```
#ifdef sccs
static char     sccsid[] = "@(#)pcnfsd.c        1.4";
#endif

/*
* Copyright (c) 1986 by Sun Microsystems, Inc.
*/

/*
* pcnfsd.c
*
* pcnfsd is intended to remedy the lack of certain critical generic network
* services by providing an simple, customizable set of RPC-based
* mechanisms. For this reason, Sun Microsystems Inc. is distributing it
* in source form as part of the PC-NFS release.
*
* Background: The first NFS networks were composed of systems running
* derivatives of the 4.2BSD release of Unix (Sun's, VAXes, Goulds and
* Pyramids). The immediate utility of the resulting networks was derived
* not only from NFS but also from the availability of a number of TCP/IP
* based network services derived from 4.2BSD. Furthermore the thorny
* question of network-wide user authentication, while remaining a
* security hole, was solved at least in terms of a convenient usage model
* by the Yellow Pages distributed data base facility, which allows
* multiple Unix systems to refer to common password and group files.
*
* The PC-NFS Dilemma: When Sun Microsystems Inc. ported NFS to PC's, two
* things became apparent. First, the memory constraints of the typical PC
* meant that it would be impossible to incorporate the pervasive TCP/IP
* based service suite in a resident fashion. Indeed it was not at all
* clear that the 4.2BSD services would prove sufficient: with the advent
* of Unix System V and (experimental) VAX-VMS NFS implementations, we had
* to consider the existence of networks with no BSD-derived Unix hosts.
* The two key types of functionality we needed to provide were remote
* login and print spooling. The second critical issue  was that of user
* authentication. Traditional time-sharing systems such as Unix and VMS
* have well- established user authentication mechanisms based upon user
* id's and passwords: by defining appropriate mappings, these could
* suffice for network-wide authentication provided that appropriate
* administrative procedures were enforced. The PC, however, is typically
* a single-user system, and the standard DOS operating environment
```

```
* provides no user authentication mechanisms. While this is acceptable
* within a single PC, it causes problems when attempting to connect to a
* heterogeneous network of systems in which access control, file space
* allocation, and print job accounting and routing may all be based upon
* a user's identity. The initial (and default) approach is to use the
* pseudo-identity 'nobody' defined as part of NFS to handle problems such
* as this. However, taking ease of use into consideration, it became
* necessary to provide a mechanism for establishing a user's identity.
*
* Initially we felt that we needed to implement two types of functionality:
* user authentication and print spooling. (Remote login is addressed by
* the Telnet module.) Since no network services were defined within the
* NFS architecture to support these, it was decided to implement them in
* a fairly portable fashion using Sun's Remote Procedure Call protocol.
* Since these mechanisms will need to be re-implemented ion a variety of
* software environments, we have tried to define a very general model.
*
* Authentication: NFS adopts the Unix model of using a pair of integers
* (uid, gid) to define a user's identity. This happens to map tolerably
* well onto the VMS system. 'pcnfsd' implements a Remote Procedure which
* is required to map a username and password into a (uid, gid) pair.
* Since we cannot predict what mapping is to be performed, and since we
* do not wish to pass clear-text passwords over the net, both the
* username and the password are mildly scrambled using a simple XOR
* operation. The intent is not to be secure (the present NFS architecture
* is inherently insecure) but to defeat "browsers".
*
* The authentication RPC will be invoked when the user enters the PC-NFS
* command:
*
* NET NAME user [password|*]
*
*
* Printing: The availability of NFS file operations simplifies the print
* spooling mechanisms. There are two services which 'pcnfsd' has to
* provide:
*   pr_init:    given the name of the client system, return the
* name of a directory which is exported via NFS and in which the client
* may create spool files.
*  pr_start: given a file name, a user name, the printer name, the client
* system name and an option string, initiate printing of the file
```

```
* on the named printer. The file name is relative to the directory
* returned by pr_init. pr_start is to be "idempotent": a request to print
* a file which is already being printed has no effect.
