```
/* main */

/* program to create and bind a socket */
/* last modified %G% version %I% */


#define CF_DEBUGS       0               /* non-switchable debug print-outs */
#define CF_DEBUG        0               /* switchable at invocation */
#define CF_DEBUGMALL    1               /* debug memory-allocations */


/* revision history:

        = 1989-03-01, David A›D› Morano
        This subroutine was originally written.

        = 1998-06-01, David A›D› Morano
        I enhanced the program a little.

        = 2013-03-01, David A›D› Morano
        I added logging of requests to a file. This would seem to be an
        appropriate security precaution.

*/

/* Copyright ' 1989,1998,2013 David A›D› Morano.  All rights reserved. */

/*************************************************************************

        Synopsis:

        $ openport

        All transactions are done on STDIN.


*************************************************************************/


#include        <envstandards.h>        /* MUST be first to configure */

#include        <sys/types.h>
#include        <sys/param.h>
#include        <sys/socket.h>
#include        <limits.h>
#include        <stropts.h>
#include        <unistd.h>
#include        <fcntl.h>
#include        <time.h>
#include        <stdlib.h>
#include        <string.h>
#include        <pwd.h>
#include        <grp.h>
#include        <netdb.h>

#include        <vsystem.h>
#include        <getbufsize.h>
#include        <bits.h>
#include        <keyopt.h>
#include        <bfile.h>
#include        <vecpstr.h>
#include        <userinfo.h>
#include        <msgbuf.h>
#include        <getax.h>
#include        <sockaddress.h>
#include        <exitcodes.h>
```

```
#include        <localmisc.h>

#include        "config.h"
#include        "defs.h"
#include        "userports.h"
#include        "openport.h"
#include        "proglog.h"


/* local defines */

#define MBUFLEN         MSGBUFLEN

#define INETADDRSTRLEN  ((INETXADDRLEN*2)+6)

#define NDF             "/tmp/openport.deb"


/* external subroutines */

extern int      snsds(char *,int,cchar *,cchar *) ;
extern int      sncpy3(char *,int,const char *,const char *,const char *) ;
extern int      mkpath2(char *,const char *,const char *) ;
extern int      mkpath3(char *,const char *,const char *,const char *) ;
extern int      sninetaddr(char *,int,uint,const char *) ;
extern int      sfshrink(const char *,int,char **) ;
extern int      sfskipwhite(cchar *,int,cchar **) ;
extern int      matstr(const char **,const char *,int) ;
extern int      matostr(const char **,int,const char *,int) ;
extern int      cfdeci(const char *,int,int *) ;
extern int      optbool(const char *,int) ;
extern int      optvalue(const char *,int) ;
extern int      getportnum(const char *,const char *) ;
extern int      getuid_user(cchar *,int) ;
extern int      vecpstr_adduniq(VECPSTR *,const char *,int) ;
extern int      hasalldig(const char *,int) ;
extern int      isdigitlatin(int) ;
extern int      isNotPresent(int) ;
extern int      isFailOpen(int) ;

extern int      printhelp(void *,cchar *,cchar *,cchar *) ;
extern int      proginfo_setpiv(PROGINFO *,cchar *,const PIVARS *) ;

extern int      proguserlist_begin(PROGINFO *) ;
extern int      proguserlist_end(PROGINFO *) ;

#if     CF_DEBUGS || CF_DEBUG
extern int      debugopen(const char *) ;
extern int      debugprintf(const char *,...) ;
extern int      debugclose() ;
#endif

extern cchar    *getourenv(cchar **,cchar *) ;

extern char     *strdcpy1w(char *,int,const char *,int) ;
extern char     *strnchr(const char *,int,int) ;


/* external variables */


/* local structures */

struct query {
        const char      *uidp ;
        const char      *protop ;
```

```
        const char      *portp ;
        int             uidl ;
        int             protol ;
        int             portl ;
} ;

struct prototupple {
        int             pf ;
        int             ptype ;
        int             proto ;
        const char      *name ;
} ;


/* forward references */

static int      usage(PROGINFO *) ;

static int      procuserinfo_begin(PROGINFO *,USERINFO *) ;
static int      procuserinfo_end(PROGINFO *) ;

static int      procopts(PROGINFO *,KEYOPT *) ;
static int      procargs(PROGINFO *,ARGINFO *,BITS *,int,
                        cchar *,cchar *,cchar *) ;
static int      process(PROGINFO *,cchar *,cchar *,VECPSTR *,int) ;
static int      procbind(PROGINFO *,USERPORTS *,int) ;
static int      proclist(PROGINFO *,USERPORTS *,const char *,VECPSTR *) ;
static int      proclistall(PROGINFO *,USERPORTS *,bfile *) ;
static int      proclistusers(PROGINFO *,USERPORTS *,bfile *,VECPSTR *) ;
static int      proclistquery(PROGINFO *,USERPORTS *,bfile *,VECPSTR *) ;

static int      parsequery(struct query *,const char *,int) ;
static int      getdefport(const char *,const char *,int) ;
static int      openbind(int,int,int,struct sockaddr *,int) ;
static int      getprotoname(int,int,int,const char **) ;


/* local variables */

static volatile int     if_exit ;
static volatile int     if_intr ;

static const char       *argopts[] = {
        "ROOT",
        "VERSION",
        "VERBOSE",
        "HELP",
        "sn",
        "af",
        "ef",
        "of",
        "if",
        "lf",
        "db",
        "query",
        NULL
} ;

enum argopts {
        argopt_root,
        argopt_version,
        argopt_verbose,
        argopt_help,
        argopt_sn,
        argopt_af,
        argopt_ef,
```

```
        argopt_of,
        argopt_if,
        argopt_lf,
        argopt_db,
        argopt_query,
        argopt_overlast
} ;

static const PIVARS     initvars = {
        VARPROGRAMROOT1,
        VARPROGRAMROOT2,
        VARPROGRAMROOT3,
        PROGRAMROOT,
        VARPRNAME
} ;

static const MAPEX      mapexs[] = {
        { SR_NOENT, EX_NOUSER },
        { SR_PERM, EX_NOPERM },
        { SR_AGAIN, EX_TEMPFAIL },
        { SR_DEADLK, EX_TEMPFAIL },
        { SR_NOLCK, EX_TEMPFAIL },
        { SR_TXTBSY, EX_TEMPFAIL },
        { SR_ACCESS, EX_NOPERM },
        { SR_REMOTE, EX_PROTOCOL },
        { SR_NOSPC, EX_TEMPFAIL },
        { SR_INTR, EX_INTR },
        { SR_EXIT, EX_TERM },
        { 0, 0 }
} ;

static const char       *progopts[] = {
        "binder",
        NULL
} ;

enum progopts {
        progopt_binder,
        progopt_overlast
} ;

static const char       *modes[] = {
        "query",
        "binder",
        NULL
} ;

static const struct prototupple socknames[] = {
