

```

::::::::::::: who0.c :::::::::::::::
#include      <stdio.h>
#include      <fcntl.h>
#include      <utmp.h>
#include      <stdlib.h>
#include      <unistd.h>
/*
 *      who version 0
 *      main outline but no substance
 */
main()
{
    int      fd;                      /* for file des of utmp */
    struct utmp current_record;        /* hold info from file */
    int      reclen = sizeof(struct utmp);

    fd = open( UTMP_FILE, O_RDONLY );
    if ( fd == -1 )
    {
        perror( "who0" );
        exit(1);
    }

    while ( read ( fd , &current_record , reclen ) == reclen )
        show_info( &current_record );

    close ( fd );
}
::::::::::::: who1.c :::::::::::::::
#include      <stdio.h>
#include      <sys/types.h>
#include      <utmp.h>
#include      <fcntl.h>
#include      <stdlib.h>
#include      <unistd.h>

/*
 *      who version 1          - read /var/run/utmp and list info therein
 */

#define INFOFILE      UTMP_FILE

void show_info( struct utmp *utbufp );

int main(int ac, char **av)
{
    struct utmp      utbuf;           /* read info into here */
    int              utmpfd;          /* read from this descriptor */

    if ( (utmpfd = open( INFOFILE, O_RDONLY )) == -1 ){
        fprintf(stderr, "%s: cannot open %s\n", *av, INFOFILE);
        exit(1);
    }

    while ( read( utmpfd, &utbuf, sizeof(utbuf) ) == sizeof(utbuf) )
        show_info( &utbuf );
    close( utmpfd );
    return 0;                          /* went ok */
}

/*
 *      show info()
 *
 *      displays the contents of the utmp struct
 *      in human readable form
 *      *note* these sizes should not be hardwired
 */
void show_info( struct utmp *utbufp )
{
    // printf("%-8.8s", utbufp->ut_name);          /* the logname */
    printf("%s", utbufp->ut_name);                /* the logname */
    printf(" ");                                  /* a space */
    printf("%-8.8s", utbufp->ut_line);             /* the tty */
    printf(" ");                                  /* a space */
    printf("%10ld", (long) utbufp->ut_time);        /* login time */
    printf(" ");                                  /* a space */
    printf("(%2.56s)", utbufp->ut_host);            /* the host */
    printf("\n");                                 /* newline */
}

```

```

:.....: who2.c :.....:
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <utmp.h>
#include <fcntl.h>
#include <time.h>
#include <stdlib.h>
/*
 *      who version 2          - read /var/run/utmp and list info therein
 *                          - suppresses empty records
 *                          - formats time nicely
 */

/* UTMP_FILE is a symbol defined in utmp.h */
/* note: compile with -DTEXTDATE for dates format: Feb  5, 1978      */
/*      otherwise, program defaults to NUMDATE (1978-02-05)          */

#define TEXTDATE
#ifndef DATE_FMT
#ifdef TEXTDATE
#define DATE_FMT          "%b %e %H:%M"          /* text format */
#else
#define DATE_FMT          "%Y-%m-%d %H:%M"        /* the default */
#endif
#endif

int main(int ac, char *av[])
{
    struct utmp    utbuf;          /* read info into here */
    int            utmpfd;         /* read from this descriptor */
    void            show_info(struct utmp *);

    if ( (utmpfd = open( UTMP_FILE, O_RDONLY )) == -1 ){
        fprintf(stderr,"%s: cannot open %s\n", *av, UTMP_FILE);
        exit(1);
    }

    while ( read( utmpfd, &utbuf, sizeof(utbuf) ) == sizeof(utbuf) )
        show_info( &utbuf );
    close( utmpfd );
    return 0;
}
/*
 *      show info()
 *
 *          displays the contents of the utmp struct
 *          in human readable form
 *          * displays nothing if record has no user name
 */
void show_info( struct utmp *utbufp )
{
    void showtime(time_t, char *);

    if ( utbufp->ut_type != USER_PROCESS )
        return;

    printf("%-8s", utbufp->ut_name);          /* the logname */
    printf(" ");                             /* a space */
    printf("%-12.12s", utbufp->ut_line);      /* the tty */
    printf(" ");                             /* a space */
    showtime( utbufp->ut_time , DATE_FMT);   /* display time */
    if ( utbufp->ut_host[0] != '\0' )
        printf(" (%.256s)", utbufp->ut_host); /* the host */
    printf("\n");                           /* newline */
}

#define MAXDATELEN    100

void showtime( time_t timeval , char *fmt )
/*
 * displays time in a format fit for human consumption.
