第九章 进程关系

对当前系统上login而言，下面些许选项已失效。具体哪些继续有效，请参阅其源码(shadow-4.1.5.1/src/login.c)。

NAME

login - 在系统上开始一个会话

SYNOPSIS

login [-p] [-h host] [username] [ENV=VAR...]

login [-p] [-h host] -f username

login [-p] -r host

描述：

login程序用于与系统建立一个新的会话。它被调用后，会在用户的终端上面自动地返回"login: "提示符。login或许是一个特殊的shell，也许并不能以一个子进程来调用它。当调用来自一个shell时，login应该以exec来执行登录，这将会使用户退出当前的shell(从而防止新登录的用户被返回调用者的session)。试图从任何shell执行login时，该shell将会产生一个错误信息(貌似centos会这样，gentoo、ubuntu、mac OS并非如此，genoo中非root用户也会提示错误)。

然后在适当的情况下提示用户输入密码。禁止回显可以防止密码泄露。在登录断开前，仅允许少数的几次无效密码验证。

如果你的账号已经开启了密码过期校验，在继续操作前，或许会提示你输入新密码。你将被强制提供旧的密码和新的密码后才能继续。

更多信息请参阅passwd(1)。

安全登录后，你将被告知相关的系统消息及邮件的存在。或许你可以关闭系统消息文件(/etc/motd)的显示，一般是在你的宿主目录中，创建一个0长度的.hushlogin文件。根据你信箱的情况，邮件消息将会是"You have new mail.", "You have mail.","No Mail."几个当中的一个。

你的用户及组ID，将根据它们在/etc/passwd文件中的值被设置。$HOME, $SHELL, $PATH, $LOGNAME, $MAIL这些变量根据它们在该文件条目中相应的字段来设置。ulimit, umask, nice或许也要根据GECOS字段(如`:Joe Smith,Room 1007,(234)555-8910,(234)555-0044,email:`)的条目来设置。

在某些安装中，环境变量$TERM(如：TERM=xterm)将被像在/etc/ttytype中规定的那样，初始化终端类型到你的tty线路上。

```
tongue ~ # grep agetty /etc/inittab
c1:12345:respawn:/sbin/agetty 38400 tty1 linux
c2:2345:respawn:/sbin/agetty 38400 tty2 linux
c3:2345:respawn:/sbin/agetty 38400 tty3 linux
c4:2345:respawn:/sbin/agetty 38400 tty4 linux
c5:2345:respawn:/sbin/agetty 38400 tty5 linux
c6:2345:respawn:/sbin/agetty 38400 tty6 linux
```

```
tongue ~ # ls -Rr /etc/terminfo/
/etc/terminfo/:
x  v  s  r  l  d  a

/etc/terminfo/x:
xterm-xfree86  xterm-color  xterm

/etc/terminfo/v:
vt52  vt220  vt200  vt102  vt100
```

一个初始化脚本也是能被你的命令解释器所执行的。更多关于该函数的信息，请参阅其手册的相关部分。

一个子系统登录，表示登录shell的首字符以一个"*"的方式存在。给定的主目录，将被作为该用户实际登录到一个新文件系统的根目录。

login程序不负责从utmp文件中删除用户。清除终端会话的所有权是getty和init的责任。如果你是从shell中不通过

exec使用login，即使在你的"subsession"登出后，你使用的用户将任然出现被登录的情况(who命令可以验证，不过事实与其不同)。

OPTIONS

-f 不进行验证，用户已预先被验证。

注意：这种情况下，username是必须的。

-h 欲登录的远程计算机名。

-p 保护环境。

-r 为rlogin执行自动登录协议。

-r、-h、-f选项仅在login被root调用时才能被使用。

注意事项

该版本的login有一些编译选项，任何特定的站点仅能使用其中的一部分。

文件的位置受系统配置的影响。

如其它程序，login的外观可被伪造。如果不可信的用户可对该机器进行物理访问，一个攻击者就可以获取下一个坐在该机器前用户的密码。在linux中，使用SAK机制，让用户启用一个可信的路径，来防止该类攻击的发生。

CONFIGURATION

下面配置变量在/etc/login.defs中，通过它们来改变该工具的行为:

CONSOLE (string)

