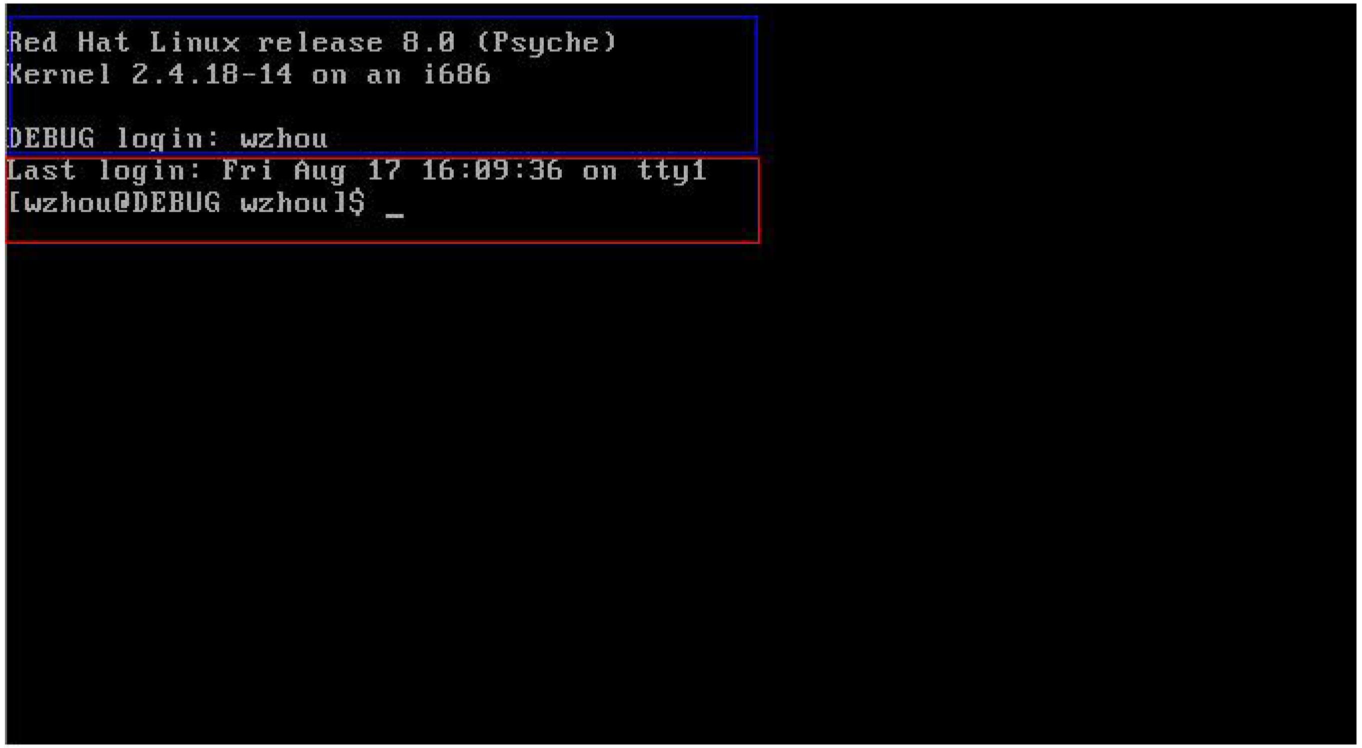
登录(login)程序分析笔记

前言

当用户在登录界面上输入用户名后，login 程序就会起来用密码来验证该用户名的合法性，接着就是一系列的与该用户相关的初始化。



```
Red Hat Linux release 8.0 (Psyche)
Kernel 2.4.18-14 on an i686

DEBUG login: wzhou
Last login: Fri Aug 17 16:09:36 on tty1
[wzhou@DEBUG wzhou]$ _
```

上图中用蓝线圈起来的是 mingetty 的输出，而红线圈起来的是 login 的输出。Mingetty 接受用户输入的用户名，然后把任务就交给了 login(注意验证用户的密码是 login 的事)。

下面是 mingetty.c 中与此相关的代码。

```
   while ((logname = get_logname ()) == 0);
   execl (_PATH_LOGIN, _PATH_LOGIN, "--", logname, NULL);
   error ("%s: can't exec " _PATH_LOGIN ": %s", tty, sys_errlist[errno]);
```

while ((logname = get_logname ()) == 0);
接受用户输入登录名，logname 即指向获得的登录名。

execl (_PATH_LOGIN, _PATH_LOGIN, "--", logname, NULL);
这里的_PATH_LOGIN 即是/bin/login，传递给它登录名 logname。

OK，接下来就是 login 的事了。

login process 伴随着系统的整个运行期间，随时恭候着用户的登录。

```
[wzhou@dcmp10 ~]$ ps aux | grep login
root     29464  0.0  0.0  4392 1032 pts/6    Ss   Sep04   0:00 login -h 13.187.243.55 -p
root     10669  0.0  0.1  3016 1220 pts/12   Ss   Sep05   0:00 login -h 13.187.243.55 -p
root     27727  0.0  0.1  4156 1168 pts/17   Ss   Sep07   0:00 login -h 13.187.243.55 -p
root     27763  0.0  0.0  3540 1032 pts/18   Ss   Sep07   0:00 login -h 13.187.243.55 -p
root     20979  0.0  0.1  3360 1232 pts/10   Ss   Sep17   0:00 login -h 13.187.243.55 -p
root      2686  0.0  0.1  2864 1228 pts/3    Ss   Sep25   0:00 login -h 13.187.243.54 -p
root      7064  0.0  0.1  2564 1232 pts/7    Ss   Sep25   0:00 login -h 13.187.243.55 -p
wzhou    11309  0.0  0.0  4992  672 pts/2    S+   13:49   0:00 grep login
[wzhou@dcmp10 ~]$
```

本笔记就是分析 `login` 到底干了点什么。

login 手册

```
NAME
      login - sign on

SYNOPSIS
      login [ name ]
      login -p
      login -h hostname
      login -f name

DESCRIPTION
      login is used when signing onto a system.  It can also be used to switch from one user to another at any time
      (most modern shells have support for this feature built into them, however).

      If an argument is not given, login prompts for the username.

      If the user is not root, and if /etc/nologin exists, the contents of this file are printed to the screen, and
      the login is terminated.  This is typically used to prevent logins when the system is being taken down.

      If  special  access restrictions are specified for the user in /etc/usertty, these must be met, or the log in
      attempt will be denied and a syslog message will be generated. See the section on  "Special  Access  Restric-
      tions".
```

```
       If  the  user  is root, then the login must be occurring on a tty listed in /etc/securetty.  Failures will be
       logged with the syslog facility.

       After these conditions have been checked, the password will be  requested  and  checked  (if  a  password  is
       required  for  this  username).   Ten  attempts are allowed before login dies, but after the first three, the
       response starts to get very slow.  Login failures are reported via the syslog  facility.   This  facility  is
       also used to report any successful root logins.

       If  the file .hushlogin exists, then a "quiet" login is performed (this disables the checking of mail and the
       printing of the last login time and message of the day).  Otherwise, if  /var/log/lastlog  exists,  the  last
       login time is printed (and the current login is recorded).

       Random administrative things, such as setting the UID and GID of the tty are performed.  The TERM environment
       variable is preserved, if it exists (other environment variables are preserved if the  -p  option  is  used).
       Then  the  HOME,  PATH,  SHELL,  TERM,  MAIL,  and  LOGNAME  environment variables are set.  PATH defaults to
       /usr/local/bin:/bin:/usr/bin          for          normal          users,          and          to
       /usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin for root.  Last, if this is not a "quiet" login,
       the message of the day is printed and the file with the user's name in /var/spool/mail will be checked, and a
       message printed if it has non-zero length.

       The  user's  shell  is  then  started.  If no shell is specified for the user in /etc/passwd, then /bin/sh is
       used.  If there is no directory specified in /etc/passwd, then / is used (the home directory is  checked  for
       the .hushlogin file described above).

OPTIONS
       -p     Used by getty(8) to tell login not to destroy the environment
```

-f Used to skip a second login authentication. This specifically does not work for root, and does not appear to work well under Linux.

-h Used by other servers (i.e., telnetd(8)) to pass the name of the remote host to login so that it may be placed in utmp and wtmp. Only the superuser may use this option.

SPECIAL ACCESS RESTRICTIONS

The file /etc/securetty lists the names of the ttys where root is allowed to log in. One name of a tty device without the /dev/ prefix must be specified on each line. If the file does not exist, root is allowed to log in on any tty.

