```
# clean

clean:;         rm -f *.o PasswordLookup* lookupd lookup z* _* core      120

grind:;         iprint READ-ME Makefile
                tgrind -lc $(HFILES) $(CFILES)
                tgrind -lpepy -d ../../pepy/grindefs $(RYFILES)
                @echo $(MANUALS) | \
                        tr " " "\012" | \
                        sed -e "s%.*%itroff -man &%" | \
                        sh -ve

true:;                                                                   130
```

Figure 11.2: Makefile for the Password Lookup Service

11.3 Interactive Initiator

The interactive initiator of the service is placed in *lookup.c*. This file is shown in Figure 11.3 starting on page 183.

```
/* lookup.c - password lookup service -- initiator */

#include <stdio.h>
#include "ryinitiator.h"                /* for generic interctive initiators */
#include "PasswordLookup-ops.h"         /* operation definitions */
#include "PasswordLookup-types.h"       /* type definitions */

/* DATA */

static char *myservice = "passwdstore";                                    10

static char *mycontext = "isode passwd lookup demo";
static char *mypci = "isode passwd lookup demo pci";


                                        /* ARGUMENTS */
int     do_lookupUser (), do_lookupUID (), do_help (), do_quit ();

                                        /* RESULTS */
int     lookup_result ();                                                  20

                                        /* ERRORS */
int     lookup_error ();


static struct dispatch dispatches[] = {
    "lookupUser", operation_PasswordLookup_lookupUser,
    do_lookupUser, free_PasswordLookup_UserName,
    lookup_result, lookup_error,
    "find user by name",                                                   30

    "lookupUID", operation_PasswordLookup_lookupUID,
    do_lookupUID, free_PasswordLookup_UserID,
    lookup_result, lookup_error,
```

```
    "find user by id",

    "help", 0,
    do_help, NULLIFP,
    NULLIFP, NULLIFP,
    "print this information",

    "quit", 0,
    do_quit, NULLIFP,
    NULLIFP, NULLIFP,
    "terminate the association and exit",

    NULL
};

/* MAIN */

/* ARGSUSED */

main (argc, argv, envp)
int       argc;
char  **argv,
     **envp;
{
    (void) ryinitiator (argc, argv, myservice, mycontext, mypci,
                        table_PasswordLookup_Operations, dispatches, do_quit);

    exit (0);                   /* NOTREACHED */
}

/* ARGUMENTS */

/* ARGSUSED */

static int  do_lookupUser (sd, ds, args, arg)
int       sd;
struct dispatch *ds;
char  **args;
register struct type_PasswordLookup_UserName **arg;
```

```
{
    char   *cp;

    if ((cp = *args++) == NULL) {
        advise (NULLCP, "usage:  lookupUser username");
        return NOTOK;
    }

    if ((*arg = str2qb (cp, strlen (cp), 1)) == NULL)
        adios (NULLCP, "out of memory");

    return OK;
}

/*  */

/* ARGSUSED */

static int  do_lookupUID (sd, ds, args, arg)
int         sd;
struct dispatch *ds;
char  **args;
register struct type_PasswordLookup_UserID **arg;
{
    char   *cp;

    if ((cp = *args++) == NULL) {
        advise (NULLCP, "usage:  lookupUID userid");
        return NOTOK;
    }

    if ((*arg = (struct type_PasswordLookup_UserID *) calloc (1, sizeof **arg))
            == NULL)
        adios (NULLCP, "out of memory");

    (*arg) -> parm = atoi (cp);

    return OK;
}
```

```
/*  */

/* ARGSUSED */

static int  do_help (sd, ds, args, dummy)
int         sd;
register struct dispatch *ds;                                   120
char  **args;
caddr_t *dummy;
{
    printf ("\nCommands are:\n");
    for (ds = dispatches; ds -> ds_name; ds++)
            printf ("%s\t%s\n", ds -> ds_name, ds -> ds_help);

    return NOTOK;
}
                                                                130
/*  */

/* ARGSUSED */

static int  do_quit (sd, ds, args, dummy)
int         sd;
struct dispatch *ds;
char  **args;
caddr_t *dummy;
{                                                               140
    struct AcSAPrelease acrs;
    register struct AcSAPrelease   *acr = &acrs;
    struct AcSAPindication  acis;
    register struct AcSAPindication *aci = &acis;
    register struct AcSAPabort *aca = &aci -> aci_abort;

    if (AcRelRequest (sd, ACF_NORMAL, NULLPEP, 0, NOTOK, acr, aci) == NOTOK)
            acs_adios (aca, "A-RELEASE.REQUEST");

    if (!acr -> acr_affirmative) {                              150
            (void) AcUAbortRequest (sd, NULLPEP, 0, aci);
```

```
        adios (NULLCP, "release rejected by peer:  %d", acr -> acr_reason);
    }

    ACRFREE (acr);

    exit (0);
}

/* RESULTS */                                                          160

/* ARGSUSED */

static int  lookup_result (sd, id, dummy, result, roi)
int         sd,
            id,
            dummy;
register struct type_PasswordLookup_Passwd *result;
struct RoSAPindication *roi;
{                                                                      170
    print_qb (result -> name);
    putchar (':');
    print_qb (result -> passwd);
    printf (":%d:%d:", result -> uid -> parm, result -> gid -> parm);
    print_qb (result -> gecos);
    putchar (':');
    print_qb (result -> dir);
    putchar (':');
    print_qb (result -> shell);
    putchar ('\n');                                                    180

    return OK;
}


static    print_qb (q)
register struct qbuf *q;
{
    register struct qbuf *p;
                                                                       190
```

```
    if (q == NULL)
        return;

    for (p = q -> qb_forw; p != q; p = p -> qb_forw)
        printf ("%*.*s", p -> qb_len, p -> qb_len, p -> qb_data);
}

/* ERRORS */

/* ARGSUSED */                                                    200

static int  lookup_error (sd, id, error, parameter, roi)
int     sd,
        id,
        error;
caddr_t parameter;
struct RoSAPindication *roi;
{
    register struct RyError *rye;
                                                                  210
    if (error == RY_REJECT) {
        advise (NULLCP, "%s", RoErrString ((int) parameter));
        return OK;
    }

    if (rye = finderrbyerr (table_PasswordLookup_Errors, error))
        advise (NULLCP, "%s",  rye -> rye_name);
    else
        advise (NULLCP, "Error %d", error);
                                                                  220
    return OK;
}
```

