NAME

e_syscall - error generating versions of system call routines

SYNOPSIS

```
#include <errfct.h>

int  e_access(name, mode, inhflag)
     char *name;
     int mode;
int  e_acct(name, inhflag)
     char *name;
int  e_attach(sub, gp, inhflag)
     int sub;
     int gp;
int  e_block(sema, inhflag)
     int sema;
int  e_brk(addr, inhflag)
     char *addr;
int  e_chan(gr, inhflag)
     int gr;
int  e_chdir(dirname, inhflag)
     char *dirname;
int  e_chmod(name, mode, inhflag)
     char *name;
     int mode;
int  e_chown(name, owner, group, inhflag)
     char *name;
     int owner;
     int group;
int  e_chroot(dirname, inhflag)
     char *dirname;
int  e_close(fildes, name, inhflag)
     int fildes;
int  e_connect(fd, ch, side, inhflag)
     int fd;
     int ch;
     int side;
int  e_creat(name, mode, inhflag)
     char *name;
     int mode;
int  e_csignal(index, gp, sig, inhflag)
     int index;
     int gp;
     int sig;
int  e_detach(sub, gp, inhflag)
     int sub;
     int gp;
int  e_dismaus(vaddr, inhflag)
     char *vaddr;
int  e_dup(fildes, name, inhflag)
     int fildes;
     char *name;
char * e_enabmaus(mausdes, inhflag)
```

```
        int mausdes;
    int e_errlog(flag, inhflag)  /* SC5 only */
        int flag;
    int e_execl(name, arg0, arg1 , ... 0, inhflag)
        char *name;
        char *arg0;
    int e_execle(name, arg0, arg1, ... 0, envp, inhflag)
        char *name;
        char *arg0;
        char *envp[];
    int e_execlp(name, arg0, arg1, ... 0, inhflag)
        char *name;
        char *arg0;
    int e_execv(name, argv, inhflag)
        char *name;
        char *argv[];
    int e_execve(name, argv, envp, inhflag)
        char *name;
        char *argv[];
        char *envp[];
    int e_execvp(name, argv, inhflag)
        char *name;
        char *argv[];
    int e_extract(sub, ch, side, inhflag)
        int sub;
        int ch;
        int side;
    int e_fcntl( fildes, request, argument, name, inhflag)
        int fildes;
        int request;
        int argument;
        char *name;
    int e_fork( inhflag)
    int e_freemaus(mausdes, inhflag)
        int mausdes;
    int e_fstat(fildes, buf, name, inhflag)
        int fildes;
        struct stat *buf;
        char *name;
    int e_getmaus(name, mode, inhflag)
        char *name;
        int mode;
    int e_gtty(fildes, arg, name, inhflag)      /* SC5 only*/
        int fildes;
        struct SGBUF *arg;
        char *name;
    int e_ioctl(fildes, request, argp, name, inhflag)
        char *fildes;
        int request;
        struct sgttyb *argp;
        char *name;
    int e_join(fd, xd, inhflag)
        int fd;
```

```
        int xd;
int     e_kill(pid, sig, inhflag)
        int pid;
        int sig;
int     e_link(name1, name2, inhflag)
        char *name1;
        char *name2;
int     e_lock(sema, inhflag)
        int sema;
long    e_lseek( fildes, offset, whence, name, inhflag)
        int fildes;
        long offset;
        int whence;
        char *name;
int     e_mknod(name, mode, addr, inhflag)
        char *name;
        int mode;
        int addr;
int     e_mount(special, name, rwflag, inhflag)
        char *special;
        char *name;
        int rwflag;
int     e_mpx(name, mode, inhflag)
        char *name;
        int mode;
int     e_msgdisab( inhflag)
int     e_msgenab( inhflag)
int     e_nice(priority, inhflag)
        int priority;
int     e_open(name, mode, inhflag)
        char *name;
        int mode;
int     e_p(sema, inhflag)
        int sema;
int     e_pause( inhflag)
int     e_pipe(fildes, inhflag)
        int *fildes;
int     e_post(sema, inhflag)
        int sema;
int     e_ptrace(request, pid, addr, data, inhflag)
        int request;
        int pid;
        int addr;
        int data;
int     e_rdsem(sema, inhflag)
        int sema;
int     e_read(fildes, buffer, nbytes, name, inhflag)
        int fildes;
        char *buffer;
        int nbytes;
        char *name;
int     e_recv(buf, size, type, inhflag)
        char *buf;
```

```
        int size;
        char *type;
int     e_recvw(buf, size, type, inhflag)
        char *buf;
        int size;
        char *type;
char *  e_sbrk(incr, inhflag)
        int *incr;
int     e_send(buf, size, topid, type, inhflag)
        char *buf;
        int size;
        int topid;
        int type;
int     e_sendw(buf, size, topid, type, inhflag)
        char *buf;
        int size;
        int topid;
        int type;
int     e_setgid(gid, inhflag)
        int gid;
int     e_setpgrp(pid, inhflag)
        int pid;
int     e_setsem(sema, value, inhflag)
        int sema;
        int value;
int     e_setuid(uid, inhflag)
        int uid;
int     (*e_signal(sig, func, inhflag) ) ()
        int sig;
        int (*func)();
int     e_stat(name, buf, inhflag)
        char *name;
        struct stat *buf;
int     e_stime(tp, inhflag)
        long *tp;
int     e_stty(fildes, arg, name, inhflag)
        int fildes;
        struct SGBUF *arg;
        char *name;
char *  e_switmaus(mausdes, vaddr, inhflag)
        int mausdes;
        char *vaddr;
int     e_sync( inhflag)
long    e_tell(fildes, name, inhflag)
        int fildes;
        char *name;
int     e_test(sema, inhflag)
        int sema;
long    e_time(tloc, inhflag)
        long *tloc;
long    e_times(buffer, inhflag)
        struct tbuffer *buffer;
int     e_tlock(sema, inhflag)
```

```
        int sema;
daddr_t  e_ulimit(newlimit, inhflag)
        daddr_t newlimit;
int     e_umask(mask, inhflag)
        int mask;
int     e_umount(special, inhflag)
        char *special;
int     e_unlink(name, inhflag)
        char *name;
int     e_unlock(sema, inhflag)
        int sema;
int     e_utime(file, times, inhflag)
        char *file;
        struct utimbuf times;
int     e_v(sema, inhflag)
        int sema;
int     e_wait(wait, status, inhflag)
        int wait;
        int *status;
int     e_write(fildes, buffer, nbytes, name, inhflag)
        int fildes;
        char *buffer;
        int nbytes;
        char *name;
int     e_xread(fildes, buffer, nbytes, name, inhflag)
        int fildes;
        char *buffer;
        int nbytes;
        char *name;
```