On most modern Linux systems PAM (Pluggable Authentication Modules) is used. On systems that do not use PAM, the file /etc/usertty specifies additional access restrictions for specific users. If this file does not exist, no additional access restrictions are imposed. The file consists of a sequence of sections. There are three possible section types: CLASSES, GROUPS and USERS. A CLASSES section defines classes of ttys and host-name patterns, A GROUPS section defines allowed ttys and hosts on a per group basis, and a USERS section defines allowed ttys and hosts on a per user basis.

Each line in this file in may be no longer than 255 characters. Comments start with # character and extend to the end of the line.

The CLASSES Section

A CLASSES section begins with the word CLASSES at the start of a line in all upper case. Each following line until the start of a new section or the end of the file consists of a sequence of words separated by tabs or spaces. Each line defines a class of ttys and host patterns.

The word at the beginning of a line becomes defined as a collective name for the ttys and host patterns specified at the rest of the line. This collective name can be used in any subsequent GROUPS or USERS section. No such class name must occur as part of the definition of a class in order to avoid problems with recursive classes.

An example CLASSES section:

```
CLASSES
myclass1      tty1 tty2
myclass2      tty3 @.foo.com
```

This defines the classes myclass1 and myclass2 as the corresponding right hand sides.

The GROUPS Section

A GROUPS section defines allowed ttys and hosts on a per Unix group basis. If a user is a member of a Unix group according to /etc/passwd and /etc/group and such a group is mentioned in a GROUPS section in /etc/usertty then the user is granted access if the group is.

A GROUPS section starts with the word GROUPS in all upper case at the start of a line, and each following line is a sequence of words separated by spaces or tabs. The first word on a line is the name of the group and the rest of the words on the line specifies the ttys and hosts where members of that group are allowed access. These specifications may involve the use of classes defined in previous CLASSES sections.

An example GROUPS section.

```
GROUPS
sys      tty1 @.bar.edu
stud     myclass1 tty4
```

This example specifies that members of group sys may log in on tty1 and from hosts in the bar.edu domain. Users in group stud may log in from hosts/ttys specified in the class myclass1 or from tty4.

The USERS Section

A USERS section starts with the word USERS in all upper case at the start of a line, and each following line is a sequence of words separated by spaces or tabs. The first word on a line is a username and that user is allowed to log in on the ttys and from the hosts mentioned on the rest of the line. These specifications may involve classes defined in previous CLASSES sections. If no section header is specified at the top of the file, the first section defaults to be a USERS section.

An example USERS section:

```
USERS
zacho        tty1 @130.225.16.0/255.255.255.0
blue     tty3 myclass2
```

This lets the user zacho login only on tty1 and from hosts with IP addreses in the range 130.225.16.0 - 130.225.16.255, and user blue is allowed to log in from tty3 and whatever is specified in the class myclass2.

There may be a line in a USERS section starting with a username of *. This is a default rule and it will be applied to any user not matching any other line.

If both a USERS line and GROUPS line match a user then the user is allowed access from the union of all the ttys/hosts mentioned in these specifications.

Origins

The tty and host pattern specifications used in the specification of classes, group and user access are called origins. An origin string may have one of these formats:

- o The name of a tty device without the /dev/ prefix, for example tty1 or ttyS0.

- o The string @localhost, meaning that the user is allowed to telnet/rlogin from the local host to the same host. This also allows the user to for example run the command: xterm -e /bin/login.

- o A domain name suffix such as @.some.dom, meaning that the user may rlogin/telnet from any host whose domain name has the suffix .some.dom.

- o A range of IPv4 addresses, written @x.x.x.x/y.y.y.y where x.x.x.x is the IP address in the usual dotted quad decimal notation, and y.y.y.y is a bitmask in the same notation specifying which bits in the address to compare with the IP address of the remote host. For example @130.225.16.0/255.255.254.0 means that the user may rlogin/telnet from any host whose IP address is in the range 130.225.16.0 - 130.225.17.255.

Any of the above origins may be prefixed by a time specification according to the syntax:

```
timespec   ::= '[' <day-or-hour> [':' <day-or-hour>]* ']'
day        ::= 'mon' | 'tue' | 'wed' | 'thu' | 'fri' | 'sat' | 'sun'
hour       ::= '0' | '1' | ... | '23'
```

```
      hourspec    ::= <hour> | <hour> '-' <hour>
      day-or-hour ::= <day> | <hourspec>

      For  example,  the origin [mon:tue:wed:thu:fri:8-17]tty3 means that log in is allowed on mondays through fri-
      days between 8:00 and 17:59 (5:59 pm) on tty3.  This also shows that an hour range a-b includes  all  moments
      between  a:00 and b:59. A single hour specification (such as 10) means the time span between 10:00 and 10:59.

      Not specifying any time prefix for a tty or host means log in from that origin is allowed any  time.  If  you
      give  a time prefix be sure to specify both a set of days and one or more hours or hour ranges. A time speci-
      fication may not include any white space.

      If no default rule is given then users not matching any line /etc/usertty are allowed to log in from anywhere
      as is standard behavior.

FILES
      /var/run/utmp
      /var/log/wtmp
      /var/log/lastlog
      /var/spool/mail/*
      /etc/motd
      /etc/passwd
      /etc/nologin
      /etc/usertty
      .hushlogin

SEE ALSO
```

```
      init(8), getty(8), mail(1), passwd(1), passwd(5), environ(7), shutdown(8)

BUGS
      The undocumented BSD -r option is not supported.  This may be required by some rlogind(8) programs.

      A recursive login, as used to be possible in the good old days, no longer works; for most purposes su(1) is a
      satisfactory substitute. Indeed, for security reasons, login does a vhangup() system call to remove any  pos-
      sible  listening  processes  on the tty. This is to avoid password sniffing. If one uses the command "login",
      then the surrounding shell gets killed by vhangup() because it's no longer the true owner of the  tty.   This
      can be avoided by using "exec login" in a top-level shell or xterm.

AUTHOR
      Derived from BSD login 5.40 (5/9/89) by Michael Glad (glad@daimi.dk) for HP-UX
      Ported to Linux 0.12: Peter Orbaek (poe@daimi.aau.dk)
```

login 分析

Login 程序主要可以分为以下几个主要部分：

1. Login 首先检查登录者是否为超级用户，如果不是超级用户，并且存在/etc/nologin 文件，则输出该文件内容，并中止登录过程；主要由 checknologin（）实现
2. 如果登录用户是超级用户，那么 login 必须在/etc/securetty/中指定的 tty 列表中实现登录，否则将导致登录失败。同样可以不指定/etc/securetty 文件，此时，超级用户可以在任何 tty 上登录
3. login 接下来将提示输入登录密码（由 getpass（）调用完成），并进行验证，如果密码不对，则提示重新登录
4. 顺利经过密码验证后，login 还将检查是否存在.hushlogin 文件，如果该文件存在，则执行一次"quiet"登录（所谓的 quiet 登录指的是，登录时不再提示邮件 mail，不再显示最后一次登录时间，不输出任何消息。启动级别为 3 时，正常情况下输出这些信息）
5. login 接下来设置登录 tty 的用户 ID 和组 ID，并设置相应的环境变量，包括 HOME、PATH、SHELL、TERM、LOGNAME 等。对于普通用户来说，PATH 缺省被设置成 /usr/local/bin:/bin:/usr/bin ；对于超级用户来说，PATH 被设置成 /usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
6. login 的最后一步是为用户启动 shell。如果在/etc/passwd 中没有为用户指定 shell，那么将使用/bin/sh，如果在/etc/passwd 中没有给出当前工作目录，则使用"/"。

在 login 以后那就是你的天地了。

login 源码

```
/* This program is derived from 4.3 BSD software and is
   subject to the copyright notice below.

   The port to HP-UX has been motivated by the incapability
   of 'rlogin'/'rlogind' as per HP-UX 6.5 (and 7.0) to transfer window sizes.

   Changes:

   - General HP-UX portation. Use of facilities not available
     in HP-UX (e.g. setpriority) has been eliminated.
     Utmp/wtmp handling has been ported.

   - The program uses BSD command line options to be used
     in connection with e.g. 'rlogind' i.e. 'new login'.

   - HP features left out:      password expiry
                  '*' as login shell, add it if you need it

   - BSD features left out:       quota checks
                            password expiry
                  analysis of terminal type (tset feature)

