/* * Copyright (c) 1983 Regents of the University of California. * All rights reserved. The Berkeley software License Agreement * specifies the terms and conditions for redistribution. */

#ifndef lint
static char sccsid[] = "@@(#)ctl.c 5.2 (Berkeley) 3/13/86"
#endif

/* This file handles haggling with the various talk daemons to get a socket to talk to. sockt is opened and connected in the progress */

#include "talk_ctl.h"

struct sockaddr_in daemon_addr = { AF_INET };

struct sockaddr_in ctl_addr = { AF_INET };

struct sockaddr_in my_addr = { AF_INET };

/* inet addresses of the two machines */

struct in_addr my_machine_addr;

struct in_addr his_machine_addr;

u_short daemon_port; /* port number of the talk daemon */

int ctl_sockt;

int sockt;

int invitation_waiting = 0;

CTL_MSG msg;

open_sockt()
{
  int length;

  my_addr.sin_addr = my_machine_addr;
  my_addr.sin_port = 0;
  sockt = socket(AF_INET, SOCK_STREAM, 0);
  if (sockt <= 0)
    p_error("Bad socket");
  if (bind(sockt, &my_addr, sizeof(my_addr)) != 0)
    p_error("Binding local socket");
  length = sizeof(my_addr);
  if (getsockname(sockt, &my_addr, &length) == −1)
    p_error("Bad address for socket");
}

/* open the ctl socket */

open_ctl()
{
  int length;

  ctl_addr.sin_port = 0;
  ctl_addr.sin_addr = my_machine_addr;
  ctl_sockt = socket(AF_INET, SOCK_DGRAM, 0);
  if (ctl_sockt <= 0)
    p_error("Bad socket");
  if (bind(ctl_sockt, &ctl_addr, sizeof(ctl_addr), 0) != 0)
    p_error("Couldn't bind to control socket");
  length = sizeof(ctl_addr);
  if (getsockname(ctl_sockt, &ctl_addr, &length) == −1)
    p_error("Bad address for ctl socket");
}

/* print_addr is a debug print routine */

print_addr(addr)
#ifndef lint
static char sccsid[] = "@(#)ctl_transact.c 5.2 (Berkeley) 3/13/86";
#endif

#include "talk_ctl.h"
#include <sys/time.h>
#define CTL_WAIT 2 /* time to wait for a response, in seconds */

/* SOCK_DGRAM is unreliable, so we must repeat messages if we have 
* not received an acknowledgement within a reasonable amount 
* of time */
ctl_transact(target, msg, type, rp) 
struct in_addr target;
CTL_MSG msg;
int type;
CTL_RESPONSE *rp;

int read_mask, ctl_mask, nready, cc;
struct timeval wait;

msg.type = type;
daemon_addr.sin_addr = target;
daemon_addr.sin_port = daemon_port;
ctl_mask = 1 << ctl_sockt;

/* Keep sending the message until a response of 
* the proper type is obtained. */
do {
    wait.tv_sec = CTL_WAIT;
    wait.tv_usec = 0;
    /* resend message until a response is obtained */
    do {
        cc = sendto(ctl_sockt, (char *)&msg, sizeof(msg), 0,
                    &daemon_addr, sizeof(daemon_addr));
        if (cc != sizeof(msg)) {
            if (errno == EINTR)
                continue;
            perror("Error on write to talk daemon");
        }
        read_mask = ctl_mask;
        nready = select(32, &read_mask, 0, 0, &wait);
        if (nready < 0) {
            if (errno == EINTR)
                continue;
            perror("Error waiting for daemon response");
        }
    } while (nready == 0);
    /* Keep reading while there are queued messages 
    * (this is not necessary, but just saves extra 
    * request/acknowledgements being sent) */
    do {
        cc = recv(ctl_sockt, (char *)rp, sizeof(*rp), 0);
        if (cc < 0) {
            if (errno == EINTR)
                continue;
            perror("Error on read from talk daemon");
        }
    } while (rp->vers != TALK_VERSION || rp->type != type);
    rp->id_num = ntohl(rp->id_num);
    rp->addr.sa_family = ntohs(rp->addr.sa_family);
}
/* Copyright (c) 1983 Regents of the University of California. * All rights reserved. The Berkeley software License Agreement * specifies the terms and conditions for redistribution. */

