Lecture #9: UNIX System Calls

Reading: *none*
Handouts: 2cat.c

1. Greetings and felicitations
2. File descriptors
   - stdio analogues: open, close, read, write, lseek
3. File status information
   - attributes of file
   - getting them
4. Time
   - internal representation of time
   - converting time to printable format
5. Users and groups
   - what information is stored and where
   - getpwent, getgrent, etc.

2cat.c

```
/*
 * 2cat -- a program to print a file twice
 * This demonstrates the use of file descriptors
 *
 * Matt Bishop, bishop@cs.ucdavis.edu    11/9/96
 */
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>

/*
 * macros
 */
#define BUFSIZ      1024    /* size of buffer for file I/O */

/*
 * forward declarations
 */
void perror(char *);        /* print system error message */

/*
 * print the file with the given file descriptor
 * returns 0 on success, -1 on error
 * does not print any error message
 */
int cat(int fd)
{
            register int n;       /* number of bytes read */
            char buf[BUFSIZ];     /* buffer for file I/O */

            /*
             * grab BUFSIZ bytes and dump them to
             * the standard output
             */
            while((n = read(fd, buf, BUFSIZ)) > 0)
                    /* if error, end this function at once */
                    if (write(1, buf, n) != n)
                            return(-1);

            /* if read failed, return error; else okay */
            return(n < 0 ? -1 : 0);
}

/*
 * print the named file twice
 * returns 0 on success, 1 on error
 * prints system error message if there is a problem
 */
int twocat(char *name)
{
            register int fd;              /* file's descriptor */
            register int retval = 0;      /* value to return */

            /*
```

```
             * open the file
             */
            if ((fd = open(name, O_RDONLY)) < 0){
                    perror(name);
                    return(-1);
            }
            /*
             * display the file, rewind it, and redisplay it
             * if there's an error, print the system error
             * message and quit
             */
            if (cat(fd) < 0 || lseek(fd, 0L, 0) != 0 || cat(fd) < 0){
                    perror(name);
                    retval = 1;
            }
            /*
             * close it up and return status
             */
            (void) close(fd);
            return(retval);
}

/*
 * main routine
 */
int main(int argc, char *argv[])
{
            register int i;              /* counter in a for loop */
            register int estat = 0;      /* exit status */

            /*
             * bad usage message
             */
            if (argc == 1){
                    (void) write(2, "Usage: ", strlen("Usage: "));
                    (void) write(2, argv[0], strlen(argv[0]));
                    (void) write(2, " file [ ... ]\n",
                                        strlen(" file [ ... ]\n"));
                    return(1);
            }

            /*
             * print each file twice as you go
             */
            for(i = 1; i < argc; i++)
                    estat += twocat(argv[i]);

            /*
             * return number of failures
             */
            return(estat);
}
```

user.c

```
/*
 * program to print information about named user(s)
 * demonstrates mapping user name (etc.) and date
 *
 * Matt Bishop, bishop@cs.ucdavis.edu   11/9/96
 */
#include <ctype.h>
#include <stdio.h>
#include <grp.h>
#include <pwd.h>
#include <sys/time.h>
#include <time.h>

/*
 * forward declarations
 */
void setgrent(void);      /* initialize group file -- not in grp.h!!! */
void endgrent(void);      /* clean up group file -- not in grp.h!!! */
int gettimeofday(struct timeval *, struct timezone *); /* get time of day */

/*
 * prints the current time of day
 */
int prdate(void)
{
            struct timeval tod;   /* time of day */
            struct timezone tz;   /* time zone */
            char *atod;           /* date as a string */

            /*
             * get the time of day
             */
            if (gettimeofday(&tod, &tz) < 0)
                    return(1);
            /*
             * convert it to a string and print it
             */
            if ((atod = ctime((time_t *)&tod.tv_sec)) == NULL)
                    atod = "Time not available\n";
            printf("%s", atod);
            return(0);
}

/*
 * print secondary group list
 * this is a list of groups the user is in
 * name is name of user, gid is his/her GID
 * if empty list, print *none*
 */
void prsgid(char *name, int gid)
{
            register struct group *gp;   /* points to group info */
            register char **p;           /* walks user list of group */
            register int count = 0;      /* how many groups printed */
```

```
            /*
             * initialize group file for scan
             */
            setgrent();

            /*
             * loop through groups, looking for this user
             * in its member list
             */
            while ((gp = getgrent()) != NULL){
                    for(p = gp->gr_mem; *p; p++)
                            if (strcmp(*p, name) == 0){
                                    /* bingo! */
                                    printf(" %s(%d)", gp->gr_name,
                                                            gp->gr_gid);
                                    count++;
                                    break;
                            }
            }

            /*
             * clean up and leave
             */
            endgrent();
            if (count == 0)
                    printf(" *none*");
}

/*
 * given a GID gid, print the group name or *unknown* if none
 * also the GID.
 */
void grname(int gid)
{
            struct group *gp;            /* points to group info */

            /*
             * if you find it, print it; else say *unknown*
             */
            if ((gp = getgrgid(gid)) == NULL)
                    printf("*unknown*(%d)\n", gid);
            else
                    printf("%s(%d)\n", gp->gr_name, gid);
}

/*
 * print user information about user name; if name is number,
 * take as UID
 */
void getuinfo(char *name)
{
            register int n = 0;   /* UID if name is UID */
            register char *s;     /* used to check if name is UID or name */
            struct passwd *pw;    /* points to user info */

            /*
             * get the user information structure
```

```
             */
            for(s = name; *s && isdigit(*s); s++)
                    n = n * 10 + *s - '0';
            if (!*s){
                    /* it's an integer */
                    pw = getpwuid(n);
            }
            else{
                    /* it's not an integer */
                    pw = getpwnam(name);
            }

            /*
             * check for bad name/UID
             */
            if (pw == NULL){
                    printf("%s: unknown %s\n", name, *s ? "name" : "UID");
                    return;
            }

            /*
             * now dump the user information
             */
            printf("user name: %s\n", pw->pw_name);
            printf("hashed password: %s\n", pw->pw_passwd);
            printf("UID: %d\n", pw->pw_uid);
            /* do the principle GID as groupname(gid) */
            printf("principle GID: ");
            grname(pw->pw_gid);
            /* do the secondary GID as a list of groupname(gid)s */
            printf("secondary GIDs:");
            prsgid(pw->pw_name, pw->pw_gid);
            putchar('\n');
            printf("GECOS field: %s\n", pw->pw_gecos);
            printf("home directory: %s\n", pw->pw_dir);
            printf("login shell: %s\n", pw->pw_shell);
}

/*
 * the main routine
 */
int main(int argc, char *argv[])
{
            register int i;              /* used to walk argument list */

            /*
             * print the date
             */
            if (prdate()){
                    perror(argv[0]);
                    return(1);
            }
            /*
             * print the user info one user at a time
             * with a separating header
             */
            for(i = 1; i < argc; i++){
                    printf("========== %s ==========\n", argv[i]);
```

```
                getuinfo(argv[i]);
        }
        return(0);
}
```