```

```
- BSD features thrown in:       Security logging to syslogd.
                                This requires you to have a (ported) syslog
               system -- 7.0 comes with syslog

               'Lastlog' feature.

- A lot of nitty gritty details have been adjusted in favour of
  HP-UX, e.g. /etc/securetty, default paths and the environment
  variables assigned by 'login'.

- We do *nothing* to setup/alter tty state, under HP-UX this is
  to be done by getty/rlogind/telnetd/some one else.

Michael Glad (glad@daimi.dk)
Computer Science Department
Aarhus University
Denmark

1990-07-04

1991-09-24 glad@daimi.aau.dk: HP-UX 8.0 port:
- now explictly sets non-blocking mode on descriptors
- strcasecmp is now part of HP-UX

1992-02-05 poe@daimi.aau.dk: Ported the stuff to Linux 0.12
From 1992 till now (1997) this code for Linux has been maintained at
```

```
   ftp.daimi.aau.dk:/pub/linux/poe/

   1999-02-22 Arkadiusz Mi烁iewicz <misiek@pld.ORG.PL>
    - added Native Language Support
   Sun Mar 21 1999 - Arnaldo Carvalho de Melo <acme@conectiva.com.br>
    - fixed strerr(errno) in gettext calls
 */

/*
 * Copyright (c) 1980, 1987, 1988 The Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by the University of California, Berkeley.  The name of the
 * University may not be used to endorse or promote products derived
 * from this software without specific prior written permission.
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */

/*
```

```
 * login [ name ]
 * login -h hostname (for telnetd, etc.)
 * login -f name  (for pre-authenticated login: datakit, xterm, etc.)
 */

/* #define TESTING */

#ifdef TESTING
#include "param.h"
#else
#include <sys/param.h>
#endif

#include <stdio.h>
#include <ctype.h>
#include <unistd.h>
#include <getopt.h>
#include <memory.h>
#include <time.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <sys/file.h>
#include <termios.h>
#include <string.h>
#define index strchr
```

```
#define rindex strrchr
#include <sys/ioctl.h>
#include <sys/wait.h>
#include <signal.h>
#include <errno.h>
#include <grp.h>
#include <pwd.h>
#include <utmp.h>
#include <setjmp.h>
#include <stdlib.h>
#include <string.h>
#include <sys/syslog.h>
#include <sys/sysmacros.h>
#include <netdb.h>
#include "pathnames.h"
#include "my_crypt.h"
#include "login.h"
#include "xstrncpy.h"
#include "nls.h"

#ifdef __linux__
#  include <sys/sysmacros.h>
#  include <linux/major.h>
#endif

#ifdef TESTING
```

```
#  include "utmp.h"
#else
#  include <utmp.h>
#endif

#ifdef SHADOW_PWD
#  include <shadow.h>
#endif

#ifdef USE_PAM
#  include <security/pam_appl.h>
#  include <security/pam_misc.h>
#  define PAM_MAX_LOGIN_TRIES   3
#  define PAM_FAIL_CHECK if (retcode != PAM_SUCCESS) { \
       fprintf(stderr,"\n%s\n",pam_strerror(pamh, retcode)); \
       syslog(LOG_ERR,"%s",pam_strerror(pamh, retcode)); \
       pam_end(pamh, retcode); exit(1); \
   }
#  define PAM_END { \
    pam_setcred(pamh, PAM_DELETE_CRED); \
    retcode = pam_close_session(pamh,0); \
    pam_end(pamh,retcode); \
}
#endif

#ifndef __linux__
```

```
#  include <tzfile.h>
#endif
#include <lastlog.h>

#define SLEEP_EXIT_TIMEOUT 5

#ifdef __linux__
#define DO_PS_FIDDLING
#endif

#ifdef DO_PS_FIDDLING
#include "setproctitle.h"
#endif

#if 0
/* from before we had a lastlog.h file in linux */
struct  lastlog
{ long ll_time;
  char ll_line[12];
  char ll_host[16];
};
#endif

#ifndef USE_PAM
static void getloginname (void);
static void checknologin (void);
static int rootterm (char *ttyn);
#endif
static void timedout (int);
static void sigint (int);
static void motd (void);
static void dolastlog (int quiet);

#ifdef CRYPTOCARD
#include "cryptocard.h"
#endif

#ifdef KERBEROS
#include <kerberos/krb.h>
#include <sys/termios.h>
char   realm[REALM_SZ];
int kerror = KSUCCESS, notickets = 1;
#endif

#ifdef USE_TTY_GROUP
#  define TTY_MODE 0620
#else
#  define TTY_MODE 0600
#endif

#define    TTYGRPNAME "tty"      /* name of group to own ttys */

```

```
#ifndef MAXPATHLEN
#  define MAXPATHLEN 1024
#endif

/*
 * This bounds the time given to login.  Not a define so it can
 * be patched on machines where it's too small.
 */
#ifndef __linux__
int timeout = 300;
#else
int     timeout = 60;     /* used in cryptocard.c */
#endif

struct passwd *pwd;      /* used in cryptocard.c */
#if USE_PAM
static struct passwd pwdcopy;
#endif
char    hostaddress[4];      /* used in checktty.c */
char    *hostname;    /* idem */
static char   *username, *tty_name, *tty_number;
static char   thishost[100];
static int failures = 1;
static pid_t  pid;

#ifndef __linux__
```

```
struct sgttyb sgttyb;
struct tchars tc = {
    CINTR, CQUIT, CSTART, CSTOP, CEOT, CBRK
};
struct ltchars ltc = {
    CSUSP, CDSUSP, CRPRNT, CFLUSH, CWERASE, CLNEXT
};
#endif

/* Nice and simple code provided by Linus Torvalds 16-Feb-93 */
/* Nonblocking stuff by Maciej W. Rozycki, macro@ds2.pg.gda.pl, 1999.
   He writes: "Login performs open() on a tty in a blocking mode.
   In some cases it may make login wait in open() for carrier infinitely,
   for example if the line is a simplistic case of a three-wire serial
   connection. I believe login should open the line in the non-blocking mode
   leaving the decision to make a connection to getty (where it actually
   belongs). */
static void
opentty(const char * tty) {
    int i, fd, flags;

    fd = open(tty, O_RDWR | O_NONBLOCK);
    if (fd == -1) {
        syslog(LOG_ERR, _("FATAL: can't reopen tty: %s"),
              strerror(errno));
        sleep(1);
        exit(1);
    }

    flags = fcntl(fd, F_GETFL);
    flags &= ~O_NONBLOCK;
    fcntl(fd, F_SETFL, flags);

    for (i = 0; i < fd; i++)
        close(i);
    for (i = 0; i < 3; i++)
        if (fd != i)
            dup2(fd, i);
    if (fd >= 3)
        close(fd);
}

/* In case login is suid it was possible to use a hardlink as stdin
   and exploit races for a local root exploit. (Wojciech Purczynski). */
/* More precisely, the problem is  ttyn := ttyname(0); ...; chown(ttyn);
   here ttyname() might return "/tmp/x", a hardlink to a pseudotty. */
/* All of this is a problem only when login is suid, which it isnt. */
static void
check_ttyname(char *ttyn) {
    struct stat statbuf;

    if (lstat(ttyn, &statbuf)
	|| !S_ISCHR(statbuf.st_mode)
	|| (statbuf.st_nlink > 1 && strncmp(ttyn, "/dev/", 5))) {
	syslog(LOG_ERR, _("FATAL: bad tty"));
	sleep(1);
	exit(1);
    }
}

/* true if the filedescriptor fd is a console tty, very Linux specific */
static int
consoletty(int fd) {
#ifdef __linux__
    struct stat stb;

    if ((fstat(fd, &stb) >= 0)
    && (major(stb.st_rdev) == TTY_MAJOR)
    && (minor(stb.st_rdev) < 64)) {
    return 1;
    }
#endif
    return 0;
}

#if USE_PAM
/*
 * Log failed login attempts in _PATH_BTMP if that exists.
 * Must be called only with username the name of an actual user.
 * The most common login failure is to give password instead of username.
 */
#define    _PATH_BTMP "/var/log/btmp"
static void
logbtmp(const char *line, const char *username, const char *hostname) {
    struct utmp ut;

    memset(&ut, 0, sizeof(ut));

    strncpy(ut.ut_user, username ? username : "(unknown)",
        sizeof(ut.ut_user));

    strncpy(ut.ut_id, line + 3, sizeof(ut.ut_id));
    xstrncpy(ut.ut_line, line, sizeof(ut.ut_line));

#if defined(_HAVE_UT_TV)    /* in <utmpbits.h> included by <utmp.h> */
    gettimeofday(&ut.ut_tv, NULL);
#else
    {
        time_t t;
        time(&t);
        ut.ut_time = t;      /* ut_time is not always a time_t */
    }
#endif