如果被定义，文件的完整路径名(每行一个)需包含设备名或一个以“:”为分隔符的设备列表之一。root将仅被允许在这些设备上登录。

如果没被定义，root可以从任何设备登录。

设备的指定，前缀不应该为/dev/。

CONSOLE_GROUPS (string)

当在控制台登录时，设置组列表到用户的附加组中(确定通过控制台设定)，默认为空。

小心使用，它可能会让用户永久取得这些组的访问，即使不在控制台上登录。

DEFAULT_HOME (boolean)

表明即登录被允许，我们也不能cd到家目录。默认为no。

设置为yes，如果不能cd到他的家目录，用户将登录到系统根(/)目录。

ENV_HZ (string)

如果设置，它将使用定义的HZ环境变量给登录时的用户。该值的前面可为HZ=。Linux上常见的值为HZ=100。

ENV_PATH (string)

如果被设置，它将使用定义的PATH环境变量给登录时的普通用户。该值前面可为 PATH=，或者是一个以冒号分割的列表(如 bin:/usr/bin)。

默认值为PATH=/bin:/usr/bin。

ENV_SUPATH (string)

如果被设置，它将使用定义的PATH环境变量给登录时的超级用户。该变量前面可以是PATH=，或者以冒号分割的路径列表(如 /sbin:/bin:/usr/sbin:/usr/bin)。默认值为PATH=/bin:/usr/bin。

ENV_TZ (string)

如果被设置，它将使用定义的TZ环境变量给登录时的用户。该时区变量名的前面可以是TZ=(如 TZ=CST6CDT)，或者完整路径包含了时区文件(如 /etc/tzname)。

如果指定了一个完整的路径，但该文件不存在或无法读取，那么默认将置该值为TZ=CST6CDT。

ENVIRON_FILE (string)

如果文件存在且可读，登录环境将由此读取。每行的格式应如name=value。

行的其实如果是‘#’字符，则将改行当做注释并忽略。

ERASECHAR (number)

终端擦除字符(010 = backspace, 0177 = DEL)。

该值可为十进制加前缀"0"，或者十六进制加“0x”。

FAIL_DELAY (number)

前一个登录失败后，延时多久才允许下一次重新登录，单位(秒)。

FAILLOG_ENAB (boolean)

开启登录失败信息记录到/var/log/faillog。

FAKE_SHELL (string)

如果被设置，login将用该shell代替用户在/etc/passwd文件中指定的shell来执行。

FTMP_FILE (string)

如果被定义，登录失败将已utmp格式记录到该文件。

HUSHLOGIN_FILE (string)

如果被定义，该文件可以抑制在登录期间繁冗的信息。如果给它指定一个完整的路径，如果用户名或shell在该文件中被找到，那么安静模式将被开启，或该文件存在于登录用户的家目录。