*
* Intent: The first versions of these procedures are implementations for Sun
* 2.0/3.0 software, which will also run on VAX 4.2BSD systems. The intent
* is to build up a set of implementations for different architectures
* (Unix System V, VMS, etc.). Users are encouraged to submit their own
* variations for redistribution. If you need a particular variation which
* you don't see here, either code it yourself (and, hopefully, send it to
* us at Sun) or contact your Customer Support representative.
*/


#include <sys/types.h>
#include <stdio.h>

#ifdef OS2
#undef min
#include <stdlib.h>
#include <types.h>
#include <io.h>
#include <io.h>
#include <string.h>
#include <process.h>
#include <direct.h>
#define mkdir(p, m) mkdir(p)
#define INCL_DOS
#include <os2.h>
extern char *crypt(char *, char*);
#else
#include <sys/file.h>
#endif

#ifdef SHADOWPWD
#include <shadow.h>
#include <shadow/pwauth.h>
#else
#include <pwd.h>
#endif
```

```
#include <rpc/rpc.h>
#include <signal.h>
#include <sys/stat.h>

/*  #define DEBUG 1  */

#ifdef DEBUG
int             buggit = 0;
#endif

#ifdef OS2
#   define LP_SPOOL     "C:\\spool\\pcnfs"
#   define LP_BIN       "print.com"
#   define DIRSEP       "\\"
#else
#   define LP_SPOOL     "/usr/spool/lp"
#   define LP_BIN       "/usr/ucb/lpr"
#   define DIRSEP       "/"
#endif


/*
* *************** RPC parameters ********************
*/
#define PCNFSDPROG      (long)150001
#define PCNFSDVERS      (long)1
#define PCNFSD_AUTH     (long)1
#define PCNFSD_PR_INIT  (long)2
#define PCNFSD_PR_START (long)3

/*
* ************* Other #define's ***********************
*/
#ifndef MAXPATHLEN
#define MAXPATHLEN 1024
#endif
#define zchar           0x5b


/*
* *********** XDR structures, etc. *********************
```

```
*/
enum arstat {
**AUTH\_RES\_OK, AUTH\_RES\_FAKE, AUTH\_RES\_FAIL**

};
enum pirstat {
**PI\_RES\_OK, PI\_RES\_NO\_SUCH\_PRINTER, PI\_RES\_FAIL**

};
enum psrstat {
**PS\_RES\_OK, PS\_RES\_ALREADY, PS\_RES\_NULL, PS\_RES\_NO\_FILE,**

        **PS\_RES\_FAIL**

};

struct auth_args {
**char           \*aa\_ident;**

        **char           \*aa\_password;**

};

struct auth_results {
**enum arstat     ar\_stat;**

        **long            ar\_uid;**

        **long            ar\_gid;**

};

struct pr_init_args {
**char           \*pia\_client;**

        **char           \*pia\_printername;**

};

struct pr_init_results {
**enum pirstat    pir\_stat;**
```

```
        **char          \*pir\_spooldir;**
};

struct pr_start_args {
**char          \*psa\_client;**
        **char          \*psa\_printername;**
        **char          \*psa\_username;**
        **char          \*psa\_filename;       /\* within the spooldir \*/**
        **char          \*psa\_options;**
};

struct pr_start_results {
**enum psrstat    psr\_stat;**
};


/*
* ***************** Misc. ************************
*/

char           *authproc();
char           *pr_start();
char           *pr_init();
struct stat     statbuf;

char            pathname[MAXPATHLEN];
char            new_pathname[MAXPATHLEN];
char            spoolname[MAXPATHLEN];


/*
* ************** Support procedures ***********************
*/
```

```
scramble(s1, s2)
**char            \*s1;**

        **char            \*s2;**

{
**while (\*s1) {**

                **\*s2++ = (\*s1 ^ zchar) \& 0x7f;**

                **s1++;**

        **}**

        **\*s2 = 0;**

}

free_child()
{
**int             pid;**

        **int             pstatus;**


        **pid = wait(\&pstatus);        /\* clear exit of child process \*/**



#ifdef DEBUG
**if (buggit || pstatus)**

                **fprintf(stderr, "FREE\_CHILD: process #%d exited with status
0X%x\\r\\n",**

