```
1. utmp/wtmp
 a. /etc/utmp contains the currently logged in users ("who" gets information
    from this file)
 b. /usr/adm/wtmp contains the history of logins and logouts

 Each entry of both files is of type specified by the following structure:

  struct utmp {
        char ut_user[8];              /* user name */
        char ut_line[12];             /* terminal name */
        char ut_host[16];             /* host name, when remote */
        time_t ut_time;               /* login/logout time */
  };

  #define ut_name ut_user
  #define long time_t;          // in <sys/types.h>

 1. A "login" entry is completely specified.
 2. A "logout" entry has a null string for ut_name.
 3. A "shutdown" or "reboot" entry has an ut_line field containing a "~"
   (tilde).  The ut_name field is usually the name of the program that did
   the shutdown, or "reboot" at reboot.

 Note: /etc/inittab describes how the INIT process should set up the system
       in a certain run−level (single user/multi user etc, MINIX does not
       have /etc/inittab).

2. gettyent and endttyent
 The gettyent and endttyent functions provide an interface to /etc/ttytab.
 Getttyent() opens the ttytab file if not already open and reads one
 entry (it returns NULL for EOF).
 Endttyent() closes the ttytab file.

 struct ttyent {
        char *ty_name;          // Name of the terminal device
        char *ty_type;          // Terminal type name (see termcap(3))
        char **ty_getty;        // Program to run (normally getty)
        char **tty_init;        // Initialization command, normally stty
 };

3. Wait() and waitpid()
   #include <sys/types.h>
   #include <sys/wait.h>

   pid_t wait(int *status);     // status can be NULL if no info is needed
   pid_t waitpid (pid_t pid, int *status, int options);
   typedef int pid_t;   // in <sys/types.h>

   Wait() blocks the caller until a signal is received or one of its
   child processes terminates. If there are no children, it returns −1.
   On return from a successful wait call, status is nozero.
   The high byte of status contains the low byte of the argument to exit()
   supplied by the child process.  The low byte of the status contains
   the termination status of the process (see the on−line manual for
   the macros to evaluate the status).

   Waitpid() is used when the parent cannot be blocked (specified in options)
   or it wants to wait for one particular child (specified in pid; if pid is
   −1, it waits for any child).

   If WNOHANG is specified in "options," waitpid() does not block the caller.
   It returns −1 if there are no stopped (but not exited −−− just wants
   to report its status) or exited children.

3. Overview of init.c

   open /dev/null for stdin
   open /dev/log (i.e., console) for stdout and stderr

   set up signal handers for SIGHUP, SIGTERM, and SIGABORT
   executes /etc/rc (sh executes it)
   log system reboot information in /usr/adm/wtmp

```



```
    check = 1  // check == 1 if new terminal processes may have to be spawned
    while(1) {
      while ((pid = waitpid(−1, NULL, check ? WNOHANG: 0)) > 0) {
         // a process has terminated
         if the process is a terminal controlling process,
           put information in both /etc/utmp and /usr/adm/wtmp,
           clear the terminal slot entry and check = 1
      }
      if (a terminal line hang−up) {
         clear error counts of all the terminals
         check = 1
      }
      if (abort signal is received (CTL−ALT−DEL)) {
         execute startup() to have a child process execute "shutdown"
      }
      if (not terminating && check) {
         for each terminal entry in ttytab,
           if error count is not reached the limit,
             execute startup () −− spawn a process and have
                                   it execute getty;
      {
      check = 0 // no need to spawn a new process until it receives
                // a signal or a process dies.
    }

    startup () // for simplicity, consider a case in which no errors occur
    {
      create a pipe for error messages from a child (that will execute getty)
      to the parent (init);

      pid = fork();
      if (fork () failed) {
         // the parent (init)
         sleep(10); return;
      }
      if (a child) {  // a terminal controlling process
         close(input pipe);
         fcntl(); // have the output pipe closed when exec is issued.
         setsid(); // create a new session (process group) and make
                   // this process the session (group) leader
         line = /dev/tty_name specified in /etc/ttytab;

         close(0)  // close /dev/null
         close(1)  // close /dev/log
         open (/dev/tty_name, O_RDWR); // open stdin (descriptor 0)
         duplicate stdin to stdout (descriptor 1);
         if (/etc/ttytab specifies the initialization command (tty_init)) {
             fork();
             if (ground child) {
                 set alarm for 10 sec.
                 executes the initialization command
                 (if exec succeeds, the alarm is stopped)
             }
             if (child) {
                 waitpid(ground child) for the termination of the
                 initialization command.
           }
         }
         dup2(0, 2); // to open /dev/tty_name to be the stderr

         execute ttyp−>getty (i.e., getty()).
         // if startup() is called from line 160 (startup(0, &TT_REBOOT)),
         // the child executes "shutdown now CTRL−ALT−DEL"
         // Note that the output pipe is closed if the exec is successful
         // because of the fcntl() call above.
      }
      // parent (init)
      record the child (the terminal controlling process) pid in slotp.
      close (the output pipe);
      if error is reported by the child process, write error messages in
         /dev/log
      close (the input pipe);
      if (abort flag is off)
```

```
147             write "login" information (pid, tty_name, etc) in /etc/utmp;
148           reset the error count of this terminal to 0;
149       }
```