Figure 11.3: Initiator for the Password Lookup Service

11.4 Responder

The responder for the service is placed in *lookupd.c*. This file is shown in Figure 11.4 starting on page 189. Note that if actual network addresses were associated with the service definitions made earlier, then the responder could operate using the static discipline.

```
/* lookup_service.c - password lookup service -- responder */

#include <stdio.h>
#include <pwd.h>
#include "ryresponder.h"                /* for generic idempotent responders */
#include "PasswordLookup-ops.h"         /* operation definitions */
#include "PasswordLookup-types.h"       /* type definitions */


#define xalloc(p, type) \                                                  10
        ((p) = (type) calloc (1, sizeof *(p)))

#define salloc(s) \
        str2qb ((s), strlen (s), 1)


#ifdef  SYS5
struct passwd *getpwnam (), *getpwuid ();
#endif
                                                                           20

/* DATA */

static char *myservice = "passwdstore";

static char *mycontext = "isode passwd lookup demo";


                                        /* OPERATIONS */
int     op_lookupUser (), op_lookupUID ();                                 30

static struct dispatch dispatches[] = {
```

```
    "lookupUser", operation_PasswordLookup_lookupUser, op_lookupUser,

    "lookupUID", operation_PasswordLookup_lookupUID, op_lookupUID,

    NULL
};


/* MAIN */

/* ARGSUSED */

main (argc, argv, envp)
int       argc;
char  **argv,
     **envp;
{
    ryresponder (argc, argv, PLocalHostName (), myservice, mycontext,
                    dispatches, table_PasswordLookup_Operations,
                    NULLIFP, NULLIFP);

    exit (0);                           /* NOTREACHED */
}


/* OPERATIONS */

static int  op_lookupUser (sd, ryo, rox, in, roi)
int       sd;
struct RyOperation *ryo;
struct RoSAPinvoke *rox;
caddr_t  in;
struct RoSAPindication *roi;
{
    int     result;
    char   *cp;
    register struct type_PasswordLookup_UserName   *arg =
                    (struct type_PasswordLookup_UserName   *) in;
```

```
    if (rox -> rox_nolinked == 0) {
        advise (LLOG_NOTICE, NULLCP,
                "RO-INVOKE.INDICATION/%d:  %s, unknown linkage %d",
                sd, ryo -> ryo_name, rox -> rox_linkid);
        return ureject (sd, ROS_IP_LINKED, rox, roi);
    }
    if (debug)
        advise (LLOG_DEBUG, NULLCP, "RO-INVOKE.INDICATION/%d:  %s",
                sd, ryo -> ryo_name);

    if ((cp = qb2str (arg)) == NULL)
        result = error (sd, error_PasswordLookup_congested, (caddr_t) NULL,
                        rox, roi);
    else {
        result = lookup (sd, getpwnam (cp), rox, roi);
        free (cp);
    }

    return result;
}


static int  op_lookupUID (sd, ryo, rox, in, roi)
int         sd;
struct RyOperation *ryo;
struct RoSAPinvoke *rox;
caddr_t   in;
struct RoSAPindication *roi;
{
    register struct type_PasswordLookup_UserID    *arg =
                (struct type_PasswordLookup_UserID    *) in;

    if (rox -> rox_nolinked == 0) {
        advise (LLOG_EXCEPTIONS, NULLCP,
                "RO-INVOKE.INDICATION/%d:  %s, unknown linkage %d",
                sd, ryo -> ryo_name, rox -> rox_linkid);
        return ureject (sd, ROS_IP_LINKED, rox, roi);
    }
    if (debug)
```