                        **pid, pstatus);**

#endif
**return;**
```

```
}

/*
* *************** XDR procedures *****************
*/
bool_t
xdr_auth_args(xdrs, aap)
**XDR             \*xdrs;**

        **struct auth\_args \*aap;**

{
**return (xdr\_string(xdrs, \&aap->aa\_ident, 32) \&\&**

                **xdr\_string(xdrs, \&aap->aa\_password, 64));**

}

bool_t
xdr_auth_results(xdrs, arp)
**XDR             \*xdrs;**

        **struct auth\_results \*arp;**

{
**return (xdr\_enum(xdrs, (int \*) \&arp->ar\_stat) \&\&**

                **xdr\_long(xdrs, \&arp->ar\_uid) \&\&**

                **xdr\_long(xdrs, \&arp->ar\_gid));**

}

bool_t
xdr_pr_init_args(xdrs, aap)
**XDR             \*xdrs;**

        **struct pr\_init\_args \*aap;**

{
```

```
**return (xdr\_string(xdrs, \&aap->pia\_client, 64) \&\&**
                **xdr\_string(xdrs, \&aap->pia\_printername, 64));**
}

bool_t
xdr_pr_init_results(xdrs, arp)
**XDR             \*xdrs;**
        **struct pr\_init\_results \*arp;**
{
**return (xdr\_enum(xdrs, (int \*) \&arp->pir\_stat) \&\&**
                **xdr\_string(xdrs, \&arp->pir\_spooldir, 255));**
}

bool_t
xdr_pr_start_args(xdrs, aap)
**XDR             \*xdrs;**
        **struct pr\_start\_args \*aap;**
{
**return (xdr\_string(xdrs, \&aap->psa\_client, 64) \&\&**
                **xdr\_string(xdrs, \&aap->psa\_printername, 64) \&\&**
                **xdr\_string(xdrs, \&aap->psa\_username, 64) \&\&**
                **xdr\_string(xdrs, \&aap->psa\_filename, 64) \&\&**
                **xdr\_string(xdrs, \&aap->psa\_options, 64));**
}

bool_t
xdr_pr_start_results(xdrs, arp)
**XDR             \*xdrs;**
```

```
        **struct pr\_start\_results \*arp;**
{
**return (xdr\_enum(xdrs, (int \*) \&arp->psr\_stat));**
}



/*
* ********************** main ********************
*/
main(argc, argv)
**int             argc;**
        **char          \*\*argv;**
{
**int             f1, f2, f3;**


                **extern         xdr\_string\_array();**



#ifdef OS2
        **if (argc < 2) {**
                        **strcpy(spoolname, LP\_SPOOL);**
                        **spoolname\[0] = 'A' - 1 + \_getdrive();**
                **}**
#else
**if (fork() == 0) {**
```

```
                **if (argc < 2)**
                        **strcpy(spoolname, LP\_SPOOL);**
#endif
        **else**
                        **strcpy(spoolname, argv\[1]);**
#ifdef DEBUG
        **if (argc > 2)**
                        **buggit++;**
#endif
        **if (stat(spoolname, \&statbuf) || !(statbuf.st\_mode \& S\_IFDIR)) {**
                        **fprintf(stderr,**
                                **"pcnfsd: invalid spool directory %s\\r\\n",
spoolname);**
                        **exit(1);**
                **}**


/*  Comment out for now
        **if ((f1 = open("/dev/null", O\_RDONLY)) == -1) {**
                        **fprintf(stderr, "pcnfsd: couldn't open /dev/null\\r\\n");**
                        **exit(1);**
                **}**
```

```
                **if ((f2 = open("/dev/console", O\_WRONLY)) == -1) {**
                        **fprintf(stderr, "pcnfsd: couldn't open /dev/console\\r\\n");**
                        **exit(1);**
                **}**
                **if ((f3 = open("/dev/console", O\_WRONLY)) == -1) {**
                        **fprintf(stderr, "pcnfsd: couldn't open /dev/console\\r\\n");**
                        **exit(1);**
                **}**
                **dup2(f1, 0);**
                **dup2(f2, 1);**
                **dup2(f3, 2);**


end of commented out stuff */
        **registerrpc(PCNFSDPROG, PCNFSDVERS, PCNFSD\_AUTH, authproc,**
                        **xdr\_auth\_args, xdr\_auth\_results);**