        { PF_INET, SOCK_STREAM, IPPROTO_TCP, "tcp" },
        { PF_INET, SOCK_STREAM, 0, "tcp" },
        { PF_INET, SOCK_DGRAM, IPPROTO_UDP, "udp" },
        { PF_INET, SOCK_DGRAM, 0, "udp" },
#ifdef  PF_INET6
        { PF_INET6, SOCK_STREAM, IPPROTO_TCP, "tcp6" },
        { PF_INET6, SOCK_STREAM, 0, "tcp6" },
        { PF_INET6, SOCK_DGRAM, IPPROTO_UDP, "udp6" },
        { PF_INET6, SOCK_DGRAM, 0, "udp6" },
#endif /* PF_INET6 */
        { 0, 0, NULL }
} ;

static const char       *defprotos[] = {
        "tcp",
        "udp",
        "ddp",
        NULL
```

```
} ;


/* exported subroutines */


int main(int argc,cchar **argv,cchar **envv)
{
        PROGINFO        pi, *pip = &pi ;
        ARGINFO         ainfo ;
        BITS            pargs ;
        KEYOPT          akopts ;
        bfile           errfile ;

#if     (CF_DEBUGS || CF_DEBUG) && CF_DEBUGMALL
        uint            mo_start = 0 ;
#endif

        int             argr, argl, aol, akl, avl, kwi ;
        int             ai, ai_max, ai_pos ;
        int             rs, rs1 ;
        int             cfd = FD_STDIN ;
        int             ex = EX_INFO ;
        int             f_optminus, f_optplus, f_optequal ;
        int             f_version = FALSE ;
        int             f_usage = FALSE ;
        int             f_help = FALSE ;
        int             f_inopen = FALSE ;

        const char      *argp, *aop, *akp, *avp ;
        const char      *argval = NULL ;
        const char      *pr = NULL ;
        const char      *sn = NULL ;
        const char      *afname = NULL ;
        const char      *efname = NULL ;
        const char      *ofname = NULL ;
        const char      *ifname = NULL ;
        const char      *dbfname = NULL ;
        const char      *cp ;


        if_exit = 0 ;
        if_intr = 0 ;

#if     CF_DEBUGS || CF_DEBUG
        if ((cp = getourenv(envv,VARDEBUGFNAME)) != NULL) {
            rs = debugopen(cp) ;
            debugprintf("main: starting DFD=%d\n",rs) ;
        }
#endif /* CF_DEBUGS */

#if     (CF_DEBUGS || CF_DEBUG) && CF_DEBUGMALL
        uc_mallset(1) ;
        uc_mallout(&mo_start) ;
#endif

        rs = proginfo_start(pip,envv,argv[0],VERSION) ;
        if (rs < 0) {
            ex = EX_OSERR ;
            goto badprogstart ;
        }

        if ((cp = getenv(VARBANNER)) == NULL) cp = BANNER ;
        rs = proginfo_setbanner(pip,cp) ;

/* initialize */
```

```
        pip->verboselevel = 1 ;
        pip->f.logprog = TRUE ;

/* start parsing the arguments */

        if (rs >= 0) rs = bits_start(&pargs,1) ;
        if (rs < 0) goto badpargs ;

        rs = keyopt_start(&akopts) ;
        pip->open.akopts = (rs >= 0) ;

        ai_max = 0 ;
        ai_pos = 0 ;
        argr = argc ;
        for (ai = 0 ; (ai < argc) && (argv[ai] != NULL) ; ai += 1) {
            if (rs < 0) break ;
            argr -= 1 ;
            if (ai == 0) continue ;

            argp = argv[ai] ;
            argl = strlen(argp) ;

            f_optminus = (*argp == '-') ;
            f_optplus = (*argp == '+') ;
            if ((argl > 1) && (f_optminus || f_optplus)) {
                const int       ach = MKCHAR(argp[1]) ;

                if (isdigitlatin(ach)) {

                    argval = (argp + 1) ;

                } else if (ach == '-') {

                    ai_pos = ai ;
                    break ;

                } else {

                    aop = argp + 1 ;
                    akp = aop ;
                    aol = argl - 1 ;
                    f_optequal = FALSE ;
                    if ((avp = strchr(aop,'=')) != NULL) {
                        f_optequal = TRUE ;
                        akl = avp - aop ;
                        avp += 1 ;
                        avl = aop + argl - 1 - avp ;
                        aol = akl ;
                    } else {
                        avp = NULL ;
                        avl = 0 ;
                        akl = aol ;
                    }

                    if ((kwi = matostr(argopts,2,akp,akl)) >= 0) {

                        switch (kwi) {

/* program root */
                        case argopt_root:
                            if (f_optequal) {
                                f_optequal = FALSE ;
                                if (avl)
                                    pr = avp ;
                            } else {
```

```
                            if (argr > 0) {
                                argp = argv[++ai] ;
                                argr -= 1 ;
                                argl = strlen(argp) ;
                                if (argl)
                                    pr = argp ;
                            } else
                                rs = SR_INVALID ;
                        }
                        break ;

/* version */
                    case argopt_version:
                        f_version = TRUE ;
                        if (f_optequal)
                            rs = SR_INVALID ;
                        break ;

/* verbose mode */
                    case argopt_verbose:
                        pip->verboselevel = 2 ;
                        if (f_optequal) {
                            f_optequal = FALSE ;
                            if (avl) {
                                rs = optvalue(avp,avl) ;
                                pip->verboselevel = rs ;
                            }
                        }
                        break ;

                    case argopt_help:
                        f_help = TRUE ;
                        break ;

/* program search-name */
                    case argopt_sn:
                        if (f_optequal) {
                            f_optequal = FALSE ;
                            if (avl)
                                sn = avp ;
                        } else {
                            if (argr > 0) {
                                argp = argv[++ai] ;
                                argr -= 1 ;
                                argl = strlen(argp) ;
                                if (argl)
                                    sn = argp ;
                            } else
                                rs = SR_INVALID ;
                        }
                        break ;

/* argument-list file */
                    case argopt_af:
                        if (f_optequal) {
                            f_optequal = FALSE ;
                            if (avl)
                                afname = avp ;
                        } else {