```

```
    ut.ut_type = LOGIN_PROCESS; /* XXX doesn't matter */
    ut.ut_pid = pid;
    if (hostname) {
        xstrncpy(ut.ut_host, hostname, sizeof(ut.ut_host));
        if (hostaddress[0])
            memcpy(&ut.ut_addr, hostaddress, sizeof(ut.ut_addr));
    }
#ifdef HAVE_updwtmp      /* bad luck for ancient systems */
    updwtmp(_PATH_BTMP, &ut);
#endif
}
#endif /* USE_PAM */

int
main(int argc, char **argv)
{
    extern int optind;
    extern char *optarg, **environ;
    struct group *gr;
    register int ch;
    register char *p;
    int ask, fflag, hflag, pflag, cnt, errsv;
    int quietlog, passwd_req;
    char *domain, *ttyn;
    char tbuf[MAXPATHLEN + 2], tname[sizeof(_PATH_TTY) + 10];
    char *termenv;
    char *childArgv[10];
    char *buff;
    int childArgc = 0;
#ifdef USE_PAM
    int retcode;
    pam_handle_t *pamh = NULL;
    struct pam_conv conv = { misc_conv, NULL };
    pid_t childPid;
#else
    char *salt, *pp;
#endif
#ifdef CHOWNVCS
    char vcsn[20], vcsan[20];
#endif

    pid = getpid();

    signal(SIGALRM, timedout);
    alarm((unsigned int)timeout);
    signal(SIGQUIT, SIG_IGN);
    signal(SIGINT, SIG_IGN);

    setlocale(LC_ALL, "");
    bindtextdomain(PACKAGE, LOCALEDIR);
    textdomain(PACKAGE);

```

```
        setpriority(PRIO_PROCESS, 0, 0);
#ifdef HAVE_QUOTA
        quota(Q_SETUID, 0, 0, 0);
#endif
#ifdef DO_PS_FIDDLING
        initproctitle(argc, argv);
#endif

        /*
         * -p is used by getty to tell login not to destroy the environment
         * -f is used to skip a second login authentication
         * -h is used by other servers to pass the name of the remote
         *    host to login so that it may be placed in utmp and wtmp
         */
        gethostname(tbuf, sizeof(tbuf));
        xstrncpy(thishost, tbuf, sizeof(thishost));
        domain = index(tbuf, '.');

        username = tty_name = hostname = NULL;
        fflag = hflag = pflag = 0;
        passwd_req = 1;

        while ((ch = getopt(argc, argv, "fh:p")) != -1)
          switch (ch) {       login 接收三个选项：
        case 'f':             -p    Used by getty(8) to tell login not to destroy the environment
          fflag = 1;          -f    Used  to  skip  a second login authentication. This specifically does not work
```

```
      break;                  for root, and does not  appear to work well under Linux.
                         -h    Used by other servers (i.e., telnetd(8)) to pass the name of the remote host to
    case 'h':                 login so that  it  may be placed in utmp and wtmp.  Only the superuser may use this
      if (getuid()) {         option.
          fprintf(stderr,
              _("login: -h for super-user only.\n"));
          exit(1);
      }
      hflag = 1;
      if (domain && (p = index(optarg, '.')) &&
          strcasecmp(p, domain) == 0)
        *p = 0;

      hostname = strdup(optarg);    /* strdup: Ambrose C. Li */
      {
          struct hostent *he = gethostbyname(hostname);

          /* he points to static storage; copy the part we use */
          hostaddress[0] = 0;
          if (he && he->h_addr_list && he->h_addr_list[0])
              memcpy(hostaddress, he->h_addr_list[0],
                 sizeof(hostaddress));
      }
      break;

    case 'p':
      pflag = 1;
      break;

    case '?':
    default:
      fprintf(stderr,
          _("usage: login [-fp] [username]\n"));
      exit(1);
      }
    argc -= optind;
    argv += optind;
    if (*argv) {                    检查 login 时是否提供了用户名，比如是 login 还是 login wzhou
    char *p = *argv;
    username = strdup(p);           获得登录的用户名
    ask = 0;
    /* wipe name - some people mistype their password here */
    /* (of course we are too late, but perhaps this helps a little ..) */
    while(*p)
        *p++ = ' ';
    } else
        ask = 1;                 运行 login 时没有提供用户名，则置需要向用户提问要用户名的标志

    for (cnt = getdtablesize(); cnt > 2; cnt--)
      close(cnt);

    ttyn = ttyname(0);              返回 login process file handle 0(标准输入)对应的 tty 设备名

    if (ttyn == NULL || *ttyn == '\0') {
    /* no snprintf required - see definition of tname */
    sprintf(tname, "%s??", _PATH_TTY);
    ttyn = tname;
    }

    check_ttyname(ttyn);

    if (strncmp(ttyn, "/dev/", 5) == 0)
    tty_name = ttyn+5;
    else
    tty_name = ttyn;

    if (strncmp(ttyn, "/dev/tty", 8) == 0)
    tty_number = ttyn+8;
    else {
    char *p = ttyn;
    while (*p && !isdigit(*p)) p++;
    tty_number = p;
    }

#ifdef CHOWNVCS
    /* find names of Virtual Console devices, for later mode change */
    snprintf(vcsn, sizeof(vcsn), "/dev/vcs%s", tty_number);
    snprintf(vcsan, sizeof(vcsan), "/dev/vcsa%s", tty_number);
```

```
#endif

    /* set pgid to pid */
    setpgrp();
    /* this means that setsid() will fail */

    {
    struct termios tt, ttt;

    tcgetattr(0, &tt);
    ttt = tt;
    ttt.c_cflag &= ~HUPCL;

    /* These can fail, e.g. with ttyn on a read-only filesystem */
    chown(ttyn, 0, 0);
    chmod(ttyn, TTY_MODE);

    /* Kill processes left on this tty */
    tcsetattr(0,TCSAFLUSH,&ttt);
    signal(SIGHUP, SIG_IGN); /* so vhangup() wont kill us */       忽略 SIGHUP signal
    vhangup();
    signal(SIGHUP, SIG_DFL);

    /* open stdin,stdout,stderr to the tty */
    opentty(ttyn);              该函数使得 file handle 的 0, 1, 2 都指向 tty

```

```
517      /* restore tty modes */
518      tcsetattr(0,TCSAFLUSH,&tt);
519      }
520
521      openlog("login", LOG_ODELAY, LOG_AUTHPRIV);
522
523  #if 0
524      /* other than iso-8859-1 */
525      printf("\033(K");
526      fprintf(stderr,"\033(K");
527  #endif
528
529  #ifdef USE_PAM
530      /*
531       * username is initialized to NULL
532       * and if specified on the command line it is set.
533       * Therefore, we are safe not setting it to anything
534       */
535
536      retcode = pam_start("login",username, &conv, &pamh);
537      if(retcode != PAM_SUCCESS) {
538      fprintf(stderr, _("login: PAM Failure, aborting: %s\n"),
539          pam_strerror(pamh, retcode));
540      syslog(LOG_ERR, _("Couldn't initialize PAM: %s"),
541             pam_strerror(pamh, retcode));
542      exit(99);
```

L529 到 L692 都是 PAM 相关的认证，关于 PAM 另文介绍，这是 Linux 凌乱的安全体系中的一种很灵活的安全架构。本文只分析经典（也是原始）的认证

```
    }
    /* hostname & tty are either set to NULL or their correct values,
       depending on how much we know */
    retcode = pam_set_item(pamh, PAM_RHOST, hostname);
    PAM_FAIL_CHECK;
    retcode = pam_set_item(pamh, PAM_TTY, tty_name);
    PAM_FAIL_CHECK;

    /*
     * Andrew.Taylor@cal.montage.ca: Provide a user prompt to PAM
     * so that the "login: " prompt gets localized. Unfortunately,
     * PAM doesn't have an interface to specify the "Password: " string
     * (yet).
     */
    retcode = pam_set_item(pamh, PAM_USER_PROMPT, _("login: "));
    PAM_FAIL_CHECK;

#if 0
    /*
     * other than iso-8859-1
     * one more time due to reset tty by PAM
     */
    printf("\033(K");
    fprintf(stderr,"\033(K");
#endif