#ifndef lint
static char scssid[] = "@(#)display.c 5.1 (Berkeley) 6/6/85";
#endif

/* The window 'manager', initializes curses and handles the actual * displaying of text */
#include "talk.h"

xwin_t my_win; xwin_t his_win; WINDOW *line_win;

int curses_initialized = 0;

/* max HAS to be a function, it is called with * a argument of the form --foo at least once. */
max(a,b)
int a, b;
{
return (a > b ? a : b);
}

/* Display some text on somebody's window, processing some control * characters while we are at it. */
display(win, text, size)
register xwin_t *win;
register char *text;
int size;
{
register int i;
char cch;
for (i = 0; i < size; i++) {
  if (*text == '\n') { /* erase character */
    xscroll(win, 0); text++;
    continue;
  }
  else
    waddch(win->x_win, *text);
  getyx(win->x_win, win->x_line, win->x_col);
}
/* * Read the character at the indicated position in win */
readwin(win, line, col)
WINDOW *win;

*/

endcol = win->x_col;
xcol = endcol - 1;
while (xcol >= 0) {
  c = readwin(win->x_win, win->x_line, xcol);
  if (c != EOF) { /* * On word erase search backwards until we find * the beginning of a word or the beginning of * the line. */
    if (*text == win->werase) { /* line kill */
      if (*text == win->kill) { /* line kill */
        wmove(win->x_win, win->x_line, xcol+1);
        wrefresh(win->x_win);
        continue;
      }
      if (*text == '\f') {
        wmove(win->x_win, win->x_line, xcol+1);
        text++;
        continue;
      }
    }
    text++;
    c = winch(win->x_win, xcol);
    if (c != EOF) { /* line kill */
      if (*text == win->kill) { /* line kill */
        wmove(win->x_win, win->x_line, xcol+1);
        wrefresh(win->x_win);
        continue;
      }
      if (*text == '\f') {
        wmove(win->x_win, win->x_line, xcol+1);
        text++;
        continue;
      }
    }
    xscroll(win, 0);
  }
  waddch(win->x_win, c);
  getyx(win->x_win, win->x_line, win->x_col);
}
}
*/

/* * Read the character at the indicated position in win */
readwin(win, line, col)
WINDOW *win;

*/

getyx(win, oldline, oldcol);
register int c;
getyx(win, oldline, oldcol);
wmove(win, line, col);
c = winch(win);
wmove(win, oldline, oldcol);
return (c);
*/

/* line kill */
if (*text == win->kill) { /* line kill */
  wmove(win->x_win, win->x_line, xcol+1);
  wrefresh(win->x_win);
  continue;
}
if (*text == '\f') {
  wmove(win->x_win, win->x_line, xcol+1);
  text++;
  continue;
}
if (win->x_col == COLS-1) { /* * check for wraparound */
xscroll(win, 0);
}
if (*text != ' ' && *text != ' ')
  waddch(win->x_win, *text);
  getyx(win->x_win, win->x_line, win->x_col);
  text++;
  continue;
if (win->x_col == COLS-1) /* * check for wraparound */
  waddch(win->x_win, *text);
  getyx(win->x_win, win->x_line, win->x_col);
  text++;
  continue;
/* * On word erase search backwards until we find * the beginning of a word or the beginning of * the line. */
if (*text == win->werase) { /* line kill */
  int oldline, oldcol;
  register int c;
  getyx(win, oldline, oldcol);
  wmove(win, line, col);
c = winch(win);
  wmove(win, oldline, oldcol);
  return (c);
*/
/* Scroll a window, blanking out the line following the current line */
/* so that the current position is obvious */
xscroll(win, flag)
    register xwin_t *win;
    int flag;
{
    if (flag == -1) {
        wmove(win->x_win, 0, 0); win->x_line = 0; win->x_col = 0;
        return;
    }
    win->x_line = (win->x_line + 1) % win->x_nlines;
    win->x_col = 0;
    wmove(win->x_win, win->x_line, win->x_col);
    wclrtoeol(win->x_win);
    wmove(win->x_win, win->x_line + 1, win->x_col);
    wclrtoeol(win->x_win);
    wmove(win->x_win, win->x_line, win->x_col);
}