                            if (argr > 0) {
                                argp = argv[++ai] ;
                                argr -= 1 ;
                                argl = strlen(argp) ;
                                if (argl)
                                    afname = argp ;
                            } else
```

```
                            rs = SR_INVALID ;
                        }
                        break ;

/* error file name */
                    case argopt_ef:
                        if (f_optequal) {
                            f_optequal = FALSE ;
                            if (avl)
                                efname = avp ;
                        } else {
                            if (argr > 0) {
                                argp = argv[++ai] ;
                                argr -= 1 ;
                                argl = strlen(argp) ;
                                if (argl)
                                    efname = argp ;
                            } else
                                rs = SR_INVALID ;
                        }
                        break ;

/* output file name */
                    case argopt_of:
                        if (f_optequal) {
                            f_optequal = FALSE ;
                            if (avl)
                                ofname = avp ;
                        } else {
                            if (argr > 0) {
                                argp = argv[++ai] ;
                                argr -= 1 ;
                                argl = strlen(argp) ;
                                if (argl)
                                    ofname = argp ;
                            } else
                                rs = SR_INVALID ;
                        }
                        break ;

/* input file name */
                    case argopt_if:
                        if (f_optequal) {
                            f_optequal = FALSE ;
                            if (avl)
                                ifname = avp ;
                        } else {
                            if (argr > 0) {
                                argp = argv[++ai] ;
                                argr -= 1 ;
                                argl = strlen(argp) ;
                                if (argl)
                                    ifname = argp ;
                            } else
                                rs = SR_INVALID ;
                        }
                        break ;

/* log filename */
                    case argopt_lf:
                        if (f_optequal) {
                            f_optequal = FALSE ;
                            if (avl)
                                pip->lfname = avp ;
                        } else {
                            if (argr > 0) {
```

```
                                argp = argv[++ai] ;
                                argr -= 1 ;
                                argl = strlen(argp) ;
                                if (argl)
                                    pip->lfname = argp ;
                            } else
                                rs = SR_INVALID ;
                        }
                        break ;

/* data-base filename */
                    case argopt_db:
                        if (f_optequal) {
                            f_optequal = FALSE ;
                            if (avl)
                                dbfname = avp ;
                        } else {
                            if (argr > 0) {
                                argp = argv[++ai] ;
                                argr -= 1 ;
                                argl = strlen(argp) ;
                                if (argl)
                                    dbfname = argp ;
                            } else
                                rs = SR_INVALID ;
                        }
                        break ;

/* query mode */
                    case argopt_query:
                        pip->f.query = TRUE ;
                        if (f_optequal) {
                            f_optequal = FALSE ;
                            if (avl) {
                                rs = optbool(avp,avl) ;
                                pip->f.query = (rs > 0) ;
                            }
                        }
                        break ;

/* handle all keyword defaults */
                    default:
                        rs = SR_INVALID ;
                        break ;

                    } /* end switch */

                } else {

                    while (akl--) {
                        const int   kc = (*akp & 0xff) ;

                        switch (kc) {

/* debug */
                        case 'D':
                            pip->debuglevel = 1 ;
                            if (f_optequal) {
                                f_optequal = FALSE ;
                                if (avl) {
                                    rs = optvalue(avp,avl) ;
                                    pip->debuglevel = rs ;
                                }
                            }
                            break ;
```

```
/* quiet mode */
                        case 'Q':
                            pip->f.quiet = TRUE ;
                            break ;

/* version */
                        case 'V':
                            f_version = TRUE ;
                            break ;

/* all mode */
                        case 'a':
                            pip->f.all = TRUE ;
                            if (f_optequal) {
                                f_optequal = FALSE ;
                                if (avl) {
                                    rs = optbool(avp,avl) ;
                                    pip->f.all = (rs > 0) ;
                                }
                            }
                            break ;

/* binder mode */
                        case 'b':
                            pip->f.binder = TRUE ;
                            if (f_optequal) {
                                f_optequal = FALSE ;
                                if (avl) {
                                    rs = optbool(avp,avl) ;
                                    pip->f.binder = (rs > 0) ;
                                }
                            }
                            break ;

/* options */
                        case 'o':
                            if (argr > 0) {
                                argp = argv[++ai] ;
