

```

/* ac.c */

/* Copyright (C) 1993, 1996, 1997 Free Software Foundation, Inc.

This file is part of the GNU Accounting Utilities

The GNU Accounting Utilities are free software; you can redistribute
them and/or modify them under the terms of the GNU General Public
License as published by the Free Software Foundation; either version
2, or (at your option) any later version.

The GNU Accounting Utilities are distributed in the hope that they will
be useful, but WITHOUT ANY WARRANTY; without even the implied warranty
of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with the GNU Accounting Utilities; see the file COPYING. If
not, write to the Free Software Foundation, 675 Mass Ave, Cambridge,
MA 02139, USA. */

#include "config.h"
#include <stdio.h>

#ifdef HAVE_STDLIB_H
#include <stdlib.h>
#endif

#ifdef HAVE_STRING_H
#include <string.h>
#endif

#include <sys/types.h>

#ifdef TIME_WITH_SYS_TIME
# include <sys/time.h>
# include <time.h>
#else
# if HAVE_SYS_TIME_H
# include <sys/time.h>
# else
# include <time.h>
# endif
#endif

#include "common.h"
#include "utmp_rd.h"
#include "getopt.h"
#include "hashtab.h"
#include "version.h"

/* static stuff */

static char *months[] = {
    "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",
    "Aug", "Sep", "Oct", "Nov", "Dec"
};

#define number_precision 8

/* globals */

char *program_name; /* name of the program, for usage & errs */

int debugging_enabled = 0; /* Nonzero means print internal

```

```

        information relating to entries
        read, data structures, etc. */

long time_warp_leniency = 60; /* Number of seconds that an entry can
        be "before" the previous one -- to
        deal with simultaneous init(8)
        writes, or to overlook problems
        with sun's wtmp files. */

long time_warp_suspicious = 31536000; /* Anything smaller than this
        jump will be considered
        normal. A year is pretty
        safe to assume... */

int print_file_problems = 0; /* when there's a problem with the
        file's format, scream! */

int print_year = 0; /* Print the year when display
        dates. */

time_t last_time = 0; /* time of last event (error checking)
        needs to be a global so that the
        signal handler can get at its value
        when the program is interrupted */

int nasty_reboot = 0; /* most ac's count the time that the
        user was logged in before a reboot
        against them. Off by default. */

int nasty_supplant = 0; /* if the same tty appears for a login
        without an intervening logout,
        there's a problem. Most ac's count
        the time between the two logins
        against the user. Off by
        default. */

int nasty_time_warp = 0; /* if the running date in the wtmp
        file jumps backwards, charge the
        user until midnight of the day they
        logged in. Off by default. */

int print_all_days = 0; /* Print a record for every day of the
        year (if we're printing days).
        This is slower, because we end up
        making a bunch more calls to
        mktime, but it's useful for
        printing out usage graphs that have
        equal spacing... */

time_t next_midnight; /* variable keeps track of the next
        midnight relative to the time of
        the current entry being processed */

int print_individual_totals = 0; /* -p flag */
int print_midnight_totals = 0; /* -d flag */
int print_zero_totals = 0; /* -z flag */

/* A hash table to hold user totals. The names themselves are the
    keys for this table. */

struct hashtab *user_totals = NULL;

struct user_data {

```

```

    unsigned long time;
};

/* A hash table which contains the names that the user wants
   printed. The names themselves are the keys for this table. */

struct hashtab *names = NULL;

/* A table for the currently logged-in ttys. The UT_LINE fields are
   the keys for this table. */

struct hashtab *login_table;

struct login_data {
    char ut_name[NAME_LEN];
    time_t time;
};

/* prototypes */

void main PARAMS((int, char *[]));
void give_usage PARAMS((void));
void do_statistics PARAMS((char *));
void log_in PARAMS((struct utmp *));
void update_user_time PARAMS((char *, time_t, char *));
void log_out PARAMS((struct utmp *));
void log_everyone_out PARAMS((time_t, int, int, char *));
void parse_entries PARAMS((void));
void do_totals PARAMS((time_t *, time_t, int, int, char *));
void update_system_time PARAMS((time_t));
time_t midnight_after_me PARAMS((time_t));

/* code */

void
main (int argc, char *argv[])
{
    int c;
    int other_wtmp_file_specified = 0; /* nonzero if the user used the
        '-f' or '--other-file' flags */

    program_name = argv[0];

    /* Init the utmp reader for reading forwards in the files. */

    utmp_init (0);

    while (1)
    {
        int option_index = 0;

        static struct option long_options[] = {
            { "complain", no_argument, NULL, 1 },
            { "reboots", no_argument, NULL, 2 },
            { "supplants", no_argument, NULL, 3 },
            { "timewarps", no_argument, NULL, 4 },
            { "print-zeros", no_argument, NULL, 5 },
            { "debug", no_argument, NULL, 6 },
            { "tw-leniency", required_argument, NULL, 7 },