ISSUE_FILE (string)
如果被定义，该文件将在登录提示符出现前被显示。注意，仅在tty，网络登录(pty)未必。
KILLCHAR (number)
结束终端字符(025 = CTRL/U)。
该值十进制前缀可为"0"，十六进制可为"0x"。
LASTLOG_ENAB (boolean)
启用日志记录/var/log/lastlog的登录时间。
LOGIN_RETRIES (number)
登录验证失败最大尝试次数。
LOGIN_STRING (string)
输入密码提示字符串。默认为"Password: "，或由它翻译的字符串。如果你设置该变量，该提示符将被转变。
如果字符串包含%s，这将替换为用户名。
LOGIN_TIMEOUT (number)
登录的最大时长。
LOG_OK_LOGINS (boolean)
记录成功的登录。
LOG_UNKFAIL_ENAB (boolean)
当记录登录失败时，记录未知的用户名。
注意：登录时未知用户名很可能是其与密码写反了，这个安全问题需评估。
MAIL_CHECK_ENAB (boolean)
登入系统后，校验并显示mailbox的状态。
如果shell在启动时，邮件文件已经被校验过("mailx -e"或类似)。你应该禁止它。
MAIL_DIR (string)
mail spool目录。当与它相应的用户账户修改、删除时，这需要操作mailbox。如果不指定，编译时默认被使用。
MAIL_FILE (string)
定义用户mail spool文件位置到它相应的家目录。
MAIL_DIR和MAIL_FILE会成为useradd, usermod, 和userdel创建、移动、删除用户mail spool的变量。
如果MAIL_CHECK_ENAB设置为yes，定义的环境变量MAIL也会被它使用。
MOTD_FILE (string)
如果被定义，以“:”分割的"message of the day"文件列表会在登入是显示。
NOLOGINS_FILE (string)
如果被定义，该文件的存在会抑制非root的登录。该文件内容应该是一个，为说明为什么不让登录的消息。
PORTTIME_CHECKS_ENAB (boolean)
在/etc/porttime中指定时间校验的限制。
QUOTAS_ENAB (boolean)
开启ulimit、umask和来自passwd文件gecos域niceness的设定。
TTYGROUP (string), TTYPERM (string)
终端权限：登录的tty将属于TTYGROUP组，权限将被设置为 TTYPERM。
默认，终端权限被设置为0600。
TTYGROUP可以是它的组名，或其组ID。
如果你有一个可写的程序，其被"setgid"为指定的组，它属于这个终端，定义TTYGROUP的组id和TTYPERM为0620。否则注释掉TTYGROUP，置TTYPERM为622或600。
TTYTYPE_FILE (string)
如果被定义，文件映射tty线路到TERM环境变量参数。每行的格式如"vt100 tty01"。
ULIMIT (number)
默认ulimit值。
UMASK (number)
文件权限掩码初始化到该值。如果没指定，掩码默认被初始化为022。
useradd、newusers使用该掩码，在家目录创建时设置它的模式。
它同样使用login定义用户初始化umask。注意，该mask可以覆盖掉用户GECOS设定的相关值。
USERGROUPS_ENAB (boolean)
为非root用户开启umask的组位设置到它相同的所属位(如：022 -> 002, 077 -> 007)，如果uid与gid，及用户名与主组名相同。
如果设置为yes，如果它不包含更多成员，userdel将删除用户组，useradd将默认创建一个与其用户名相应的组。
FILES
/var/run/utmp
当前登录会话的列表。
/var/log/wtmp
以前登录会话的列表。
/etc/passwd

用户账号信息。
/etc/shadow
用户账号安全信息。
/etc/motd
天文件的系统消息。
/etc/nologin
防止非root用户登录。
/etc/ttytype
终端类型列表。
$HOME/.hushlogin
禁止系统消息打印到终端。
/etc/login.defs
shadow密码套件配置。

SEE ALSO
mail(1), passwd(1), sh(1), su(1), login.defs(5), nologin(5), passwd(5), securetty(5), getty(8).