                                argr -= 1 ;
                                argl = strlen(argp) ;
                                if (argl) {
                                    KEYOPT  *kop = &akopts ;
                                    rs = keyopt_loads(kop,argp,argl) ;
                                }
                            } else
                                rs = SR_INVALID ;
                            break ;

                        case 'q':
                            pip->verboselevel = 0 ;
                            break ;

/* verbose mode */
                        case 'v':
                            pip->verboselevel = 2 ;
                            if (f_optequal) {
                                f_optequal = FALSE ;
                                if (avl) {
                                    rs = optvalue(avp,avl) ;
                                    pip->verboselevel = rs ;
                                }
                            }
                            break ;

                        case '?':
                            f_usage = TRUE ;
```

```
                                break ;

                            default:
                                rs = SR_INVALID ;
                                break ;

                            } /* end switch */
                            akp += 1 ;

                            if (rs < 0) break ;
                        } /* end while */

                    } /* end if (individual option key letters) */

                } /* end if (digits as argument or not) */

            } else {

                rs = bits_set(&pargs,ai) ;
                ai_max = ai ;

            } /* end if (key letter/word or positional) */

            ai_pos = ai ;

        } /* end while (all command line argument processing) */

        if (efname == NULL) efname = getenv(VAREFNAME) ;
        if (efname == NULL) efname = getenv(VARERRORFNAME) ;
        if (efname == NULL) efname = BFILE_STDERR ;
        if ((rs1 = bopen(&errfile,efname,"wca",0666)) >= 0) {
            pip->efp = &errfile ;
            pip->open.errfile = TRUE ;
            bcontrol(&errfile,BC_SETBUFLINE,TRUE) ;
        } else if (! isFailOpen(rs1)) {
            if (rs >= 0) rs = rs1 ;
        }

        if ((rs >= 0) && (pip->debuglevel == 0)) {
            if ((cp = getenv(VARDEBUGLEVEL)) != NULL) {
                rs = optvalue(cp,-1) ;
                pip->debuglevel = rs ;
            }
        }

        if (rs < 0)
            goto badarg ;

#if     CF_DEBUG
        if (DEBUGLEVEL(2))
            debugprintf("main: debuglevel=%u\n",pip->debuglevel) ;
#endif

        if (f_version) {
            bprintf(pip->efp,"%s: version %s\n",pip->progname,VERSION) ;
        }

/* get some program information */

        if (rs >= 0) {
            if ((rs = proginfo_setpiv(pip,pr,&initvars)) >= 0) {
                rs = proginfo_setsearchname(pip,VARSEARCHNAME,sn) ;
            }
        }

        if (rs < 0) {
```

```
            ex = EX_OSERR ;
            goto retearly ;
        }

        if (pip->debuglevel > 0) {
            bprintf(pip->efp,"%s: pr=%s\n", pip->progname,pip->pr) ;
            bprintf(pip->efp,"%s: sn=%s\n", pip->progname,pip->searchname) ;
        } /* end if */

        if (f_usage)
            usage(pip) ;

/* help file */

        if (f_help)
            printhelp(NULL,pip->pr,pip->searchname,HELPFNAME) ;

        if (f_version || f_help || f_usage)
            goto retearly ;


        ex = EX_OK ;

/* some initialization */

        if ((rs >= 0) && (pip->n == 0) && (argval != NULL)) {
            rs = optvalue(argval,-1) ;
            pip->n = rs ;
        }

        if (afname == NULL) afname = getenv(VARAFNAME) ;

        if (pip->lfname == NULL) pip->lfname = getenv(VARLFNAME) ;

        if (pip->tmpdname == NULL) pip->tmpdname = getenv(VARTMPDNAME) ;
        if (pip->tmpdname == NULL) pip->tmpdname = TMPDNAME ;

        if (pip->logsize == 0) pip->logsize = LOGSIZE ;

        rs = procopts(pip,&akopts) ;

        if (pip->f.binder) {
            dbfname = NULL ;
        } else {
            if (dbfname == NULL) dbfname = getenv(VARDBFNAME) ;
        }
        if (dbfname == NULL) dbfname = USERPORTSFNAME ;

        if ((rs >= 0) && (pip->debuglevel > 0)) {
            cchar       *pn = pip->progname ;
            cchar       *fmt ;
            cchar       *ms ;
            fmt = "%s: mode=%s\n" ;
            ms = (pip->f.binder) ? modes[1] : modes[0] ;
            bprintf(pip->efp,fmt,pn,ms) ;
            fmt = "%s: db=%s\n" ;
            bprintf(pip->efp,fmt,pn,dbfname) ;
        }

        if ((rs >= 0) && (ifname != NULL) && (ifname[0] != '\0')) {
            if (ifname[0] != '-') {
                f_inopen = TRUE ;
                rs = uc_open(ifname,O_RDWR,0777) ;
                cfd = rs ;
            }
        }
```

```
/* OK, we finally do our thing */

        memset(&ainfo,0,sizeof(ARGINFO)) ;
        ainfo.argc = argc ;
        ainfo.ai = ai ;
        ainfo.argv = argv ;
        ainfo.ai_max = ai_max ;
        ainfo.ai_pos = ai_pos ;

        if (rs >= 0) {
            USERINFO    u ;
            if ((rs = userinfo_start(&u,NULL)) >= 0) {
                if ((rs = procuserinfo_begin(pip,&u)) >= 0) {
                    if ((rs = proglog_begin(pip,&u)) >= 0) {
                        if ((rs = proguserlist_begin(pip)) >= 0) {
                            {
                                ARGINFO         *aip = &ainfo ;
                                const char      *dfn = dbfname ;
                                const char      *afn = afname ;
                                const char      *ofn = ofname ;
                                rs = procargs(pip,aip,&pargs,cfd,dfn,ofn,afn) ;
                            }
                            rs1 = proguserlist_end(pip) ;
                            if (rs >= 0) rs = rs1 ;
                        } /* end if (proguserlist) */
                        rs1 = proglog_end(pip) ;
                        if (rs >= 0) rs = rs1 ;
                    } /* end if (proglog) */
                    rs1 = procuserinfo_end(pip) ;
                    if (rs >= 0) rs = rs1 ;
                } /* end if (procuserinfo) */
                rs1 = userinfo_finish(&u) ;
                if (rs >= 0) rs = rs1 ;
            } else {
                cchar   *pn = pip->progname ;
                cchar   *fmt ;
                ex = EX_NOUSER ;
                fmt = "%s: userinfo failure (%d)\n" ;