DESCRIPTION

Each routine calls the corresponding system call routine in Section 2, and returns the same value returned by the system call. If an error is detected:

1) If the error is overload related (eg, inability to open a file because the inode table is full) and if the program has previously called e_setrep (or e_setup(3L) with appropriate arguments - see e_setup(3L)), then the system call is repeated for the time specified in the e_setrep call or until the error clears.

2) If another type of error is detected or if the overload error is not resolved, then an SCCS output message (OM) is formatted and stored away.

3) If an OM is created and if the "error class" (as determined by the errno variable - see Intro(2)) is not "inhibited", then the OM is outputted by means of sccerr or glberr. The inhibiting is controlled by the "inhflag" argument. Its value may be any of the following defines (<errfct.h>) or the logical ORing of two or more of them:

Define	Error Class Inhibited	Errno
INHPERM	Permission (su or owner)	EPERM
INHNOENT	File not found	ENOENT
INHINTR	System call interrupted	EINTR
INHNXIO	Non existing special file	ENXIO
INHNOEXEL	No execute permission	ENOEXEC
INHAGAIN	Process table overflow	EAGAIN
INHACCES	File access permission	EACCESS
INHNOTDIR	File should be a directory	ENOTDIR
INHNFILE	File or inode table overflow	ENFILE
INHFBIG	File too big	EFBIG
INHZBIG	Argument list too big	EZBIG
INHNOSPC	Out of disk space	ENOSPC
NOERR	Inhibit for all	-

If no inhibiting is desired, use **ALLERR** for the inhibit flag.

Numerous options are available - see e_setup(3L).

The routine e_setname(3L) or e_setup(3L) must be called prior to these routines to set up the program name field of the OM.

4) OM's stored away may be modified and then output - see e_new(3L) and e_output(3L).

Note that several routines have an extra "name" argument. This should be a pointer to the name of the file being operated upon or zero if the name is not known.

LIBRARY

/lib/lib1.a

SEE ALSO

e_stdio(3L) e_setup(3L) e_new(3L) e_output(3L) intro(2)

DIAGNOSTICS

Same as corresponding routines in Section 2.