                **registerrpc(PCNFSDPROG, PCNFSDVERS, PCNFSD\_PR\_INIT, pr\_init,**
                        **xdr\_pr\_init\_args, xdr\_pr\_init\_results);**
                **registerrpc(PCNFSDPROG, PCNFSDVERS, PCNFSD\_PR\_START, pr\_start,**
                        **xdr\_pr\_start\_args, xdr\_pr\_start\_results);**
                **svc\_run();**
```

```
                **fprintf(stderr, "pcnfsd: error: svc\_run returned\\r\\n");**
                **exit(1);**
#ifndef OS2
**}**
#endif
}


/*
* ******************** RPC procedures **************
*/
char            *
authproc(a)
**struct auth\_args \*a;**
{
**static struct auth\_results r;**
        **char            username\[32];**
        **char            password\[64];**
        **int             c1, c2;**
        **struct passwd  \*p;**
#ifdef SHADOWPWD
**struct spwd \*spwd;**
#endif
**r.ar\_stat = AUTH\_RES\_FAIL; /\* assume failure \*/**
        **scramble(a->aa\_ident, username);**
        **scramble(a->aa\_password, password);**
```

```
#ifdef DEBUG
**if (buggit)**
                **fprintf(stderr, "AUTHPROC username=%s\\r\\n", username);**
#endif
**p = getpwnam(username);**
        **if (p == NULL)**
                **return ((char \*) \&r);**
#ifdef SHADOWPWD
**if (!(spwd = getspnam(username)))**
                **return ((char \*) \&r);**
        **else**
                **p->pw\_passwd = spwd->sp\_pwdp;**


        **if (p->pw\_name \&\& p->pw\_passwd\[0] == '@') {**
                **if (pw\_auth(p->pw\_passwd+1, username, PW\_LOGIN))**
                        **return ((char \*) \&r);**
        **} else {**
                **if (!valid(password, p))**
                        **return ((char \*) \&r);**
        **}**
```

```
#else
**c1 = strlen(password);**
        **c2 = strlen(p->pw\_passwd);**
        **if ((c1 \&\& !c2) || (c2 \&\& !c1) ||**
                **(strcmp(p->pw\_passwd, crypt(password, p->pw\_passwd)))) {**
                **return ((char \*) \&r);**
        **}**
#endif
**r.ar\_stat = AUTH\_RES\_OK;**
        **r.ar\_uid = p->pw\_uid;**
        **r.ar\_gid = p->pw\_gid;**
        **return ((char \*) \&r);**
}


char            *
pr_init(pi_arg)
**struct pr\_init\_args \*pi\_arg;**

{
**int             dir\_mode = 0777;**
        **static struct pr\_init\_results pi\_res;**


        **/\* get pathname of current directory and return to client \*/**
        **strcpy(pathname, spoolname);  /\* first the spool area \*/**
        **strcat(pathname, DIRSEP);     /\* append a slash \*/**
```

```
        **strcat(pathname, pi\_arg->pia\_client);**
        **/\* now the host name \*/**

        **mkdir(pathname, dir\_mode);   /\* ignore the return code \*/**

        **if (stat(pathname, \&statbuf) || !(statbuf.st\_mode \& S\_IFDIR)) {**
                **fprintf(stderr,**
                        **"pcnfsd: unable to create spool directory %s\\r\\n",**
                        **pathname);**
                **pathname\[0] = 0;/\* null to tell client bad vibes \*/**
                **pi\_res.pir\_stat = PI\_RES\_FAIL;**
        **} else {**
                **pi\_res.pir\_stat = PI\_RES\_OK;**
        **}**
        **pi\_res.pir\_spooldir = \&pathname\[0];**
        **chmod(pathname, dir\_mode);**


#ifdef DEBUG
**if (buggit)**
                **fprintf(stderr, "PR\_INIT pathname=%s\\r\\n", pathname);**
```

```
#endif

**return ((char \*) \&pi\_res);**

}

char            *
pr_start(ps_arg)
**struct pr\_start\_args \*ps\_arg;**

{
**static struct pr\_start\_results ps\_res;**

        **int             pid;**

        **int             free\_child();**

        **char            printer\_opt\[64];**

        **char            username\_opt\[64];**

