460 Last LAB ASSIGNMENT

```
                  LAST LAB ASSIGNMENT
                DUE and DEMO: CLOSE WEEK

1. Wanix in ~samples/LAST/

1.1. Download all the files.
     sdimage is a SDC image with NO partitions. It can be mounted as a loop
     device under Linux directly.

1.3. Read the mk script to see how to generate a NEW user mode command
     Example:  mk test ===> generate user mode command from test.c and copy
                            test to /bin of the sdimage


2. Syscalls in Wanix Kernel

   ====================== Wani System Call Functions ======================
   Number  Name        Usage                       Function
   ------ ---------  --------------------------  ----------------------
     0    getpid      getpid()                    get process pid
     1    getppid     getppid()                   get parent pid
     2    getpri      getpri()                    get priority
     3    setpri      setpri(pri)                 set priority
     4    getuid      getuid()                    get uid
     5    chuid       chuid(uid,gid)              set uid,gid
     6    yield       yield()                     switch process
     9    exit        exit(value)                 terminate process
    10    fork        fork()                      fork child process
   *11    exec        exec(cmd_line)              change image to a file
    12    wait        wait(&status)               wait for child to die
    13    vfork       vfork()                     fork child process
    14    thread      thread(fd,stack,flag,prt)   create thread
    15    mutex_creat   mutex_creat()
    16    mutex_lock    mutex_lock(&mutex)
    17    mutex_unlock  mutex_unlock(&mutex)
    18    mutex_destroy mutex_destroy(&mutex)
   ----------------------------------------------  ----------------------
    20    mkdir       mkdir(pathname)             make directory
    21    rmdir       rmdir(pathname)             rm   directory
    22    creat       creat(pathname)             creat file
    23    link        link(oldname, newname)      hard link to file
    24    unlink      unlink(pathname)            unlink
    25    symlink     symlink(oldname,newname)    create symbolic link
    26    readlink    readlink(name, buf[ ])      read symlink
    27    chdir       chdir(pathname)             change dir
    28    getcwd      getcwd(buf[ ])              get cwd pathname
    29    stat        stat(filename, &stat_buf)    stat file
    30    fstat       fstat(fd, &stat_buf)        stat file descriptor
    31    open        open(filename, flag)        open file for RD_ONLY,
                            O_WRONLY|O_CREAT, O_WRONLY|O_APPEND, O_RDWR
    32    close       close(fd)                   close file descriptor
   *33    lseek       lseek(fd, position)         lseek
    34    read        read(fd, buf[ ], nbytes)    read file
    35    write       write(fd,buf[ ], nbytes)    write to file
    36    pipe        pipe(pd[ ])                 carete pipe
    37    chmod       chmod(filename, mode)       change permission
    38    chown       chown(filname, uid)         change file owner
    39    touch       touch(filename)             change file time
    40    settty      settty(tty_name)            set proc.tty name
    41    gettty      gettty(buf[ ])              get proc.tty name
    42    dup         dup(fd)                     dup file descriptor
    43    dup2        dup2(fd1, fd2)              dup fd1 to fd2
    44    ps          ps()                        ps in kernel
    45    mount       mount(FS, mountPoint)       mount file system
    46    umount      umount(mountPoint)          umount file system
    47    getSector   getSector(sector, buf[ ])   read CDROM sector
    48    cd_cmd      cd_cmd(cmd)                 issue cmd to Cd driver
   ----------------------------------------------  ----------------------
    50    kill        kill(sig#, pid)             send signal to pid
    51    signal      signal(sig#, catcher)       install sigan1 handler
    52    pause       pause(t)                    pause for t seconds
    53    itimer      itimer(sec, action)         set timer request
    54    send        send(msg, pid)              send msg to pid
    55    recv        sender=recv(msg)            receive msg
    56    tjoin       tjoin(n)                    threads join
    57    texit       texit(value)                thread exit
    58    hits        hits()                      I/O buffer hit ratio
    59    color       color(v)                    change display color
    60    sync        sync()                      sync file system
   ========================================================================


Wanix syscalls are organized into 4 groups.
Group 1 (0-19) is for process management.
Group 2 (20-49) is for file system operations.
Group 3 is for signals and signal processing, and
Group 4 is for micesleous system calls.

All the syscall functions in Groups 1-3 are compatible with those of Unix/Linux.
The only exception is exec(cmd_line). In Unix, when a user enters a command line
        a.out  arg1  arg2 ... argn
it is used in execl() as execl(a.out, a.out, arg1, arg2,...,argn, 0); or as
argc, argv[] in          execv(argc, argv);

In Wanix, the entire command line is used in the exec() call. For example, if
you enter cat filename to the Wanix sh, the child sh uses

         exec("cat filename");

to change its execution image to the cat program. However, the entire command
line is passed to the new image when execution starts. Parsing the command line
into argc and argv[] is done by a C start up code, crt0, in the new image.

