Changing user IDs and group IDs

- Prototype

```
#include <sys/types.h>
#include <unistd.h>
int setuid (uid_t uid);
int setgid (gid_t gid);
```

- Rules

1. If the process has superuser privilege, the setuid function sets – real user ID, effective user ID , saved set-user-ID to uid
2. If the process doesnot have superuser privilege, but uid equals either real user ID or saved set-user-ID, setuid sets only effective user ID to uid
3. If neither of the two conditions is true, errno is set to EPERM and an error is returned

ID	exec	exec
	Set-user-ID bit off	Set-user-Id bit on
Real user ID	unchanged	unchanged
Effective user ID	unchanged	Set from user ID of program file
Saved set user ID	copied from effective user ID	copied from effective user ID

ID	Superuser	Unprivileged user
Real user ID	Set to uid	unchanged
Effective user ID	Set to uid	Set to uid
Saved set-user ID	Set to uid	unchanged

5.9 setreuid and setregid

```
#include <sys/types.h>
#include <unistd.h>
int setreuid (uid_t ruid, uid_t euid);
int setregid (gid_t rgid,gid_t egid);
```

seteuid and setegid

```
#include <sys/types.h>
#include <unistd.h>
int seteuid (uid_t euid);
int setegid (gid_t egid);
```

Interpreter files

- Files which begin with a line of the form

  #! pathname [optional argument]

  most common example : #! /bin/bash
- The actual file execed by kernel is the one specified in the pathname

```
/*example of interpreter file*/
#!/bin/awk -f
BEGIN
 {
     for (i = 0; i < ARGC; i++)
               printf "ARGV[%d] = %s\n", i, ARGV[i]
     exit
}
```

- Uses of interpreter files

1. They hide the fact that certain programs are scripts in some other language
2. They provide an efficiency gain
3. They help us write shell scripts using shells other than /bin/sh

5.10 system function

- It helps us execute a command string within a program
- System is implemented by calling fork, exec and waitpid

```
#include <stdlib.h>
 int system (const char  *cmdstring);
```

- Return values of system function
- -1 – if either fork fails or waitpid returns an error other than EINTR
  - 127 -- If exec fails [as if shell has executed exit]
  - termination status of shell -- if all three functions succeed

```
#include        <sys/types.h>
#include        <sys/wait.h>
#include        <errno.h>
#include        <unistd.h>
int system(const char *cmdstring)
        /* version without signal handling */
{
 pid_t  pid;
 int    status;
 if (cmdstring == NULL)
             return(1);
 /* always a command processor with Unix */
       if ( (pid = fork()) < 0)
 {
             status = -1;
   /* probably out of processes */

       } else if (pid == 0)
   {                          /* child */
       execl("/bin/sh", "sh", "-c", cmdstring,
                              (char *) 0);
               _exit(127);          /* execl error */
■  }
   else {                           /* parent */
         while (waitpid(pid, &status, 0) < 0)
               if (errno != EINTR) {
                     status = -1;
```

```
                    /* error other than EINTR from waitpid() */
                     break;
              }
       }
    return(status);
}

    /*calling system function*/
            #include        <sys/types.h>
            #include        <sys/wait.h>
            #include        "ourhdr.h"
            int main(void)
            {
                    int             status;
                    if ( (status = system("date")) < 0)
                            err_sys("system() error");
                    pr_exit(status);
            if ( (status = system("nosuchcommand")) < 0)
                            err_sys("system() error");
                    pr_exit(status);
```

```
        if ( (status = system("who; exit 44")) < 0)
                err_sys("system() error");
        pr_exit(status);
        exit(0);
}
```