 * Uses localtime to convert the timeval into a struct of elements
 * (see localtime(3)) and uses strftime to format the data
 */
{
    char    result[MAXDATELEN];

    struct tm *tp = localtime(&timeval);    /* convert time */
    strftime(result, MAXDATELEN, fmt, tp);   /* format it */
    fputs(result, stdout);
}

```

```
::::::::::::: Makefile :::::::::::::::
#
# makefile for lecture 2
#

CC      = cc
CFLAGS = -Wall -Wextra -g

who1: who1.c
      $(CC) $(CFLAGS) -o who1 who1.c

who2: who2.c
      $(CC) $(CFLAGS) -o who2 who2.c

who3: who3.o utmplib.o
      $(CC) $(CFLAGS) -o who3 who3.o utmplib.o

llcopy: llcopy.c
      $(CC) $(CFLAGS) llcopy.c -o llcopy

utmplib.o: utmplib.c
      $(CC) $(CFLAGS) -c utmplib.c

who3.o: who3.c
      $(CC) $(CFLAGS) -c who3.c

clean:
      rm -f who1 who2 who3 llcopy utmplib.o
#
# tests if llcopy works like cp
#
copytest: llcopy
      last -20 > data
      llcopy data data.copy
      if diff data data.copy ; then echo success ; else echo error ; fi
```

```
::::::::::::: llcopy.c ::::::::::::::
#include      <stdio.h>
#include      <unistd.h>
#include      <fcntl.h>
#include      <stdlib.h>

/**
 **      low level copy
 **                      uses read and write with tunable buffer size
 **
 **      usage: llcopy src dest
 **/

#define BUFFERSIZE      4096
#define COPYMODE        0644

void fatal( char *, char *);

int main( int ac, char *av[] )
{
    int      in_fd, out_fd, n_chars;
    char      buf[BUFFERSIZE];

                                /*
                                *      check args
                                */

    if ( ac != 3 ){
        fprintf( stderr, "usage: %s source destination\n", *av);
        exit(1);
    }

                                /*
                                *      open files
                                */

    if ( ( in_fd=open(av[1], O_RDONLY) ) == -1 )
        fatal("Cannot open ", av[1]);

    if ( ( out_fd=creat( av[2], COPYMODE ) ) == -1 )
        fatal( "Cannot creat", av[2]);

                                /*
                                *      copy files
                                */

    while ( (n_chars = read( in_fd , buf, BUFFERSIZE )) > 0 )
        if ( write( out_fd, buf, n_chars ) != n_chars )
            fatal("Write error to ", av[2]);

                                /*
                                *      close them up
                                */

    if ( n_chars == -1 ){
        perror(av[1]);
        fatal("error reading input","");
    }
    if ( close(in_fd) == -1 )
        fatal("Error closing input file:", av[1]);
    if ( close(out_fd) == -1 )
        fatal("Error closing output file:", av[2]);
    return 0;
}

void
fatal(char *s1, char *s2)
{
    fprintf(stderr,"Error: %s%s\n", s1, s2);
    exit(1);
}
```

```

:.....: who3.c :.....:
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <utmp.h>
#include <fcntl.h>
#include <time.h>
#include <stdlib.h>
#include "utmp.h" /* include interface def for utmp.h */

/*
 * who version 3.0 - read /var/run/utmp and list info therein
 * - suppresses empty records
 * - formats time nicely
 * - buffers input (using utmp.h)
 */

#define TEXTDATE
#ifndef DATE_FMT
#define DATE_FMT "%b %e %H:%M" /* text format */
#else
#define DATE_FMT "%Y-%m-%d %H:%M" /* the default */
#endif
#endif

void show_info( struct utmp * );

int main(int ac, char **av)
{
    struct utmp *utbufp; /* holds pointer to next rec */

    if ( utmp_open( UTMP_FILE ) == -1 ){
        fprintf(stderr,"%s: cannot open %s\n", *av, UTMP_FILE);
        exit(1);
    }
    while ( ( utbufp = utmp_next() ) != NULL ) /* get ptr to next rec */
        show_info( utbufp );
    utmp_close( );
    return 0;
}

/*
 * show_info()
 * displays the contents of the utmp struct
 * in human readable form
 * displays nothing if record has no user name
 */
void show_info( struct utmp *utbufp )
{
    void showtime( time_t , char * );

    if ( utbufp->ut_type != USER_PROCESS )
        return;

    printf("%-8s", utbufp->ut_name); /* the logname */
    printf(" "); /* a space */
    printf("%-12.12s", utbufp->ut_line); /* the tty */
    printf(" "); /* a space */
    showtime( utbufp->ut_time, DATE_FMT ); /* display time */
    if ( utbufp->ut_host[0] != '\0' )
        printf(" (%s)", utbufp->ut_host); /* the host */
    printf("\n"); /* newline */
}

#define MAXDATELEN 100

void showtime( time_t timeval , char *fmt )
/*
 * displays time in a format fit for human consumption.