        **char            clientname\_opt\[64];**

        **struct passwd  \*p;**

        **long          rnum;**

        **char          snum\[20];**

        **int           z;**



#ifdef OS2
**strcpy(printer\_opt, "/D:");**

        **/\* Strangely, sometimes there are garbage characters after the eighth**

         **\* character. And since there can at most be eight, we truncate here. \*/**
```

```
        **if (strlen(ps\_arg->psa\_filename) > 8)**
          **ps\_arg->psa\_filename\[8] = 0;**
#else
**strcpy(printer\_opt, "-P");**
#endif
**strcpy(username\_opt, "-J");**
        **strcpy(clientname\_opt, "-C");**



#ifdef SIGCHLD
**signal(SIGCHLD, free\_child); /\* when child terminates it sends \*/**
#endif
**/\* a signal which we must get \*/**
        **strcpy(pathname, spoolname);  /\* build filename \*/**
        **strcat(pathname, DIRSEP);**
        **strcat(pathname, ps\_arg->psa\_client);       /\* /spool/host \*/**
        **strcat(pathname, DIRSEP);     /\* /spool/host/ \*/**
        **strcat(pathname, ps\_arg->psa\_filename);     /\* /spool/host/file \*/**



#ifdef DEBUG
**if (buggit) {**
                **fprintf(stderr, "PR\_START pathname=%s\\r\\n", pathname);**
                **fprintf(stderr, "PR\_START username= %s\\r\\n",
ps\_arg->psa\_username);**
```

```
                **fprintf(stderr, "PR\_START client= %s\\r\\n", ps\_arg->psa\_client);**
        **}**
#endif
**strcat(printer\_opt, ps\_arg->psa\_printername);**
        **/\* make it (e.g.) -Plw        \*/**
        **strcat(username\_opt, ps\_arg->psa\_username);**
        **/\* make it (e.g.) -Jbilly     \*/**
        **strcat(clientname\_opt, ps\_arg->psa\_client);**
        **/\* make it (e.g.) -Cmypc      \*/**

        **if (stat(pathname, \&statbuf)) {**
                **/\***
                 **\* We can't stat the file. Let's try appending '.spl' and**
                 **\* see if it's already in progress.**
                 **\*/**


#ifdef DEBUG
        **if (buggit)**
                        **fprintf(stderr, "...can't stat it.\\r\\n");**
#endif
```

```
        **strcat(pathname, ".spl");**
                **if (stat(pathname, \&statbuf)) {**
                        **/\***
                         **\* It really doesn't exist.**
                         **\*/**


#ifdef DEBUG
                **if (buggit)**
                                **fprintf(stderr, "...PR\_START returns
PS\_RES\_NO\_FILE\\r\\n");**
#endif
                **ps\_res.psr\_stat = PS\_RES\_NO\_FILE;**
                        **return ((char \*) \&ps\_res);**
                **}**
                **/\***
                 **\* It is already on the way.**
                 **\*/**


#ifdef DEBUG
        **if (buggit)**
                        **fprintf(stderr, "...PR\_START returns
PS\_RES\_ALREADY\\r\\n");**
```

```
#endif
        **ps\_res.psr\_stat = PS\_RES\_ALREADY;**
                **return ((char \*) \&ps\_res);**
        **}**
        **if (statbuf.st\_size == 0) {**
                **/\***
                 **\* Null file - don't print it, just kill it.**
                 **\*/**
                **unlink(pathname);**


#ifdef DEBUG
        **if (buggit)**
                        **fprintf(stderr, "...PR\_START returns PS\_RES\_NULL\\r\\n");**
#endif
        **ps\_res.psr\_stat = PS\_RES\_NULL;**
                **return ((char \*) \&ps\_res);**
        **}**
        **/\***
         **\* The file is real, has some data, and is not already going out.**
         **\* We rename it by appending '.spl' and exec "lpr" to do the**
```

```
         **\* actual work.**
         **\*/**
        **strcpy(new\_pathname, pathname);**