5.11 Process accounting

- Process accounting : when enabled kernel writes an accounting record each time a process terminates
- Accounting records : 32 bytes of binary data

```
Struct acct
 {
  char ac_flag;
  char ac_stat;
  uid_t ac_uid;
  gid_t ac_gid;
  dev_t ac_ttty;
  time_t ac_btime;
  comp_t ac_utime;
  comp_t ac_stime;
  comp_t ac_etime;
  comp_t ac_mem;
  comp_t ac_io;
  comp_t ac_rw;
  char ac_comm;
}
```

```
/*prog: to generate accounting data */

      #include<sys/types.h>
      #include<sys/acct.h>
      #include "ourhdr.h"
      #define ACCTFILE   "/var/adm/pacct"
      static unsigned long   compt2ulong(comp_t);
      int main(void)
      {
            struct acct                acdata;
            FILE                       *fp;
      if ( (fp = fopen(ACCTFILE, "r")) == NULL)
                  err_sys("can't open %s", ACCTFILE);
          while
     (fread(&acdata, sizeof(acdata), 1, fp) == 1)
    {     printf("%-*.*s  e = %6ld, chars = %7ld, "
                         "stat = %3u: %c %c %c %c\n",
                  sizeof(acdata.ac_comm),
                               sizeof(acdata.ac_comm),
                  acdata.ac_comm,
              compt2ulong(acdata.ac_etime),
                  compt2ulong(acdata.ac_io),
                 (unsigned char)   acdata.ac_stat,
#ifdef  ACORE
    /* SVR4 doesn't define ACORE */
          acdata.ac_flag & ACORE ? 'D' : ' ',
#else
          ' ',
#endif
#ifdef  AXSIG
               /* SVR4 doesn't define AXSIG */
                   acdata.ac_flag & AXSIG ? 'X' : ' ',
#else
   ' ',
#endif
acdata.ac_flag & AFORK ? 'F' : ' ',
          acdata.ac_flag & ASU   ? 'S' : ' ');
                      }
           if (ferror(fp))
                                err_sys("read error");
                      exit(0);
           }
```

```
static unsigned long
  compt2ulong(comp_t comptime)
/* convert comp_t to unsigned long */
{
            unsigned long  val;
            int                                               exp;
            val = comptime & 017777;
                                   /* 13-bit fraction */
            exp = (comptime >> 13) & 7;
                                        /* 3-bit exponent (0-7) */
            while (exp-- > 0)
                        val *= 8;
            return(val);
}
```

5.12 User identification

To obtain the login name

```
        #include <unistd.h>
        char *getlogin (void);
```

Process times

```
#include <sys/times.h>
clock_t times (struct tms *buf);
        Struct tms
        {
           clock_t tms_utime;
           clock_t tms_stime;
           clock_t tms_cutime;
           clock_t tms_cstime;
        }
```

```
#include<sys/times.h>
#include "ourhdr.h"
static void
   pr_times (clock_t, struct tms *, struct tms *);
static void       do_cmd(char *);
int main (int argc, char *argv[ ])
{          int     i;
         for (i = 1; i < argc; i++)
         do_cmd(argv[i]);
          /* once for each command-line arg */
         exit(0);
}
     static void
     do_cmd (char *cmd)
     /* execute and time the "cmd" */
     {
               struct tms        tmsstart, tmsend;
               clock_t           start, end;
               int                         status;
               fprintf(stderr, "\ncommand: %s\n", cmd);
               if ( (start = times(&tmsstart)) == -1)
               /* starting values */
                         err_sys("times error");
   if ( (status = system(cmd)) < 0)
                 /* execute command */
                      err_sys("system() error");
            if ( (end = times(&tmsend)) == -1)
                                /* ending values */
                      err_sys("times error");
            pr_times(end-start, &tmsstart, &tmsend);

            pr_exit(status);
    }
    static void
     pr_times (clock_t real, struct tms *tmsstart,
                                    struct  tms *tmsend)
    {  static long clktck = 0;
            if (clktck == 0)
       /* fetch clock ticks per second first time */
            if ( (clktck = sysconf(_SC_CLK_TCK)) < 0)
                    err_sys("sysconf error");
            fprintf (stderr, "  real:  %7.2f\n",  real / (double) clktck);
    fprintf (stderr, "  user:  %7.2f\n",(tmsend->tms_utime - tmsstart> tms_utime) /  (double)
    clktck);
fprintf(stderr, "  sys:   %7.2f\n",
(tmsend->tms_stime - tmsstart->tms_stime) / (double) clktck);
fprintf(stderr, "   child user:   %7.2f\n",(tmsend->tms_cutime - tmsstart-> tms_cutime) /
(double) clktck);
fprintf (stderr, "   child sys:   %7.2f\n",  (tmsend->tms_cstime - tmsstart-> tms_cstime) /
(double) clktck);
}
```