```
/* openport */

/* proxy-open a socket port for the calling program */


#define CF_DEBUGS       0               /* non-switchable debug print-outs */


/* revision history:

        = 1998-10-20, David A›D› Morano
        This subroutine was originally written so that dangerous daemon programs
        do not have to be run as 'root' in order to bind to a priveledge port
        (for those transport providers that have priveledged ports). A good
        example of a dangerous daemon program that this capability is especially
        useful for is the infamous SENDMAIL daemon!

*/

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

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

        This subroutine will open a socket port as a proxy for our calling
        process.

        Synopsis:

        int openport(pf,ptype,proto,sap)
        int             pf ;
        int             ptype ;
        int             proto ;
        SOCKADDRESS     *sap ;

        Arguments:

        pf              protocol family (PF_xxx)
        ptype           protocol type (eg: SOCK_DGRAM)
        proto           protocol
        sap             pointer to socket address

        Returns:

        <0              error
        >=0             FD


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


#include        <envstandards.h>

#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        <stdlib.h>
#include        <string.h>

#include        <vsystem.h>
#include        <nulstr.h>
#include        <sockaddress.h>
#include        <vecstr.h>
```

```
#include        <spawnproc.h>
#include        <localmisc.h>

#include        "openport.h"
#include        "openportmsg.h"


/* type-defs */

#ifndef TYPEDEF_CCHAR
#define TYPEDEF_CCHAR   1
typedef const char      cchar ;
#endif


/* local defines */

#ifndef VARHOME
#define VARHOME         "HOME"
#endif

#ifndef VARUSERNAME
#define VARUSERNAME     "USERNAME"
#endif

#define PROG_OPENPORT   "openport"

#define ENVBUFLEN       (MAXPATHLEN + 20)
#define MBUFLEN         MSGBUFLEN

#define NENVS           10


/* external subroutines */

extern int      sncpy2(char *,int,const char *,const char *) ;
extern int      mkpath3(char *,const char *,const char *,const char *) ;
extern int      sfbasename(const char *,int,const char **) ;
extern int      matkeystr(const char **,const char *,int) ;
extern int      perm(const char *,uid_t,gid_t,gid_t *,int) ;
extern int      getusername(char *,int,uid_t) ;
extern int      vecstr_envadd(VECSTR *,const char *,const char *,int) ;
extern int      isNotPresent(int) ;
extern int      isNotAccess(int) ;

#if     CF_DEBUGS
extern int      debugprintf(const char *,...) ;
#endif

extern char     *strwcpy(char *,const char *,int) ;


/* external variables */

extern cchar    **environ ;


/* local structures */


/* forward references */

static int procprog(int,int,int,SOCKADDRESS *,cchar *,cchar *,int,cchar *) ;
static int procspawn(cchar *,cchar *,cchar **,cchar **,
                int,int,int,SOCKADDRESS *) ;
static int procspawn_begin(SPAWNPROC *,cchar *,cchar **,cchar **) ;
```

```
static int procspawn_end(pid_t,int *) ;

static int procexchange(cchar *,int,int,int,int,SOCKADDRESS *) ;

static int      loadenvs(VECSTR *,cchar *,cchar *,cchar *,int) ;
static int      getprog(char *) ;


/* local variables */

static const char       *prs[] = {
        "/usr/extra",
        "/usr/preroot",
        NULL
} ;


/* exported subroutines */


int openport(int pf,int ptype,int proto,SOCKADDRESS *sap)
{
        const int       ulen = USERNAMELEN ;
        int             rs ;
        int             fd = -1 ;
        char            ubuf[USERNAMELEN+1] ;

        if (sap == NULL) return SR_FAULT ;

#if     CF_DEBUGS
        debugprintf("openport: ent pf=%u ptype=%u proto=%u\n",
            pf,ptype,proto) ;
#endif

        if ((rs = getusername(ubuf,ulen,-1)) >= 0) {
            char        progfname[MAXPATHLEN+1] = { 0 } ; /* LINT */
            if ((rs = getprog(progfname)) >= 0) {
                const int       pl = rs ;
                int             bl ;
                const char      *prog = progfname ;
                const char      *bn ;
