. Delay type 2 lasts about .16 seconds and is suitable for the VT05 and the TI 700. Delay type 3 is unimplemented and is 0.

New-line delay type 1 is dependent on the current column and is tuned for Teletype model 37's. Type 2 is useful for the VT05 and is about .10 seconds. Type 3 is unimplemented and is 0.

Tab delay type 1 is dependent on the amount of movement and is tuned to the Teletype model 37. Other types are unimplemented and are 0.

Characters with the wrong parity, as determined by bits 200 and 100, are ignored.

In raw mode, every character is passed immediately to the program without waiting until a full line has been typed. No erase or kill processing is done; the end-of-file character (EOT), the interrupt character (DEL) and the quit character (FS) are not treated specially.

Mode 020 causes input carriage returns to be turned into new-lines; input of either CR or LF causes LF-CR both to be echoed (used for GE TermiNet 300's and other terminals without the newline function).

The hangup mode 01 causes the line to be disconnected when the last process with the line open closes it or terminates. It is useful when a port is to be used for some special purpose; for example, if it is associated with an ACU used to place outgoing calls.

This system call is also used with certain special files other than typewriters, but since none of them are part of the standard system the specifications will not be given.

SEE ALSO

stty (I), gtty (II)

DIAGNOSTICS

The error bit (c-bit) is set if the file descriptor does not refer to a typewriter. From C, a negative value indicates an error.

KSOS

It is not possible to set speeds or parity from the UNIX Emulator. In no case are more than two delay algorithms available. Hangup mode is not supported.

NAME

sync - update super-block

SYNOPSIS

(sync = 36.; not in assembler)
sys sync

DESCRIPTION

Sync causes all information in core memory that should be on disk to be written out. This includes modified super blocks, modified i-nodes, and delayed block I/O.

It should be used by programs which examine a file system, for example icheck, df, etc. It is mandatory before a boot.

SEE ALSO

sync (VIII), update (VIII)

DIAGNOSTICS

-

KSOS

This call causes the Emulator Family buffer cache to be flushed.

NAME

time - get date and time

SYNOPSIS

```
(time = 13.)
sys  time

time(tvec)
int tvec[2];
```

DESCRIPTION

Time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds. From as, the high order word is in the r0 register and the low order is in r1. From C, the user-supplied vector is filled in.

SEE ALSO

date (I), stime (II), ctime (III)

DIAGNOSTICS

-

KSOS

NAME

times - get process times

SYNOPSIS

```
(times = 43.; not in assembler)
sys  times; buffer

times(buffer)
struct tbuffer *buffer;
```

DESCRIPTION

Times returns time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/60 seconds.

After the call, the buffer will appear as follows:

```
struct tbuffer {
        int     proc_user_time;
        int     proc_system_time;
        int     child_user_time[2];
        int     child_system_time[2];
};
```

The children times are the sum of the children's process times and their children's times.

SEE ALSO

time (I)

DIAGNOSTICS

-

BUGS

The process times should be 32 bits as well.

KSOS

NAME

umount - dismount file system

SYNOPSIS

(umount = 22.)
sys umount; special

DESCRIPTION

Umount announces to the system that special file special is no longer to contain a removable file system. The file associated with the special file reverts to its ordinary interpretation; see mount (II).

SEE ALSO

umount (VIII), mount (II)

DIAGNOSTICS

Error bit (c-bit) set if no file system was mounted on the special file or if there are still active files on the mounted file system.

KSOS

This call has been subsumed by the NKSR. Attempted use from the UNIX Emulator will result in an error.

NAME

unlink - remove directory entry

SYNOPSIS

```
(unlink = 10.)
sys  unlink; name

unlink(name)
char *name;
```

DESCRIPTION

Name points to a null-terminated string. Unlink removes the entry for the file pointed to by name from its directory. If this entry was the last link to the file, the contents of the file are freed and the file is destroyed. If, however, the file was open in any process, the actual destruction is delayed until it is closed, even though the directory entry has disappeared.

SEE ALSO

rm (I), rmdir (I), link (II)

DIAGNOSTICS

The error bit (c-bit) is set to indicate that the file does not exist or that its directory cannot be written. Write permission is not required on the file itself. From C, a -1 return indicates an error.

KSOS

NAME

wait - wait for process to terminate

SYNOPSIS

```
(wait = 7.)
sys  wait
(process ID in r0)
(status in r1)

wait(status)
int *status;
```

DESCRIPTION

Wait causes its caller to delay until one of its child processes terminates. If any child has died since the last wait, return is immediate; if there are no children, return is immediate with the error bit set (resp. with a value of -1 returned). The normal return yields the process ID of the terminated child (in r0). In the case of several children several wait calls are needed to learn of all the deaths.

If no error is indicated on return, the r1 high byte (resp. the high byte stored into status) contains the low byte of the child process r0 (resp. the argument of exit) when it terminated. The r1 (resp. status) low byte contains the termination status of the process. See signal (II) for a list of termination statuses (signals); 0 status indicates normal termination. A special status (0177) is returned for a stopped process which has not terminated and can be restarted. See ptrace (II). If the 0200 bit of the termination status is set, a core image of the process was produced by the system.

SEE ALSO

exit (II), fork (II), signal (II)

DIAGNOSTICS

The error bit (c-bit) is set if there are no children not previously waited for. From C, a returned value of -1 indicates an error.

KSOS

NAME

write - write on a file

SYNOPSIS

```
(write = 4.)
(file descriptor in r0)
sys  write; buffer; nbytes

write(fildes, buffer, nbytes)
char *buffer;
```

DESCRIPTION

A file descriptor is a word returned from a successful open, creat, dup, pipe, or port call.

Buffer is the address of nbytes contiguous bytes which are written on the output file. The number of characters actually written is returned (in r0). It should be regarded as an error if this is not the same as requested.

Writes which are multiples of 512 characters long and begin on a 512-byte boundary in the file are more efficient than any others.

SEE ALSO

creat (II), open (II), pipe (II), eofp (II)

DIAGNOSTICS

The error bit (c-bit) is set on an error: bad descriptor, buffer address, or count; physical I/O errors. From C, a returned value of -1 indicates an error.

KSOS

ILMED
8