```

```
    /* if fflag == 1, then the user has already been authenticated */
    if (fflag && (getuid() == 0))
    passwd_req = 0;
    else
    passwd_req = 1;

    if(passwd_req == 1) {
    int failcount=0;

    /* if we didn't get a user on the command line, set it to NULL */
    pam_get_item(pamh,  PAM_USER, (const void **) &username);
    if (!username)
        pam_set_item(pamh, PAM_USER, NULL);

    /* there may be better ways to deal with some of these
       conditions, but at least this way I don't think we'll
       be giving away information... */
    /* Perhaps someday we can trust that all PAM modules will
       pay attention to failure count and get rid of MAX_LOGIN_TRIES? */

    retcode = pam_authenticate(pamh, 0);
    while((failcount++ < PAM_MAX_LOGIN_TRIES) &&
          ((retcode == PAM_AUTH_ERR) ||
           (retcode == PAM_USER_UNKNOWN) ||
           (retcode == PAM_CRED_INSUFFICIENT) ||
           (retcode == PAM_AUTHINFO_UNAVAIL))) {
```

```
       pam_get_item(pamh, PAM_USER, (const void **) &username);

       syslog(LOG_NOTICE,_("FAILED LOGIN %d FROM %s FOR %s, %s"),
          failcount, hostname, username, pam_strerror(pamh, retcode));
       logbtmp(tty_name, username, hostname);

       fprintf(stderr,_("Login incorrect\n\n"));
       pam_set_item(pamh,PAM_USER,NULL);
       retcode = pam_authenticate(pamh, 0);
    }

    if (retcode != PAM_SUCCESS) {
       pam_get_item(pamh, PAM_USER, (const void **) &username);

       if (retcode == PAM_MAXTRIES)
       syslog(LOG_NOTICE,_("TOO MANY LOGIN TRIES (%d) FROM %s FOR "
           "%s, %s"), failcount, hostname, username,
            pam_strerror(pamh, retcode));
       else
       syslog(LOG_NOTICE,_("FAILED LOGIN SESSION FROM %s FOR %s, %s"),
           hostname, username, pam_strerror(pamh, retcode));
       logbtmp(tty_name, username, hostname);

       fprintf(stderr,_("\nLogin incorrect\n"));
       pam_end(pamh, retcode);
       exit(0);
```

```
        }

        retcode = pam_acct_mgmt(pamh, 0);

        if(retcode == PAM_NEW_AUTHTOK_REQD) {
            retcode = pam_chauthtok(pamh, PAM_CHANGE_EXPIRED_AUTHTOK);
        }

        PAM_FAIL_CHECK;
    }

    /*
     * Grab the user information out of the password file for future usage
     * First get the username that we are actually using, though.
     */
    retcode = pam_get_item(pamh, PAM_USER, (const void **) &username);
    PAM_FAIL_CHECK;

    if (!username || !*username) {
        fprintf(stderr, _("\nSession setup problem, abort.\n"));
        syslog(LOG_ERR, _("NULL user name in %s:%d. Abort."),
           __FUNCTION__, __LINE__);
        pam_end(pamh, PAM_SYSTEM_ERR);
        exit(1);
    }
    if (!(pwd = getpwnam(username))) {
        fprintf(stderr, _("\nSession setup problem, abort.\n"));
        syslog(LOG_ERR, _("Invalid user name \"%s\" in %s:%d. Abort."),
           username, __FUNCTION__, __LINE__);
        pam_end(pamh, PAM_SYSTEM_ERR);
        exit(1);
    }

    /*
     * Create a copy of the pwd struct - otherwise it may get
     * clobbered by PAM
     */
    memcpy(&pwdcopy, pwd, sizeof(*pwd));
    pwd = &pwdcopy;
    pwd->pw_name   = strdup(pwd->pw_name);
    pwd->pw_passwd = strdup(pwd->pw_passwd);
    pwd->pw_gecos  = strdup(pwd->pw_gecos);
    pwd->pw_dir    = strdup(pwd->pw_dir);
    pwd->pw_shell  = strdup(pwd->pw_shell);
    if (!pwd->pw_name || !pwd->pw_passwd || !pwd->pw_gecos ||
    !pwd->pw_dir || !pwd->pw_shell) {
        fprintf(stderr, _("login: Out of memory\n"));
        syslog(LOG_ERR, "Out of memory");
        pam_end(pamh, PAM_SYSTEM_ERR);
        exit(1);
    }
    username = pwd->pw_name;
```

```

        /*
         * Initialize the supplementary group list.
         * This should be done before pam_setcred because
         * the PAM modules might add groups during pam_setcred.
         */
        if (initgroups(username, pwd->pw_gid) < 0) {
            syslog(LOG_ERR, "initgroups: %m");
            fprintf(stderr, _("\nSession setup problem, abort.\n"));
            pam_end(pamh, PAM_SYSTEM_ERR);
            exit(1);
        }

        retcode = pam_open_session(pamh, 0);
        PAM_FAIL_CHECK;

        retcode = pam_setcred(pamh, PAM_ESTABLISH_CRED);
        PAM_FAIL_CHECK;

    #else /* ! USE_PAM */       非 PAM 认证

        for (cnt = 0;; ask = 1) {

        if (ask) {
            fflag = 0;
            getloginname();       在终端上输出“ login: ”,获得用户登录名，由全局变量 username
```

```
}

/* Dirty patch to fix a gigantic security hole when using
   yellow pages. This problem should be solved by the
   libraries, and not by programs, but this must be fixed
   urgently! If the first char of the username is '+', we
   avoid login success.
   Feb 95 <alvaro@etsit.upm.es> */

if (username[0] == '+') {
    puts(_("Illegal username"));
    badlogin(username);
    sleepexit(1);
}

/* (void)strcpy(tbuf, username); why was this here? */
if ((pwd = getpwnam(username))) {     从/etc/passwd 中获得与登录用户相关信息，存放在 pwd（struct passwd 结构中

 struct passwd {
      char    *pw_name;      /* user name */
      char    *pw_passwd;    /* user password */
      uid_t   pw_uid;        /* user id */
      gid_t   pw_gid;        /* group id */
      char    *pw_gecos;     /* real name */
      char    *pw_dir;       /* home directory */
      char    *pw_shell;     /* shell program */
```

```
            };

)
#  ifdef SHADOW_PWD                        从/etc/shadow 中获得与登录用户相关信息
        struct spwd *sp;

        if ((sp = getspnam(username)))
```

```
struct spwd
{
  char *sp_namp;               /* login name */
  char *sp_pwdp;               /* encrypted password */
  sptime sp_lstchg;             /* date of last change */
  sptime sp_min;               /* minimum number of days between changes */
  sptime sp_max;               /* maximum number of days between changes */
  sptime sp_warn;              /* number of days of warning before password
                               expires */
  sptime sp_inact;             /* number of days after password expires
                               until the account becomes unusable. */
  sptime sp_expire;             /* days since 1/1/70 until account expires*/
  unsigned long sp_flag;        /* reserved for future use */
};
```

```
          pwd->pw_passwd = sp->sp_pwdp;
#  endif
        salt = pwd->pw_passwd;
    } else
      salt = "xx";

    if (pwd) {
        initgroups(username, pwd->pw_gid);
        checktty(username, tty_name, pwd); /* in checktty.c */
    }

    /* if user not super-user, check for disabled logins */
    if (pwd == NULL || pwd->pw_uid)
      checknologin();     检查有无/etc/nologin 文件，如果有则表示禁止该用户登录，输出/etc/nologin 中的内容

    /*
     * Disallow automatic login to root; if not invoked by
     * root, disallow if the uid's differ.
     */
    if (fflag && pwd) {
        int uid = getuid();

        passwd_req = pwd->pw_uid == 0 ||
          (uid && uid != pwd->pw_uid);
    }

```

```
/*
 * If trying to log in as root, but with insecure terminal,
 * refuse the login attempt.
 */
if (pwd && pwd->pw_uid == 0 && !rootterm(tty_name)) {
   fprintf(stderr,
       _("%s login refused on this terminal.\n"),
       pwd->pw_name);

   if (hostname)
     syslog(LOG_NOTICE,
        _("LOGIN %s REFUSED FROM %s ON TTY %s"),
        pwd->pw_name, hostname, tty_name);
   else
     syslog(LOG_NOTICE,
        _("LOGIN %s REFUSED ON TTY %s"),
        pwd->pw_name, tty_name);
   continue;
}

/*
 * If no pre-authentication and a password exists
 * for this user, prompt for one and verify it.
 */
if (!passwd_req || (pwd && !*pwd->pw_passwd))
  break;
```