get_addrs(my_machine_name, his_machine_name)
    char *my_machine_name, *his_machine_name;
{
    struct hostent *hp;
    struct servent *sp;
    msg.pid = htonl(getpid());
    /* look up the address of the local host */
    hp = gethostbyname(my_machine_name);
    if (hp == (struct hostent *) 0) {
        fprintf(stderr, "talk: %s: Can't figure out network address.
                my_machine_name); exit(-1);
    }
    bcopy(hp->h_addr, (char *)&my_machine_addr, hp->h_length);
    /* If the callee is on-machine, just copy the * network address, otherwise do a lookup... */
    if (strcmp(his_machine_name, my_machine_name)) {
        hp = gethostbyname(his_machine_name);
        if (hp == (struct hostent *) 0 ) {
            fprintf(stderr, "talk: %s/%s: service is not registered.
                    my_machine_name); exit(-1);
        }
        bcopy(hp->h_addr, (char *)&his_machine_addr, hp->h_length);
    } else
        his_machine_addr = my_machine_addr;
    /* find the server's port */
    sp = getservbyname("ntalk", "udp");
    if (sp == 0) {
        fprintf(stderr, "talk: %s/%s: service is not registered.
                    my_machine_name); exit(-1);
    }
    daemon_port = sp->s_port;
/* * Copyright (c) 1983 Regents of the University of California. * All rights reserved. The Berkeley software License Agreement * specifies the terms and conditions for redistribution. */

#ifndef lint
static char sccsid[] = 
"@(#)get_names.c 5.2 (Berkeley) 3/13/86"; #endif not lint

#include "talk.h"
#include <sys/param.h>
#include "talkd.h"
char *getlogin();
char *ttyname();
char *rindex();
static any();
extern CTL_MSG msg;

/* * Determine the local and remote user, tty, and machines * /
get_names(argc, argv)
int argc;
char *argv[];
char *hostname[MAXHOSTNAMELEN];
char *his_name, *my_name;
char *my_machine_name, *his_machine_name;
char *my_tty, *his_tty;
register char *cp;
if (argc < 2 ) {
    printf("Usage: talk user \[ttyname\]\n
    exit(−1);
} else {
    my_name = getlogin();
    if (my_name == NULL) {
        printf("You don’t exist. Go away\n\n    exit(−1);
} else {
    gethostname(hostname, sizeof (hostname));
    my_machine_name = hostname;
    my_tty = rindex(ttyname(0), ‘/’) + 1;
    /* check for, and strip out, the machine name of the target */
    for (cp = argv[1]; *cp && !any(*cp, "/@:!.\n
    if (*cp == ‘0’ ) { /* this is a local to local talk */
        his_name = argv[1];
        his_machine_name = my_machine_name;
} else {
    if (*cp == ‘!’ ) { /* this is a local to local talk */
        his_name = argv[1];
        his_machine_name = my_machine_name;
} else {
    if (*cp == ‘@’ ) { /* user@Host */
        his_name = argv[1];
        his_machine_name = cp;
} else { /* host, user or host:user or host:user */
        his_name = cp;
        his_machine_name = argv[1];
} else {
    if (argc > 2)
    if (argc < 2 ) {
        printf("Usage: talk user \[ttyname\]\n
    exit(−1);
} else {
    msg.vers = TALK_VERSION;
    msg.addr.sa_family = htons(AF_INET);
    msg_id_num = htonl(0);
    strncpy(msg.l_name, my_name, NAME_SIZE);
    msg.l_name[NAME_SIZE − 1] = ‘\0’;
    strncpy(msg.r_name, his_name, NAME_SIZE);
    msg.r_name[NAME_SIZE − 1] = ‘\0’;
    strncpy(msg.r_tty, his_tty, TTY_SIZE);
    msg.r_tty[TTY_SIZE − 1] = ‘\0’;
} */

# include "talkd.h"
#include <sys/param.h>
#include "talk.h"
char *getlogin();
char *ttyname();
char *rindex();
static any();
extern CTL_MSG msg;