```

```

{ "version", no_argument, NULL, 8 },
{ "help", no_argument, NULL, 9 },
{ "daily-totals", no_argument, NULL, 10 },
{ "individual-totals", no_argument, NULL, 11 },
{ "file", required_argument, NULL, 12 },
{ "compatibility", no_argument, NULL, 13 },
{ "print-year", no_argument, NULL, 14 },
{ "all-days", no_argument, NULL, 15 },
{ "tw-suspicious", required_argument, NULL, 16 },
{ 0, 0, 0, 0 }
};

c = getopt_long (argc, argv, "adf:hpVyz", long_options, &option_index);

if (c == EOF)
break;

switch (c)
{
case 1:
print_file_problems = 1;
break;
case 2:
nasty_reboot = 1;
break;
case 3:
nasty_supplant = 1;
break;
case 4:
nasty_time_warp = 1;
break;
case 'z':
case 5:
print_zero_totals = 1;
break;
case 6:
debugging_enabled = 1;
print_file_problems = 1;
break;
case 7:
time_warp_leniency = atol (optarg);
if (time_warp_leniency < 0)
fatal ("time warp leniency value has to be non-negative");
break;
case 'V':
case 8:
printf ("%s: GNU Accounting Utilities (release %s)\n",
program_name, VERSION_STRING);
exit (0);
break;
case 10:
case 'd':
print_midnight_totals = 1;
break;
case 11:
case 'p':
print_individual_totals = 1;
break;
case 12:
case 'f':
add_utmp_file (optarg);
other_wtmp_file_specified = 1;
break;
case 13:
/* u*x compatibility */

```

```

    nasty_time_warp = 1;
    nasty_supplant = 1;
    nasty_reboot = 1;
    break;
case 'y':
case 14:
    print_year = 1;
    break;
case 'a':
case 15:
    print_all_days = 1;
    break;
case 16:
    time_warp_suspicious = atol (optarg);
    if (time_warp_suspicious < 0)
        fatal ("time warp suspicious value has to be non-negative");
    if (time_warp_suspicious <= time_warp_leniency)
        fatal ("time warp suspicious value has to greater than the time warp lenie
    break;
case 'h':
case 9:
    /* This should fall through to default! */
default:
    give_usage ();
    exit (1);
    break;
}
}

/* Init the hash table for usernames.  Don't init it if we don't
have any extra arguments so we can quickly check if we have
anything in the list. */

if (optind < argc)
{
    names = hashtab_init (0);

    while (optind < argc)
        hashtab_create (names, argv[optind++], 0);

    if (debugging_enabled)
        hashtab_dump_keys (names, stddebug);
}

if (! other_wtmp_file_specified)
    add_utmp_file (WTMP_FILE_LOC);

/* Create hash tables for user totals and logins. */

user_totals = hashtab_init (0);
login_table = hashtab_init (0);

/* Do it! */

parse_entries ();

/* At this point, there are no more entries in the wtmp file.  We
need to do statistics for everything up to now, so update all of
the people who are logged in presently and print today's
statistics. */

{
    time_t now = time ((time_t *)0);

    /* If we're processing a file with times from the future, we can't

```

```

do any of this "last day" business. The best we can do is use
the time from the last record in the file. */

if (now < last_time)
    now = last_time;

do_totals (&next_midnight, now, TRUE, TRUE, "midnight logout");

log_everyone_out (now, TRUE, FALSE, "catch-up");
}

do_statistics (print_midnight_totals
               ? (print_year ? "Today\t" : "Today")
               : "");

exit (0);          /* guarantee the proper return value */
}

/* guess what this does... */
void
give_usage (void)
{
    char *usage = "\
Usage: %s [-dhpVy] [-f <file>] [people] ... \n\
        [--daily-totals] [--individual-totals] [--file <file>] \n\
        [--complain] [--reboots] [--supplants] [--timewarps] [--print-year] \n\
        [--compatibility] [--print-zeros] [--debug] [--tw-lenieny <value>] \n\
        [--tw-suspicious <value>] [--version] [--help] \n";

    printf (usage, program_name);
    print_wtmp_file_location ();
}

/* Since the routines in ac & last are so similar, just include them
   from another file. */

#include "al_share.cpp"

/* since the sys clock has changed, each entry's login time has to be
   * adjusted... */
void
update_system_time (time_t the_time)
{
    struct hashtab_order ho;
    struct hashtab_elem *he;

    for (he = hashtab_first (login_table, &ho);
         he != NULL;
         he = hashtab_next (&ho))
    {
        struct login_data *l = hashtab_get_value (he);
        l->time += the_time;
    }
}

/* Log all entries out at THE_TIME. Update statistics if
   UPDATE_TIME_FLAG is non-zero, and print out the entries preceded by
   DEBUG_STR. If CHANGE_LOGIN_FLAG is non-zero, reset the login times
   to THE_TIME. */