如果是 root 用户登录则检查是否是通过安全的 tty 登录，不是则拒绝登录。所谓安全的 tty 登录就是是否在文件/etc/securetty 中被记录了

如果密码为空，也可以，就像
root::0:0:root:/root:/bin/bash

```
                                                          中的密码项为空
    setpriority(PRIO_PROCESS, 0, -4);                     提升本 process 的优先级(为了获得密码，还特意提升进程优先级)
    pp = getpass(_("Password: "));                        在终端上显示“Password: ”，获得用户输入的密码，由 pp 指向

#  ifdef CRYPTOCARD
    if (strncmp(pp, "CRYPTO", 6) == 0) {
        if (pwd && cryptocard()) break;
    }
#  endif /* CRYPTOCARD */

    p = crypt(pp, salt);                                  对用户输入的 passwd 进行加密，获得密文
    setpriority(PRIO_PROCESS, 0, 0);                      恢复本 process 的优先级

#  ifdef KERBEROS                    Kerberos 认证
    /*
     * If not present in pw file, act as we normally would.
     * If we aren't Kerberos-authenticated, try the normal
     * pw file for a password.  If that's ok, log the user
     * in without issueing any tickets.
     */

    if (pwd && !krb_get_lrealm(realm,1)) {
        /*
         * get TGT for local realm; be careful about uid's
         * here for ticket file ownership
         */
```

```
798           setreuid(geteuid(),pwd->pw_uid);
799           kerror = krb_get_pw_in_tkt(pwd->pw_name, "", realm,
800                           "krbtgt", realm, DEFAULT_TKT_LIFE, pp);
801           setuid(0);
802           if (kerror == INTK_OK) {
803           memset(pp, 0, strlen(pp));
804           notickets = 0;    /* user got ticket */
805           break;
806           }
807       }
808   #  endif /* KERBEROS */
809       memset(pp, 0, strlen(pp));
810
811       if (pwd && !strcmp(p, pwd->pw_passwd))          比较两个密文是否相同
812         break;                                        相同则 break 出 L694 行的循环，即认证通过，到 L829 行运行
813
814       printf(_("Login incorrect\n"));                 输出“ Login incorrect ”，表示密码不对
815       badlogin(username); /* log ALL bad logins */    在 syslog 中记录登录失败记录
816       failures++;
817
818       /* we allow 10 tries, but after 3 we start backing off */
819       if (++cnt > 3) {
820           if (cnt >= 10) {
821           sleepexit(1);                               超过 10 次登录都失败，则退出 login
822           }
823           sleep((unsigned int)((cnt - 3) * 5));
```

```
    }
    }
#endif /* !USE_PAM */
登录用户合法
    /* committed to login -- turn off timeout */
    alarm((unsigned int)0);             关闭所有闹钟

#ifdef HAVE_QUOTA           如果系统对登录用户数有限制
    if (quota(Q_SETUID, pwd->pw_uid, 0, 0) < 0 && errno != EINVAL) {
    switch(errno) {
      case EUSERS:          有太多登录用户数了
        fprintf(stderr,
            _("Too many users logged on already.\nTry again later.\n"));
        break;
      case EPROCLIM:
        fprintf(stderr,
            _("You have too many processes running.\n"));
        break;
      default:
        perror("quota (Q_SETUID)");
    }
    sleepexit(0);    /* %% */
    }
#endif

    /* paranoia... */
```

```
#ifdef SHADOW_PWD
    endspent();
#endif
    endpwent();
    当运行到这儿时，常规情况下，运行 login 的是 account 是 root(如果你手工在命令行上输入 login，那另当别论)
    /* This requires some explanation: As root we may not be able to
       read the directory of the user if it is on an NFS mounted
       filesystem. We temporarily set our effective uid to the user-uid
       making sure that we keep root privs. in the real uid.

       A portable solution would require a fork(), but we rely on Linux
       having the BSD setreuid() */

    {
    char tmpstr[MAXPATHLEN];
    uid_t ruid = getuid();
    gid_t egid = getegid();

    /* avoid snprintf - old systems do not have it, or worse,
       have a libc in which snprintf is the same as sprintf */
    if (strlen(pwd->pw_dir) + sizeof(_PATH_HUSHLOGIN) + 2 > MAXPATHLEN)
        quietlog = 0;
    else {
    在 home 目录下查找.hushlogin 文件
        sprintf(tmpstr, "%s/%s", pwd->pw_dir, _PATH_HUSHLOGIN);
        setregid(-1, pwd->pw_gid);
```

```
            setreuid(0, pwd->pw_uid);
            quietlog = (access(tmpstr, R_OK) == 0);
            setuid(0); /* setreuid doesn't do it alone! */
            setreuid(ruid, 0);
            setregid(-1, egid);
        }
    }

    /* for linux, write entries in utmp and wtmp */
```

检查 home 目录下有无.hushlogin 文件。然后置是否安静登录标志

utmp 文件用于记录当前系统用户是哪些人及相关信息，这些信息由下面的结构表示

```
struct utmp {
  short ut_type;             /* type of login */
  pid_t ut_pid;              /* PID of login process */
  char ut_line[UT_LINESIZE];  /* device name of tty - "/dev/" */
  char ut_id[4];             /* init id or abbrev. ttyname */
  char ut_user[UT_NAMESIZE];  /* user name */
  char ut_host[UT_HOSTSIZE];  /* hostname for remote login */
  struct exit_status ut_exit; /* The exit status of a process
                               marked as DEAD_PROCESS */

  /* The ut_session and ut_tv fields must be the same size when
     compiled 32- and 64-bit.  This allows data files and shared
     memory to be shared between 32- and 64-bit applications */
#if __WORDSIZE == 64 && defined __WORDSIZE_COMPAT32
```

```
      int32_t ut_session;        /* Session ID, used for windowing */
      struct {
        int32_t tv_sec;          /* Seconds */
        int32_t tv_usec;         /* Microseconds */
      } ut_tv;                   /* Time entry was made */
    #else
       long int ut_session;       /* Session ID, used for windowing */
       struct timeval ut_tv;      /* Time entry was made */
    #endif

      int32_t ut_addr_v6[4];      /* IP address of remote host */
      char __unused[20];          /* Reserved for future use */
    };

      {
      struct utmp ut;
      struct utmp *utp;

      utmpname(_PATH_UTMP);           设置 utmp 的文件路径为/var/run/utmp
      setutent();

      /* Find pid in utmp.
   login sometimes overwrites the runlevel entry in /var/run/utmp,
   confusing sysvinit. I added a test for the entry type, and the problem
   was gone. (In a runlevel entry, st_pid is not really a pid but some number
   calculated from the previous and current runlevel).
```

```
Michael Riepe <michael@stud.uni-hannover.de>
    */
```