/* * Determine the local and remote user, tty, and machines * /
get_names(argc, argv)
int argc;
char *argv[];
char *hostname[MAXHOSTNAMELEN];
char *his_name, *my_name;
char *my_machine_name, *his_machine_name;
char *my_tty, *his_tty;
register char *cp;
if (argc < 2 ) {
    printf("Usage: talk user \[ttyname\]\n
    exit(−1);
} else {
    my_name = getlogin();
    if (my_name == NULL) {
        printf("You don’t exist. Go away\n\n    exit(−1);
} else {
    gethostname(hostname, sizeof (hostname));
    my_machine_name = hostname;
    my_tty = rindex(ttyname(0), ‘/’) + 1;
    /* check for, and strip out, the machine name of the target */
    for (cp = argv[1]; *cp && !any(*cp, "/@:!.\n
    if (*cp == ‘0’ ) { /* this is a local to local talk */
        his_name = argv[1];
        his_machine_name = my_machine_name;
} else {
    if (*cp == ‘!’ ) { /* this is a local to local talk */
        his_name = argv[1];
        his_machine_name = my_machine_name;
} else {
    if (*cp == ‘@’ ) { /* user@Host */
        his_name = argv[1];
        his_machine_name = cp;
} else { /* host, user or host:user or host:user */
        his_name = cp;
        his_machine_name = argv[1];
} else {
    if (argc > 2)
/* * Copyright (c) 1983 Regents of the University of California. * All rights reserved. The Berkeley software license Agreement * specifies the terms and conditions for redistribution. */

#ifndef lint
static char sccsid[] = "@(#)init_disp.c 5.1(Berkeley) 6/6/85";
#endif not lint

/* * Initialization code for the display package, * as well as the signal handling routines. */

#include "talk.h"
#include <signal.h>

/*  * Set up curses, catch the appropriate signals, * and build the various windows. */

init_display() {
    void sig_sent();
    struct sigvec sigv;

    initscr();
    sigvec(SIGTSTP, (struct sigvec *)0, &sigv);
    sigv.sv_mask |= sigmask(SIGALRM);
    curcsa_initialized = 1;
    clear();
    refresh();
    noecho();
    crmode();
    signal(SIGINT, sig_sent);
    signal(SIGPIPE, sig_sent);
    // curses takes care of 5 */
    my_win.x_nlines = LINES / 2;
    my_win.x_ncols = COLS;
    my_win.x_win = newwin(my_win.x_nlines, my_win.x_ncols, 0, 0);
    scrollok(my_win.x_win, FALSE);
    wclear(my_win.x_win);
    his_win.x_nlines = LINES / 2 - 1;
    his_win.x_ncols = COLS;
    his_win.x_win = newwin(his_win.x_nlines, his_win.x_ncols, my_win.x_nlines + 1, 0);
    scrollok(his_win.x_win, FALSE);
    wclear(his_win.x_win);
    line_win = newwin(1, COLS, my_win.x_nlines, 0);
    box(line_win, '.', '.');
    wrefresh(line_win);
    /* let them know we are working on it */
    current_state = "No connection yet";

    struct ltchars ltc;
    ioctl(0, TIOCGETP, &tty);
    ioctl(0, TIOCGLTC, (struct sgttyb *)&ltc);
    my_win.cerase = tty.sg_erase;
    my_win.kill = tty.sg_killer;
    if (ltc.t_werasc == (char) -1) my_win.werase = 'W'; /* control W */
    else my_win.werase = ltc.t_werasc;
    buf[0] = my_win.cerase;
    buf[1] = my_win.kill;
    buf[2] = my_win.werase;
    cc = write(sockt, buf, sizeof(buf));
    if (cc != sizeof(buf)) p_error("Lost the connection");
    cc = read(sockt, buf, sizeof(buf));
    if (cc != sizeof(buf)) p_error("Lost the connection");
    his_win.cerase = buf[0];
    his_win.kill = buf[1];
    his_win.werase = buf[2];
}

void sig_sent() {
    message("Connection closing Exiting");
    quit();
}

/* * All done talking...hang up the phone and reset terminal thingy's */
quit() {
    if (curses_initialized) {
        wmove(his_win.x_win, his_win.x_nlines-1, 0);
        wclrtoeol(his_win.x_win);
        wrefresh(his_win.x_win);
        endwin();
    }
    if (invitation_waiting) send_delete();
    exit(0);
}