3. Operation of the Wanix system:
   The sdimage is an EXT2 FS containing

            /---|
                |----bin/ : All binary executables are in /bin
                |
                |---dev/ : special files tty0   c 4 0
                |                        fd0    b 2 0
                |                        sdc    b 3 0
                |                        ttyS0 c 5 0
                |                        ttyS1 c 5 1
                |                        ttyS2 c 5 2
                |---etc/ : passwd file
                |
                |---boot/: bootable Wanix kernels
                |
                |---user/: users HOME directories

  After mounting the root file system, P0 creats P1, whose Umode image is
  the /bin/init program. P1 will go Umode directly to execute /bin/init,
  in which it forks children P2 on /dev/tty0. P1 may also fork P3 /dev/ttyS0
  and P4 on /dev/ttyS1, etc. Then P1 waits for any child to die. Henceforth, P1
  plays the role of the INIT process (P1) in Unix.

                      Proc#2:
  (1). P2 is a login process. It executes /bin/login on the console terminal
       /dev/tty0. Special files /dev/ttyS0, /dev/ttyS1, etc. are for serail
       terminals.

  (2). In /bin/login, P2 opens its tty special file (/dev/tty0) as stdin(0),
       stdout(1) and stderr(2). Then it displays (to its stdout)
       login:

  and waits for a user to login. When a user tries to login, it reads the
  user name and password (from its stdin), opens the /etc/passwd file to
  authenticate the user. Each line of /etc/passwd has the format:

       username:password:gid:uid:fullname:HOMEDIR:program
  e.g. root:xxxxxxx:1000:0:superuser:/root:sh

  (Use plain text for password OR devise your own encryption schemes)

  If the user has a valid account in /etc/passwd, P2 becomes the user's process
  (by taking on the user's uid). It chdir to user's HOME directory and execute
  the listed program, which is normally the sh (/bin/sh).

  (3). then (in sh) it loops forever (until "logout"):
       {
         prompts for a command line, e.g. cmdLine="cat filename"
         if (cmd == "logout")
            syscall to die;
         if (cmd == "cd")
            syscall to chdir;

         // if just ONE cmd:
         pid = fork();
         if (pid)
            pid = wait(&status);
         else
            exec(cmdLine);
       }

  When the child proc terminates (by exit() syscall to WANIX kernel), it wakes
  up sh, which prompts for another cmdLine, etc.

 (4). When sh dies, it wakes up its parent, INIT, which forks another login
      process.
========================================================================

4. OBJECTIVES:

   The purpose of this assignment is for you to write YOUR OWN
   init, login, sh and other user command programs. Among these, sh is the
   most important. Accordingly, it will carry the most weight.

========================================================================

                   5. ASSIGNMENTS:
                  DUE in CLOSED week
              Demo and INTERVIEW during DEMO

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
5-1. Develop YOUR OWN programs for
           init                     // for process P1
           login                    // for login processes
           ls    [filename]         // ls CWD or filename; NO time field
           cat   [filename]         // cat filename or stdin to stdout
           more  [filename]         // more as in Linux
           grep  pattern [filename] // print lines containing pattern
           l2u   [f1 f2]            // convert lower case to upper case
           cp    file1 file2        // copy files

    NOTE: [filename means OPTIONAL. For cat, more, grep, l2u
       If no filenames, use stdin for IN, stdout for OUT, both may be redirected

5-2. Write YOUR OWN sh.c to support I/O redirections and pipes:

    Examples:  ls  [filename]       // as ls -l in Linux; NO time field
               cat [filename]       // as cat in Linux
               cat [filename] >  newfile
               cat [filename] >> appendFile
               a.out < inFile       // read inputs from inFile
               cat filename | more
               cat filename | grep test
               cat filename | l2u | grep LINE
               cat < infile | cat | grep print | more

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

REFERENCE: READ Chapter 8 of TEXTBOOK

NOTE: (1). You must implement ALL the listed programs in 5-1.
          Your init must support multiple logins from console and serial ports.

      (2). All programs must behave the same as they do in Linux.
          Test ls, cat, more, grep under Linux to see how they behave first.
========================================================================

                Specific Examples:
cat [filename]: if no filename:  cat from stdin
    abcd
    abcd
    (control-D) terminate cat process

cat filename: show filename contents on stdout as cat in Linux

cat f1 > f2; ls: f2 and f1 MUST be the same size
             if their size differ, your cat is WRONG!
cat f | cat | grep print: show lines containig "print"
------------------------------------------------------------------------
more [filename]: if no filename: show inputs from stdin to stdout
                                 same as cat

more f: show ONE screen of f;
       <enter>: show one more line
       <space>: show on more screen (of 25 lines)

cat f | more:
       <enter>: show one more line
       <space>: show on more screen (of 25 lines)
------------------------------------------------------------------------
grep pattern [filename]:

grep abcde (grep from stdin)
     line1
     abcde  <== matches with abcde
     abcde  <== echo line containing pattern
     line2

grep printf f: show lines containing "printf" in f

cat f | grep print ==> show lines containing "print"

------------------------------------------------------------------------
ls [filename]: same as ls -l in Linux, except do NOT print time field.
------------------------------------------------------------------------
                Grading standards

init: your init MUST fork logins on tty0, ttyS0, ttyS1        10
login: allow login and logout on different terminals          10
sh  : simple command:                                         10
      simple command with I/O redirection                      5
      one pipe:                                                5
      multiple pipes:                                         20
------------------------------------------------------------------------
ls  : behave SAME as ls -l in Linux                           10
cat, more, grep: MUST work as specified:                      24
l2u, cp :                                                      6
```