下面就是填充这些信息，然后写入 utmp 文件

```
    while ((utp = getutent()))
        if (utp->ut_pid == pid
            && utp->ut_type >= INIT_PROCESS
            && utp->ut_type <= DEAD_PROCESS)
            break;

    /* If we can't find a pre-existing entry by pid, try by line.
       BSD network daemons may rely on this. (anonymous) */
    if (utp == NULL) {
        setutent();
        ut.ut_type = LOGIN_PROCESS;
        strncpy(ut.ut_line, tty_name, sizeof(ut.ut_line));
        utp = getutline(&ut);
    }

    if (utp) {
       memcpy(&ut, utp, sizeof(ut));
    } else {
       /* some gettys/telnetds don't initialize utmp... */
       memset(&ut, 0, sizeof(ut));
    }

    if (ut.ut_id[0] == 0)
      strncpy(ut.ut_id, tty_number, sizeof(ut.ut_id));

    strncpy(ut.ut_user, username, sizeof(ut.ut_user));
    xstrncpy(ut.ut_line, tty_name, sizeof(ut.ut_line));
#ifdef _HAVE_UT_TV      /* in <utmpbits.h> included by <utmp.h> */
    gettimeofday(&ut.ut_tv, NULL);
#else
    {
       time_t t;
       time(&t);
       ut.ut_time = t;   /* ut_time is not always a time_t */
               /* glibc2 #defines it as ut_tv.tv_sec */
    }
#endif
    ut.ut_type = USER_PROCESS;
    ut.ut_pid = pid;
    if (hostname) {
       xstrncpy(ut.ut_host, hostname, sizeof(ut.ut_host));
       if (hostaddress[0])
          memcpy(&ut.ut_addr, hostaddress, sizeof(ut.ut_addr));
    }

    pututline(&ut);
    endutent();
```

```

#ifdef HAVE_updwtmp
    updwtmp(_PATH_WTMP, &ut);
#else
#if 0
    /* The O_APPEND open() flag should be enough to guarantee
      atomic writes at end of file. */
    {
       int wtmp;

       if((wtmp = open(_PATH_WTMP, O_APPEND|O_WRONLY)) >= 0) {
       write(wtmp, (char *)&ut, sizeof(ut));
       close(wtmp);
       }
    }
#else
    /* Probably all this locking below is just nonsense,
      and the short version is OK as well. */
    {
       int lf, wtmp;
       if ((lf = open(_PATH_WTMPLOCK, O_CREAT|O_WRONLY, 0660)) >= 0) {
       flock(lf, LOCK_EX);
       if ((wtmp = open(_PATH_WTMP, O_APPEND|O_WRONLY)) >= 0) {       wtmp 的路径为/var/log/wtmp
          write(wtmp, (char *)&ut, sizeof(ut));
          close(wtmp);
       }
```

```
                flock(lf, LOCK_UN);
                close(lf);
                }
            }
        #endif
        #endif
            }

            dolastlog(quietlog);            report the most recent login of all users or of a given user
                                            /var/log/lastlog
            chown(ttyn, pwd->pw_uid,        改变当前 login 用到的终端的 owner 为登录用户所有
              (gr = getgrnam(TTYGRPNAME)) ? gr->gr_gid : pwd->pw_gid);
            chmod(ttyn, TTY_MODE);

        #ifdef CHOWNVCS
            /* if tty is one of the VC's then change owner and mode of the
               special /dev/vcs devices as well */
            if (consoletty(0)) {
            chown(vcsn, pwd->pw_uid, (gr ? gr->gr_gid : pwd->pw_gid));
            chown(vcsan, pwd->pw_uid, (gr ? gr->gr_gid : pwd->pw_gid));
            chmod(vcsn, TTY_MODE);
            chmod(vcsan, TTY_MODE);
            }
        #endif

            setgid(pwd->pw_gid);                       改为登录用户的 GID
```

```

#ifdef HAVE_QUOTA
    quota(Q_DOWARN, pwd->pw_uid, (dev_t)-1, 0);
#endif

    if (*pwd->pw_shell == '\0')              如果在/etc/passwd 中没有制定登录 shell，则设为/bin/sh
      pwd->pw_shell = _PATH_BSHELL;

    /* preserve TERM even without -p flag */
    {
    char *ep;

    if(!((ep = getenv("TERM")) && (termenv = strdup(ep))))
      termenv = "dumb";
    }

    /* destroy environment unless user has requested preservation */
    if (!pflag)
      {
          environ = (char**)malloc(sizeof(char*));
      memset(environ, 0, sizeof(char*));
      }
    设置如下环境变量（HOME，SHELL，TERM，LOGNAME 等）

当我们敲入 env 后看到的环境变量有些就是在这里设的
```

```
    setenv("HOME", pwd->pw_dir, 0);      /* legal to override */
    if(pwd->pw_uid)
      setenv("PATH", _PATH_DEFPATH, 1);         /usr/local/bin:/bin:/usr/bin
    else
      setenv("PATH", _PATH_DEFPATH_ROOT, 1);   对 root 而言
                                     /usr/local/sbin:/usr/local/bin:/sbin:/bin: /usr/sbin:/usr/bin
    setenv("SHELL", pwd->pw_shell, 1);      登录 shell 的路径
    setenv("TERM", termenv, 1);

    /* mailx will give a funny error msg if you forget this one */
    {
      char tmp[MAXPATHLEN];
      /* avoid snprintf */
      if (sizeof(_PATH_MAILDIR) + strlen(pwd->pw_name) + 1 < MAXPATHLEN) {
          sprintf(tmp, "%s/%s", _PATH_MAILDIR, pwd->pw_name);
          setenv("MAIL",tmp,0);      用户的信箱在“/spool/mail/用户名”下面
      }
    }

    /* LOGNAME is not documented in login(1) but
       HP-UX 6.5 does it. We'll not allow modifying it.
       */
    setenv("LOGNAME", pwd->pw_name, 1);

#ifdef USE_PAM
    {
```

```
    int i;
    char ** env = pam_getenvlist(pamh);

    if (env != NULL) {
       for (i=0; env[i]; i++) {
       putenv(env[i]);
       /* D(("env[%d] = %s", i,env[i])); */
       }
    }
    }
#endif

#ifdef DO_PS_FIDDLING
    setproctitle("login", username);
#endif

    if (!strncmp(tty_name, "ttyS", 4))
      syslog(LOG_INFO, _("DIALUP AT %s BY %s"), tty_name, pwd->pw_name);

    /* allow tracking of good logins.
      -steve philp (sphilp@mail.alliance.net) */

    if (pwd->pw_uid == 0) {
    if (hostname)
      syslog(LOG_NOTICE, _("ROOT LOGIN ON %s FROM %s"),
         tty_name, hostname);
    else
      syslog(LOG_NOTICE, _("ROOT LOGIN ON %s"), tty_name);
    } else {
    if (hostname)
      syslog(LOG_INFO, _("LOGIN ON %s BY %s FROM %s"), tty_name,
        pwd->pw_name, hostname);
    else
      syslog(LOG_INFO, _("LOGIN ON %s BY %s"), tty_name,
        pwd->pw_name);
    }

    if (!quietlog) {        如果 home 目录下无.hushlogin 文件，则表示正常登录（非安静登录）
    motd();                 所谓的安静登录指的是，登录时不再提示邮件mail，不再显示最后一次登录时间，不输出任何消息
                            mod()读取/etc/motd 文件中的内容显示，一般为欢迎词等
#ifdef DO_STAT_MAIL
    /*                      检查有无新 mail，有则提示
     * This turns out to be a bad idea: when the mail spool
     * is NFS mounted, and the NFS connection hangs, the
     * login hangs, even root cannot login.
     * Checking for mail should be done from the shell.
     */
    {  检查在登录用户的信箱里有否新 mail
       struct stat st;
       char *mail;

       mail = getenv("MAIL");
```

```
        if (mail && stat(mail, &st) == 0 && st.st_size != 0) {
        if (st.st_mtime > st.st_atime)
            printf(_("You have new mail.\n"));
        else
            printf(_("You have mail.\n"));
        }
    }
#endif
    }

    signal(SIGALRM, SIG_DFL);
    signal(SIGQUIT, SIG_DFL);
    signal(SIGTSTP, SIG_IGN);

#ifdef USE_PAM
    /*
     * We must fork before setuid() because we need to call
     * pam_close_session() as root.
     */

    childPid = fork();          login process 孕育新进程
    if (childPid < 0) {
      int errsv = errno;
      /* error in fork() */
      fprintf(stderr, _("login: failure forking: %s"), strerror(errsv));
      PAM_END;
      exit(0);
    }

    if (childPid) {             login 父进程
      /* parent - wait for child to finish, then cleanup session */
      signal(SIGHUP, SIG_IGN);
      signal(SIGINT, SIG_IGN);
      signal(SIGQUIT, SIG_IGN);
      signal(SIGTSTP, SIG_IGN);
      signal(SIGTTIN, SIG_IGN);
      signal(SIGTTOU, SIG_IGN);

      wait(NULL);         login 父进程就是 suspend，等待子进程的结束
      PAM_END;
      exit(0);
    }

    /* child */         下面的代码在 login 的子进程中执行
    /*
     * Problem: if the user's shell is a shell like ash that doesnt do