/* Trade edit characters with the other talk. By agreement * the first three characters each talk transmits after * connection are the three edit characters. */

set_edit_chars() {
    char buf[3];
    int cc;
    struct sgttyb tty;
```c
#ifndef lint
static char scsccsid[] = "@(#)invite.c 5.2 (Berkeley) 3/13/86";
#endif

#include "talk_ctl.h"
#include <sys/time.h>
#include <signal.h>
#include <setjmp.h>

int local_id, remote_id;

void re_invite();
jmp_buf invitebuf;

void invite_remote()
{
    int sockfd, read_mask, template, new_sockfd;
    struct itimerval itimer;
    CTL_RESPONSE response;

    itimer.it_value.tv_sec = RING_WAIT;
    itimer.it_value.tv_usec = 0;
    itimer.it_interval = itimer.it_value;

    if (listen(sockfd, 5) != 0)
        perror("Error on attempt to listen for caller");

    msg.addr = *(struct sockaddr *)&my_addr;
    msg.addr.sa_family = htons(msg.addr.sa_family);
    msg.id_num = htonl(-1);

    invitation_waiting = 1;
    announce_invite();

    /* Shut off the automatic messages for a while, * so we can use the interrupt timer to resend the invitation */
    end_msgs();
    setitimer(ITIMER_REAL, &itimer, (struct itimerval *)0);
    signal(SIGALRM, re_invite);

    while ((new_sockfd = accept(sockfd, 0, 0)) < 0) {
        if (errno == EINTR)
            continue;
        perror("Unable to connect with your party");
        close(sockfd);
        sockfd = new_sockfd;
    }

    /* This is just an extra cleanup, so just send it */
    send_delete();
}

/* Routine called on interrupt to re−invite the callee */
void re_invite()
{
    message("Ringing your party again");

    current_line++;
    / * Force a re−announce */
    msg.id_num = htonl(remote_id + 1);
    announce_invite();

    longjmp(invitebuf, 1);
}

static char answers[] = {
    "Your party is not logged on",
    "Target machine does not recognize us",
    "Target machine cannot handle remote talk",
    "Target machine is too confused to talk to us",
    "Target machine indicates protocol mismatch",
    "Target machine indicates protocol botch (addr)",
    "Target machine indicates protocol botch (ctl addr)"
};
#define NANSWERS (sizeof answers / sizeof answers[0])

/* Transmit the invitation and process the response */
void announce_invite()
{
    CTL_RESPONSE response;

    current_state = "Trying to connect to your party’s talk daemon";
    ctl_transact(his_machine_addr, msg, ANNOUNCE, &response);
    remote_id = response.id_num;

    if (response.answer != SUCCESS) {
        if (response.answer < NANSWERS)
            message(answers[response.answer]);
        quit();
    }

    /* Leave the actual invitation on my talk daemon */
    ctl_transact(my_machine_addr, msg, LEAVE_INVITE, &response);
    local_id = response.id_num;

    /* Tell the daemon to remove your invitation */
    send_delete();
}
```

if (sendto(ctl_sockt, &msg, sizeof(msg), 0, &daemon_addr, sizeof(daemon_addr)) != sizeof(msg))
    perror("send_delete (remote)");
msg.id_num = htonl(local_id);
daemon_addr.sin_addr = my_machine_addr;
if (sendto(ctl_sockt, &msg, sizeof(msg), 0, &daemon_addr, sizeof(daemon_addr)) != sizeof(msg))
    perror("send_delete (local)");
/* We can't make the tty non-blocking, because * curses's output routines would screw up */
ioctl(0, FIONREAD, (struct sgtytcb *) &nb);
nb = read(0, buf, nb);
display(my_win, buf, nb);
/* might lose data here because socket is non-blocking */
write(sockt, buf, nb);
}
}

extern int errno;
extern int sys_nerr;
extern char *sys_errlist[];

/* p_error prints the system error message on the standard location * on the screen and then exits. i.e. a curses version of perror */
p_error(string)
char *string;
{
char *sys;
sys = "Unknown error";
if (errno < sys_nerr)
    sys = sys_errlist[errno];
wmove(my_win.x_win, current_line%my_win.x_nlines, 0);
wprintw(my_win.x_win, "[%s : %s (%d)]
", string, sys, errno);
move(LINES-1, 0);
refresh();quit();
}

/* Display string in the standard location */
message(string)
char *string;
{
wmove(my_win.x_win, current_line%my_win.x_nlines, 0);
wprintw(my_win.x_win, "[\%s]
", string);
wrefresh(my_win.x_win);
}