                bprintf(pip->efp,fmt,pn,rs) ;
            } /* end if (userinfo) */
        } else {
            cchar       *pn = pip->progname ;
            cchar       *fmt = "%s: invalid argument or configuration (%d)\n" ;
            ex = EX_USAGE ;
            bprintf(pip->efp,fmt,pn,rs) ;
            usage(pip) ;
        } /* end if (ok) */

        if (f_inopen && (cfd >= 0)) {
            f_inopen = FALSE ;
            u_close(cfd) ;
        }

#if     CF_DEBUG
        if (DEBUGLEVEL(2))
            debugprintf("main: finishing rs=%d\n",rs) ;
#endif

/* done */
        if ((rs < 0) && (ex == EX_OK)) {
            switch (rs) {
            case SR_INVALID:
                ex = EX_USAGE ;
                if (! pip->f.quiet) {
                    bprintf(pip->efp,"%s: invalid query (%d)\n",
```

```
                        pip->progname,rs) ;
                }
                break ;
            case SR_NOENT:
                ex = EX_CANTCREAT ;
                break ;
            case SR_AGAIN:
                ex = EX_TEMPFAIL ;
                break ;
            default:
                ex = mapex(mapexs,rs) ;
                break ;
            } /* end switch */
        } else if (if_exit) {
            ex = EX_TERM ;
        } else if (if_intr)
            ex = EX_INTR ;

retearly:
        if (pip->debuglevel > 0) {
            bprintf(pip->efp,"%s: exiting ex=%u (%d)\n",
                pip->progname,ex,rs) ;
        }

#if     CF_DEBUG
        if (DEBUGLEVEL(2))
            debugprintf("main: exiting ex=%u (%d)\n",ex,rs) ;
#endif

        if (pip->efp != NULL) {
            pip->open.errfile = FALSE ;
            bclose(pip->efp) ;
            pip->efp = NULL ;
        }

        if (pip->open.akopts) {
            pip->open.akopts = FALSE ;
            keyopt_finish(&akopts) ;
        }

        bits_finish(&pargs) ;

badpargs:
        proginfo_finish(pip) ;

badprogstart:

#if     (CF_DEBUGS || CF_DEBUG) && CF_DEBUGMALL
        {
            uint        mo ;
            uc_mallout(&mo) ;
            debugprintf("main: final mallout=%u\n",(mo-mo_start)) ;
            uc_mallset(0) ;
        }
#endif

#if     (CF_DEBUGS || CF_DEBUG)
        debugclose() ;
#endif

        return ex ;

/* the bad things */
badarg:
        ex = EX_USAGE ;
        bprintf(pip->efp,"%s: invalid argument specified (%d)\n",
```

```
            pip->progname,rs) ;
        usage(pip) ;
        goto retearly ;

}
/* end subroutine (main) */


/* local subroutines */


static int usage(PROGINFO *pip)
{
        int             rs = SR_OK ;
        int             wlen = 0 ;
        const char      *pn = pip->progname ;
        const char      *fmt ;

        fmt = "%s: USAGE> %s [{ -b | -query [<query>] | -a | <user(s)> }]\n" ;
        if (rs >= 0) rs = bprintf(pip->efp,fmt,pn,pn) ;
        wlen += rs ;

        fmt = "%s:  [-af <afile>]\n" ;
        if (rs >= 0) rs = bprintf(pip->efp,fmt,pn) ;
        wlen += rs ;

        fmt = "%s:  [-Q] [-D] [-v[=<n>]] [-HELP] [-V]\n" ;
        if (rs >= 0) rs = bprintf(pip->efp,fmt,pn) ;
        wlen += rs ;

        return (rs >= 0) ? wlen : rs ;
}
/* end subroutine (usage) */


static int procopts(PROGINFO *pip,KEYOPT *kop)
{
        int             rs = SR_OK ;
        int             c = 0 ;
        cchar           *cp ;

        if ((cp = getourenv(pip->envv,VAROPTS)) != NULL) {
            rs = keyopt_loads(kop,cp,-1) ;
        }

        if (rs >= 0) {
            KEYOPT_CUR  kcur ;
            if ((rs = keyopt_curbegin(kop,&kcur)) >= 0) {
                int     oi ;
                int     kl, vl ;
                cchar   *kp, *vp ;

                while ((kl = keyopt_enumkeys(kop,&kcur,&kp)) >= 0) {

                    if ((oi = matostr(progopts,3,kp,kl)) >= 0) {

                        vl = keyopt_fetch(kop,kp,NULL,&vp) ;

                        switch (oi) {

                        case progopt_binder:
                            pip->f.binder = TRUE ;
                            if (vl > 0) {
                                rs = optbool(vp,vl) ;
                                pip->f.binder = (rs > 0) ;
                            }
```

```
                            break ;

                        } /* end switch */

                        c += 1 ;
                    } else
                        rs = SR_INVALID ;

                    if (rs < 0) break ;
                } /* end while (looping through key options) */

                keyopt_curend(kop,&kcur) ;
            } /* end if (keyopt-cur) */
        } /* end if (ok) */

        return (rs >= 0) ? c : rs ;
}
/* end subroutine (procopts) */


static int procuserinfo_begin(PROGINFO *pip,USERINFO *uip)
{
        int             rs = SR_OK ;

        pip->nodename = uip->nodename ;
        pip->domainname = uip->domainname ;
        pip->username = uip->username ;
        pip->gecosname = uip->gecosname ;
        pip->realname = uip->realname ;
        pip->name = uip->name ;
        pip->fullname = uip->fullname ;
        pip->mailname = uip->mailname ;
        pip->org = uip->organization ;
        pip->logid = uip->logid ;
        pip->pid = uip->pid ;
        pip->uid = uip->uid ;
        pip->euid = uip->euid ;
        pip->gid = uip->gid ;
        pip->egid = uip->egid ;

        if (rs >= 0) {
            const int   hlen = MAXHOSTNAMELEN ;
            char        hbuf[MAXHOSTNAMELEN+1] ;
            const char  *nn = pip->nodename ;
            const char  *dn = pip->domainname ;
            if ((rs = snsds(hbuf,hlen,nn,dn)) >= 0) {
                const char      **vpp = &pip->hostname ;
                rs = proginfo_setentry(pip,vpp,hbuf,rs) ;
            }
        }

        return rs ;
}
/* end subroutine (procuserinfo_begin) */


static int procuserinfo_end(PROGINFO *pip)
{
        int             rs = SR_OK ;

        if (pip == NULL) return SR_FAULT ;

        return rs ;
}
/* end subroutine (procuserinfo_end) */
```

```
static int procargs(PROGINFO *pip,ARGINFO *aip,BITS *bop,int cfd,
                cchar *dfn,cchar *ofn,cchar *afn)
{
        VECPSTR         al ;
        int             rs ;
        int             rs1 ;
        int             c = 0 ;

        if ((rs = vecpstr_start(&al,4,0,0)) >= 0) {
            int         pan = 0 ;
            int         cl ;
            cchar       *pn = pip->progname ;
            cchar       *fmt ;
            cchar       *cp ;

            if (rs >= 0) {
                int     ai ;
                int     f ;