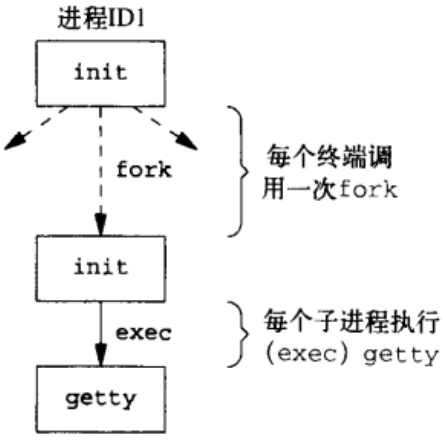


图9-1 init为允许终端登录而调用的进程

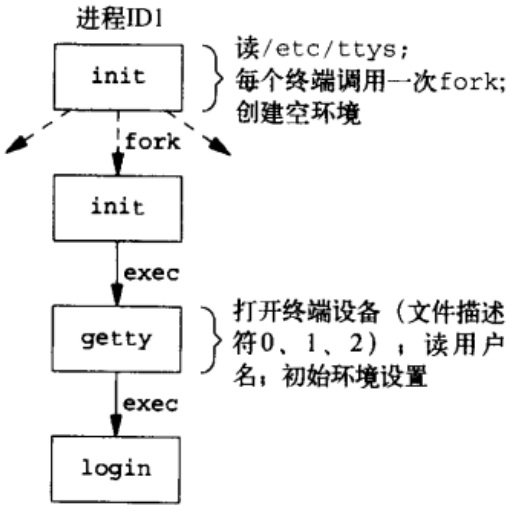


图9-2 调用login后的进程状态

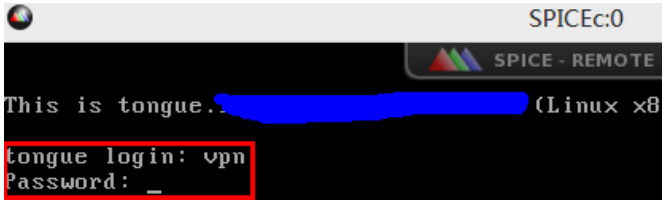
getty为终端设备调用open函数，以读、写方式将终端打开。如果设备是调制解调器，则open可能会在设备驱动程序中滞留，直到用户拨号调制解调器，并且呼叫被应答。一旦设备被打开，则文件描述符0、1、2就被设置到该设备。然后getty输出“login:”之类的信息，

```
execle("/bin/login", "login", "-p", username, (char *)0, envp);
```

需要理解的重点是：当通过终端（见图9-3）或网络（见图9-5）登录时，我们得到一个登录shell，其标准输入、输出和标准出错连接到一个终端设备或者伪终端设备上。在下一节中我们会了解到这一登录shell是一个POSIX.1会话的开始，而此终端或伪终端则是会话的控制终端。

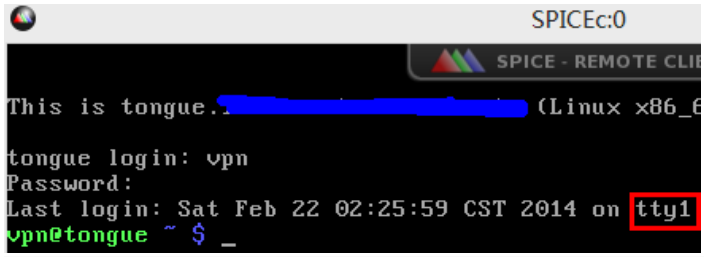
则文件描述符0、1、2就被设置到该设备。然后getty输出“login:”之类的信息，

```
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|tty1'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root     18325 18325 18325     1 18325 tty1     Ss+  agetty
```



login能执行多项工作。因为它得到了用户名，所以能调用getpwnam取得相应用户的口令文件登录项。然后调用getpass(3)以显示提示“Password:”，接着读用户键入的口令(自然，禁止回送用户键入的口令)。它调用crypt(3)将用户键入的口令加密，并与该用户在阴影口令文件中登录项的pw_passwd字段相比较。如果用户几次键入的口令都无效，则login以参数1调用exit表示登录过程失败。父进程（init）了解到子进程的终止情况后，将再次调用fork，其后接着执行getty，对此终端重复上述过程。

```
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|tty1'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root     18325 18325 18325     1 18325 tty1     Ss+  login
```



如果用户正确登录，login就将执行如下工作：

- 将当前工作目录更改为该用户的起始目录（chdir）。
- 调用chown改变该终端的所有权，使登录用户成为它的所有者。
- 将对该终端设备的访问权限改变成用户读和写。
- 调用setgid及initgroups设置进程的组ID。
- 用login所得到的所有信息初始化环境：起始目录（HOME）、shell（SHELL）、用户名（USER和LOGNAME），以及一个系统默认路径（PATH）。
- login进程改变为登录用户的用户ID（setuid）并调用该用户的登录shell，如下：

```
execl("/bin/sh", "-sh", (char *)0);
```

argv[0]的第一个字符“-”是一个标志，表示该shell被调用为登录shell，shell可以查看此字符，并相应地修改其启动过程。

```
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|tty1'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0  1001     0 root     18332 18325 18325     1 18325 tty1     Ss   login
 1001  1001  1001 vpn      18332 18325 18332 18325 18332 tty1     S+   bash
```

```
1186      (void) signal (SIGINT, SIG_IGN);
1187      child = fork ();
1188      if (child < 0) {
1189          /* error in fork() */
1190          fprintf (stderr, _("%s: failure forking: %s"),
1191                   Prog, strerror (errno));
1192          PAM_END;
1193          exit (0);
1194      } else if (child != 0) {
1195          /*
1196           * parent - wait for child to finish, then cleanup
1197           * session
1198           */
1199          wait (NULL);  这里解释了上图login为何未消失的原因
1200          PAM_END;
1201          exit (0);
1202      }
~/shadow-4.1.5.1/src/login.c [FORMAT=unix:utf-8] [TYPE=C] [COL=0
```

```
 89         (void) execle (SHELL, "sh", "-", file, (char *)0, envp);
~/shadow-4.1.5.1/libmisc/shell.c [FORMAT=unix:utf-8] [TYPE=C] [COL=0
```