                if ((bl = sfbasename(prog,pl,&bn)) > 0) {
                    rs = procprog(pf,ptype,proto,sap,prog,bn,bl,ubuf) ;
                    fd = rs ;
                } else {
                    rs = SR_INVALID ;
                }
            } /* end if (getprog) */
        } /* end if (getusername) */

#if     CF_DEBUGS
        debugprintf("openport: ret rs=%d fd=%d\n",rs,fd) ;
#endif

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


/* local subroutines */


static int procprog(pf,pt,proto,sap,prog,bn,bl,un)
int             pf ;
int             pt ;
int             proto ;
```

```
SOCKADDRESS     *sap ;
const char      prog[] ;
const char      bn[] ;
int             bl ;
const char      un[] ;
{
        VECSTR          envs ;
        int             rs ;
        int             rs1 ;
        int             fd = -1 ;
        if ((rs = vecstr_start(&envs,5,0)) >= 0) {
            if ((rs = loadenvs(&envs,un,prog,bn,bl)) >= 0) {
                NULSTR  n ;
                cchar   *name ;
                if ((rs = nulstr_start(&n,bn,bl,&name)) >= 0) {
                    int         i = 0 ;
                    int         j ;
                    cchar       *sargv[5] ;
                    cchar       *senvv[NENVS +2+ 1] ;
                    cchar       *ep ;
                    sargv[i++] = name ;
                    sargv[i++] = "-b" ;
                    sargv[i] = NULL ;
                    i = 0 ;
                    if ((j = matkeystr(environ,VARHOME,-1)) >= 0) {
                        senvv[i++] = environ[j] ;
                    }
                    for (j = 0 ; vecstr_get(&envs,j,&ep) >= 0 ; j += 1) {
                        if (ep != NULL) {
                            if (i >= NENVS) {
                                rs = SR_NOANODE ;
                                break ;
                            }
                            senvv[i++] = ep ;
                        }
                    } /* end for */
                    senvv[i] = NULL ;
                    if (rs >= 0) {
                        rs = procspawn(un,prog,sargv,senvv,pf,pt,proto,sap) ;
                        fd = rs ;
                    } /* end if (ok) */
                    rs1 = nulstr_finish(&n) ;
                    if (rs >= 0) rs = rs1 ;
                } /* end if (nulstr) */
            } /* end if (loadenvs) */
            rs1 = vecstr_finish(&envs) ;
            if (rs >= 0) rs = rs1 ;
            if ((rs < 0) && (fd >= 0)) u_close(fd) ;
        } /* end if (vecstr) */
        return (rs >= 0) ? fd : rs ;
}
/* end subroutine (procprog) */


static int procspawn(un,prog,sargv,senvv,pf,pt,proto,sap)
cchar           *un ;
cchar           *prog ;
cchar           **sargv ;
cchar           **senvv ;
int             pf ;
int             pt ;
int             proto ;
SOCKADDRESS     *sap ;
{
                    SPAWNPROC   psa ;
        int             rs ;
```

```
        int             rs1 ;
        int             fd = -1 ;
        if ((rs = procspawn_begin(&psa,prog,sargv,senvv)) >= 0) {
                        const pid_t     pid = rs ;
                        const int       cfd = psa.fd[0] ;
                        int             cs ;

                        rs = procexchange(un,cfd,pf,pt,proto,sap) ;
                        fd = rs ;
                        u_close(cfd) ;

                        rs1 = procspawn_end(pid,&cs) ;
                        if (rs >= 0) rs = rs1 ;
                    if ((rs < 0) && (fd >= 0)) u_close(fd) ;
        } /* end if (spawned program) */
        return (rs >= 0) ? fd : rs ;
}
/* end subroutine (procspawn) */


static int procspawn_begin(psp,prog,sargv,senvv)
SPAWNPROC       *psp ;
cchar           *prog ;
cchar           **sargv ;
cchar           **senvv ;
{
        int             rs ;
        memset(psp,0,sizeof(struct spawnproc)) ;
        psp->opts |= SPAWNPROC_OIGNINTR ;
        psp->opts |= SPAWNPROC_OSETPGRP ;
        psp->disp[0] = SPAWNPROC_DOPEN ;
        psp->disp[1] = SPAWNPROC_DCLOSE ;
        psp->disp[2] = SPAWNPROC_DCLOSE ;
        rs = spawnproc(psp,prog,sargv,senvv) ;
        return rs ;
}
/* end subroutine (procspawn_begin) */