                cchar   **argv = aip->argv ;
                for (ai = 1 ; ai < aip->argc ; ai += 1) {

                    f = (ai <= aip->ai_max) && (bits_test(bop,ai) > 0) ;
                    f = f || ((ai > aip->ai_pos) && (argv[ai] != NULL)) ;
                    if (f) {
                        cp = argv[ai] ;
                        if (cp[0] != '\0') {
                            pan += 1 ;
                            rs = vecpstr_adduniq(&al,cp,-1) ;
                            if (rs < INT_MAX) c += 1 ;
                        }
                    }

                    if ((rs >= 0) && if_exit) rs = SR_EXIT ;
                    if ((rs >= 0) && if_intr) rs = SR_INTR ;
                    if (rs < 0) break ;
                } /* end for (handling positional arguments) */
            } /* end if (ok) */

            if ((rs >= 0) && (afn != NULL) && (afn[0] != '\0')) {
                bfile   afile, *afp = &afile ;

                if (strcmp(afn,"-") == 0)
                    afn = BFILE_STDIN ;

                if ((rs = bopen(afp,afn,"r",0666)) >= 0) {
                    const int   llen = LINEBUFLEN ;
                    char        lbuf[LINEBUFLEN + 1] ;

                    while ((rs = breadline(afp,lbuf,llen)) > 0) {
                        int     len = rs ;

                        if (lbuf[len - 1] == '\n') len -= 1 ;
                        lbuf[len] = '\0' ;

                        if ((cl = sfskipwhite(lbuf,len,&cp)) > 0) {
                            if (cp[0] != '#') {
                                pan += 1 ;
                                rs = vecpstr_adduniq(&al,cp,cl) ;
                                if (rs < INT_MAX) c += 1 ;
                            }
                        }

                        if ((rs >= 0) && if_exit) rs = SR_EXIT ;
                        if ((rs >= 0) && if_intr) rs = SR_INTR ;
                        if (rs < 0) break ;
```

```
                    } /* end while (reading lines) */

                    rs1 = bclose(afp) ;
                    if (rs >= 0) rs = rs1 ;
                } else {
                    if (! pip->f.quiet) {
                        fmt = "%s: inaccessible argument-list (%d)\n" ;
                        bprintf(pip->efp,fmt,pn,rs) ;
                        bprintf(pip->efp,"%s: afile=%s\n",pn,afn) ;
                    }
                } /* end if */

            } /* end if (processing file argument file list) */

/* OK, we're good to go */

#if     CF_DEBUG
            if (DEBUGLEVEL(2)) {
                debugprintf("main: f_bind=%u\n",pip->f.binder) ;
                debugprintf("main: f_all=%u\n",pip->f.all) ;
                debugprintf("main: f_query=%u\n",pip->f.query) ;
            }
#endif

            if (rs >= 0) {
                rs = process(pip,dfn,ofn,&al,cfd) ;
            }

#if     CF_DEBUG
            if (DEBUGLEVEL(2))
                debugprintf("main: process() rs=%d\n",rs) ;
#endif

            rs1 = vecpstr_finish(&al) ;
            if (rs >= 0) rs = rs1 ;
        } /* end if (vecpstr) */

        proglog_printf(pip,"done (%d)",((rs>=0)?c:rs)) ;

        return (rs >= 0) ? c : rs ;
}
/* end subroutine (procargs) */


static int process(PROGINFO *pip,cchar *dbfn,cchar *ofn,VECPSTR *alp,int cfd)
{
        USERPORTS       db ;
        int             rs ;
        int             rs1 ;

        {
            cchar       *ms = (pip->f.binder) ? modes[1] : modes[0] ;
            proglog_printf(pip,"mode=%s",ms) ;
            proglog_printf(pip,"db=%s",dbfn) ;
        }

        if ((rs = userports_open(&db,dbfn)) >= 0) {
            if (pip->f.binder) {
                rs = procbind(pip,&db,cfd) ;
            } else {
                rs = proclist(pip,&db,ofn,alp) ;
            }
            rs1 = userports_close(&db) ;
            if (rs >= 0) rs = rs1 ;
        } /* end if (userports) */
```

```
        return rs ;
}
/* end subroutine (process) */


static int proclist(PROGINFO *pip,USERPORTS *dbp,cchar *ofn,VECPSTR *alp)
{
        bfile           ofile, *ofp = &ofile ;
        int             rs ;
        int             rs1 ;

        if ((ofn == NULL) || (ofn[0] == '\0') || (ofn[0] == '-'))
            ofn = BFILE_STDOUT ;

        if ((rs = bopen(ofp,ofn,"wct",0666)) >= 0) {

            if (pip->f.all) {
                rs = proclistall(pip,dbp,ofp) ;
            } else if (pip->f.query) {
                rs = proclistquery(pip,dbp,ofp,alp) ;
            } else {
                rs = proclistusers(pip,dbp,ofp,alp) ;
            } /* end if */

            rs1 = bclose(ofp) ;
            if (rs >= 0) rs = rs1 ;
        } /* end if (file-output) */

        return rs ;
}
/* end subroutine (proclist) */


static int proclistall(PROGINFO *pip,USERPORTS *dbp,bfile *ofp)
{
        USERPORTS_CUR   cur ;
        USERPORTS_ENT   ent ;
        int             rs ;
        int             rs1 ;

        if (pip == NULL) return SR_FAULT ;

        if ((rs = userports_curbegin(dbp,&cur)) >= 0) {
            cchar       *fmt = "%10u %16s %16s\n" ;

            while ((rs1 = userports_enum(dbp,&cur,&ent)) >= 0) {

#if     CF_DEBUG
                if (DEBUGLEVEL(3))
                    debugprintf("main/proclistall: %10u %16s %16s\n",
                        ent.uid,ent.protocol,ent.portname) ;
#endif

                rs = bprintf(ofp,fmt,ent.uid,ent.protocol,ent.portname) ;

                if (rs < 0) break ;
            } /* end while */
            if ((rs >= 0) && (rs1 != SR_NOTFOUND)) rs = rs1 ;

            rs1 = userports_curend(dbp,&cur) ;
            if (rs >= 0) rs = rs1 ;
        } /* end if (userports) */

        return rs ;
}
/* end subroutine (proclistall) */
```

```
static int proclistusers(PROGINFO *pip,USERPORTS *dbp,bfile *ofp,VECPSTR *alp)
{
        USERPORTS_CUR   cur ;
        USERPORTS_ENT   ent ;
        uid_t           uid ;
        int             rs ;
        int             rs1 ;
        int             i ;
        const char      *up ;

        if (pip == NULL) return SR_FAULT ;

        for (i = 0 ; vecpstr_get(alp,i,&up) >= 0 ; i += 1) {
            if (up == NULL) continue ;

            if ((rs = getuid_user(up,-1)) >= 0) {
                uid = rs ;
                if ((rs = userports_curbegin(dbp,&cur)) >= 0) {
                    cchar       *fmt = "%10u %16s %16s\n" ;

                    while ((rs1 = userports_fetch(dbp,&cur,uid,&ent)) >= 0) {
                        {
                            rs = bprintf(ofp,fmt,
                                ent.uid,ent.protocol,ent.portname) ;