```

```

void
log_everyone_out (time_t the_time, int update_time_flag,
                 int change_login_flag, char *debug_str)
{
    struct hashtab_order ho;
    struct hashtab_elem *he;

    for (he = hashtab_first (login_table, &ho);
         he != NULL;
         he = hashtab_next (&ho))
    {
        struct login_data *l = hashtab_get_value (he);

        if (update_time_flag)
            update_user_time (l->ut_name, the_time - l->time, debug_str);

        if (change_login_flag)
            l->time = the_time;
        else
            hashtab_delete (he);
    }
}

/* Put a terminal into the hash table. */

void
log_in (struct utmp *entry)
{
    struct hashtab_elem *he;

    if (entry->ut_line[0] == '\0')
    {
        if (print_file_problems)
        {
            utmp_print_file_and_line (stddebug);
            fprintf (stddebug,
                  ": problem: trying to hash rec with ut_line == NULL\n");
        }
        return;
    }

    he = hashtab_find (login_table, entry->ut_line, TTY_LEN);

    if (he != NULL)
    {
        struct login_data *l = hashtab_get_value (he);

        if (print_file_problems)
        {
            char *ttyname = hashtab_get_key (he);
            utmp_print_file_and_line (stddebug);
            fprintf (stddebug, ": problem: duplicate record for line '%.*s'\n",
                  TTY_LEN, ttyname);
        }

        /* we should just write over the old one -- nasty ac's charge
        the user being for being logged in until this new tty entry
        appears, so the flag NASTY_SUPPLANT is included */

        if (nasty_supplant)
            update_user_time (l->ut_name, entry->ut_time - l->time, "supplant");

        strncpy (l->ut_name, entry->ut_name, NAME_LEN);
        l->time = entry->ut_time;
    }
}

```

```

    }
else
{
    /* If we get here, we didn't find the entry in the list, so add
       a new one. */

    struct login_data l;

    strncpy (l.ut_name, entry->ut_name, NAME_LEN);
    l.time = entry->ut_time;

    he = hashtab_create (login_table, entry->ut_line, TTY_LEN);
    hashtab_set_value (he, &l, sizeof (l));
}
}

/* Remove an entry from the hash table. */

void
log_out (struct utmp *entry)
{
    struct hashtab_elem *he;

    if (entry->ut_line[0] == '\0')
    {
        if (print_file_problems)
        {
            utmp_print_file_and_line (stddebug);
            fprintf (stddebug,
                    ": problem: trying to hash rec with ut_line == NULL\n");
        }
        return;
    }

    /* Match the most recent login on the terminal. */

    he = hashtab_find (login_table, entry->ut_line, TTY_LEN);

    if (he != NULL)
    {
        struct login_data *l = hashtab_get_value (he);

        update_user_time (l->ut_name, entry->ut_time - l->time, "logout");

        hashtab_delete (he);
    }
else
{
    /* There's a problem with the wtmp file, since we couldn't find
       any login corresponding to this logout. */

    if (print_file_problems)
    {
        utmp_print_file_and_line (stddebug);
        fprintf (stddebug, ": problem: missing login record for '%.*s'\n",
                TTY_LEN, entry->ut_line);
    }
}
}

/* fills in the days between entries, calculating how much time a user
 * has racked up by midnight of each period