 * Uses localtime to convert the timeval into a struct of elements
 * (see localtime(3)) and uses strftime to format the data
 */
{
    char result[MAXDATELEN];

    struct tm *tp = localtime(&timeval); /* convert time */
    strftime(result, MAXDATELEN, fmt, tp); /* format it */
    fputs(result, stdout);
}

```

```

::::::::::::: utmplib.h :::::::::::::::
#ifndef UTMPLIB_H
#define UTMPLIB_H
/* utmplib.h - header file with decls of functions in utmplib.c */

#include <utmp.h>

int      utmp_open(char *);
struct utmp *utmp_next();
int      utmp_close();
#endif

::::::::::::: utmplib.c :::::::::::::::
/* utmplib.c - functions to buffer reads from utmp file
 *
 * functions are
 *      int utmp_open( filename )      - open file
 *      returns -1 on error
 *      struct utmp *utmp_next( )     - return pointer to next struct
 *      returns NULL on eof or read error
 *      int utmp_close()              - close file
 *
 * reads NRECS per read and then doles them out from the buffer
 * hist: 2012-02-14 slightly modified error handling (thanks mk)
 * note: this version is extended from the handout version: this tracks cache hits
 */
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <utmp.h>
#include <unistd.h>

#define NRECS    32
#define UTSIZE   (sizeof(struct utmp))

static struct utmp utmpbuf[NRECS]; /* storage */
static int num_recs;                /* num stored */
static int cur_rec;                 /* next to go */
static int fd_utmp = -1;            /* read from */

static int utmp_reload();

/*
 * utmp_open -- connect to specified file
 * args: name of a utmp file
 * rets: -1 if error, fd for file otherwise
 */
int utmp_open( char *filename )
{
    fd_utmp = open( filename, O_RDONLY ); /* open it */
    cur_rec = num_recs = 0;               /* no recs yet */
    return fd_utmp;                       /* report */
}

/*
 * utmp_next -- return address of next record in file
 * args: none
 * rets: address of record in buffer
 * note: this location will be reused on next buffer reload
 */
struct utmp *utmp_next()
{
    struct utmp *recp;

    if ( fd_utmp == -1 ) /* error ? */
        return NULL;
    if ( cur_rec==num_recs && utmp_reload() <= 0 ) /* any more ? */
        return NULL;

    recp = &(utmpbuf[cur_rec]); /* get address of next record */
    cur_rec++;
    return recp;
}

```

```
static int utmp_reload()
/*
 *   read next bunch of records into buffer
 *   rets: 0=>EOF, -1=>error, else number of records read
 */
{
    int    amt_read;

    amt_read = read(fd_utm, utmpbuf, NRECS*UTSIZE); /* read data    */
    if ( amt_read < 0 ) /* mark errors as EOF */
        return -1;

    num_recs = amt_read/UTSIZE; /* how many did we get? */
    cur_rec  = 0; /* reset pointer */
    return num_recs; /* report results */
}

/*
 * utmp_close -- disconnect
 * args: none
 * rets: ret value from close(2) -- see man 2 close
 */
int utmp_close()
{
    int rv = 0;
    if ( fd_utm != -1 ){
        rv = close( fd_utm ); /* don't close if not */
        fd_utm = -1; /* open */
    } /* record as closed */
    return rv;
}
```