     * setsid() or setpgrp(), then a ctrl-\, sending SIGQUIT to every
     * process in the pgrp, will kill us.
     */

    /* start new session */
    setsid();

    /* make sure we have a controlling tty */
    opentty(ttyn);        把 login 子进程的 file handle 0，1，2 指向父进程的 tty 设备（即父子进程公有该 tty 设备）
    openlog("login", LOG_ODELAY, LOG_AUTHPRIV);   /* reopen */

    /*
     * TIOCSCTTY: steal tty from other process group.
     */
    if (ioctl(0, TIOCSCTTY, 1))
        syslog(LOG_ERR, _("TIOCSCTTY failed: %m"));
#endif
    signal(SIGINT, SIG_DFL);

    /* discard permissions last so can't get killed and drop core */
    if(setuid(pwd->pw_uid) < 0 && pwd->pw_uid) {      子进程的 UID 与 GID 还为 root，现在把它设成登录用户的 UID 与 GID
    syslog(LOG_ALERT, _("setuid() failed"));
    exit(1);
    }

    /* wait until here to change directory! */
    if (chdir(pwd->pw_dir) < 0) {                         把当前目录切换到登录用户的 home 目录
    printf(_("No directory %s!\n"), pwd->pw_dir);
    if (chdir("/"))                                       如果登录用户没有设 home 目录，则设为根目录
      exit(0);
    pwd->pw_dir = "/";
    printf(_("Logging in with home = \"/\".\n"));
}
下面就是 login 进程的最后一步，就是执行用户的登录 shell
/* if the shell field has a space: treat it like a shell script */
if (strchr(pwd->pw_shell, ' ')) {      如果指向登录 shell 的字符串中有空格
buff = malloc(strlen(pwd->pw_shell) + 6);

if (!buff) {
    fprintf(stderr, _("login: no memory for shell script.\n"));
    exit(0);
}

strcpy(buff, "exec ");
strcat(buff, pwd->pw_shell);
childArgv[childArgc++] = "/bin/sh";
childArgv[childArgc++] = "-sh";
childArgv[childArgc++] = "-c";
childArgv[childArgc++] = buff;            “/bin/sh -sh -c exec pwd->pw_shell”
} else {
tbuf[0] = '-';
xstrncpy(tbuf + 1, ((p = rindex(pwd->pw_shell, '/')) ?
            p + 1 : pwd->pw_shell),
    sizeof(tbuf)-1);

childArgv[childArgc++] = pwd->pw_shell;
childArgv[childArgc++] = tbuf;            “pwd->pw_shell”
}
```

```

    childArgv[childArgc++] = NULL;

    execvp(childArgv[0], childArgv + 1);      login 子进程执行登录 shell

    errsv = errno;                            不应该在执行到这儿，否则那就是错了

    if (!strcmp(childArgv[0], "/bin/sh"))
    fprintf(stderr, _("login: couldn't exec shell script: %s.\n"),
        strerror(errsv));
    else
    fprintf(stderr, _("login: no shell: %s.\n"), strerror(errsv));

    exit(0);
}

#ifndef USE_PAM
static void
getloginname(void) {                    向用户询问登录名
    int ch, cnt, cnt2;
    char *p;
    static char nbuf[UT_NAMESIZE + 1];

    cnt2 = 0;
    for (;;) {
    cnt = 0;
```

```
       printf(_("\n%s login: "), thishost); fflush(stdout);
       for (p = nbuf; (ch = getchar()) != '\n'; ) {
           if (ch == EOF) {
           badlogin("EOF");
           exit(0);
           }
           if (p < nbuf + UT_NAMESIZE)
             *p++ = ch;

           cnt++;
           if (cnt > UT_NAMESIZE + 20) {
           fprintf(stderr, _("login name much too long.\n"));
           badlogin(_("NAME too long"));
           exit(0);
           }
       }
       if (p > nbuf) {
         if (nbuf[0] == '-')
           fprintf(stderr,
               _("login names may not start with '-'.\n"));
         else {
             *p = '\0';
             username = nbuf;       登录名由全局变量指向
             break;
         }
       }
```

```

	cnt2++;
	if (cnt2 > 50) {
	    fprintf(stderr, _("too many bare linefeeds.\n"));
	    badlogin(_("EXCESSIVE linefeeds"));
	    exit(0);
	}
	}
}
#endif

/*
 * Robert Ambrose writes:
 * A couple of my users have a problem with login processes hanging around
 * soaking up pts's.  What they seem to hung up on is trying to write out the
 * message 'Login timed out after %d seconds' when the connection has already
 * been dropped.
 * What I did was add a second timeout while trying to write the message so
 * the process just exits if the second timeout expires.
 */

static void
timedout2(int sig) {
	struct termio ti;

	/* reset echo */
```

```
    ioctl(0, TCGETA, &ti);
    ti.c_lflag |= ECHO;
    ioctl(0, TCSETA, &ti);
    exit(0);         /* %% */
}

static void
timedout(int sig) {
    signal(SIGALRM, timedout2);
    alarm(10);
    fprintf(stderr, _("Login timed out after %d seconds\n"), timeout);
    signal(SIGALRM, SIG_IGN);
    alarm(0);
    timedout2(0);
}

#ifndef USE_PAM
int
rootterm(char * ttyn)
{
    int fd;
    char buf[100],*p;
    int cnt, more = 0;

    fd = open(SECURETTY, O_RDONLY);
    if(fd < 0) return 1;

    /* read each line in /etc/securetty, if a line matches our ttyline
       then root is allowed to login on this tty, and we should return
       true. */
    for(;;) {
    p = buf; cnt = 100;
    while(--cnt >= 0 && (more = read(fd, p, 1)) == 1 && *p != '\n') p++;
    if(more && *p == '\n') {
        *p = '\0';
        if(!strcmp(buf, ttyn)) {
        close(fd);
        return 1;
        } else
          continue;
    } else {
        close(fd);
        return 0;
    }
    }
}
#endif /* !USE_PAM */

jmp_buf motdinterrupt;

void
motd(void) {
    int fd, nchars;
    void (*oldint)(int);
    char tbuf[8192];

    if ((fd = open(_PATH_MOTDFILE, O_RDONLY, 0)) < 0)       打开登录欢迎文件/etc/motd
      return;
    oldint = signal(SIGINT, sigint);
    if (setjmp(motdinterrupt) == 0)
      while ((nchars = read(fd, tbuf, sizeof(tbuf))) > 0)   输出该欢迎文件中的内容
    write(fileno(stdout), tbuf, nchars);
    signal(SIGINT, oldint);
    close(fd);
}

void
sigint(int sig) {
    longjmp(motdinterrupt, 1);
}

#ifndef USE_PAM                /* PAM takes care of this */
void
checknologin(void) {
    int fd, nchars;
    char tbuf[8192];

    if ((fd = open(_PATH_NOLOGIN, O_RDONLY, 0)) >= 0) {        打开/etc/nologin 文件
    while ((nchars = read(fd, tbuf, sizeof(tbuf))) > 0)        输出该文件中的内容
      write(fileno(stdout), tbuf, nchars);
    close(fd);
    sleepexit(0);
    }
}
#endif

void
dolastlog(int quiet) {
    struct lastlog ll;
    int fd;

    if ((fd = open(_PATH_LASTLOG, O_RDWR, 0)) >= 0) {
    lseek(fd, (off_t)pwd->pw_uid * sizeof(ll), SEEK_SET);
    if (!quiet) {
        if (read(fd, (char *)&ll, sizeof(ll)) == sizeof(ll) &&
        ll.ll_time != 0) {
            time_t ll_time = (time_t) ll.ll_time;

            printf(_("Last login: %.*s "),
               24-5, ctime(&ll_time));

            if (*ll.ll_host != '\0')
                printf(_("from %.*s\n"),
                   (int)sizeof(ll.ll_host), ll.ll_host);
            else
                printf(_("on %.*s\n"),
                   (int)sizeof(ll.ll_line), ll.ll_line);
        }
        lseek(fd, (off_t)pwd->pw_uid * sizeof(ll), SEEK_SET);
    }
    memset((char *)&ll, 0, sizeof(ll));
    time(&ll.ll_time);
    xstrncpy(ll.ll_line, tty_name, sizeof(ll.ll_line));
    if (hostname)
        xstrncpy(ll.ll_host, hostname, sizeof(ll.ll_host));

    write(fd, (char *)&ll, sizeof(ll));
    close(fd);
    }
}

void
badlogin(const char *name) {
    if (failures == 1) {
    if (hostname)
      syslog(LOG_NOTICE, _("LOGIN FAILURE FROM %s, %s"),
         hostname, name);
    else
      syslog(LOG_NOTICE, _("LOGIN FAILURE ON %s, %s"),
         tty_name, name);
    } else {
    if (hostname)
      syslog(LOG_NOTICE, _("%d LOGIN FAILURES FROM %s, %s"),
         failures, hostname, name);
    else
      syslog(LOG_NOTICE, _("%d LOGIN FAILURES ON %s, %s"),
         failures, tty_name, name);
    }
}

/* Should not be called from PAM code... */
void
sleepexit(int eval) {
    sleep(SLEEP_EXIT_TIMEOUT);
    exit(eval);
}
```

环境

`login` 源码摘自 `util-linux-2.12r package`

联系

Walter Zhou

mailto:z-l-dragon@hotmail.com