static int procspawn_end(pid_t pid,int *csp)
{
        return u_waitpid(pid,csp,0) ;
}
/* end subroutine (procspawn_end) */


static int procexchange(un,cfd,pf,ptype,proto,sap)
cchar           un[] ;
int             cfd ;
int             pf ;
int             ptype ;
int             proto ;
SOCKADDRESS     *sap ;
{
        struct openportmsg_request      m0 ;
        struct openportmsg_response     m1 ;
        struct strrecvfd        fds ;
        int             rs ;
        int             rs1 ;
        int             size ;
        int             ml ;
        int             fd = -1 ;
        char            mbuf[MBUFLEN+1] ;

        size = sizeof(struct openportmsg_request) ;
        memset(&m0,0,size) ;
```

```
        m0.msgtype = openportmsgtype_request ;
        m0.pf = pf ;
        m0.ptype = ptype ;
        m0.proto = proto ;
        m0.sa = *sap ;
        strwcpy(m0.username,un,USERNAMELEN) ;

        rs = openportmsg_request(&m0,0,mbuf,MBUFLEN) ;
        ml = rs ;

#if     CF_DEBUGS
        debugprintf("openport: openportmsg_request() rs=%d\n",rs) ;
#endif

        if (rs >= 0)
            rs = uc_writen(cfd,mbuf,ml) ;

#if     CF_DEBUGS
        debugprintf("openport: uc_writen() rs=%d\n",rs) ;
#endif

        if (rs >= 0) {
            const int   mt = openportmsgtype_response ;

            rs1 = u_read(cfd,mbuf,MBUFLEN) ;
            ml = rs1 ;

#if     CF_DEBUGS
            debugprintf("openport: uc_readn() rs=%d\n",rs1) ;
#endif

            if (rs1 >= 0) {
                rs1 = openportmsg_response(&m1,1,mbuf,ml) ;

#if     CF_DEBUGS
                debugprintf("openport: openportmsg_response() rs=%d\n",
                    rs1) ;
                debugprintf("openport: m1.msgtype=%u m1.rs=%d\n",
                    m1.msgtype,m1.rs) ;
#endif

            }

            if ((rs1 > 0) && (m1.msgtype == mt)) {
                if (m1.rs >= 0) {
                    rs = u_ioctl(cfd,I_RECVFD,&fds) ;
                    fd = fds.fd ;
                } else
                    rs = m1.rs ;
            } else
                rs = SR_PROTO ;

        } /* end if (write was successful) */

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


static int loadenvs(vecstr *elp,cchar *un,cchar *prog,cchar *bn,int bl)
{
        int             rs = SR_OK ;
        int             c = 0 ;
        if (rs >= 0) {
            rs = vecstr_envadd(elp,"_",prog,-1) ;
            if (rs < INT_MAX) c += 1 ;
```

```
        }
        if (rs >= 0) {
            rs = vecstr_envadd(elp,"_EF",prog,-1) ;
            if (rs < INT_MAX) c += 1 ;
        }
        if (rs >= 0) {
            rs = vecstr_envadd(elp,"_A0",bn,bl) ;
            if (rs < INT_MAX) c += 1 ;
        }
        if (rs >= 0) {
            rs = vecstr_envadd(elp,"USERNAME",un,-1) ;
            if (rs < INT_MAX) c += 1 ;
        }
        return (rs >= 0) ? c : rs ;
}
/* end subroutine (loadenvs) */


static int getprog(char *progfname)
{
        struct ustat    sb ;
        int             rs = SR_LIBACC ;
        int             i ;
        int             rl = 0 ;
        int             len = 0 ;
        cchar           *bin = "sbin" ;
        cchar           *pn = PROG_OPENPORT ;

        for (i = 0 ; prs[i] != NULL ; i += 1) {
            if ((rs = mkpath3(progfname,prs[i],bin,pn)) >= 0) {
                rl = rs ;
                if ((rs = u_stat(progfname,&sb)) >= 0) {
                    if (S_ISREG(sb.st_mode)) {
                        const int       am = (R_OK|X_OK) ;
                        if ((rs = perm(progfname,-1,-1,NULL,am)) >= 0) {
                            len = rl ;
                        } else if (isNotAccess(rs)) {
                            rs = SR_OK ;
                        }
                    } /* end if (is-reg-file) */
                } else if (isNotPresent(rs)) {
                    rs = SR_OK ;
                } /* end if (stat) */
            } /* end if (mkpath) */
            if (len > 0) break ;
            if (rs < 0) break ;
        } /* end for (prs) */

        if ((rs >= 0) && (len == 0)) rs = SR_LIBACC ;

        return (rs >= 0) ? len : rs ;
}
/* end subroutine (getprog) */
```