总结：

一个会话仅有一个控制终端，一个控制终端也仅被一个会话所拥有。

一个会话包含一个或多个进程组。通过控制tty结构，来指定哪一个为前台进程组。

通常由shell的管道线或父子结构的进程，将几个进程编成一组。

控制终端是一个系统直接与用户交互的对象，用户把自己的想法通过它来告诉系统，系统也用它来把结果反馈给用户。它就像一个唱戏的舞台 。

console(控制台、控制台终端)，面向物理硬件；

terminal(终端、控制终端)，面向系统。

区别它两的标准是，在系统未启动前，是否可以控制计算机。可以控制计算机的是console，需要等系统启动后，才能控制系统的是terminal。比如console就可以操控计算机的BIOS、grub等，但terminal却不行。console包含了terminal的功能及特性。

习题：

9-1、按书上本章的逻辑，终端登录的shell其父进程应该是pid为1的init进程。如果仅考虑正常的退出，utmp的清除可由shell来做，但若加入其它不确定因素(比如：线路中断、shell进程被强制退出等)，那么shell本身就无法处理结束前的操作。而它的父进程却可侦测到它的退出(wait子进程)，并做相应的善后处理。

网络登录，其父进程一般为其相应端口的侦听进程。该进程负责创建子进程并使其处理用户建立新连接，也可以用它来处理子进程结束的操作。所以其在utmp的处理中，有能力代替tty版的init角色。不过当该服务重启时，子进程的处理，值得琢磨(是强制结束该进程的所有子进程呢，还是把该进程的所有子进程都交给init，两种不同的处理方式，直接影响到utmp的清除操作)。

总结来看，utmp的清除操作具体由谁来做，要看谁可以捕获登录进程的异常结束。

```
/* The structure describing an entry in the user accounting database.  */
struct utmp
{
  short int ut_type;        /* Type of login.  */
  pid_t ut_pid;         /* Process ID of login process.  */
  char ut_line[UT_LINESIZE];    /* Devicename.  */
  char ut_id[4];        /* Inittab ID.  */
  char ut_user[UT_NAMESIZE];    /* Username.  */
  char ut_host[UT_HOSTSIZE];    /* Hostname for remote login.  */
  struct exit_status ut_exit;   /* Exit status of a process marked
/usr/include/bits/utmp.h [FORMAT=unix:utf-8] [TYPE=CPP] [COL=001] [ROW=058
```

9-2、如下

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>

int main(int argc, const char *argv[]) {
    pid_t pid;
    struct stat sb;
    fstat(stdout->_fileno, &sb);
    printf("parent stdout dev = %d/%d, inode: %ld!\n", major(sb.st_dev), minor(sb.st_dev), sb.st_ino);
    if (0 > (pid = fork())) {
        printf("error by fork!\n");
    } else if (0 == pid) {
        printf("child pid: %d, sid: %d!\n", getpid(), getsid(getpid()));
        system("ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|9-2'");
        pid = setsid();
        printf("after the new session: \n");
        printf("child pid: %d, sid: %d!\n", getpid(), getsid(0));
        system("ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|9-2'");
        fstat(stdout->_fileno, &sb);
        printf("child stdout dev = %d/%d, inode: %ld!\n", major(sb.st_dev), minor(sb.st_dev), sb.st_ino);
        return 0;
    }
    printf("parent pid: %d, sid: %d!\n", getpid(), getsid(getpid()));
    sleep(1);
    return 0;
}
/opt/drill_ground/aupe/9-2.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [ROW=001/29(3%)]
```

```
tongue aupe # gcc -g 9-2.c -o 9-2
tongue aupe # ./9-2
parent stdout dev = 0/9, inode: 3!
parent pid: 28962, sid: 28862!
child pid: 28963, sid: 28862!
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root     28962 28862 28962 28862 28962 pts/0    S+   9-2
    0     0     0 root     28962 28862 28962 28962 28963 pts/0    S+   9-2
after the new session:
child pid: 28963, sid: 28963!
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root     28962 28862 28962 28862 28962 pts/0    S+   9-2
    0     0     0 root        -1 28963 28963 28962 28963 ?        Ss   9-2
child stdout dev = 0/9, inode: 3!
```