        **strcat(new\_pathname, ".spl");**


#ifdef DEBUG
**if (buggit)**
                **fprintf(stderr, "...renaming %s -> %s\\r\\n", pathname,
new\_pathname);**
#endif
**/\***
         **\* See if the new filename exists so as not to overwrite it.**
         **\*/**



        **for(z = 0; z <100; z++) {**
                **if (!stat(new\_pathname, \&statbuf)){**
                        **strcpy(new\_pathname, pathname);  /\* rebuild a new name \*/**
                        **sprintf(snum,"%ld",random());         /\* get some number \*/**
                        **strncat(new\_pathname, snum, 3);**
                        **strcat(new\_pathname, ".spl");                /\* new spool file
\*/**
```

```
#ifdef DEBUG
                **if (buggit)**
                                        **fprintf(stderr, "...created new spl file -> %s\\r\\n",
new\_pathname);**



#endif
        **} else**
                        **break;**
        **}**


        **z = rename(pathname, new\_pathname);**
#ifdef OS2
**while (z \&\& errno == EISOPEN) {**
                **/\* You can't rename an open file under OS/2, \*/**
                **/\* it may still be opened by the nfsd server. \*/**
                **DosSleep(100); /\* wait a little bit \*/**
                **z = rename(pathname, new\_pathname); /\* and retry \*/**
        **}**
#endif
**if (z) {**
                **/\***
                 **\* CAVEAT: Microsoft changed rename for Microsoft C V3.0.**
```

```
                 **\* Check this if porting to Xenix.**
                 **\*/**
                **/\***
                 **\* Should never happen.**
                 **\*/**
                **fprintf(stderr, "pcnfsd: spool file rename (%s->%s) failed.\\r\\n",**
                        **pathname, new\_pathname);**
                **ps\_res.psr\_stat = PS\_RES\_FAIL;**
                **return ((char \*) \&ps\_res);**
        **}**


#ifdef OS2
**pid = spawnlp(P\_WAIT, LP\_BIN, LP\_BIN, printer\_opt, new\_pathname, 0);**
        **if (pid < 0) {**
          **perror("pcnfsd: spawn print failed");**
        **} else {**
          **unlink(new\_pathname);**
#ifdef DEBUG
  **if (buggit)**
            **fprintf(stderr, "...spawned child, result = %d\\r\\n", pid);**
#endif
#ifdef DEBUG
```

```
  **if (buggit)**
            **fprintf(stderr, "...PR\_START returns PS\_RES\_OK\\r\\n");**
#endif
  **ps\_res.psr\_stat = PS\_RES\_OK;**
          **return ((char \*) \&ps\_res);**
        **}**
#else /* !OS2 */
**pid = fork();**
        **if (pid == 0) {**
#ifdef DEBUG
        **if (buggit)**
                        **fprintf(stderr, "...print options =%s\\r\\n",
ps\_arg->psa\_options);**
#endif
        **if (ps\_arg->psa\_options\[1] == 'd') {**
                        **/\***
                         **\* This is a Diablo print stream. Apply the ps630**
                         **\* filter with the appropriate arguments.**
                         **\*/**
#ifdef DEBUG
                **if (buggit)**
                                **fprintf(stderr, "...run\_ps630 invoked\\r\\n");**
#endif
                **run\_ps630(new\_pathname, ps\_arg->psa\_options);**
```

```
                **}**
                **execlp(LP\_BIN,**
                        **"lpr",**
                        **"-s",**
                        **"-r",**
                        **printer\_opt,**
                        **username\_opt,**
                        **clientname\_opt,**
                        **new\_pathname,**
                        **0);**
                **perror("pcnfsd: exec lpr failed");**
                **exit(0);      /\* end of child process \*/**
        **} else if (pid == -1) {**