                        }
                        if (rs < 0) break ;
                    } /* end while */
                    if ((rs >= 0) && (rs1 != SR_NOTFOUND)) rs = rs1 ;

                    userports_curend(dbp,&cur) ;
                } /* end if (cursor) */
            } else if (rs == SR_INVALID) {
                rs = SR_OK ;
            }

            if (rs < 0) break ;
        } /* end for (queries) */

        return rs ;
}
/* end subroutine (proclistusers) */


/* ARGSUSED */
static int proclistquery(PROGINFO *pip,USERPORTS *dbp,bfile *ofp,VECPSTR *alp)
{
        struct passwd   pw ;
        struct query    q ;
        const int       pwlen = getbufsize(getbufsize_pw) ;
        int             rs ;
        int             pl = 0 ;
        int             c = 0 ;
        char            *pwbuf ;

#if     CF_DEBUG
        debugprintf("main/proclistquery: ent\n") ;
#endif

        if ((rs = uc_malloc((pwlen+1),&pwbuf)) >= 0) {
            int         ulen = USERNAMELEN ;
            uid_t       uid ;
            int         i ;
            const char  *up ;
            char        ubuf[USERNAMELEN+1] ;
```

```
            for (i = 0 ; vecpstr_get(alp,i,&up) >= 0 ; i += 1) {
                if (up == NULL) continue ;

#if     CF_DEBUG
                debugprintf("main/proclistquery: q=>%s<\n",up) ;
#endif

                pl = parsequery(&q,up,-1) ;

#if     CF_DEBUG
                debugprintf("main/proclistquery: parsequery() pl=%d\n",pl) ;
                debugprintf("main/proclistquery: uid=%t\n",q.uidp,q.uidl) ;
                debugprintf("main/proclistquery: proto=%t\n",
                        q.protop,q.protol) ;
                debugprintf("main/proclistquery: port=%t\n",q.portp,q.portl) ;
#endif

                if (pl == 0) continue ;

                if (q.uidl > 0) {
                    strdcpy1w(ubuf,ulen,q.uidp,q.uidl) ;
                    rs = getpw_name(&pw,pwbuf,pwlen,ubuf) ;
                    uid = pw.pw_uid ;
#if     CF_DEBUG
                    debugprintf("main/proclistquery: getpw_name() rs=%d\n",rs) ;
#endif
                    if ((rs == SR_NOTFOUND) && hasalldig(ubuf,-1)) {
                        int     v ;
                        rs = cfdeci(ubuf,-1,&v) ;
                        uid = v ;
                    }
                } else {
                    rs = SR_OK ;
                    uid = pip->uid ;
                }

                if (rs >= 0) {
                    char        protostr[32+1] ;
                    int         port ;
                    protostr[0] = '\0' ;
                    if (q.protop != NULL) {
                        strdcpy1w(protostr,32,q.protop,q.protol) ;
                    }
                    rs = getdefport(protostr,q.portp,q.portl) ;
                    port = rs ;
#if     CF_DEBUG
                    debugprintf("main/proclistquery: getdefport() rs=%d\n",rs) ;
#endif
                    if (rs == SR_NOTFOUND) rs = SR_PERM ;
                    if (rs >= 0) {
                        rs = userports_query(dbp,uid,protostr,port) ;
#if     CF_DEBUG
                        debugprintf("main/proclistquery: "
                            "userports_query() rs=%d\n",rs) ;
#endif
                        if (rs >= 0) c += 1 ;
                        else if (rs == SR_NOTFOUND) rs = SR_PERM ;
                    }
                } /* end if */

                if (rs < 0) break ;
            } /* end for (queries) */
            uc_free(pwbuf) ;
        } /* end if (memory-allocation) */

#if     CF_DEBUG
```

```
        debugprintf("main/proclistquery: ret rs=%d c=%u\n",rs,c) ;
#endif

        return (rs >= 0) ? c : rs ;
}
/* end subroutine (proclistquery) */


static int procbind(PROGINFO *pip,USERPORTS *dbp,int cfd)
{
        struct openportmsg_request      m0 ;
        struct openportmsg_response     m1 ;
        const uid_t     uid_cur = pip->uid ;
        const int       mlen = MBUFLEN ;
        int             rrs = SR_BADFMT ;
        int             rs ;
        int             rs1 ;
        int             port ;
        int             sal ;
        int             size ;
        int             ml ;
        int             fd = 0 ;
        int             f_ok = FALSE ;
        const char      *protoname ;
        char            mbuf[MBUFLEN+1] ;

/* read in the arguments passed from caller */

        if ((rs = u_read(cfd,mbuf,mlen)) >= 0) {
            cchar       *fmt ;
            ml = rs ;

            if (pip->open.logprog)
                proglog_printf(pip,"msgtype=%u",(mbuf[0] & 0xff)) ;

            if ((rs = openportmsg_request(&m0,1,mbuf,ml)) >= 0) {

#if     CF_DEBUG
                if (DEBUGLEVEL(4))
                    debugprintf("main/procbind: openportmsg_request() rs=%d",
                        rs) ;
#endif

                rrs = SR_PROTO ;
                if (m0.msgtype == openportmsgtype_request) {
                    SOCKADDRESS *saddrp = (SOCKADDRESS *) &m0.sa ;
                    int         pf = m0.pf ;
                    int         ptype = m0.ptype ;
                    int         proto = m0.proto ;

                    if (pip->open.logprog) {
                        proglog_printf(pip,"pf=%u pt=%u proto=%u",
                            pf,ptype,proto) ;
                    }

/* get the protoname */

                    rrs = getprotoname(pf,ptype,proto,&protoname) ;

                    if (pip->open.logprog) {
                        proglog_printf(pip,"protoname(%d)=%s",
                            rrs,protoname) ;
                    }

/* get the port out of there */
```

```
                    if (rrs >= 0) {
                        rrs = sockaddress_getport(saddrp) ;
                        port = rrs ;

                        if (pip->open.logprog) {
                            int         af ;
                            int         v = ((rrs >= 0) ? port : rrs) ;
                            char        addrbuf[INETXADDRLEN+1] ;
                            char        addrstr[INETADDRSTRLEN+1] ;
                            af = sockaddress_getaf(saddrp) ;
                            sockaddress_getaddr(saddrp,addrbuf,INETXADDRLEN) ;
                            sninetaddr(addrstr,INETADDRSTRLEN,af,addrbuf) ;
                            fmt = "af=%u addr=%s port=%d" ;
                            proglog_printf(pip,fmt,af,addrstr,v) ;
                        }

                    }

                    if (rrs >= 0) {
                        f_ok = TRUE ;