May 28, 02 16:22 io.c
Printed by Jacques Beigbeder

May 28, 02 16:22 look_up.c
Printed by Jacques Beigbeder

May 28, 02 16:22 io.c

Copyright (c) 1983 Regents of the University of California.
* All rights reserved. The Berkeley software License Agreement
* specifies the terms and conditions for redistribution.
*/

#ifndef lint
static char sccsid[] = "@(#)look_up.c 52 (Berkeley) 3/13/86"
#endif not lint

#include "talk_ctl.h"

/* See if the local daemon has an invitation for us. */
check_local()
{
    CTL_RESPONSE response;
    register CTL_RESPONSE *rp = &response;
    /* the rest of msg was set up in get_names */
    msg.ctl_addr = *(struct sockaddr *)ctl_addr;
    msg.ctl_addr.sa_family = htons(msg.ctl_addr.sa_family);
    /* must be initiating a talk */
    if (!look_for_invite(rp))
        return (0);
    /* There was an invitation waiting for us, so connect with the other (hopefully waiting) party */
    current_state = "Waiting to connect with caller";
    do |
        if (rp->addr.sa_family != AF_INET)
            p_error("Response uses invalid network address");
        errno = 0;
        if (connect(sockt, &rp->addr, sizeof(rp->addr)) != -1)
            return (1);
    while (errno == EINTR);
    if (errno == ECONNREFUSED) |
        /* The caller gave up, but his invitation somehow was not cleared. Clear it and initiate an * invitation. (We know there are no newer invitations, * the talkd works LIFO.) */
        ctl_transact(his_machine_addr, msg, DELETE, rp);
        close(sockt);
        open_sockt();
        return (0);
    p_error("Unable to connect with initiator");
    /*NOTREACHED*/
|
/* Look for an invitation on 'machine' */
look_for_invite(rp)
    CTL_RESPONSE *rp;
    {
        struct in_addr machine_addr;
        current_state = "Checking for invitation on caller's machine";
        ctl_transact(his_machine_addr, msg, LOOK_UP, rp);
        /* the switch is for later options, such as multiple invitations */
        switch (rp->answer) |
            case SUCCESS: |
                msg.id_num = htonl(rp->id_num);
                break;
|
 Wednesday March 02, 2011
 17/21

 Wednesday March 02, 2011
18/21
/* there wasn't an invitation waiting for us */
return (0);
}

/* A package to display what is happening every MSG_INTERVAL seconds
  * if we are slow connecting. */

#define MSG_INTERVAL 4
#define LONG_TIME 100000

cchar *current_state;
int current_line = 0;

static struct timeval undo = { LONG_TIME, 0};
disp_msg()
{
    message(current_state);
}
start_msgs()
{
    message(current_state);
signal(SIGALRM, disp_msg);
itimer.it_value = itimer.it_interval = wait;
seltimer(ITIMER_REAL, itimer, (struct timeval *)0);
}
end_msgs()
{
    signal(SIGALRM, SIG_IGN);
timerclear(&itimer.it_value);
timerclear(&itimer.it_interval);
seltimer(ITIMER_REAL, itimer, (struct timeval *)0);
}
/* * Copyright (c) 1983 Regents of the University of California. * All rights reserved. The Berkeley software License Agreement * specifies the terms and conditions for redistribution. */

#ifndef lint
char copyright[] = "@(#) Copyright (c) 1983 Regents of the University of California.
All rights reserved."
#endif not lint

#ifndef lint
static char sccsid[] = "@(#)talk.c 5.1 (Berkeley) 6/6/85";
#endif not lint

#include "talk.h"

/* * talk: A visual form of write. Using sockets, a two way * connection is set up between the two people talking. * With the aid of curses, the screen is split into two * windows, and each users text is added to the window, * one character at a time... * * Written by Kipp Hickman * * Modified to run under 4.1a by Clem Cole and Peter Moore * Modified to run between hosts by Peter Moore, 8/19/82 * Modified to run under 4.1c by Peter Moore 3/17/83 */

main(argc, argv)
int argc;
char *argv[];
{
    get_names(argc, argv);
    init_display();
    open_ctl();
    open_sockt();
    start_msgs();
    if (!check_local() )
        invite_remote();
    end_msgs();
    set_edit_chars();
    talk();
}