                **perror("pcnfsd: fork failed");**


#ifdef DEBUG
        **if (buggit)**
                        **fprintf(stderr, "...PR\_START returns PS\_RES\_FAIL\\r\\n");**
#endif
        **ps\_res.psr\_stat = PS\_RES\_FAIL;**
```

```
                **return ((char \*) \&ps\_res);**
        **} else {**



#ifdef DEBUG
        **if (buggit)**
                        **fprintf(stderr, "...forked child #%d\\r\\n", pid);**
#endif


#ifdef DEBUG
        **if (buggit)**
                        **fprintf(stderr, "...PR\_START returns PS\_RES\_OK\\r\\n");**
#endif
        **ps\_res.psr\_stat = PS\_RES\_OK;**
                **return ((char \*) \&ps\_res);**
        **}**
#endif /* OS2 */
}


char            *
mapfont(f, i, b)
**char              f;**
        **char              i;**
        **char              b;**
{
```

```
**static char     fontname\[64];**

        **fontname\[0] = 0;     /\* clear it out \*/**

        **switch (f) {**
        **case 'c':**
                **strcpy(fontname, "Courier");**
                **break;**
        **case 'h':**
                **strcpy(fontname, "Helvetica");**
                **break;**
        **case 't':**
                **strcpy(fontname, "Times");**
                **break;**
        **default:**
                **strcpy(fontname, "Times-Roman");**
                **goto exit;**
        **}**
        **if (i != 'o' \&\& b != 'b') { /\* no bold or oblique \*/**
                **if (f == 't') /\* special case Times \*/**
                        **strcat(fontname, "-Roman");**
```

```
                **goto exit;**
        **}**
        **strcat(fontname, "-");**
        **if (b == 'b')**
                **strcat(fontname, "Bold");**
        **if (i == 'o')         /\* o-blique \*/**
                **strcat(fontname, f == 't' ? "Italic" : "Oblique");**



exit:   return (&fontname[0]);
}

/*
* run_ps630 performs the Diablo 630 emulation filtering process. ps630 is
* currently broken in the Sun release: it will not accept point size or
* font changes. If your version is fixed, define the symbol
* PS630_IS_FIXED and rebuild pcnfsd.
*/


run_ps630(file, options)
**char           \*file;**
        **char           \*options;**
{
**char            tmpfile\[256];**
        **char             commbuf\[256];**
        **int              i;**
```

```
        **strcpy(tmpfile, file);**
        **strcat(tmpfile, "X"); /\* intermediate file name \*/**


#ifdef PS630_IS_FIXED
**sprintf(commbuf, "ps630 -s %c%c -p %s -f ",**
                **options\[2], options\[3], tmpfile);**
        **strcat(commbuf, mapfont(options\[4], options\[5], options\[6]));**
        **strcat(commbuf, " -F ");**
        **strcat(commbuf, mapfont(options\[7], options\[8], options\[9]));**
        **strcat(commbuf, "  ");**
        **strcat(commbuf, file);**
#else
**/\***
         **\* The pitch and font features of ps630 appear to be broken at**
         **\* this time. If you think it's been fixed at your site, define**
         **\* the compile-time symbol `ps630\_is\_fixed'.**
         **\*/**
        **sprintf(commbuf, "/usr/local/bin/ps630 -p %s %s", tmpfile, file);**
#endif

**if (i = system(commbuf)) {**
```

```
                **/\***
                 **\* Under (un)certain conditions, ps630 may return -1**
                 **\* even if it worked. Hence the commenting out of this**
                 **\* error report.**
                 **\*/**
                 **/\* fprintf(stderr, "\\r\\n\\nrun\_ps630 rc = %d\\r\\n", i) \*/ ;**
                **/\* exit(1); \*/**
        **}**
        **if (rename(tmpfile, file)) {**
                **perror("run\_ps630: rename");**
                **exit(1);**
        **}**
}
```