```

```

*
* side-effect (desirable) -- changes the value passed in NEXT_MIDNIGHT
* to catch up with CURRENT_TIME
*/
void
do_totals (time_t *next_midnight, time_t current_time,
           int update_time_flag, int change_login_flag, char *debug_str)
{
    while (*next_midnight < current_time)
    {
        log_everyone_out (*next_midnight, update_time_flag, change_login_flag,
                          debug_str);

        if (print_midnight_totals)
        {
            /* We need the proper label for do_statistics: we can get it
               relative to next_midnight. */

            char month_day_string[255];
            time_t temp_time = *next_midnight - 10;
            struct tm *temp_tm = localtime (&temp_time);
            if (print_year)
                sprintf (month_day_string, "%s %2d %4d",
                        months[temp_tm->tm_mon], temp_tm->tm_mday,
                        1900 + temp_tm->tm_year);
            else
                sprintf (month_day_string, "%s %2d",
                        months[temp_tm->tm_mon], temp_tm->tm_mday);

            do_statistics (month_day_string);
        }

        /* Get the next day ONLY if we're printing totals at midnight
           and the user wants to see all of the days (because it's much
           slower). */

        *next_midnight =
        midnight_after_me ((print_midnight_totals && print_all_days)
                          ? *next_midnight
                          : current_time);
    }
}

/* print out statistics and clear the user totals
* if !PRINT_INDIVIDUAL_TOTALS && !PRINT_MIDNIGHT_TOTALS
* don't print anything until the end
* if PRINT_MIDNIGHT_TOTALS && !PRINT_INDIVIDUAL_TOTALS
* print totals for each day, & clear user times
* if PRINT_MIDNIGHT_TOTALS && PRINT_INDIVIDUAL_TOTALS
* print totals, individ user times, & clear times
*/
void
do_statistics (char *date_string)
{
    unsigned long total;
    double float_total;
    struct hashtab_order ho;
    struct hashtab_elem *he;

    total = 0;
    float_total = 0.0;

    for (he = hashtab_first (user_totals, &ho);

```

```

    he != NULL;
    he = hashtable_next (&ho)
}
char *username = hashtable_get_key (he);

/* print if the user table is null or the name is on the command line */
if ((names == NULL) || hashtable_find (names, username, NAME_LEN))
{
    struct user_data *u = hashtable_get_value (he);

    total += u->time;

    if ((print_individual_totals)
        && !((u->time == 0) && !print_zero_totals))
        printf ("%s%-*.s %*.2f\n",
            (print_year ? "\t\t" : "\t"),
            NAME_LEN, NAME_LEN, username,
            number_precision, (double) u->time / 3600.0);
}

    hashtable_delete (he);
}

float_total = (double) total / 3600.0;

if (print_midnight_totals)
{
    if (!((float_total == 0.0) && !print_zero_totals))
        printf ("%s\t\ttotal %*.2f\n", date_string,
            number_precision + 3, float_total);
}
else
    printf ("%stotal %*.2f\n",
        (print_year ? "\t\t" : "\t"),
        number_precision + 3, float_total);
}

/* put something into the user table */
void
update_user_time (char *name, time_t the_time, char *debug_label)
{
    struct hashtable_elem *he;

    /* If we've rolled off the end of the range of time_t, it's possible
       to have negative values for THE_TIME. Yuck. Notify the user and
       quit. */

    if (the_time < 0)
    {
        utmp_print_file_and_line (stdout);
        fprintf (stderr,
            ": problem: update user %.*s (for %s) with %ld seconds?!\n",
            NAME_LEN, name, debug_label, (long) the_time);
        fatal ("Possible overflow of time_t! Can't continue.");
    }

    if (debugging_enabled
        && ((names == NULL) || hashtable_find (names, name, NAME_LEN)))
        fprintf (stderr,
            "\t\t\t\t\t%*.2f %-*.*s (%s)\n",
            NAME_LEN, (double) the_time / 3600.0,
            NAME_LEN, NAME_LEN, name, debug_label);

    he = hashtable_find (user_totals, name, NAME_LEN);

```

```
if (he == NULL)
{
    struct user_data u;

    u.time = 0;
    he = hashtab_create (user_totals, name, NAME_LEN);
    hashtab_set_value (he, &u, sizeof (u));
}

/* Now we have an entry -- add the time. */

{
    struct user_data *u = hashtab_get_value (he);
    u->time += the_time;
}

}

/* return the time of midnight that will happen after our time */
time_t
midnight_after_me (time_t now_time)
{
    struct tm *tm_ptr, temp_tm;

    tm_ptr = localtime (&now_time);
    memcpy ((void *)&temp_tm, (void *)tm_ptr, sizeof (struct tm));
    temp_tm.tm_sec = 0;
    temp_tm.tm_min = 0;
    temp_tm.tm_hour = 0;
    temp_tm.tm_mday++;      /* next day */
    temp_tm.tm_isdst = -1; /* mktime should figure out whether
                           DST is enabled */

    return mktime (&temp_tm);
}
```