                        if (port < IPPORT_RESERVED) {

                            rrs = userports_query(dbp,uid_cur,protoname,port) ;
                            f_ok = (rrs >= 0) ;

#if     CF_DEBUG
                            if (DEBUGLEVEL(4))
                                debugprintf("main: userports_query() rs=%d\n",
                                    rs) ;
#endif

                            if (pip->open.logprog) {
                                proglog_printf(pip,"query=%d",rrs) ;
                            }

                        } /* end if */
                    } /* end if (validating port) */

/* make the socket */

                    if ((rs >= 0) && (rrs >= 0) && f_ok) {
                        struct sockaddr *sap = (struct sockaddr *) saddrp ;

#if     CF_DEBUG
                        if (DEBUGLEVEL(4))
                            debugprintf("main: pf=%d ptype=%u p=%u\n",
                                pf,ptype,proto) ;
#endif
                        if ((rrs = sockaddress_getlen(saddrp)) >= 0) {
                            sal = rrs ;

#if     CF_DEBUGS
                            debugprinthex("openbind: openbind<",80,sap,sal) ;
#endif
                            rrs = openbind(pf,ptype,proto,sap,sal) ;
                            fd = rrs ;
                            f_ok = (rrs >= 0) ;
#if     CF_DEBUG
                            if (DEBUGLEVEL(2))
                                debugprintf("main: openbind() rs=%d\n",rrs) ;
#endif

                            if (pip->open.logprog) {
                                proglog_printf(pip,"openfd=%d",rrs) ;
                            }
                        } /* end if (sockaddress_getlen) */
```

```
                    } /* end if (ok) */

                } /* end if (valid request) */

                if (rs >= 0) { /* prepare the reponse */

                    size = sizeof(struct openportmsg_response) ;
                    memset(&m1,0,size) ;
                    m1.msgtype = openportmsgtype_response ;
                    m1.rs = rrs ;

                    if ((rs = openportmsg_response(&m1,0,mbuf,mlen)) >= 0) {
                        ml = rs ;

                        rs1 = uc_writen(cfd,mbuf,ml) ;

                        if ((rs1 >= 0) && f_ok) {

#if     CF_DEBUG
                            if (DEBUGLEVEL(2))
                                debugprintf("main: sending FD=%u\n",fd) ;
#endif

                            rs1 = u_ioctl(cfd,I_SENDFD,fd) ;

#if     CF_DEBUG
                            if (DEBUGLEVEL(2))
                                debugprintf("main: u_ioctl() rs=%d\n",rs1) ;
#endif

                            if (pip->open.logprog) {
                                proglog_printf(pip,"sendfd(%d)=%d",
                                    rs1,fd) ;
                            }

                        } /* end if (sending FD) */

                        if (fd >= 0) {
                            u_close(fd) ;
                            fd = -1 ;
                        }
                        if (rs >= 0) rs = rs1 ;
                    } /* end if */

                } /* end if (ok) */
            } /* end if (request) */
        } /* end if (read data) */

        return rs ;
}
/* end subroutine (procbind) */


/* parse out a query */
static int parsequery(struct query *qp,const char *sp,int sl)
{
        const char      *tp ;
        int             pl = 0 ;

        if (sl < 0) sl = strlen(sp) ;

        memset(qp,0,sizeof(struct query)) ;

        qp->uidp = NULL ;
        qp->protop = NULL ;
```

```
        qp->portp = NULL ;
        if ((tp = strnchr(sp,sl,':')) != NULL) {
            cchar       *cp = (tp+1) ;
            int         cl = ((sp+sl)-(tp+1)) ;

            qp->uidp = sp ;
            qp->uidl = (tp-sp) ;

            if ((tp = strnchr(cp,cl,':')) != NULL) {
                qp->protop = cp ;
                qp->protol = (tp-cp) ;
                qp->portp = (tp+1) ;
                qp->portl = ((cp+cl)-(tp+1)) ;
                pl = qp->portl ;
            } else {
                qp->portp = cp ;
                qp->portl = cl ;
                pl = qp->portl ;
            } /* end if */

        } else {

            qp->portp = sp ;
            qp->portl = sl ;
            pl = qp->portl ;

        } /* end if */

        return pl ;
}
/* end subroutine (parsequery) */


/* get a port number */
static int getdefport(cchar *protostr,cchar *pp,int pl)
{
        int             rs = SR_OK ;
        int             i ;
        char            portstr[32+1] ;

        strdcpy1w(portstr,32,pp,pl) ;

#if     CF_DEBUGS
        debugprintf("main/getdefport: protostr=%s\n",protostr) ;
        debugprintf("main/getdefport: portstr=%s\n",portstr) ;
#endif

        if (protostr[0] == '\0') {
            for (i = 0 ; defprotos[i] != NULL ; i += 1) {
                protostr = defprotos[i] ;
                rs = getportnum(protostr,portstr) ;
                if (rs != SR_NOTFOUND) break ;
            } /* end for */
        } else {
            rs = getportnum(protostr,portstr) ;
        }

#if     CF_DEBUGS
        debugprintf("main/getdefport: ret rs=%d\n",rs) ;
#endif

        return rs ;
}
/* end subroutine (getdefport) */
```

```
static int openbind(int pf,int pt,int proto,SOCKADDR *sap,int sal)
{
        int             rs ;
        int             fd = 0 ;

#if     CF_DEBUGS
        debugprinthex("openbind: ",80,sap,sal) ;
#endif

        if ((rs = u_socket(pf,pt,proto)) >= 0) {
            fd = rs ;
            if ((rs = uc_reuseaddr(fd)) >= 0) {
                rs = u_bind(fd,sap,sal) ;
            }
            if (rs < 0) u_close(fd) ;
        } /* end if (u_socket) */

        return (rs >= 0) ? fd : rs ;
}
/* end subroutine (open-bind) */


static int getprotoname(int pf,int ptype,int proto,cchar **rpp)
{
        int             rs = SR_NOTFOUND ;
        int             i ;
        int             f = FALSE ;
        const char      *pn ;

        for (i = 0 ; socknames[i].name != NULL ; i += 1) {
            f = TRUE ;
            f = f && (socknames[i].pf == pf) ;
            f = f && (socknames[i].ptype == ptype) ;
            f = f && (socknames[i].proto == proto) ;
            if (f) break ;
        } /* end for */

        if (f) {
            rs = SR_OK ;
            pn = socknames[i].name ;
            if (strcmp(pn,"tcp6") == 0) pn = "tcp" ;
            if (rpp != NULL) *rpp = pn ;
        }

        return rs ;
}
/* end subroutine (getprotoname) */
```